RAIRO-Oper. Res. 46 (2012) 41–62
DOI: 10.1051/ro/2012007

RAIRO Operations Research
www.rairo-ro.org

ALGORITHMS FOR THE TWO DIMENSIONAL BIN PACKING PROBLEM WITH PARTIAL CONFLICTS

Khaoula Hamdi-Dhaoui[1], Nacima Labadie[1] and Alice Yalaoui[1]

Abstract. The two-dimensional bin packing problem is a well-known problem for which several exact and approximation methods were proposed. In real life applications, such as in Hazardous Material transportation, transported items may be partially incompatible, and have to be separated by a safety distance. This complication has not yet been considered in the literature. This paper introduces this extension called the two-dimensional bin packing problem with partial conflicts (2BPPC) which is a 2BP with distance constraints between given items to respect, if they are packed within a same bin. The problem is NP-hard since it generalizes the BP, already NP-hard. This study presents a mathematical model, two heuristics and a multi-start genetic algorithm for this new problem.

Keywords. Bin-packing, distance constraint, conflicts, genetic algorithm.

Mathematics Subject Classification. 05-XX, 90-XX.

1. Introduction

Packing problems form an important class of combinatorial optimization problems that have been studied under different variants. For a survey, see for example [4]. The Bin packing problem (BP) consists in packing items with one, two or three dimensions in a minimal number of bins. Many packing problems involve

Received July 12, 2011. Accepted March 27, 2012.

[1] ICD – LOSI – University of Technology of Troyes, UMR-STMR-CNRS-6279, Troyes, France.
khaoula.hamdi@gmail.com

Article published by EDP Sciences

© EDP Sciences, ROADEF, SMAI 2012

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

the insertion of two dimensional items. They mostly differ on the objective function to minimize. Among these problems, we find the two-dimensional bin packing problem (2BP) that consists in packing a set of rectangular items into a minimum number of identical rectangular bins.

In some fields (hazardous materials transportation, for example), transported items can be totally incompatible and have then to be transported and stored in different warehouses. In this case, we need to solve a bin packing problem with conflicts (BPC). The BPC is a natural generalization of the BP that aims to pack items into a minimum number of bins subject to incompatibility restrictions. Gendreau *et al.* [7] proposed six heuristics and two lower bounds for the BPC. One heuristic is a direct adaptation of the first fit decreasing (FFD) algorithm of Coffman *et al.* [4]. Three heuristics are based on graph coloring using different procedures. The remaining two heuristics are based on finding cliques using Johnson greedy heuristic [11]. Most of the available studies in the literature present approximation methods for the BPC. The first study has been proposed by Jansen [10]. Recently, Fernandes-Muritiba *et al.* [6] provided different new lower and upper bounds and two approaches to solve exactly the BPC. Khanafer *et al.* [14] developed new lower bounds and heuristics as well.

The two-dimensional version of the bin packing problem with conflicts (2BPC) has been introduced by Khanafer *et al.* [14]. The authors proposed at first lower bounds and pretreatment procedures [13]. The same authors developed later a new approach based on a tree-decomposition [12]. They also achieved a bi-objective study in which the number of violated conflicts and the number of used bins are the two criteria to minimize [13].

The concept of distance constraint in BP problems was already introduced in the literature. In [16], each item has to be separated from all other items and from the bin borders by a given distance associated to the item. There is therefore no contact between the edges of all items. Stoyan and Yoskov [16] proposed a mathematical model and a hybrid method, combining a branch and bound and a reduced gradient method, to solve this problem. They consider both cases where the items are either rectangles or circles. The problem was generalized to the three dimensional case by Stoyan and Chugay [15]. In this last study, items are either cylinders or parallelepiped and the packing area is considered of some given shape. In another variant studied by Beaumont *et al.* [1], the maximal distance between two items belonging to the same bin has to be smaller than a given threshold.

The two dimensional Bin-Packing with Partial Conflicts (2BPPC) is defined as a 2BP in which some items are partially conflicting. When such items are assigned to a same bin, a given safety distance must be kept between them. The introduction of this new problem is motivated by real applications such as hazardous waste collection.

This paper provides a first study for the two-dimensional bin packing problem with partial conflicts (2BPPC). In this work, some classical heuristics usually used to solve the BP problem are adapted to our problem and a multi-start genetic algorithm is also developed. In Section 2, a mathematical model is proposed. In

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

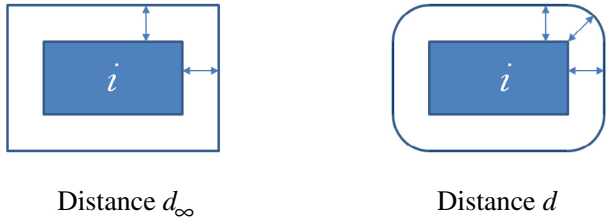


FIGURE 1. Safety distance.

Section 3, two heuristics are developed to obtain the first results for this new problem. In Section 4, the different components of the metaheuristic are detailed. Section 5 is dedicated to computational results and a conclusion ends this paper.

2. Bin packing broblem with partial conflicts

2.1. Problem presentation

A 2BPPC instance consists in a set $A = \{1, \ldots, n\}$ of items which have to be packed in a minimum number of identical bins. A bin is defined by its height H and its width W. An item i has a height h_i and a width w_i ($h_i, w_i \in \mathbb{N}$). A solution of the problem consists in assigning each item i to a bin and defining its position, denoted by (x_i, y_i) which corresponds to the coordinates of its bottom left-hand corner in the bin of its insertion, without overlapping while keeping a safety distance D between partially conflicting items (Fig. 1). The considered distance is denoted d_∞ and is defined as follows: let i and j be two items inserted in the same bin. Their positions in the bin are: (x_i, y_i) and (x_j, y_j). $d_\infty(i, j) = \max\{|x_i - x_j|, |y_i - y_j|\}$.

In this study, the distance d_∞ is chosen because it is the one often used in hazardous material transportation. In this sector, the guideline is to keep a prefixed horizontal or vertical spacing between any two materials partially conflicting. For example, corrosive material and flammable solid material are in partial conflict as indicated in Hazardous material classification data-base [5]. hence when they have to be packed within the same bin (a storage area or a vehicle), a safety distance of 1 horizontal or vertical meter must separate them. Figure 1 shows the difference on using the distance d_∞ and the Euclidean distance d. Further more, the distance constraints (4 and 5) are expressed in a linear way for d_∞, contrary to the case where the Euclidean distance d is used.

In Figure 2, items 1 and 2 are conflicting with item i, that is why a safety distance D is kept between them.

2.2. Model

This section presents a mixed integer linear program (MILP) for the 2BPPC. Like most mathematical programming formulations for 2D-packing problems, only

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

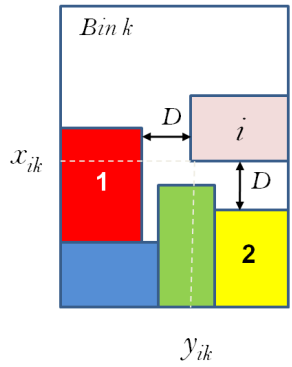


FIGURE 2. Partial conflicts.

very small instances can be handled by commercial MILP software. Hence, the main interest of this model is to provide a compact and unambiguous problem specification.

$$\text{minimize} \sum_{k=1}^{m} v_k \tag{2.1}$$

$$\sum_{i=1}^{n} z_{ik} \leq n * v_k, \qquad \forall k \in \{1, \ldots, m\} \tag{2.2}$$

$$\sum_{k=1}^{m} z_{ik} = 1, \qquad \forall i \in A \tag{2.3}$$

$$x_i + w_i \leq W \qquad \forall i \in A \tag{2.4}$$

$$y_i + h_i \leq H \qquad \forall i \in A \tag{2.5}$$

$$x_i + (w_i + D * b_{ij}) \leq x_j + M * (3 - z_{ik} - z_{jk} - a_{ij}^1) \ \forall i, j \in A, \forall k \in \{1, \ldots, m\} \tag{2.6}$$

$$x_j + (w_j + D * b_{ij}) \leq x_i + M * (3 - z_{ik} - z_{jk} - a_{ij}^2) \ \forall i, j \in A, \forall k \in \{1, \ldots, m\} \tag{2.7}$$

$$y_i + (h_i + D * b_{ij}) \leq y_j + M * (3 - z_{ik} - z_{jk} - a_{ij}^3) \ \ \forall i, j \in A, \forall k \in \{1, \ldots, m\} \tag{2.8}$$

$$y_j + (h_j + D * b_{ij}) \leq y_i + M * (3 - z_{ik} - z_{jk} - a_{ij}^4) \ \ \forall i, j \in A, \forall k \in \{1, \ldots, m\} \tag{2.9}$$

$$a_{ij}^1 + a_{ij}^2 + a_{ij}^3 + a_{ij}^4 \geq 1 \qquad \forall i, j \in A \tag{2.10}$$

$$v_k \in \{0, 1\}, \qquad \forall k \in \{1, \ldots, m\} \tag{2.11}$$

$$z_{ik} \in \{0, 1\}, \qquad \forall i \in A, \forall k \in \{1, \ldots, m\} \tag{2.12}$$

$$a_{ij}^l \in \{0, 1\}, \qquad \forall l \in \{1, \ldots, 4\} \forall i, j \in A \tag{2.13}$$

$$x_i, y_i \in \mathbb{R}^+, \qquad \forall i \in A. \tag{2.14}$$

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

We define a Cartesian coordinate system to describe a feasible packing. The origin $(0, 0)$ is located at the bin bottom left corner. The x-axis for horizontal locations and the y-axis for vertical positions respectively coincide with the bin edges. The position of an item i is expressed as the coordinates (x_i, y_i) of its bottom-left corner in the bin to which it is assigned. Let M be a huge integer which can be fixed to $\max(H, W)$. Although the number of used bins is the objective function to minimize, we need to define an upper bound m on the number of bins to limit the model size ($m = n$ in the worst case). Let $B = (b_{ij})$ be the matrix of partial conflicts. b_{ij} is equal to one if and only if the items i and j have to be separated with a minimal given distance D if they are assigned to the same bin, zero otherwise.

The following decision variables are used in the model:

- binary variables v_k, equal to one if and only if the bin k is used;
- binary variables z_{ik}, equal to one if and only if the item i is packed in the bin k;
- real variables x_i and y_i, determine the position of the item i in the used bin;
- binary variables a_{ij}^l, equal to one if and only if the equation l connecting the items i and j is used.

The objective function (1) to minimize is the number of used bins. The constraints (2) ensure that no item is assigned to an unused bin. The constraints (3) guarantee that each item is packed only once, in a single bin. The constraints (4) and (5) make sure that each item is really placed inside the bin to which it is assigned. The respect of conflicts and the packing of items without overlapping are ensured with the constraints (6)–(9). The constraints (10) guarantee that at least one constraint among the last four is satisfied when the two concerned items are assigned to the same bin. Finally, the constraints (11)–(14) fix the nature of the decision variables.

Let consider the problem described in Figure 3. The problem is composed of 5 items pertaining to 3 different classes. Items of classes 1 and 2 are partially conflicting while those of class 3 are compatible with all others items. We start by solving the problem with Cplex solver that gives an exact solution in Figure 4. Some results of the model are given in Section 5.1. The model allows to obtain exact solutions for instances with a small number of items in a reasonable time. For larger instances, heuristic and metaheuristic method are called. In the following sections, the chosen algorithms to approximately solve the 2BPPC are detailed.

3. Heuristics

3.1. Modified bottom-left-fill heuristic (BLF)

This method consists in placing an item into its lowest possible position and left justifying it. The procedure is then reiterated for each item in turn in a given order. This order is obtained by sorting out the items in a decreasing order of their heights. In case of equality, widths are considered and an item with a biggest

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

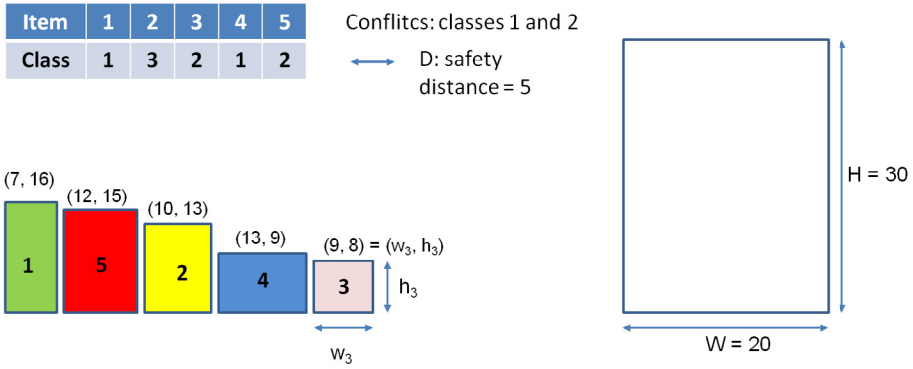


FIGURE 3. An example of bin-packing problem.

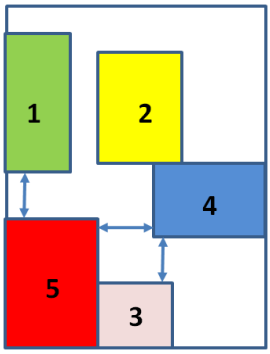


FIGURE 4. Exact resolution of the problem.

value of its width is examined first. At any stage of the heuristic, each used bin contains a set of empty spaces. These spaces are available rectangles; each of them is defined with its height, width and coordinates of its bottom left corner. To place an item, the method proceeds by examining each available rectangle in turn and finding whether the item fits into it and whether it is conflicting with the already packed items that surround the considered rectangle. Rectangles are examined in an increasing order of the ordinates of their bottom left corner. In case of equality, abscissa is considered and the rectangle with lowest value of abscissa is examined first.

In Figure 5, after the insertion of item 1, the used available rectangle is deleted while two new available rectangles are created: *Top* ($W_1 = W, H_1 = H - y_1$) is the available rectangle on top of the inserted item and *Right* ($W_2 = W - x_2, H_2 = H$) is the available rectangle on its right.

In Algorithm 1, *Item* denotes the current item for insertion. Two tests are necessary before packing any item. The first one is *Packing test* which takes the value *true* if the available space can contain the item. The second one is *Conflicts*

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

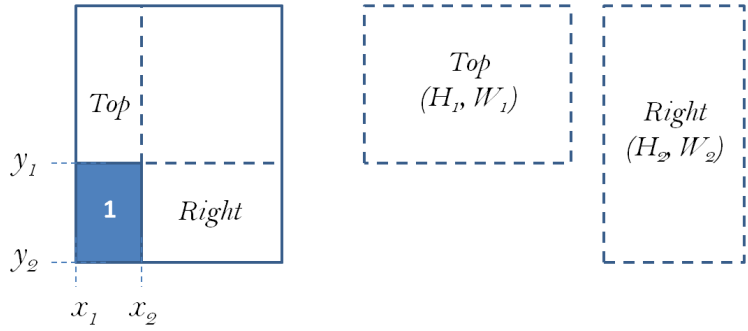


FIGURE 5. Available rectangles.

Algorithm 1. Modified Bottom-left-fill heuristic

```
Initialize a bin with one available rectangle
For Item = 1 to n do
        Repeat
                Go through available rectangles in initialized bins
                if (Packing test (Item) = True) then
                        if (Conflicts test (Item) = True) then
                                Packing (Item)
                                Update available rectangles
                        End if
                End if
        Until ((Item is packed) or (all available rectangles are scanned))
        If (Item is not packed) Initialize a new bin
end for
```

Algorithm 2. Procedure Update available rectangles

```
Create Rectangles (Top) and (Right)
For r = 1 to Nr
        If Rectangle(r) ∩ Rectangle(p) ≠ ∅
        Update Rectangle(r)
        If (W_r = 0 or H_r = 0) Delete Rectangle(r)
end for
Delete Rectangle(p)
Sort out Rectangle
```

test which takes the value *true* if there is a conflict between the item to insert and the already packed items defining the insertion rectangle. Procedure *Packing* fixes item coordinates in the corresponding valid bin, while procedure *Update* is used to create new empty rectangles and erase used ones.

The procedure *Update* is described in Algorithm 2. Nr is the number of available rectangles. Let p be the index of the available rectangle where the current item

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

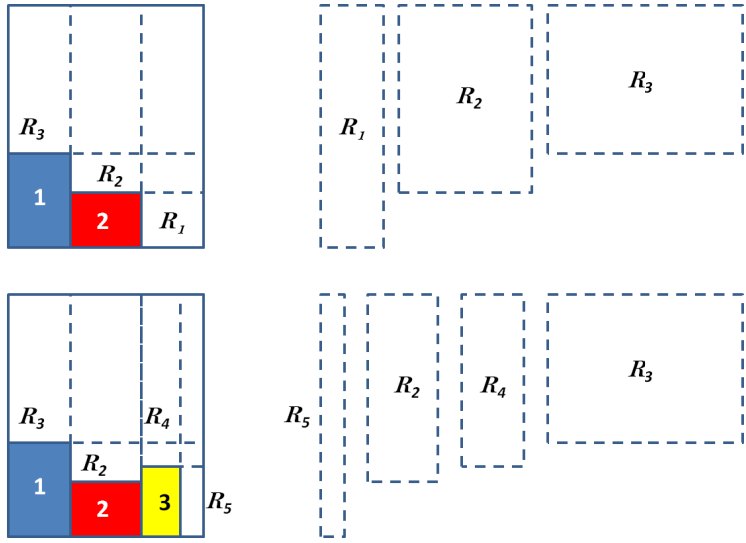


FIGURE 6. Update available rectangles.

is inserted. If there is an intersection between an existing rectangle r and the rectangle p, the dimensions of the rectangle r are modified in order to eliminate the intersection area. If the new height or width of the treated rectangle becomes null, the rectangle is deleted. Two new rectangles are created (*Top* and *right*) and the rectangle p is deleted.

In Figure 6, before inserting Item 3, three rectangles are available, R_1, R_2 and R_3 (see the top part of the figure). Item 3 is inserted in R_1. The rectangle $R2$ intersects $R1$, it is thus updated. The rectangles R_4 and R_5 are created and R_1 is deleted (see bottom part of Fig. 6).

In Figure 7, item 6 is partially conflicting with items 1, 2, 4 and 5. An available rectangle is valid for inserting item 6, if item 6 can fit in the rectangle while keeping D between item 6 and all surrounding items that are conflicting with it. The available rectangle in Figure 7 is valid for inserting item 6 and the position of insertion is (x_6, y_6). If we assume that item 3 is conflicting with item 6, this insertion becomes impossible and the following available rectangles are to be tested.

The Bottom left heuristic, described in Algorithm 1, is modified in order to fulfill the requirement of the new problem. In BLF, if an item conflicts with one or more items surrounding a valid rectangle, a safety distance D has to be kept between the two items. In order to avoid keeping so many unused empty spaces, when the first valid rectangle for packing an item corresponds to a conflicting position, the item is not packed immediately. It is removed from the list, and the following items are examined. The first item that can be packed without keeping a safety distance is then inserted. The removed item is reinserted at the end of the list and is reexamined later.

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

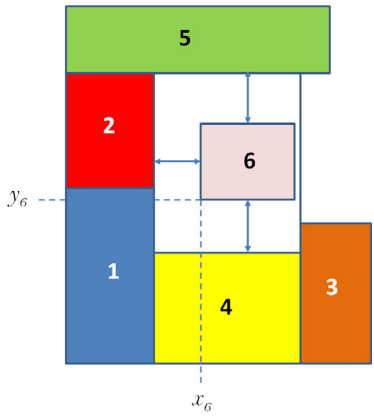
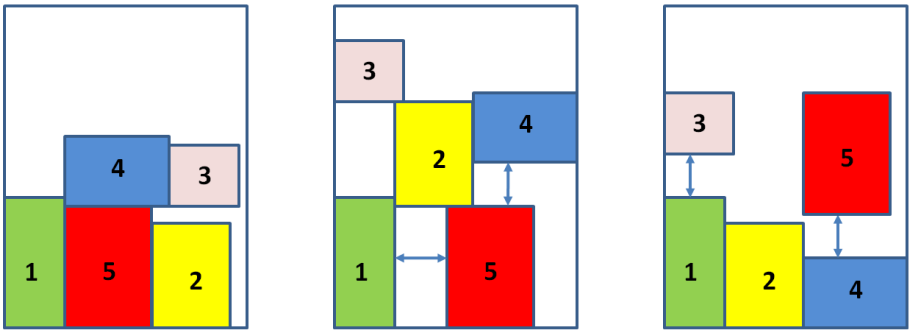


FIGURE 7. Conflicts verification.



(a) BLF on instance without conflicts

(b) BLF on instance with conflicts

(c) Modifizs BLF on instance with conflicts

FIGURE 8. BLF and modified BLF.

In Figure 8, the results obtained with BLF and modified BLF on the problem of Figure 3, with and without partial conflicts, are compared. Figure 8a describes the solution obtained with BLF without considering conflicts. Items are sorted out in a decreasing order of their heights. We notice that three conflicts are violated between items (1, 5), (4, 5) and (3, 4). Figure 8b illustrates the solution given by BLF on the instance with conflicts. A safety distance D is kept between items (1, 5) and (4, 5). In Figure 8c, the problem is resolved with modified BLF. Item 5 is partially conflicting with Item 1 that is already inserted, that is why it is not inserted immediately. It is removed from the list and items 2 and 4 are inserted. Item 3 is partially conflicting with Item 4 which is adjacent to the valid available rectangle for Item 3 insertion. Item 3 is then removed from the list. As there is

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

no more items that can be inserted without keeping a safety distance, Item 5 is inserted. Finally, Item 3 is also inserted.

3.2. Modified shelf heuristic fill – dynamic (SHF-D)

This heuristic was introduced by Ben Massoud *et al.* [2] and is mainly used for guillotine cutting problems. In this kind of problems, the cutting tool has to go from one edge of the rectangle (or the strip to cut) to the other. Shelf algorithms allow to resolve this problem. Packing is obtained by inserting items from the left to the right side, while forming shelfs. The first shelf corresponds to the rectangle bottom. The next shelf bottom is obtained with the horizontal line that coincide with the top of the highest item in the previous shelf. The treatment is repeated for all items.

In the classical version, items are sorted out in a decreasing order of heights. In this modified version, items are separated into sets corresponding to their classes (as in hazardous materials classification, items are assigned to classes and the partial conflicts are rather defined between classes than items). Then they are inserted in a way that the items of two classes treated successively are not conflicting. If it is impossible to obtain an order that separates every pair of conflicting classes, a safety distance is kept when necessary.

The procedure that sorts out the classes starts by assigning the first position to the class that has the biggest degree of conflicts. The degree of conflicts of a class corresponds to the number of classes conflicting with it. After adding a class to the list, the following class is chosen among the classes non conflicting with it.

The procedure of sorting out the classes is described in Algorithm 3. *Classes* stands for the table of classes and represents the input of the algorithm, while *List* is the output corresponding to the new order of the classes. Nc is the number of classes and c and ind are indexes of the classes. *Degree* is a table in which are recorded the degree of the classes. Finally, *Compatible* is a boolean that takes the value *true* if the considered classes are compatible, and the value *false* if not.

Each item is inserted in the lowest possible position left-justified in the first shelf where it fits entirely, if any. If none of the existing shelfs can contain it, a new shelf is initialized. This heuristic is different from other classical shelf algorithms with its tendency to reuse at best the free areas in each shelf, by inserting items on top of the others while preserving guillotine cutting constraints.

The notion of available rectangles that allows to detect all free areas is also used in this method. Conflicts verification is performed in the same manner as in modified Bottom-left-fill. In the dynamic version of the method (SHF-D), a shelf height is not fixed with the height of the first inserted item. A temporary height is updated after each insertion. Let Nr be the number of available rectangles in the current bin and Ns the number of shelfs within it. If the current item does not fit in all the $Nr - 1$ available rectangles and the last rectangle has to be used (the rectangle with the biggest ordinate of bottom-left corner), a new shelf is initialized

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

Algorithm 3. Sorting out classes procedure

Sort out *Classes* in a decreasing order of *Degree*
$c = 1$
List(c) = Classes(c)
Repeat
    $ind = c + 1$
    Compatible = false
    Repeat
        If (*Classes(ind)* conflicts with *List(c)*) then $ind = ind + 1$
        Else *Compatible = true*
    Until ((*Compatible = true*) or $(ind = Nc)$)
    If (*Compatible = true*) then *List(c) = Classes(ind)*
    Else *List(c) = Classes(c+1)*
    $c = c + 1$
Until $(c = Nc)$

Algorithm 4. Fix shelf height procedure

If $(p = Nr)$ then
        Shelf Height(Ns) $= Y_p$
        $Ns = Ns + 1$
        Initialize *Shelf(Ns)* with *Current Item*
else
        Insert *Current Item* in *Shelf(Ns)*
End if

with this item and the height of the previous shelf is fixed. In Algorithm 4, let p be the index of the valid rectangle for insertion and Y_p its ordinate.

Figure 9 illustrates the difference between SHF, SHF-D and modified SHF-D heuristics applied on the problem proposed in Figure 3. In Figure 9a, corresponding to a solution obtained by SHF, the height of item 1 fixes the height of the first shelf that contains it. Item 5 is partially conflicting with item 1, that is why a safety distance separates them. As there is no enough space in the current shelf to insert item 2, this item initializes a new shelf and fixes its height. Item 4 is partially conflicting with Item 5, thus it can not be inserted in the second shelf. It initializes the third shelf. Finally, Item 3 is partially conflicting with items 1 and 4. It is inserted in the second shelf in a position separated from these two items with a safety distance.

In Figure 9b, corresponding to the solution given by SHF-D, the shelf height is not fixed by the first inserted item. We notice that all items have been inserted without fixing the shelf height even once. Safety distances separate items (1 and 5), (5 and 4) and (3 and 4).

In Figure 9c, the problem is resolved with modified SHF-D. Items of class 1 are inserted first, then those of class 3 that is compatible with all classes. Finally, items of class 2 are inserted while keeping safety distance when needed.

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

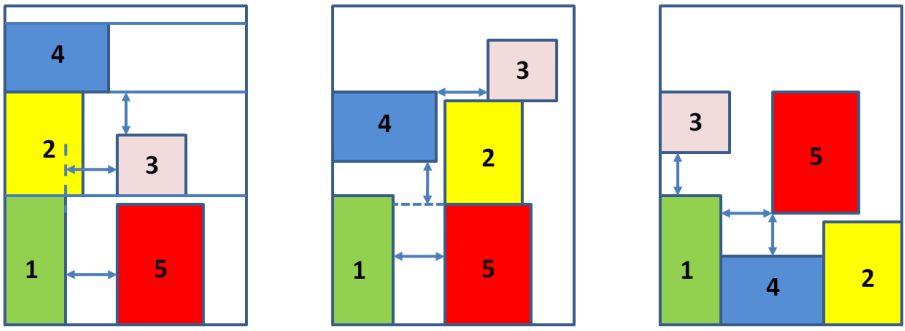


FIGURE 9. SHF-D and modified SHF-D.

4. MULTI-START GENETIC ALGORITHM (MS-GA)

In 1975, Holland [9] described an optimization method, based on analogy to the process of natural selection in biology. The biological basis for the adaptation process is evolution from one generation to the next, based on elimination of weak elements and retention of stronger elements (those with the best performance in the current environment). Of course, over the long term, it is not the strong individuals themselves which ultimately survive, but rather offspring related to them genetically in proportion to their reproductive fitness or success at reproduction. The search over this representation space is performed using so-called "genetic algorithms" (GA).

GAs are now widely recognized as effective search paradigms in several areas, particularly as a method for achieving relatively good solutions to NP-hard optimization problems of high dimensionality in few time. A GA maintains a population of strings (chromosomes) that encode candidate solutions to a problem. These strings are the analog of chromosomes in natural evolution. A fitness function defines the quality of each solution. The GA chooses parent organisms from the population in such a way that the more fit organisms are more likely to be chosen. It applies operators to generate offspring from the parents. Most commonly used operators are crossovers, which combine genetic material from two parents, and mutations, which randomly make local modifications in the chromosomes.

In this section, the developed Genetic algorithm to solve the two-dimensional Bin Packing Problem with Partial Conflicts is detailed. The general framework of the method is given in Algorithm 2. *NP* represents the population size. *Population Generation* denotes the procedure that generates chromosomes and calls the packing heuristic to build the corresponding initial solutions. *Select* is a randomized selection process of selection of pair of parents candidates to generate new offspring with the procedure *Crossover*. The procedure *Mutation* performs a local

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

Algorithm 5. Multi-start genetic algorithm

```
Gen = 0
Best = Infinity
count = Infinity
Repeat
        Gen=Gen+1
        If (count > Cmax) then Population Generation
        Repeat
                Selection(P1)
                Selection(P2)
                New offspring = Crossover(P1, P2)
                Mutation(New offspring)
        until (NP offsprings are created)
        If (Fitness(Best new offspring) < Best) then
                Best = Fitness(Best new offspring)
                count = 0
        end if
        Else count = count +1
        Sort Out(Population = Parents and Offspring)
        New Population = (NP Best solutions)
until ((Gen = NG) or Lower bound attained)
```

modification in a given solution. *NG* is the maximal number of generations and *Gen* is the corresponding counter. *Sort Out* is the procedure in charge of sorting out the list *Population* in increasing order of fitness (the fitness of a solution is the number of bins it uses to pack all items). *C*max is the maximum number of created solutions without improving the best found solution. The corresponding counter is *count*.

At each step, two parents are selected randomly using binary tournament. The parent with best quality in the pair is kept and the operation is repeated to obtain the second parent. *NP* crossovers are performed to obtain *NP* offspring. All solutions are sorted out (old and new ones), and the *NP* best solutions are conserved for the next step. The population is reinitialized after creating *C*max populations without improving the best solution. A new restart of the metaheuristic is then recorded. The process is repeated *NG* times. It can be stopped well before all iterations are performed, if a lower bound is achieved. In our implementation, a simple lower bound for the bin packing problem is used: $LB = \sum_{i=1}^{n} \frac{w_i h_i}{WH}$.

4.1. Initial solutions and chromosome's encoding

Different manners to encode the chromosomes have been proposed for bin-packing problems. Here, we chose an encoding that allows obtaining a feasible solution without reparation. Our chromosomes represent the order according to which items are treated by the packing heuristic. A packing heuristic is then called to obtain the solution.

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

To create initial solutions, modified BLF heuristic is chosen. In fact, for each given chromosome, modified BLF can build a different solution. While in modified SHF-D, items are treated in a given order that takes into account the conflicts between classes, that is why only a limited number of initial solutions can be generated. Initial chromosomes are then generated randomly and the corresponding solutions are given by modified BLF.

4.2. Test of conflicts

Conflicts are managed when constructing solutions with the packing heuristic. The test of conflicts consists in scanning the list of already inserted items and focus on their insertion positions. The compatibility of all the items directly surrounding the available rectangle for packing the current item is verified. If there exists a conflict with at least one surrounding item, a safety distance D has to be kept between the two items.

4.3. Crossover

The chromosomes of a given population are sorted out in increasing number of used bins (fitness function). If equality, the solution with higher maximum filling rate is considered of better quality. The filling rate of a bin corresponds to the quotient between the full surface and the total surface of the bin. The maximum filling rate of a solution is the highest filling rate of its bins. In the crossover, the parent with a bigger maximum filling rate is considered as the basis to build the child offspring. It is called *P1*. The second parent *P2*, serves to complete the solution.

For each item, the touching perimeter obtained with its position in *P1* is computed. It consists in the total length of its edges that touches other items in the same bin. The borders of the bin are not taken into account. The touching rate of an item is computed as the quotient of its touching perimeter by its perimeter.

Each item with a touching rate bigger than a fixed level α, appears in the offspring, with its surrounding items, in the same order as in *P1*. The items pertaining to more than one conserved sequence, are represented only once (the items that appear in more than one sequence are represented only one time and the copies are eliminated). The solution is then completed with the remaining items, according to their appearance order in *P2*. Only one offspring is obtained with each crossover.

In the example given in Figure 10, item 4 has an interesting touching rate ($\geq \alpha = 75\%$ for example where α is a parameter of the algorithm), in the solution corresponding to $P1$. It is in contact with *items* 1, 2, 3 and 6. *Item* 4 and its surrounding items appear then in the offspring $O1$ in the same order as in *P1* (3, 2, 1, 4, 6). The remaining items (5, 7, 8) are added to the child solution according to their order of appearance in *P2*: (8, 7, 5).

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

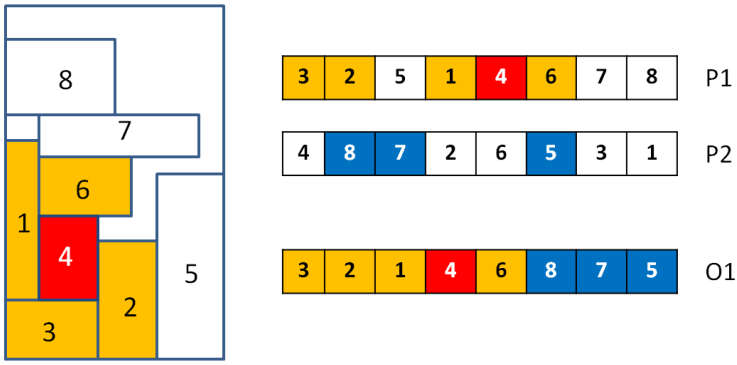


FIGURE 10. Example of crossover.

4.4. MUTATION

The mutation procedure is a local modification applied to a chromosome in order to bring a part of diversification. It allows to direct the research to a new area in the search space and escape from local optima. The used mutation is item relocation. It consists in removing an item from its current position in the chromosome to another position in the list. The position of the item to relocate and the insertion position are determined randomly according to a uniform distribution in $[1, n]$ where n is the number of elements in a chromosome. The mutation procedure is applied with a given probability (see Sect. 5.2.1).

5. EXPERIMENTAL RESULTS

Our algorithm was tested on Berkey and Wang instances [3]. The instances are composed of five groups that differ in the number of items ($n = 20, 40, 60, 80, 100$). Each group contains ten different instances. The instances characteristics are as follows:

- Set 1: w_i and h_i are generated randomly according to uniform distribution in $[1, 10]$. $W = H = 10$.
- Set 2: w_i and h_i are generated randomly according to uniform distribution in $[1, 10]$. $W = H = 30$.
- Set 3: w_i and h_i are generated randomly according to uniform distribution in $[1, 35]$. $W = H = 40$.
- Set 4: w_i and h_i are generated randomly according to uniform distribution in $[1, 35]$. $W = H = 100$.
- Set 5: w_i and h_i are generated randomly according to uniform distribution in $[1, 100]$. $W = H = 100$.
- Set 6: w_i and h_i are generated randomly according to uniform distribution in $[1, 100]$. $W = H = 300$.

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

TABLE 1. Matrix of partial conflicts.

Classes	1	2	3	4	5
1	0	0	0	0	0
2	0	0	1	1	1
3	0	1	0	0	1
4	0	1	0	0	1
5	0	1	1	1	0

TABLE 2. Results of the mathematical model.

			Without conflicts		With conflicts
Sets	N	OPT	N bins	N Opt	N bins
Set 1	20	7.1	7.7	4	9.2
Set 2	20	1	1	10	1.5
Set 3	20	5.2	5.3	8	6
Set 4	20	1	1	10	1
Set 5	20	6.5	6.9	6	7.2
Set 6	20	1	6.7	0	7.6

To meet the requirement of our problem, a new benchmark using these instances was generated by adding partial conflicts. A class index is randomly assigned to each item according to uniform distribution. We consider 5 possible classes between which partial conflicts are defined according to the symmetric matrix in Table 1:

All studied algorithms were developed with C++ and run on a PC Pentium(R) Dual-Core CPU, 2.20 GHz and 4 Go of RAM.

5.1. Mathematical model results

The model proposed in Section 2.2 has been tested on the instances of Berkey and Wang with a number of items limited to 20. The maximum computing time was fixed to 5 min (300 s). In Table 2 the results are reported. The column *N bins* corresponds to the average number of used bins for each group of 10 instances with a number of items equal to 20. The column N Opt gives the number of instances for which the optimal solution is reached by the model.

In limited computing time to 5 min, the model is not able to find all optimal solutions for instances without conflicts. Only for Sets 2 and 4, 10 optimal solutions out of 10 are reached. Instances of Set 6 are the most difficult to resolve, since no optimal solution was found and the gap between OPT and N bins is very important (670%). When the maximum computing time is raised to 900 s, the gap drops to 480%. The same behavior is observed for the instances with conflicts: the gap drops from 760% to 580%.

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

TABLE 3. Latin square.

	NP	NG	Alpha	F
1	(−)	(−)	(−)	1039
2	0	(−)	0	1035
3	(+)	(−)	(+)	1035
4	(−)	0	0	1036
5	0	0	(+)	1031
6	(+)	0	(−)	1031
7	(−)	(+)	(+)	1032
8	0	(+)	(−)	1030
9	(+)	(+)	0	1029

5.2. Heuristics and multi-start genetic algorithm results

5.2.1. Parameter setting

The parameter $C\max$ is determined in function of NG and was set to $\min(NG/4, 100)$. For values of NG smaller than 100, $C\max = NG/4$. When NG takes a big value, the population is initialized after 100 iterations without improving the best solution. The mutation is applied with a probability of 10%. This rate corresponds to a reasonable level of diversification that allows to modify only a small number of chromosomes and avoids random dispersion of the population.

To obtain the values of the remaining parameters, we start by dressing a design of experiments. This procedure allows to estimate the effect of different parameters on the obtained results. 3 parameters are concerned by the study: *NP, NG* and α. Each parameter can take one of three possible values: high, low and neutral. $NP = \{10, 20, 30\}$, $NG = \{20, 50, 100\}$ and $\alpha = \{0.6, 0.7, 0.8\}$. The applied plan is the Latin square with 9 possible situations [8]. In Table 3, these 9 situations and the corresponding results are explained. The tests are realized for all the 50 instances of the first set. The given results correspond to the sum of used bins in these instances. The sign (+) denotes the high position of each parameter and (−) the low one, while 0 is the neutral position.

Let F be a vector of 9 responses. Each response F_i corresponding to a different situation among the 9 situations of Table 3 can be explained with a limited development, in function of the parameters (x_j) as follows:

$$F_i = b_{i0} + \sum_j b_{ij} x_j + e \tag{5.1}$$

where b_{i0} is a constant and e is a polynomial function of (x_j) of a degree at least equal to 2. As the considered parameters are independent, the response F_i can be approximated by a linear function (the value of e aims towards zero). Let b be the vector of the coefficients (b_{ij}). An estimation of b is computed with the following expression, where X^T is a transposition of the matrix of parameters X, $X^T X$ is the matrix of information, $(X^T X)^{-1}$ is the matrix of dispersion and F

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

TABLE 4. Coefficients computing.

	NP	NG	α
Coeffecient	–2	–3	–0.33
Importance	38%	56%	6%

the obtained response. The matrix X is obtained by replacing the signs (+) and (–) in Table 3 by the values (+1) and (–1).

$$b = (X^T X)^{-1} X^T F. \tag{5.2}$$

In Table 4, the obtained coefficients for the studied responses are recorded. According to variance analysis, The second parameter (NG) has the most important coefficient that represents 56% of the model's responses. As seen in Table 3, the best results are obtained with $NG = 100$. The second important parameter is NP with a normalized coefficient of 38%. With $NG = 100$, we notice that the results are improved when NP increases. Finally, the last parameter α can be fixed to 0.7 as this value gives the best obtained result within preliminary tests. For the following tests, the value of NG will be set to 10 000 as it corresponds to the most interesting parameter and the results quality improves when NG increases. NP and α will be set to 30 and 0.7.

5.2.2. Instances without conflicts

In order to estimate the algorithm's quality, it has been tested on 2BP instances without conflicts from the literature. In Table 5, results of tests on the instances of Berkey and Wang without conflicts are detailed. For each line, an average result for 10 instances with the same number of items is computed. First, we report in the column *BLF* the results obtain with this heuristic. Optimal solutions available in the literature are reported [17] in column *Optimal*. The remaining columns concern our MS-GA. For each instance, 10 runs are performed and the minimum value (N Min), the maximum value (N Max) and the average value (N Average) of the objective function are recorded. The average computing time for each group of instances is given in the column *CPU* Average. The column N Opt is the number of optimal solutions found while considering the best results of our MS-GA (N Min). Finally, *Gap* determines the gap between N Min and *Optimal*.

We notice that for all sets of instances, the heuristic results are worse than the worst results obtained with the MS-GA recorded in the column N Max. We compute an average gap of 8.5% between *BLF* and N Max, of 12.11% between *BLF* and N Min and of 14.05% *BLF* and *Optimal*. The percentage of optimal solutions found by *BLF* ranges from 6% for Set 5 to 76% for Set 6, with an average of 35%.

For the best results of our MS-GA (N Min), the observed distance to optimal solutions ranges from 0% to 11.7%. The most important values are noticed for instances with small number of bins in the optimal solution. In this case, every supplementary bin has a great impact on the final gap. In average, N Min is only

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

TABLE 5. Results of BLF and MS-GA on the instances without conflicts.

		BLF	Optimal	*N*Min	*N*Max	*N*Average	CPU Average	*N*opt	Gap
Set 1	20	7.9	7.1	7.1	7.1	7.1	33.4	10	0%
	40	15	13.4	13.4	13.9	13.59	90.2	10	0%
	60	21.9	20	20.1	20.8	20.32	182.8	9	0.5%
	80	30.4	27.5	27.5	28.4	27.79	302.1	10	0%
	100	35.2	31.7	32.1	33.6	32.67	443.2	6	1.3%
	Average	22.08	19.94	20.04	20.76	20.29	210.34	9	0.5%
Set 2	20	1.6	1	1	1	1	39.2	10	0%
	40	2.2	1.9	1.9	2	1.97	109.2	10	0%
	60	3	2.5	2.5	2.7	2.55	197.1	10	0%
	80	4	3.1	3.2	3.3	3.25	309.5	9	3.2%
	100	4.5	3.9	4	4.2	4.11	448.3	9	2.6%
	Average	3.06	2.48	2.52	2.64	2.58	220.66	9.6	1.6%
Set 3	20	5.9	5.1	5.1	5.4	5.24	35	10	0%
	40	11	9.4	9.4	9.9	9.57	90.8	10	0%
	60	16.1	13.9	13.9	14.9	14.34	175.2	10	0%
	80	22.3	18.9	19.1	20.4	19.66	277.8	8	1%
	100	26.2	22.3	23.4	24.2	23.71	405.7	4	4.9%
	Average	16.3	13.92	14.18	14.96	14.5	196.9	8.4	1.9%
Set 4	20	1.3	1	1	1	1	37.3	10	0%
	40	2.1	1.9	1.9	2	1.94	100.3	10	0%
	60	2.9	2.5	2.5	2.7	2.59	177.9	10	0%
	80	3.6	3.2	3.2	3.4	3.3	275.3	10	0%
	100	4.4	3.8	3.9	4.2	4.04	397.2	9	2.6%
	Average	2.86	2.48	2.5	2.66	2.57	197.6	9.8	0.8%
Set 5	20	7.3	6.5	6.5	6.5	6.5	33.9	10	0%
	40	14	11.9	12.1	12.2	12.11	88.1	8	1.7%
	60	20.2	18	18.4	18.6	18.5	173.6	6	2.2%
	80	27.8	24.7	25.3	25.4	25.37	291.3	4	2.2%
	100	32.4	28.1	29.6	29.9	29.71	414.3	0	5.3%
	Average	20.34	17.84	18.38	18.52	18.44	200.24	5.6	3%
Set 6	20	1	1	1	1	1	39.2	10	0%
	40	2.1	1.7	1.9	1.9	1.9	114.4	8	11.7%
	60	2.6	2.1	2.2	2.3	2.23	197.8	9	4.7%
	80	3.2	3	3	3	3	290.8	10	0%
	100	3.8	3.4	3.4	3.6	3.5	414.9	10	0%
	Average	2.54	2.24	2.3	2.36	2.33	211.42	9.4	2.7%
Average		11.2	9.82	9.99	10.32	10.12	206.19	8.63	1.73%

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

1.73% distant from optimal solutions, while N Average has a gap of 3.47%. We notice that the set 5 is the most difficult to resolve. Indeed, for this group no optimal solution was reached in Set 5 – N 100. Finally, the number of optimal solutions found is 259 for the best results N Min (out of 300). In average, 86.3% of optimal solutions are reached by N Min. The values recorded by N Max are only 3.94% distant from N Min which shows the robustness of our MS-GA.

For the computing time, it does not depend on the Set of instances but on the number of items per instance. For instances with only 20 items, an average computing time of 35 seconds is recorded, while it can reach 440 s for instances with 100 items (7 min).

5.2.3. Instances with conflicts

The results of the genetic algorithm and the two presented heuristics on instances with partial conflicts are given in Table 6. Berkey and Wang instances were adapted by randomly assigning a class index (corresponding to a Hazardous material type) to each item. The conflicts are considered between classes (see Tab. 1). As for instances without conflicts, 10 runs are performed for each instance.

The two developed heuristics have very small computing time ($<10^{-4}$ s). In average, the modified SHF-D gives the best heuristic results. The MS-GA is run with the parameters values determined earlier, in Section 5.2.1. The best results of our MS-GA allow to improve the best heuristic results obtained with modified SHF-D with an average of 12.03%, while the improvement of modified BLF solutions reaches 20.6% (Tab. 6, column *Imp.*). The number of instances improved by the metaheuristic is reported in column N Imp., with regards to modified BLF solutions. The percentage of improved instances ranges from 14% (Sets 4 and 6) to 72% (Set 1), with an average of 44%.

The additional conflicts constraints lead to an average of 7% more bins compared to the case without conflicts. These results traduce the importance of this supplementary constraint of partial conflicts and its impact on packing problems. The number of instances for which the number of bins have not changed is 138 (out of 300), which represents 46% of all instances. The Set 3 is the most constrained with only 18% of its instances with conflicts for which the number of used bins did not change. For the Set 6, the partial conflicts are less important, as 78% of its instances with conflicts are resolved without raising the number of required bins.

The computing time for the instances with conflicts is higher than the one required for instances without conflicts. Especially, instances of Set 2 are too much time consuming (average of 28 min), while the average for remaining instances does not exceed 10 minutes (for 10 000 generations).

6. Conclusion and perspectives

This paper introduces a new packing problem: the two-dimensional packing problem with partial conflicts. A mathematical model is proposed and two heuristics are developed to obtain the first results. Then a multi-start genetic algorithm

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

TABLE 6. Results of the heuristics and MS-GA on the instances with conflicts.

		Heuristics		MS-GA					
		BLF-M	SHFD-M	*N*Min	*N*Max	*N*Average	CPU	Imp.	*N*Imp.
Set 1	20	8.5	8.2	7.4	7.4	7.4	46	14.86%	4
	40	15.5	15.1	14	14	14	145	10.71%	9
	60	23.1	22.6	21.2	21.9	21.3	306.4	8.96%	6
	80	31.1	30.6	29	30.1	29.4	553.2	7.24%	9
	100	36.9	35.6	33.9	34.7	34.4	864.7	8.85%	8
	Average	23.02	22.42	21.1	21.62	21.3	383.06	9.1%	7.2
Set 2	20	2	2	1.4	1.4	1.4	93.3	42.86%	6
	40	3.1	2.5	2.1	2.1	2.1	569.1	47.62%	4
	60	3.9	3.5	3	3	3	1346.6	30%	4
	80	4.8	4.4	4	4	4	2491.1	20%	3
	100	5.3	5	4.1	4.5	4.2	4134.6	29.27%	6
	Average	3.82	3.48	2.92	3	2.94	1726.94	33.95%	4.7
Set 3	20	6.6	6.3	5.5	5.5	5.5	35	20%	5
	40	11.7	11.5	10.1	10.1	10.1	124.3	15.84%	9
	60	17	16.4	15.1	15.8	15.4	248.1	12.58%	8
	80	23.2	22.5	21	21.8	21.3	456.2	10.48%	6
	100	26.7	26.6	24.8	25.7	25.2	731.3	7.66%	7
	Average	17.04	16.66	15.4	15.8	15.56	246.26	13.31%	7
Set 4	20	1.6	1.4	1	1	1	65.3	60%	2
	40	2.3	2.4	2	2	2	228.3	15%	2
	60	3.5	3	2.9	3.1	3	472.9	20.69%	1
	80	4.5	3.9	3.7	3.9	3.8	870.4	21.62%	1
	100	5.1	4.4	4.4	4.4	4.4	1344.8	15.91%	1
	Average	3.4	3.02	2.8	2.88	2.84	583.28	26.64%	1.4
Set 5	20	7.1	7.1	6.6	6.6	6.6	41.7	7.58%	3
	40	13.3	13.8	12.5	12.5	12.5	123.4	6.40%	5
	60	20.7	20.4	19	19.5	19.2	252	8.95%	6
	80	28	27.4	26	26.7	26.3	450.1	7.69%	6
	100	32.6	32.1	30.7	31.4	31.1	689.3	6.19%	3
	Average	20.34	20.16	18.96	19.34	19.14	311.3	7.36%	4.7
Set 6	20	1.4	1.2	1	1	1	24	40%	1
	40	2.1	2	1.9	1.9	1.9	215.5	10.53%	1
	60	3	2.6	2.5	2.5	2.5	481.8	20%	1
	80	4	3.3	3.3	3.3	3.3	771.8	21.21%	3
	100	4.5	3.9	3.7	3.9	3.8	1144.6	21.62%	1
	Average	3	2.6	2.48	2.52	2.5	522.74	22.67%	1.4

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press

is designed to improve heuristic results. For instances with conflicts, the genetic algorithm improves modified SHF-D results by 12.03% and 44% of instances results are improved.

The metaheuristic was also tested on instances without conflicts in order to evaluate its performance with regards to existing packing methods. The results are only 1.73% distant from optimal solutions. It succeeds to find 259 optimal solutions out of 300, which corresponds to 86.3%, in an average computing time of 206 s.

References

[1] O. Beaumont, N. Bonichon and H. Larchevêque, *Bin packing under distance constraint.* Technical Report, Université de Bordeaux, Laboratoire Bordelais de Recherche en Informatique, INRIA Bordeaux Sud-Ouest (2010).

[2] S. Ben Messaoud, C. Chu and M.L. Espinouse, An approach to solve cutting stock sheets. *Scottish Mathematical Council* **6** (2004) 5109–5113.

[3] J.O. Berkey and P.Y. Wang, Two dimensional finite bin packing algorithms. *J. Oper. Res. Soc.* **38** (2004) 423–429.

[4] E.G. Coffman, M.R. Garey and D.S. Johnson, Approximation algorithms for bin-packing – an updated survey, in *Algorithm design for computer system design,* edited by G. Ausiello, M. Lucertini and P. Serafini. Springer, Vienna (2007).

[5] Environment Canada, Compliance promotion bulletin (Compro No. 12), *regulations for the management of hazardous waste* (2002).

[6] A.E. Fernandes-Muritiba, M. Iori, E. Malaguti and P. Toth, Algorithms for the bin packing problem with conflicts. *Informs J. Comput.* **22** (2010) 401–415.

[7] M. Gendreau, G. Laporte and F. Semet, Heuristics and lower bounds for the bin packing problem with conflicts. *Comput. Oper. Res.* **31** (2004) 347–358.

[8] J. Goupy, Les plans d'expériences. *Revue Modulad* **34** (2006) 74–116.

[9] J.H. Holland, *Adaptation in natural and artifficial systems.* University of Michigan Press, Ann Arbor, MI (1975) 1–211.

[10] K. Jansen, An approximation Scheme for Bin Packing with conflicts, *Lect. Notes Comput. Sci.* **1432**. Springer, Berlin (1998).

[11] D. Johnson, Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.* **9** (1974) 272–314.

[12] A. Khanafer, F. Clautiaux and E.G. Talbi, Tree-decomposition based tabu search for the bin packing problems with conflicts, in *Metaheuristics International Conference, MIC09.* Hamburg, Germany (2009).

[13] A. Khanafer, F. Clautiaux and E.G. Talbi, *Algorithmes pour des problèmes de bin-packing mono et multi-objectif.* Ph.D. thesis, Université des Sciences et Technologies de Lilles (2010).

[14] A. Khanafer, F. Clautiaux and E.G. Talbi, New lower bounds for bin packing problems with conflicts. *Eur. J. Oper. Res.* **206** (2010) 281–288.

[15] Y.G. Stoyan and A. Chugay, Packing cylinders and rectangular parallelepipeds with distances between them into a given region. *Eur. J. Oper. Res.* **360** (2009) 446–455.

[16] Y.G. Stoyan and G.N. Yaskov, Mathematical model and solution method of optimization problem of placement of rectangles and circles taking account special constraints. *Int. Trans. Oper. Res.* **5** (1998) 45–57.

[17] Università di Bologna D.E.I.S., Operations Research, http://www.or.deis.unibo.it/research.html.

https://doi.org/10.1051/ro/2012007 Published online by Cambridge University Press