A tailored two-phase constructive heuristic for the three-dimensional Multiple Bin Size Bin Packing Problem with transportation constraints

C. Paquay[a,*], S. Limbourg[a], M. Schyns[a]

[a] *University of Liege (ULg), HEC Management School, QuantOM*

Abstract

This paper considers the three-dimensional Multiple Bin Size Bin Packing Problem which consists in packing a set of cuboid boxes into containers of various shapes with minimising unused space. The problem is extended to air cargo where bins are Unit Load Devices, especially designed for fitting in aircraft. We developed a fast constructive heuristic able to manage the different constraints met in transportation. The heuristic is split into two distinct phases. The first phase deals with the packing of boxes into identical bins using an extension of the Extreme Points. During this phase, the fragility, stability and orientations of the boxes are taken into account as well as the special shape of the bins and their weight capacity. The second phase takes into account the multiple types of available bins. If necessary, the best found loading pattern is finally enhanced with respect to weight distribution in a post processing. After parametrisation, computational experiments have been performed on data sets especially designed for this application. The heuristic requires really short computational times to achieve promising results.

Keywords: Packing, Heuristics, Air transportation, Extreme points

1. Introduction

The subject of this paper is to solve the problem of packing a set of cuboid boxes into containers of various shapes with minimising the unused space. There are few identical boxes and all have to be loaded. Regarding the containers, this work deals with the specific case of air cargo. These are then Unit Load Devices (ULD) and can be described as an assembly of components consisting of a container or of a pallet covered with a net, whose purpose is to provide standardised size units for individual pieces of baggage or cargo, and to allow for rapid

*Corresponding author

loading and unloading (Limbourg et al. (2012)). ULDs may have specific shapes to fit inside aircraft, such as truncated rectangular parallelepipeds. There are thus only several types of available bins. This problem is a combinatorial optimisation problem which has been labelled as a three-dimensional Multiple Bin Size Bin Packing Problem (MBSBPP) using the typology defined by Wäscher et al. (2007). Depending on the chosen application, other objectives can be defined such as minimising the costs of the selected ULDs. Obviously, this work can be extended to many other packing applications. For instance, ULDs can be replaced by simple Euro pallets or by the loading volume of vehicles. This specific problem has been formulated as a MIP in Paquay et al. (2016). However, the Bin Packing Problem is a NP-hard problem (Garey and Johnson (1979)) and limited results were obtained due to the problem complexity. The goal of the present work is to develop a method able to compute a good feasible packing in short computational times.

The contribution of this paper is threefold. The first contribution is the set of constraints taken into account for the MBSBPP in a constructive heuristic. In the final loading pattern, all the boxes have to be packed without overlap and lie within the ULD. The weight capacity of each loaded ULD has to be respected. The boxes can orthogonally rotate, i.e. the edges of the boxes have to be either parallel or perpendicular to those of the ULDs. Sometimes only a limited number of orientations are allowed due to the contents of the boxes. For the same reason, some boxes may be fragile and are not able to support other boxes. Moreover, each box of the loading pattern has to be correctly supported to be stable. In air transportation, when ULDs are packed inside the airplane, the centre of gravity is computed assuming each ULD has a centre of gravity close to the geometrical centre of its basis, i.e. the weight distribution is uniform. This is therefore included in our problem as a hard constraint. These constraints make the problem very specific and not deeply studied in the literature. More information about the constraints met in three-dimensional packing problems can be found in Bortfeldt and Wäscher (2013).

The second contribution is to propose a constructive heuristic composed of two main phases. During the first phase, a packing algorithm provides a loading pattern for a given set of boxes and a given ULD type. The second phase considers the different available types of ULDs and uses the packing algorithm as a subroutine. This second phase creates a lot of loading patterns. Among the feasible patterns, the pattern with the minimum volume or cost is selected and if necessary is enhanced in terms of weight distribution in a post process phase. Along the algorithm, different criteria for sorting or selecting are used and several parameters require a tuning process. To make these choices, the software package `irace` has been used (López-Ibáñez et al. (2016)).

The third contribution of this work is related to the benchmark instances which are missing for the MBSBPP as explained in Zhao et al. (2016). Several box instances have been created on the basis of a box data set which stems from a real world case. These instances are available online to allow other researchers to compare their results. Information about these data sets is provided in Section 6.

The remainder of this paper is organised as follows. In Section 2, a complete description of the problem and of the methodology is provided. The first phase of the algorithm is presented in Section 3, the second phase is addressed in Section 4 and the post processing designed to improve the weight distribution is described in Section 5. Section 6 holds the parametrisation technique and the computational results. Finally, conclusions are drawn in Section 7.

2. Description of the problem and methodology

The problem is the following one: the objective is to provide a loading pattern that minimises the unused space inside the selected ULDs considering that (1) each box is assigned to exactly one used ULD, (2) each box lies within the limits of a ULD even when this one has a special shape, (3) there is no overlap of boxes, (4) boxes can be orthogonally rotated but some rotations are not permitted for some boxes, (5) fragile boxes cannot support any other boxes, (6) the total weight of the boxes inside each ULD respects its maximum weight capacity, (7) the packing is stable and (8) the weight inside the loaded ULDs has to be uniformly distributed.

In more detail, each box i has a length l_i, a width w_i, a height h_i, and a weight m_i, all these dimensions being integers. Without loss of generality, a coordinate system is placed with its origin on the front left bottom vertex of the ULDs and the axes such that the length (resp. with, height) of the ULD lies on the x-axis, (resp. y-axis, z-axis) as described in Figure 1. The position of the box i in the loading pattern can then be described by its front left bottom vertex (x_i, y_i, z_i) and its top right rear vertex (x'_i, y'_i, z'_i) in that system.

A box may have only some orientations allowed due to it contents. To describe the allowed

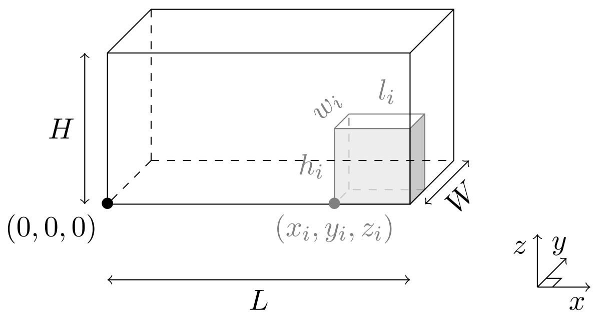


Figure 1: The coordinate system associated to a bin and the coordinates of a box i

orientations, three parameters, l^+, w^+ and h^+ describe whether a specific dimension can be in a vertical orientation. If all these parameters are set to one, each box is free to rotate in any direction, leading to six possible orientations as depicted in Figure 2. Every time one parameter is set to zero, two of these six orientations are forbidden. This type of constraint is called *orientation constraint* and can be found in Chen et al. (1995), Terno et al. (2000), Junqueira et al. (2012), Ceschia and Schaerf (2013), Paquay et al. (2016) among others.

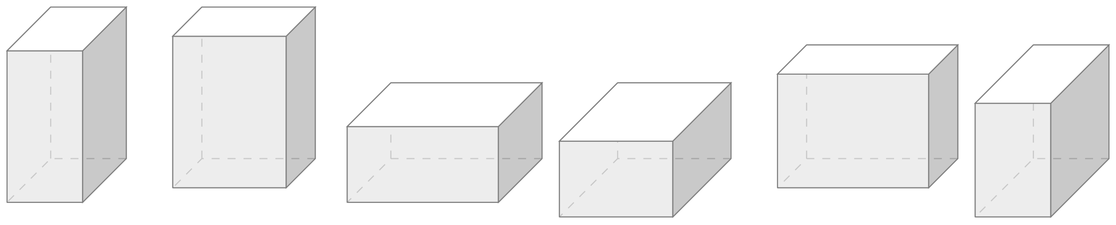


Figure 2: Six possible orientations

A fragile box cannot support other boxes on its top face, making the volume above unusable. This constraint is quite important in practice because it prevents damage to products contained in a fragile box. Different strategies have been developed to manage this feature of the cargo as in Terno et al. (2000), Junqueira et al. (2012), Ceschia and Schaerf (2013), Paquay et al. (2016).

Each ULD type j has a length L_j, a width W_j and a height H_j. It also has a maximum weight capacity C_j which implies that boxes can be loaded as long as this weight limit is not exceeded as in Terno et al. (2000), Chan et al. (2006), Ceschia and Schaerf (2013), Paquay et al. (2016). The volume of ULD j is denoted V_j. This number can also be the cost of ULD j if the objective is a cost minimisation. Some ULDs may have a special shape to fit into the fuselage of the aircraft, as full parallelepipeds whose one or several corners have been cut. There exist four possible cuts which can be described by the linear equation $z = ax + b$, where $a, b \in \mathbb{R}$. To describe these cuts, eight new parameters (two for each cut σ, $\sigma \in \{1, ..., 4\}$) are added for each bin j, $j \in \{1, ..., n\}$: $a_{\sigma j}$, $b_{\sigma j}$ $\in \mathbb{R}^+$ as shown in Figure 3. This uncommon container shape makes the problem a variant of the MBSBPP. However, if this feature is dropped, the work can be extended to more general applications in fields other than air transportation.

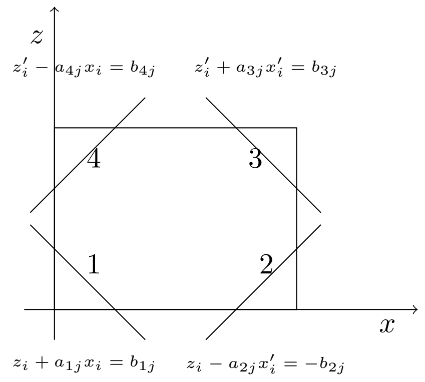

z
$z'_i - a_{4j}x_i = b_{4j}$
$z'_i + a_{3j}x'_i = b_{3j}$
4
3
1
2
x
$z_i + a_{1j}x_i = b_{1j}$
$z_i - a_{2j}x'_i = -b_{2j}$

Figure 3: Possible cuts (projection on the XZ plane)

Load stability is one of the most important types of constraint since unstable loads may result in damaged cargo and even in injuries of personnel during handling operations. For the sake of vertical stability, the bottom side of each box needs to be supported by the top face of other boxes or by the container floor (Terno et al. (2000), Jin et al. (2003), Chan et al. (2006), Junqueira et al. (2012), Ceschia and Schaerf (2013)). Deeper analysis of vertical stability can be found in Ramos et al. (2016). In the present work, the stability of a pattern is ensured by forcing the four base vertices of each box to be supported. Boxes can thus lie on the ground, be supported by top face of non fragile boxes or by the possible inclined wall of the ULDs. This choice is rather restrictive but ensure the vertical stability in any case.

According to the control and loading manual of some airline companies such as Boeing (Boeing (2008)), the centre of gravity (CG) of each ULD must lie in a determined area (depending on the ULD type) around the geometrical centre of the ULD and below a maximum height. Based on this, the CG of each ULD is considered as a point in the centre of the position occupied to calculate the CG of the plane and to ensure some weight constraints. This constraint is therefore crucial and can be met in other applications as in Davies and Bischoff

(1999), Jin et al. (2003), Chan et al. (2006), Trivella and Pisinger (2016), Paquay et al. (2016). This type of constraints can also be adapted to the case of truck loading in which a precise distribution of the cargo over the axles of the vehicles has to be respected (Pollaris et al. (2015, 2016), Alonso et al. (2016)).

To the best of our knowledge, no fast constructive heuristic has been developed to tackle all these specific constraints for the three-dimensional MBSBPP. The aim of this work is therefore to develop an algorithm able to find feasible packing for a large number of boxes for these types of bins. To this purpose, a tailored two-phase constructive heuristic is now presented. The first phase provides a loading pattern for a given set of boxes and a ULD type. Using sorting and selection criteria, a sequence of boxes is built and each box is packed one after another. The position selection improves the Extreme Point (EP) (Crainic et al. (2008)) designed for a packing problem with identical bins. The EPs represent the interesting possible positions to accommodate items and are an extension of the *Corner Points* introduced in Martello et al. (2000). They provide the means to exploit the available volume inside a packing by the shapes of the boxes already in the bins. The basic idea of the EPs is that when a box i with sizes l_i, w_i and h_i is added to a given packing with its front left bottom vertex on (x_i, y_i, z_i), new potential points, the EPs, are generated, where other boxes can be accommodated. This generation is explained in Section 3.2. However, several further developments are provided to consider all the constraints of the specific problem studied in this work. In practice, we have several types of ULDs that we can select, each type available in a determined quantity. Their selection represents the second phase of the algorithm, using the packing algorithm as a subroutine. This second phase is an extension of Kang and Park (2003). For a given packing, the boxes from one or several ULDs are removed and then packed in a smaller ULD type if it is possible. The feasible loading pattern with the minimum volume or cost is selected. The best loading pattern may have unbalanced ULDs. A post processing has thus to be developed to improve the CG position if needed. In this process, two operators have been created: a jump operator that attempts to unpack boxes from one side of the ULD to repack them on the opposite side and a shift operator that pushes the boxes along a defined direction.

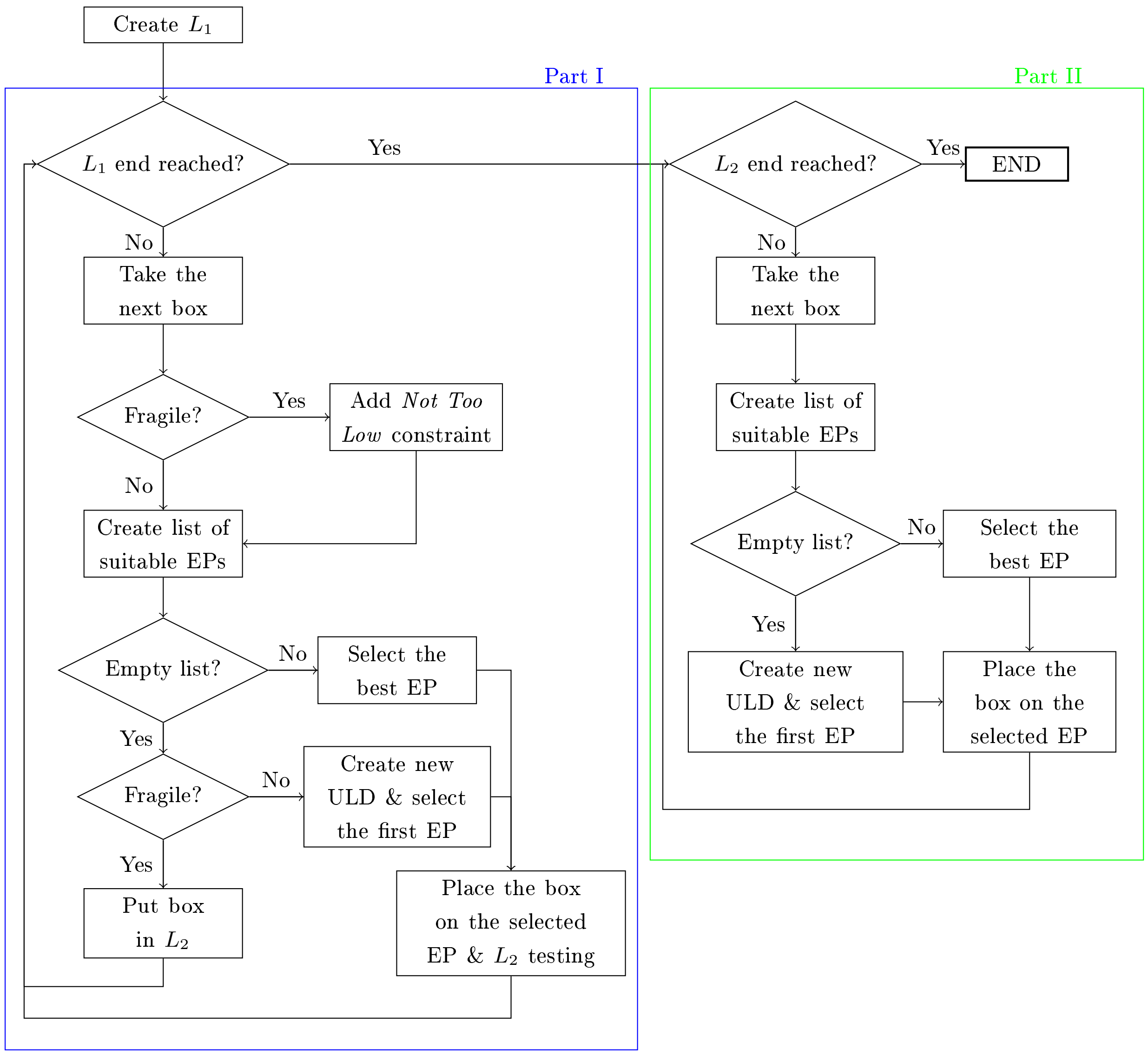
3. Packing algorithm: Phase 1

In this section, a packing algorithm is developed to provide a loading pattern for a given set of boxes and a given ULD type. The ULD is assumed available in an unlimited quantity and the aim is to use its volume in the most efficient manner.

Among the possible constructive heuristics, the First Fit Decreasing and the Best Fit Decreasing algorithms represent a promising perspective as they showed interesting results in terms of worst-case performance ratio for the one-dimensional Bin Packing Problem (Martello and Toth (1990)). However, the adaptation of these algorithms for the three-dimensional case is far from simple: several rules for sorting the items and the bins exist and placing a box in a bin can also be achieved in different ways. This challenge was taken up in Crainic et al. (2008): the authors considered the three-dimensional Single Bin Size Bin Packing Problem in which items cannot be rotated or overlap. For this purpose, Crainic et al. (2008) developed an *Extreme Point* (EP)-based rule for packing items inside a three-dimensional container. These EPs have been created to be efficient with regard to computational effort and volume utilisation. The packing algorithm proposed in this work is based on an extension of the EPs to consider all the constraints of this specific MBSBPP.

The possible fragility of the boxes needs special care in this algorithm. As a reminder, no other box can be packed on top of a fragile box. Therefore, if the top face of a fragile box is low in the ULD, then a consequent volume above this becomes unusable for packing other boxes. For this reason, two lists of boxes are used and the packing algorithm is divided into two major parts as represented in Figure 4. L_1 is the main list containing all the boxes with their best orientation. The best orientation of a non fragile box is selected with the Clustered Area Height rule from Crainic et al. (2008). If the box is fragile, the best orientation is the orientation with the minimum top face area in order to minimise the unusable space. First, the boxes from L_1 are handled one after another. During this first part, a fragile box cannot be packed too low in a ULD because it would lead to a waste of volume above its top face. If a fragile box cannot be packed, then it is added to L_2 (Part I). Second, once all the boxes from L_1 have been packed or assigned to L_2, all the fragile boxes from L_2 are loaded at any possible height (Part II).

To select the location of a box, a global list contains all the EPs existing in all the ULDs. The first EP created when a ULD is open is (0,0,0) or ($\frac{b_1}{a_1}$,0,0) if the ULD has type 1 cut. At



Create L_1
Part I
Part II
L_1 end reached?
Yes
L_2 end reached?
Yes
END
No
Take the next box
No
Take the next box
Fragile?
Yes
Add *Not Too Low* constraint
Create list of suitable EPs
No
Create list of suitable EPs
Empty list?
No
Select the best EP
Empty list?
No
Select the best EP
Yes
Create new ULD & select the first EP
Place the box on the selected EP
Yes
Fragile?
No
Create new ULD & select the first EP
Yes
Put box in L_2
Place the box on the selected EP & L_2 testing

Figure 4: Packing algorithm

each iteration, a box (from L_1 or L_2) is examined and tried to be packed. Each EP of the global list is tested to check whether the box with this orientation can be placed with its left front bottom vertex on that EP while satisfying several constraints. If it is possible, the EP is *suitable* for that box as explained in Section 3.2. From the list of suitable EPs, the best EP according to a merit function also defined in Section 3.2 is selected to accommodate the considered box. However, if the list of suitable EPs is empty, which means that the current box cannot be packed anywhere, then a new bin is created unless the box is fragile, this box would then be added to L_2 to avoid a waste of volume. Every time a box is packed, new EPs are generated since new positions become conceivable. These new EPs can be suitable for the fragile boxes previously assigned to L_2. For this reason, these EPs are analysed for the current boxes of L_2 and used to accommodate any of these if possible. This process is called *L_2 testing* as mentioned in Figure 4 and is explained further in more detail.

3.1. Creating L_1

Building the list. The first step of the packing algorithm consists in building L_1 to take into account the possibility of orthogonal rotations and the possible fragile boxes. A box initially has six different orientations, but it may happen that it is not allowed to be packed in some orientations due to its contents. Some orientations may not be feasible because the box would not fit in the proposed ULD type. Each authorised orientation is analysed and checked for whether the box with this orientation can be packed on that EP. If no orientation is acceptable, it is impossible to pack the box even in an empty ULD, and the box is kept for a different ULD type. Otherwise, a feasible orientation has to be selected. If the box is fragile, the best orientation is the orientation with the minimum top face area. If the box is not fragile, the best orientation is obtained with the *Clustered Area-Height* sorting rule from Crainic et al. (2008). The *Clustered Area-Height* rule works as follows: the set of boxes to pack or orientations is partitioned into clusters, each box being assigned to a cluster according to the size of its base area. Each cluster $A_{j,\delta}$ corresponds to a proportion of the bin base area $L \times W$:

$$A_{j,\delta} = \{\text{boxes } i \mid l_i \times w_i \in](j-1) \times L \times W \times \delta, j \times L \times W \times \delta]\}, \qquad \delta \in [0.01, 1].$$

Inside each cluster, boxes are sorted by non-increasing value of their height h_i. Finally, the clusters are sorted by decreasing value of j, which implies that the cluster with the boxes with the biggest bases is considered first. The best feasible orientation is thus selected and added to the list L_1.

In Crainic et al. (2008), the value of δ has been tuned. However, since a lot of other constraints are added, the calibration of δ has to be addressed for our specific algorithm in the experiments in Section 6.2. If δ is large, then there are few clusters among which some can hold a lot of boxes. Boxes are sorted by decreasing height value. If δ is small, then there is a larger number of clusters, possibly containing few boxes. This second sorting is thus closer to a decreasing base area sorting while the first is closer to a general decreasing height sorting.

Sorting the list. Once L_1 is created, the sequence according to which the boxes are considered has still to be decided. In Crainic et al. (2008), the authors state that the Clustered Area-Height is the most efficient sorting rule. In Section 6.2, `irace` selects the sorting rule among the Clustered Area-Height and the decreasing base area rule.

3.2. Part I

Each box of list L_1 is considered and packed if possible, one after another.

Extreme Point Suitability. For a given box, a list with all the suitable EPs is created by testing all the available EPs. An EP is *suitable* for a box i with a given orientation if, once the box i is packed with its front left bottom vertex on the EP:

1. the box i does not overlap other boxes previously packed,
2. the box i lies within the limits of the ULD (in particular, box i respects the possible special shape of the ULD),
3. the ULD weight capacity is still respected,
4. the box i does not lay on fragile boxes previously packed,
5. the CG of the set of loaded boxes does not exceed the given upper limit and
6. the box i and the previously packed boxes are stable.

These conditions can be checked with the boxes coordinates as done in Paquay et al. (2016).

If the box is fragile, an additional constraint is considered during the first part of the packing algorithm. An EP (x_{EP}, y_{EP}, z_{EP}) is suitable if, once the fragile box i is packed with its front left bottom vertex on this EP, the top face of the box has a height at least equal to a given percentage β of the ULD j height: $z_{EP} + h_i \geq \beta H_j$. This is the *Not Too Low* constraint in Figure 4. The value of β is tuned using parametrisation technique in Section 6.2. A really high value means that the *Not Too Low* constraint is really restrictive and thus fragile boxes are more likely assigned to list L_2. Conversely, a small value of β means that the constraint is easily satisfied.

Selection of the best Extreme Point. Choosing the best EP is a difficult task and many criteria can be imagined in order to optimise different objectives. This selection is achieved with a merit function as defined in Crainic et al. (2008).

In Crainic et al. (2008), the authors aim to minimise the number of bins used to pack the boxes, all the bins being identical. Their best merit function is the function maximising the utilisation of the EP's Residual Space. The *Residual Space* (RS) of a given EP is defined as the free volume available around an EP. It is composed of three components RS_x, RS_y and RS_z, each component describing the free space along each direction. When an EP is created in ULD, the RS of this new EP is equal to the distance from the EP to the side of the bin along each axis. Then, every time a box is assigned to ULD, the RS components of the EPs located in this ULD are updated. The initial value of the RS and its update are extended in this algorithm to take into account the possible special shape of the ULDs and the extension of the EP generation process. For a box i to be packed, this merit function selects the EP that minimises the difference between its RS and the box dimensions:

$$MF1 = (RS_x - l_i) + (RS_y - w_i) + (RS_z - h_i).$$

This selection is close to the selection achieved in the common Best Fit algorithm.

Since the ULD weight distribution is taken into account in this work, another specific merit function is also introduced. In order to measure the weight distribution uniformity, the CG of the packed boxes (x_{CG}, y_{CG}, z_{CG}) is compared to the geometrical centre of the ULD base (x_M, y_M, z_M). For this reason, an additional merit function that computes the distance between the two elements is defined:

$$MF2 = \sqrt{(x_{CG} - x_M)^2 + (y_{CG} - y_M)^2},$$

This merit function is a first step towards a uniform weight distribution. Note that the constraint relative to the height of the CG is included in the definition of suitable EPs. Among these two merit functions, the best function with respect to the objective function minimisation is determined in Section 6.2.

Placing the box on the selected EP and L_2 testing. Once the box is placed with its front left bottom vertex on the best EP, new EPs are generated and one tries to pack some fragile boxes from L_2 on these new points.

Regarding the box placement, the considered box is assigned to the ULD containing the selected EP with the position described by the EP. The selected EP is removed from the EP list. It may happen that placing a box prevents other existing EPs from accommodating a box. Therefore, each EP located in the same ULD is checked and deleted if unusable.

New EPs are generated when a box i is placed with its front left bottom vertex on the point (x_i, y_i, z_i) with its opposite vertex on (x'_i, y'_i, z'_i). This generation process is inspired from the procedure developed by Crainic et al. (2008). The three initial points (x'_i, y_i, z_i) (in blue in Figure 5), (x_i, y'_i, z_i) (in green in Figure 5) and (x_i, y_i, z'_i) (in red in Figure 5) represent the EP sources (and also new EPs themselves) and each point is projected along two directions until reaching a previously packed box or a container side to create new EPs as shown in Figure 5.

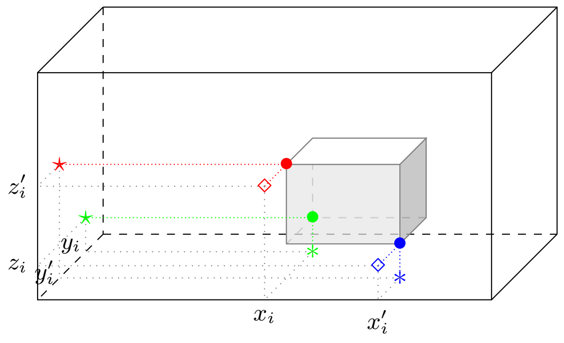


Figure 5: Projection of the source points of box i: the symbol $\star$ (resp. $\diamond$, $*$) represents the projection on the YZ plane (resp. XZ plane, XY plane)

The EP generation process is extended in this work as follows. Consider three types of boxes: one very long, another very wide and a third very high. Each of these three boxes is packed on a source point and is then pushed as far as possible along the projection direction, until it reaches either a previously packed box or the side of the ULD. The position of the front left bottom vertex of this virtual box defines a new EP. However, since the four vertices of each box have to be supported, the projection of the point (x_i, y_i, z'_i) does not lead to usable EPs. For this reason, this point is not projected unless there is a type 1 cut which can also support the box as shown in Figure 6.

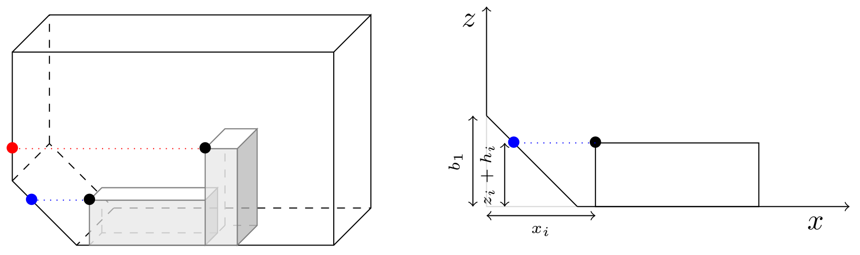


Figure 6: Projection of (x_i, y_i, z_i') along the x-axis in a ULD with a cut 1

About L_2 testing, a list of new EPs has been generated and cleared to avoid duplicates and unusable EPs. Before adding this list to the global list of EPs, these new proposed positions are tested to accommodate fragile boxes from L_2. For each box from L_2, the new EPs are tested looking for suitable EPs (with the *Not Too Low* constraint). If there is one (or more) suitable EP, then

- the fragile box is packed on the (best, if several, as defined previously) EP,
- the selected new EP is removed from the list of available EPs and
- new EPs are generated with the same method and added to the list of new EPs.

Every time a box from L_2 is packed, this box is removed from L_2 and the beginning of L_2 is started again since new EPs are available. If a box from L_2 cannot be packed on any of the new EPs, then the next box from L_2 is considered. This process stops when boxes from L_2 can no longer be packed on these new EPs. The final list of new EPs is then merged with the global EP list.

3.3. Part II

At this point of the algorithm, every box from L_1 has been either packed in a ULD, or, if it was fragile and not possible to pack without opening a new ULD, assigned to the list L_2. In Part II, the *Not Too Low* constraint is dropped and all the fragile boxes from L_2 have to be loaded. The process is now identical to that of Part I: for every box, the global list of EPs is considered and the best suitable EP, if several, is selected with the same criterion as previously. If there is no suitable EP, then a new ULD is created to accommodate the current fragile box.

4. Multiple ULDs: Phase 2

In the packing algorithm, a given set of boxes is loaded in a given ULD type available in an unlimited amount. In practice, several types of ULDs exist and can be selected, each type available in a determined quantity. Therefore, the algorithm has to be extended in order to take this possibility into account.

In Kang and Park (2003), the existence of several types of bins is considered for the one-dimensional BPP. Multiple extensions are set up in order to consider the three dimensions and to improve the solution quality. First, the whole set of boxes is packed (with the packing algorithm from Section 3) into the biggest ULD type (called type 1 ULDs). A list of type 1 ULDs that contain all the boxes is thus obtained. Then, the used ULDs are sorted by non-increasing or non-decreasing occupied volume as explained further. This was the first iteration. During the second iteration, several steps may take place. The boxes that are assigned to the last type 1 ULD of the list are repacked in the next ULD type (called type 2 ULDs). We get a loading pattern with a list of type 1 ULDs and a list of type 2 ULDs (step 1). Another loading pattern is obtained by removing the last two type 1 ULDs and to repack their boxes in type 2 ULDs (step 2). We can do the same for all the type 1 ULDs. During the second iteration, we have thus created one pattern with type 1 ULDs only during the first iteration and as many patterns as there are type 1 ULDs used in the first pattern. Every pattern built during the second iteration are then analysed to create new patterns using type 3 ULDs. The process is repeated until all ULD types have been tested. During the first (resp. second, third, etc.) iteration, boxes are packed in type 1 (resp. type 2, type 3, etc.) ULDs. Each iteration is decomposed into several steps, each step leading to a distinct loading pattern. From one iteration to another, all the patterns of the previous iteration are considered and the boxes placed in ULDs of the last type are repacked into a new ULD type. An example is shown in Figure 7. Finally, the solution with the minimum used volume is selected.

About the sorting of the loaded ULDs to be repacked, it would seem natural to sort the ULDs by non-increasing occupied volume to increase the chance to repack in a smaller ULD type. Indeed, since the last ULD is the ULD whose boxes will be repacked, the less volume there is to repack the more likely that it is possible to repack into a smaller ULD type. However, some preliminary experiments show that the least loaded ULDs often have boxes with special dimensions (for instance, a very long box) and these boxes may be hard

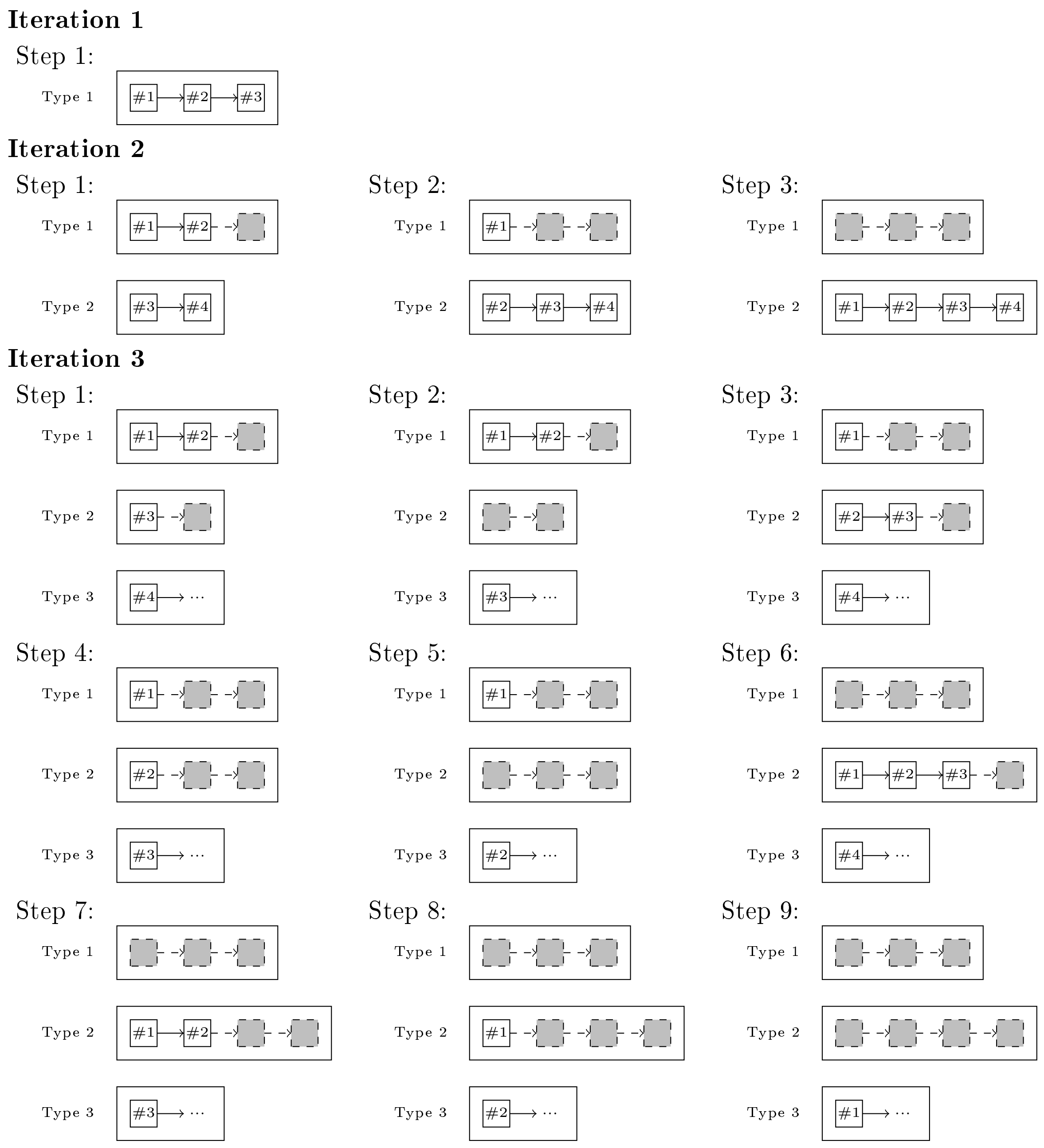


Figure 7: Creation of loading patterns by considering different ULD types

Note. In the first iteration, boxes are packed in type 1 ULDs. During the second iteration, the first step repacks the boxes from the last ULD of type 1 in two ULDs of type 2, the second step repacks the boxes from type 1 ULDs #2 and #3 and the third step repacks all the boxes in type 2 ULDs only. In the third iteration, steps 1 and 2 are based on the pattern obtained in the first step of iteration 2, steps 3, 4 and 5 are based on the pattern obtained in the second step of iteration 2 and steps 6, 7, 8 and 9 are based on the third pattern obtained in the third step of iteration 2.

or even impossible to repack in smaller ULD. For this reason, sorting by non-increasing or non-decreasing occupied volume is a choice made using `irace` parametrisation technique in Section 6.2.

Completion. This procedure can lead to a large number of potential patterns but not all these patterns are feasible because some boxes may not fit in certain ULD types. In other words, some patterns may have unpacked boxes. When all the patterns have been created, the feasible pattern with the minimum volume is retained. The incomplete patterns are then analysed. For every incomplete pattern with an occupied volume smaller than the current minimum volume, one tries to pack the unpacked boxes in the biggest ULD type. If some of the unpacked boxes do not fit in this ULD type, then the second biggest ULD type is considered. This process is repeated until every initially unpacked box is loaded. A feasible pattern is obtained with this method since each box is assumed to fit in at least one ULD. Once the pattern has been completed, its objective function value is compared to the value of the current best solution to determine the new best pattern.

5. Post processing: balance improvement

After the first two phases of the algorithm, a loading pattern with possibly distinct ULDs has been obtained. In this packing, all the constraints of the specific MBSBPP are ensured except the uniform weight distribution in the XY plane. As explained in the merit function definition in Section 3.2, a first step can be achieved when selecting the best EP for each box, but this does not ensure that the CG is in the allowed region around the geometric centre of the ULD base. For this reason, post processing has to be conducted to improve the weight distribution in the unbalanced ULDs. Three operators are developed for this purpose.

5.1. Shifts

Considering the EP generation during the packing algorithm, the CG is likely to lie too on the left or too on the front of the geometrical centre of the ULD base area.

The first method proposes two *shift* operators to improve the balance along the x and y-axes. For the ULDs presenting a CG located too on the left, the boxes are pushed as much as possible to the right until the CG reaches the threshold of the allowed region. The maximum possible global shift is first applied to all the boxes, i.e. the maximum possible overall shift is computed and applied to all the boxes at once. A *global* shift means that if the shift is increased by one more unit, then the ULD limit constraint becomes violated. If the threshold is still not achieved, then each box is considered separately and an adapted shift is set up until a constraint is violated or the threshold is reached. An *adapted* shift means that the shift does not have a general value but is specific to each box. Note that special attention is required to manage the right (resp. left) shift in ULDs with cuts of type 2 or 3 (resp. 1 or 4). In order to increase the chance of possible adapted shift, the boxes i of the ULD are considered by non-increasing x'_i (resp. y'_i) values in the case of a right (resp. back) shift and by non-decreasing x_i (resp. y_i) values in the case of a left (resp. front) shift. Ties are broken sorting by non-increasing value of their front left bottom vertex height (z_i) and finally by non-increasing value of their density. An example of the application of shift operators on one unbalanced ULD is represented in Figure 8.

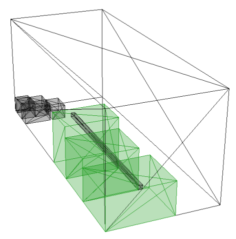
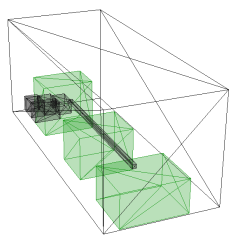
Before right and rear shifts	After right and rear shifts
	

Figure 8: An example of shift operators applications

The CG of the set of packed boxes is tested for each direction and the deviations are analysed in this order: CG too on the left, front, right and rear. Every time the answer is positive, shift operators are applied. The next deviation is then tested and the same process is applied until the balance is reached or if there is no more possible improvement.

Many patterns can be improved with the shift operators. However, in some patterns, it may happen that heavy boxes cannot be shifted further. Therefore, a second operator, jump, is developed.

5.2. Jumps

The idea of the *jump* operator is to take one or several boxes from one side of an unbalanced ULD and to pack these boxes (make them jump) to the other side of the ULD. For instance, if the CG of the ULD is too on the left, then one tries to repack boxes from the left-hand side to the right-hand side of the ULD in order to move the CG to the right. Note that the movements leading to a CG too on the right are not allowed. Moreover, if the weight was balanced in the other direction, movements leading to an unbalanced loading along this other direction are also rejected.

In more detail, if the CG is too on the left, then the boxes assigned to this ULD are considered by non-decreasing value of their front left bottom vertex abscissa (x_i) because the more on the left the boxes are, the bigger is their influence on the CG. Ties are broken sorting by non-increasing value of their front left bottom vertex height (z_i) and finally by non-increasing value of their density. The loading pattern has still to be stable when the box to be jumped is removed from its current position and the highest boxes are less likely to support other boxes. Finally, the denser the boxes, the bigger the impact on the CG.

To repack the boxes, the available EPs of the ULD produced during the packing algorithm are analysed. The idea is to repack the leftmost boxes on rightmost EPs to move the CG as much as possible. The box i can jump from (x_i, y_i, z_i) to a new EP (x_{EP}, y_{EP}, z_{EP}) if several conditions are met:

1. the loading pattern is still feasible, in particular still stable, if box i is removed from its current position,
2. only the EPs with $x_{EP} > x_i$ are helpful and thus considered since a balance improvement is expected,
3. only the suitable (as defined in the beginning of Section 3.2 but without the *Not Too Low* constraint) EPs are considered,
4. if the CG was in the allowable area along the y-axis before box i removal, it has still to be in the allowable area after repacking box i on the new EP and
5. after repacking, the CG is not too on the right (on the other side of the allowable region).

If several EPs satisfy these conditions, the rightmost one is selected to accommodate the box i. Ties are broken selecting the EP with the smallest residual space. Afterwards, the selected EP is removed from the list as well as the potentially unusable other EPs. New EPs are generated and added to the list of available EPs. If the CG is still on the left-hand side of the allowable region, then the next box of the sorted list is considered for a jump.

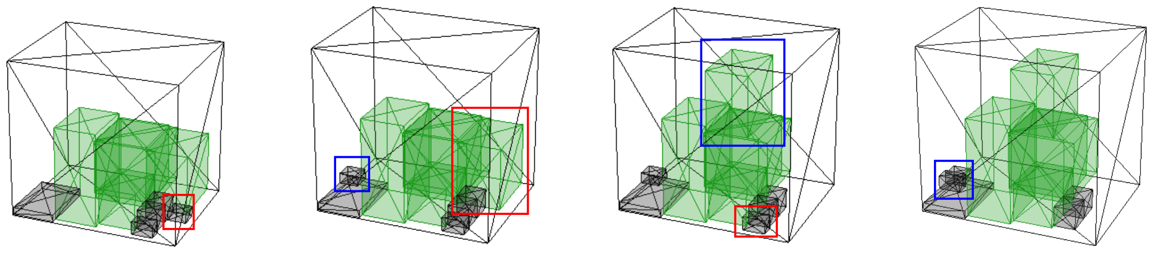


Figure 9: An example of jump operator applications to fix a CG too on the right: the former positions are circled in red, while the new positions (after jumping) are circled in blue

Naturally, the jump operator can be applied in the other directions: a jump from the right-hand side to the left-hand side, from the front to the rear and inversely. An example of jump operator applications is shown in Figure 9.

As for the shift operators, the CG of the set of packed boxes is tested for each direction and the deviations are analysed in the same order: CG too on the left, front, right and rear. Every time the answer is positive, the jump operator is applied. The next deviation is then tested and the same process is applied until the balance is reached or if there is no more improvement.

5.3. Combination of jump and shift operators

The jump and shift (J&S) operators can be combined into the same process. In practice, the jump operator is applied first. The jump operator is less accurate than the shift operators because the CG does not move one unit by one unit. Moreover, after applying shift operators, the residual space of some EPs may be reduced and thus these EPs would not be able to accommodate some boxes. In practice, the CG of the set of packed boxes is tested for each direction and the deviations are analysed in this order: CG too on the left, front, right and rear. Every time the answer is positive, jump is applied. If no box is still able to jump and the CG is still not in the allowable region for the considered direction, then the shift operators are applied. The next deviation is then tested and the same process is applied again if some deviations remain. This means that jump operators may be applied after shifts in another direction. It is therefore crucial to keep the EPs and their respective residual space up-to-date during the whole process. It may happen that deviations are not corrected at the first application of J&S operators. However, the ULD may have another deviation whose correction leads to other movements of the boxes. For this reason, unless boxes are no longer moving, a loop over the different deviations is put in place. In order to avoid infinite loops, the CGs already met are memorised and compared to the current CG. If a same CG is obtained twice, then the algorithm stops. Indeed, it may happen that when trying to correct two deviations (one along the x-axis and one along the y-axis), boxes are moved in one direction first and then in another direction, which can lead to infinite loop. The improvement in terms of weight distribution gained thanks to this operator is described in Section 6.3.2.

6. Computational experiments

This section addresses the computational experiments. Data sets are first described. These sets come from a real world case and are meant for different purposes: a first group is used to parametrise the heuristic (training data sets) and a second group is used to measure its efficiency (final data sets). Second, in Section 6.2, the parametrisation method is explained and applied to the tailored two-phase heuristic using training instances. Finally, the results of the trained heuristic are analysed in Section 6.3.

All the tests were performed on a workstation with 32.0 GB RAM and a Intel Xeon processor E5-2620 v4 running 64-bit Windows 10 Pro. Codes have been implemented in Java.

6.1. Data sets

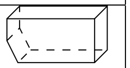
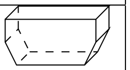
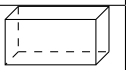
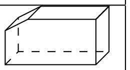
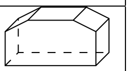
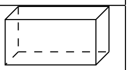
6.1.1. ULDs

Among the ULDs that can be loaded in a Boeing 777, which is the most common aircraft for cargo transportation, six common types are selected: three for the lower deck and three for the main deck. These ULDs are described in Table 1. More detail about these ULDs can be found in Boeing (2008). This set of ULDs constitutes a representative sample since all the shapes are proposed.

6.1.2. Boxes

As can be seen in Bortfeldt (2012) and Zhao et al. (2016), there are no benchmark instances for the three-dimensional MBSBPP. The closest instances are those from Ivancic et al. (1989) which were designed for a Multiple Stock-Size Cutting Stock Problem. For this reason, new data sets were needed in order to represent the specificity of the problem studied in this work.

Table 1: Description of the parameters of the proposed ULDs

IATA Code	Dimensions $L \times W \times H$ [mm]	Max. Cap. [kg]	Vol. [m^3]	$\alpha^L;\alpha^W;\alpha^H$ [mm]	Equation of the cuts	Shapes
Lower deck - containers						
LD1	2337×1534×1626	1518	5	157;152;864	cut 1: $z + \frac{826}{775}x = \frac{640150}{775}$	
LD6	4064×1534×1626	2945	9.1	318;152;864	cut 1: $z + \frac{422}{444}x = \frac{187368}{444}$ cut 2: $z - \frac{422}{445}x = -\frac{1527218}{445}$	
LD11	3175×1534×1626	2991	7.4	318;152;864	None	
Main deck - pallets						
PA	2235×3175×2997	5890	20	224;318;1219	cut 4: $z - \frac{747}{1056}x = \frac{2376000}{1056}$	
PG	2438×6058×2438	10840	31.1	244;302;1219	cut 3: $z + \frac{1054}{660}x = \frac{3483092}{660}$ cut 4: $z - \frac{1054}{660}x = \frac{913440}{660}$	
PM	2438×3175×2997	6680	18.9	244;318;1219	None	

For the computational experiments, a box data set which stems from a real world case is considered. It contains information about the dimensions and weight for 562 rectangular boxes. The main features of these boxes are given in Table 2. Unfortunately, the initial data set does not hold the authorised orientations and the fragility characteristic. Therefore, a data manipulation is set up to add these missing parameters. In more detail, if a box is too heavy (>50kg), it is assumed to be too much work to turn it, the parameters l^+, w^+ are thus assumed to be 0. The parameter h^+ equals 1 for all the boxes because the initial orientation of the boxes is supposed feasible. With regard to the stacking constraint, a box which has a density lower than 0.05 kg/dm^3 is considered as fragile. After applying these two rules, all the parameters are available for each box.

Description of the data sets. For the experimentations, instances with different sizes are interesting to analyse the behaviour of the algorithm. A data processing is used to generate box samples of different sizes ranging from 10 to 100 boxes. For each sample size, a set of 30 instances is built by random selection from the original data set. Instance sets are available from `http://hdl.handle.net/2268/206856`:

- Training data sets: in order to tune the parameters in Section 6.2, a set of representative

Table 2: Information about the initial data set

	length $[mm]$	width $[mm]$	height $[mm]$	weight [kg]	volume $[dm^3]$	density $[kg/dm^3]$
Range	[130;5250]	[100;1720]	[40;1900]	[1;1983]	[2.2;5832.96]	[0.01;5.62]
Average	927.65	620.18	618.89	106.55	571.57	0.24
Standard dev.	625.91	309.72	407.78	177.02	682.75	0.31

Number of distinct boxes (types):	199
Average number of boxes per type:	2.82

instances has to be provided. To this purpose, 30 instances with 10, 15, 20, 25 and 30 boxes each (i.e. 150 instances) have been generated;

- Final data sets: for testing the tailored two-phase constructive heuristic in Section 6.3, 30 instances with 10, 20, 30, ..., 100 boxes each (i.e. 300 instances) have been generated.

6.2. Parametrisation with `irace`

A *configuration* of an algorithm is a unique assignment of values to parameters. `irace` is a software package that provides an automatic configuration tool for tuning optimisation algorithms, that is, automatically finding the most appropriate settings of an algorithm given a set of instances of a problem, saving the effort that normally requires manual tuning (López-Ibáñez et al. (2016)). During the tuning phase, a set of training instances representative of a particular problem has to be provided to choose the best algorithm configuration. The selected algorithm configuration can then be used to solve new instances of the same problem.

A *racing method for selection* or *race* is a method that aims to find a good configuration through a sequence of steps. As explained in López-Ibáñez et al. (2016), along the racing method, if sufficient evidence is gathered that some candidate configurations perform statistically worse than at least another configuration, such candidates are discarded and the procedure is iterated over the remaining surviving configurations. The elimination of inferior candidates speeds up the procedure and allows a more reliable evaluation of the promising configurations.

To evaluate the performance, a set of training instances, which will be sampled, is provided as well as a cost function. In the present case, the cost function assigns the objective function value to each configuration when applied to a single problem instance. To discard a configuration, the non-parametric Friedman's statistical test is performed on the values provided by the cost function on several instances. In this work, the race is applied until reaching a maximum number of experiments, where an experiment is the application of a configuration to an instance.

In iterated racing, each configurable parameter is associated to a sampling distribution which is independent of the sample distributions of the other parameters. More detail about these sampling distributions can be found in López-Ibáñez et al. (2016). *Iterated racing* is a method for automatic configuration which can be decomposed in three steps:

1. sampling the new configurations according to a particular distribution,
2. selecting the best configurations from the newly sampled ones by means of racing,
3. updating the sampling distribution of each configurable parameter in order to bias the sampling towards the best configuration. The update biases the distributions to increase the probability of sampling the parameter values in the best configurations found so far.

The three steps are repeated until a termination criterion is met.

At different stages of the tailored two-phase constructive heuristic, several options are possible and choices have to be made in order to optimise the value of the objective function. For this reason, the five following parameters need calibration:

Parameters	Types	Range
β	Real	[0.50,1.00]
merit function	Integer	[1,2]
`uldOccupationSort`	Integer	[1,2]
`boxSort`	Integer	[1,2]
δ	Integer	[1,99]

Parameter β represents the ratio of the ULD height that the top face of a fragile box has to reach during the first part of the packing algorithm. Values tested for β are limited to two digits. Two merit functions have been proposed to select the best EP: the first function, MF1, uses the RS around the EP and the second, MF2, uses the distance between the CG and the geometrical centre of the ULD in the XY plane. Parameter `uldOccupationSort` describes the ordering of the loaded ULDs to be repacked in the second phase of the heuristic. If `uldOccupationSort`=1, then the ULDs are sorted by non-increasing occupied volume, while if `uldOccupationSort`=2, they are sorted by non-decreasing occupied volume. Parameter `boxSort` is equal to 1 if boxes are sorted according to the Clustered-Area Height before being packed and is equal to 2 if boxes are sorted by decreasing base area. The parameter δ is the percentage used to define the Clustered-Area Height sorting rule. Therefore, if `boxSort`=2, then δ has not to be addressed.

After 5000 experimentations, `irace` states that the best configuration is as follows:

β	merit function	`uldOccupationSort`	`boxSort`	δ
0.68	MF1	2	2	–

`irace` shows some difficulties to find a value for parameter β, it did not find significant differences between several values. This may be explained by the small impact of parameter β on the solution quality. About the merit function, according to `irace`, MF1, based on RS, gets better solution quality than MF2, based on the distance between the CG and the geometrical centre of the ULD. This can be expected since the aim of the RS utilisation is to exploit the space in the best way, choosing the minimum volume able to accommodate the given box. However, the goal of the function MF2 is to minimise the distance between the CG and the geometric centre of the ULD. The influence on the weight distribution is analysed in Section 6.3.2. Results from `irace` state that the solution quality is better if the ULDs are sorted by non-decreasing loaded volume for unpacking and repacking. This behaviour may be explained by the fact that some ULDs have few packed boxes but with a special shape. These boxes can thus be hard to repack in smaller type of ULDs. Results from `irace` suggest that it is more efficient in terms of solution quality to sort the boxes to be packed by decreasing base area.

6.3. Results analysis

In this section, several aspects of the heuristic are analysed on experiments performed on the final data sets with parameters set to the values found by `irace`. First, a good indicator of the solution quality is the filling rate of the used ULDs. The filling rate shows how efficiently the loading space is used inside the ULDs. However, the constraint related to the height of the CG may prevent a really high filling rate from happening. Second, another important point is the weight distribution inside the loaded ULDs as explained in Section 5. The computational times are finally observed.

6.3.1. Filling rate analysis

To represent the global tendency of these filling rates, boxplots are drawn with all the ULDs used for the 30 instances of each sample size in Figure 10. In addition to the size of the samples, the number of used ULDs is provided along the y-axis.

In Figure 10, one can see that the bigger the number of boxes, the bigger the filling rates of the ULDs can be. In particular, half of the ULDs used with the instances of 100 boxes have a filling rate of at least 50%, one ULD even reaches a filling rate of 72.78%. The tailored two-phase constructive heuristic is thus able to achieve ULDs with important filling rates.

The observation distribution tends to become asymmetrical when the number of boxes in the samples increases. For instance, the boxplot for the samples with 90 boxes is more left skewed than the boxplot for instances with 10 boxes. The 25% of observations of the right part of the interquartile ranges have more or less the same size, no matter the size of the samples, whereas the 25% of the left part of the interquartile ranges become more spread out when the sample size increases. In addition to the interquartile range itself, the left whisker encompasses really different values when the sample size is increasing. The observations seem to present a distribution with a negative skewness, i.e. with a tail on the left, when the sample holds a large number of boxes. In other words, it seems that, no matter the number of boxes in the instances to be solved, there are always ULDs with really small filling rates. This can be due to boxes with special dimensions (a box with a large length for example) or simply because some small boxes have to be packed at the end of the algorithm but not enough space remains in the already loaded ULDs.

6.3.2. CG deviation analysis

As explained above, two distinct merit functions have been tested: the first is based on the RS around the EPs while the second tries to reduce the distance between the CG and the geometrical centre of the ULD. Figure 11 shows the percentage of unbalanced ULDs per sample size before (in full lines) and after the application of the J&S operators (in dashed lines) for the two merit functions. One can see that the percentage of unbalanced ULDs is initially really high especially for the instances with a small number of boxes. This is due to the way of packing, starting by the front left bottom vertex. If few boxes are packed, then the weight distribution is not likely to be improved naturally by packing more boxes in this ULD. The improvements brought by the J&S operators, shown in dashed lines in Figure 11,

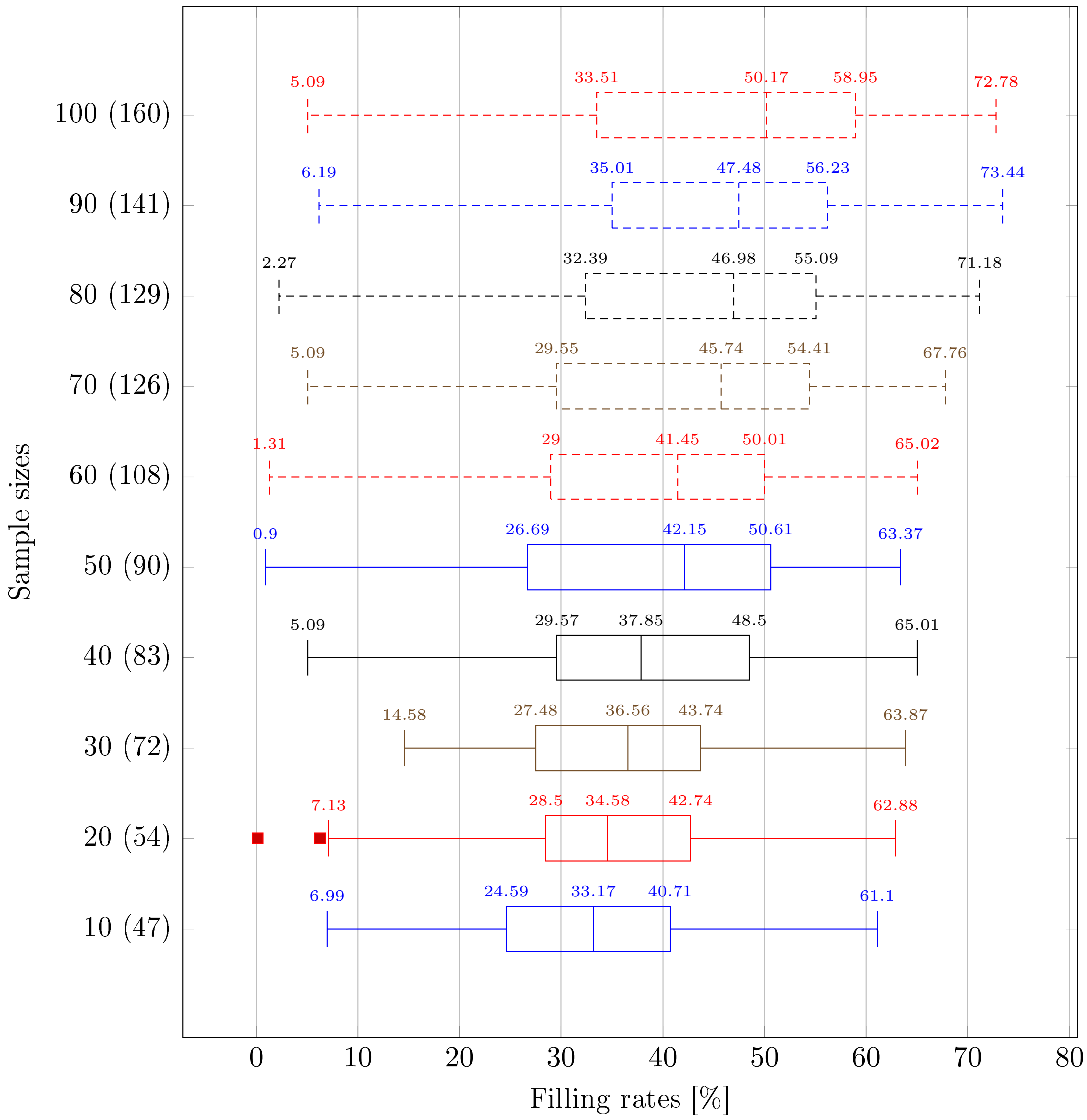


Figure 10: Filling rates in percent per sample size (each sample size holds 30 instances) for the tailored two-phase constructive heuristic. The number in brackets denotes the number of ULDs used for all the 30 instances of each sample size

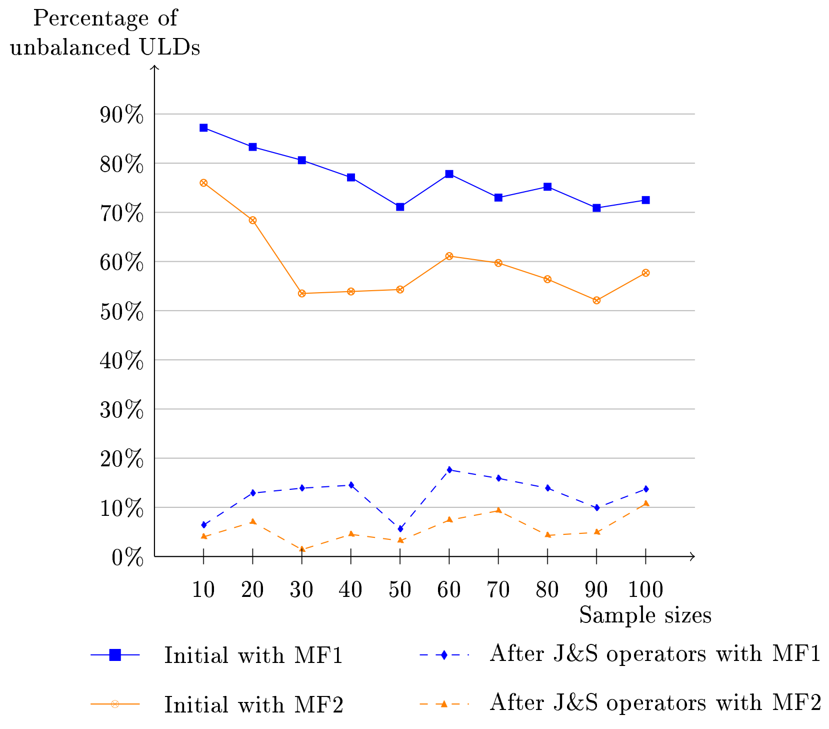


Figure 11: Average percentage of unbalanced ULDs per sample size (each sample size holds 30 instances)

are considerable.

About the influence of the merit function choice, the weight distribution is indeed improved with MF2 and the number of observed deviations is therefore reduced compared to the use of MF1 as shown in Figure 11. Furthermore, it can be seen that MF1 has a larger number of unbalanced ULDs than with MF2 already before the J&S operators application designed to improve weight distribution. This remark is still valid after the J&S operators application. Therefore, MF2 fulfils its role with respect to the weight distribution. Results represented in Figure 11 suggest that the combination with MF2 and the J&S operators is the key for a low percentage of unbalanced ULDs.

Looking closely at the deviations separately for MF1 only, the packing frame algorithm starts the packing in the front left bottom vertex as explained in Section 5. Because of this, the number of CGs initially too on the left or front is really high as shown in Figure 12. In this figure, the number of deviations before and after the J&S operators application can be observed, highlighting the weight distribution improvement. Table 3 summarises the reductions for each CG deviation. Among the 130 ULDs still unbalanced after the application of

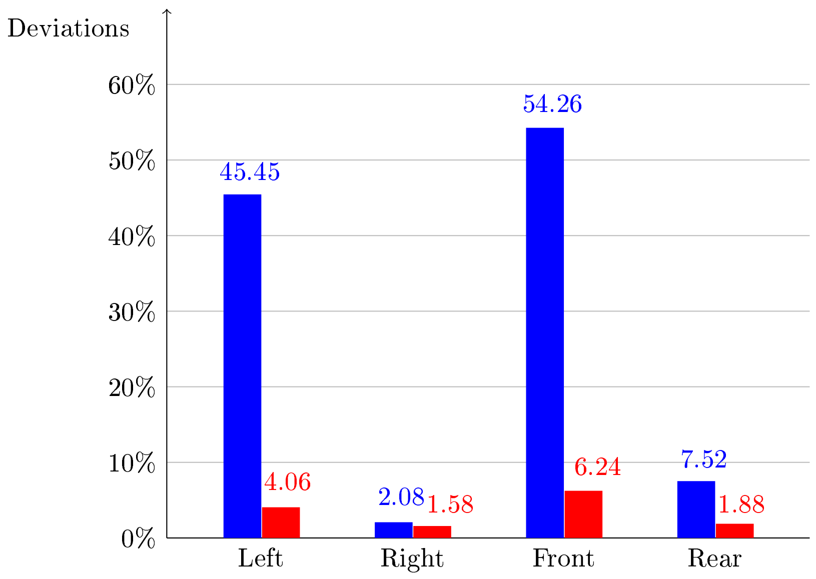


Figure 12: Percentages of deviations between the CG and the allowable area among the 1010 used ULDs (in blue before and in red after the J&S operators application)

the J&S operators, 43.85% of these ULDs still have a unbalanced weight along the x-axis, whereas 63.08% still have a CG out of the allowable region along the y-axis. The former have an average x-deviation of 8.41% and the latter have an average y-deviation of 8.42%. The remaining deviations are thus really reasonable.

Table 3: Reductions in the number of CG deviations among the 1010 used ULDs before and after the application of J&S operators

	Number of	Deviations			
	unbalanced ULDs	Left	Right	Front	Rear
Before J&S	761	459	21	548	76
	(75.35%)	(45.45%)	(2.08%)	(54.26%)	(7.52%)
After J&S	130	41	16	63	19
	(12.87%)	(4.06%)	(1.58%)	(6.24%)	(1.88%)
reduction	82.92%	91.07%	23.81%	88.50%	75.00%

6.3.3. Computational times analysis

The main advantage of the tailored two-phase constructive heuristic is its dramatic speed. Indeed, even for the instances with up to 100 boxes to be packed, the maximum computational duration does not exceed 12 seconds for any instance.

7. Conclusion

In this paper, we have addressed the three-dimensional rectangular Multiple Bin Size Bin Packing Problem with bins typical to air transportation. Due to its specificity and complexity, this problem has not been deeply studied in the literature. The presented tailored constructive heuristic is composed of two phases. The first phase proposes a packing algorithm for identical bins and takes into account several constraints inherent to transportation: the fragility, the stability and the possible different orientations of the boxes, as well as the special shape and the weight capacity of the bins. The algorithm extends the Extreme Points generation and selection to suit to the studied problem. The purpose of the second phase is to deal with multiple types of bins. Furthermore, post processing has been developed to enhance the weight distribution for the unbalanced bins. Two operators, as well as their combination, have been created for this purpose and showed promising results. Experimentations have been performed on new data sets especially designed for the Multiple Bin Size Bin Packing problem and are publicly available. The different parameters have been tuned using the parametrisation technique `irace` and the results are very encouraging. The tailored two-phase constructive heuristic requires short computational times and presents valuable solutions in terms of filling rate. As perspective, post processing for weight distribution improvement can be extended to consider combination of boxes which cannot be shifted without loosing the stability. It can also be combined to an exact approach derived from the formulation from Paquay et al. (2016). Another direction can be related to the sorting operators in which randomness may be introduced to improve the solution quality.

Acknowledgments

The project that leads to these results was partially funded by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office (grant P7/36). This paper, however, only expresses the authors' views.

References

Alonso, M., R. Alvarez-Valdes, M. Iori, F. Parreno, and J. Tamarit (2016). Mathematical models for multicontainer loading problems. *Omega*, –.

Boeing (2008). Weight and balance control and loading manual – model 777f.

Bortfeldt, A. (2012). A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. *Computers & Operations Research 39*(9), 2248 – 2257.

Bortfeldt, A. and G. Wäscher (2013). Constraints in container loading - A State-of-the-Art Review. *European Journal of Operational Research 229*, 1–20.

Ceschia, S. and A. Schaerf (2013). Local search for a multi-drop multi-container loading problem. *Journal of Heuristics 19*(2), 275–294.

Chan, F. T. S., R. Bhagwat, N. Kumar, M. Tiwari, and P. Lam (2006). Development of a decision support system for air-cargo pallets loading problem : A case study. *Expert Systems with Applications 31*, 472–485.

Chen, C., S. Lee, and Q. Shen (1995). An analytical model for the container loading problem. *European Journal of Operational Research 80*, 68–76.

Crainic, T. G., G. Perboli, and R. Tadei (2008). Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing 20*(3), 368–384.

Davies, A. and E. Bischoff (1999). Weight distribution considerations in container loading. *European Journal of Operational Research 114*, 209–527.

Garey, M. and D. Johnson (1979). *Computers and Intractability : A guide to the theory of NP-Completeness.* San Francisco: W.H. Freeman.

Ivancic, N., K. Mathur, and B. Mohanty (1989). An integer-programming based heuristic approach to the three-dimensional packing problem. *Journal of Manufacturing and Operations Management* (2), 268–289.

Jin, Z., T. Ito, and K. Ohna (2003). A three-dimensional bin packing problem and its practical algorithm. *JSME International Journal Series C : Mechanical Systems, Machine Elements and Manufacturing*(46), 60–66.

Junqueira, L., R. Morabito, and D. S. Yamashita (2012). Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers and Operations Research 39*, 74–85.

Kang, J. and S. Park (2003). Algorithms for the variable sized bin packing problem. *European Journal of Operational Research 147*(2), 365–372.

Limbourg, S., M. Schyns, and G. Laporte (2012). Automatic aircraft cargo load planning. *Journal of the Operational Research Society 63*(0), 1271–1283.

López-Ibáñez, M., J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives 3*, 43–58.

Martello, S., D. Pisinger, and D. Vigo (2000). The three-dimensional bin packing problem. *Operations Research 48*(2), 256–267.

Martello, S. and P. Toth (1990). *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc.

Paquay, C., M. Schyns, and S. Limbourg (2016). A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research 23*(1-2), 187–213.

Pollaris, H., K. Braekers, A. Caris, G. K. Janssens, and S. Limbourg (2015). Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum 37*(2), 297–330.

Pollaris, H., K. Braekers, A. Caris, G. K. Janssens, and S. Limbourg (2016). Capacitated vehicle routing problem with sequence-based pallet loading and axle weight constraints. *EURO Journal on Transportation and Logistics 5*(2), 231–255.

Ramos, A. G., J. F. Oliveira, and M. P. Lopes (2016). A physical packing sequence algorithm for the container loading problem with static mechanical equilibrium conditions. *International Transactions in Operational Research 23*(1-2), 215–238.

Terno, J., G. Scheithauer, U. Sommerweiss, and J. Riehme (2000). An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research 123*, 372–381.

Trivella, A. and D. Pisinger (2016). The load-balanced multi-dimensional bin-packing problem. *Computers & Operations Research 74*, 152–164.

Wäscher, G., H. Haußner, and H. Schumann (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research 183*, 1109–1130.

Zhao, X., J. A. Bennell, T. Bektaş, and K. Dowsland (2016). A comparative review of 3d container loading algorithms. *International Transactions in Operational Research 23*(1-2), 287–320.