Annals of Operations Research
https://doi.org/10.1007/s10479-021-04349-w

ORIGINAL RESEARCH

Mathematical models and heuristic algorithms for pallet building problems with practical constraints

Gabriele Calzavara[1] · Manuel Iori[2] · Marco Locatelli[1] · Mayron C. O. Moreira[3] · Tiago Silveira[1]

Accepted: 7 October 2021

Abstract
In the pallet building problem, we aim at loading a given set of items into one or more pallets, by satisfying specific constraints and minimizing the number of pallets used. In this paper, we address a practical case of this problem that originates from a real-world robotized application, subject to some non-trivial operational constraints. In practice, items are grouped into families and must be packed into horizontal layers. To facilitate loading/unloading operations, items of the same type packed into the same layer should be contiguous and at least one of them should be visible from the outside. We present a formal mathematical description for layer and pallet creation subproblems and then we propose heuristic, metaheuristic, matheuristic algorithms to solve the overall problem. The performance of the algorithms is assessed through extensive computational tests on real-world instances.

Keywords Pallet building problem · Practical constraints · Two-step heuristic · Reactive GRASP · Mathematical models · Real-world instances

✉ Tiago Silveira
tiago.silveira@unipr.it

Gabriele Calzavara
gabriele.calzavara@unipr.it

Manuel Iori
manuel.iori@unimore.it

Marco Locatelli
marco.locatelli@unipr.it

Mayron C. O. Moreira
mayron.moreira@ufla.br

1 Department of Engineering and Architecture, University of Parma, Parma, Italy

2 Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Reggio Emilia, Italy

3 Department of Computer Science, Federal University of Lavras, Lavras, Brazil

Published online: 25 October 2021

1 Introduction

Cutting and Packing (C&P) is one of the most widely studied fields of Operations Research and involves several interesting practical problems (Bortfeldt and Wäscher (2013); Crainic et al. (2012); Iori et al. (2021); Scheithauer (2018)). In cutting problems, a set of stock units has to be cut to produce smaller items, while in packing problems, a set of items has to be packed into one or more containers. These problems have attracted the attention of researchers both for their practical and for their theoretical interest. Some practical applications involve the production of materials that come in panels (such as wood or glass), the optimization of layouts (as in industry or newspaper paging), and the loading and subsequent transportation of items in containers, to mention a few.

Several interesting surveys and books have been published in recent years to try to review the fast-growing C&P literature, see, e.g., Crainic et al. (2012), Iori et al. (2021), Lodi et al. (2014), Scheithauer (2018) and Silva et al. (2016). A typology of C&P problems has been proposed by Wäscher et al. (2007) and, later, extended by Bortfeldt and Wäscher (2013) by considering the area of container loading and its main constraints. Specifically, Bortfeldt and Wäscher formalized the main concepts of the container loading problems and the related constraints, such as container-related, cargo-related, positioning, and load constraints.

Most C&P problems are associated with mass-production operations in a company. Therefore, improvements in the time performance of the process and the reduction in the material/space wasted are directly related to the use of efficient methods. Previously, C&P tasks were usually performed by skilled experts, but this has been changing over the years by automated-packing (robotized) systems, as is the case for the problem we face in this work.

The problem that we study is named Pallet Building Problem (PBP). Basically, through the solution of this problem, we aim at loading a given set of items into one or more pallets by meeting general and specific constraints to minimize the number of pallets used. The PBP originates from a real-world robotized application and is thus subject to some non-trivial operational constraints. Typical items include, e.g., food packaging, soft drink bottles, and cans. Items should be packed into layers, that must then be piled one over the other while respecting given stackability rules. Moreover, items are grouped into families, and, to facilitate loading/unloading operations, items of the same type packed into the same layer should be contiguous (as detailed in Sect. 3). That said, the loading of items and, therefore, the creation of pallets for this problem is a fundamental area for logistics, as it impacts not only the company's costs but also the customer's final price.

The PBP is NP-hard because it is a generalization of the Bin Packing Problem, which is known to be NP-hard (Delorme et al. (2016)). To solve it, we propose new efficient algorithms based on the most successful and recent C&P studies by embedding in them in a tailored way the additional operational constraints of the problem at hand. In our preliminary works (see Iori et al. (2020a, b, 2021)), we developed some heuristics and proposed a reactive GRASP metaheuristic, besides performing some preliminary computational tests. In this paper, we extend and conclude our work by presenting new algorithms and a mathematical formulation for the PBP. In summary, the main contributions of this paper can be sketched as follows:

(i) a real-world industrial application that addresses the concept of family, contiguity, and visibility is presented;
(ii) a formal mathematical formulation of the PBP with contiguity and visibility constraints is presented, to the best of our knowledge, for the first time in the literature;
(iii) new algorithms, made up by combining heuristics, metaheuristic, and mathematical models, are proposed to solve the problem;

(iv) extensive computational tests on instances derived from a real-world case study are presented.

The remainder of the paper is organized as follows. Section 2 reviews the related literature. Section 3 provides a formal description of the problem. Section 4 provides mathematical models for the layer creation and for the pallet creation. Section 5 presents the different solution algorithms that have been developed. Section 6 gives the outcome of extensive computational tests. Conclusions and some future research directions are given in Sect. 7.

2 Literature review

Approaches to handle cutting and packing problems started around 1960, with the adoption of simple mathematical models for 2D versions of the problem (see Gilmore and Gomory (1965)). In contrast, approaches to handle practical cases of packing problems addressing 3D models started only years later with Tsai (1987), who proposed a robotic 3D pallet loading with boxes of several sizes.

The PBP emerges as a variant of the *Container Loading Problem* (CLP), which has received large attention in the last years. Bortfeldt and Wäscher (2013) presented a comprehensive survey of the main constraints used in the literature. The authors verified that heuristic approaches are more frequently used than exact and approximation-guarantee algorithms. Instead, Silva et al. (2016) considered the *Pallet Loading Problem* (PLP). In this problem, a set of two-dimensional rectangular items needs to be packed without overlapping and allowing a 90° rotation into a two-dimensional bin. The authors proposed a broad analysis of the solution methods and some aspects concerning computational complexity, upper bounds, and data sets. In Crainic et al. (2012), a survey is presented about 2D and 3D Orthogonal Packing Problems, focusing on data structures for the packing representation and the item-positioning rules. Concerning heuristic criteria to place items, we highlight the *extreme points* in Crainic et al. (2008), that increase the amount of feasible regions for the partial packing. Recently, Iori et al. (2021) proposed a survey of 2D packing problems, considering techniques to represent and handle items, relaxation methods, as well as exact and heuristic approaches. We also refer to Wäscher et al. (2007) for a categorized typology of 2D packing problems, besides Delorme et al. (2017) for a state-of-the-art computational analysis.

The PBP can be separated into two subsequent decisions: the first one consists of creating 2D layers, while the second one involves stacking layers to form pallets and thus considers the 3D characteristics. In the following, we discuss some relevant approaches for 2D and 3D, respectively.

For what concerns heuristics for 2D problems, Chazelle (1983) described an efficient way to implement the famous bottom-left heuristic, which packs the items, one at a time in a given order, in the lowest and left-most position. In Burke et al. (2004), a new placement heuristic, called best-fit, is presented for a 2D cutting problem, allowing non-guillotine packings and rotations of 90 degrees. This technique uses a dynamic search based on the "niches", which are the available bottom-most gaps for an item in the partial packing. In terms of metaheuristics, Alvarez-Valdes et al. (2008) developed a GRASP approach for the 2D strip packing problem, which is the problem of packing items into a strip of a given width by minimizing the used height. In the constructive phase, the items are placed into rectangles following specific criteria. A new rule attempts to foresee the tallest object's future effect in the final solution to avoid spikes. The local search iteratively destroys and rebuilds portions of the current solution. Imahori and Yagiura (2010) improved the technique proposed by Burke et al.

(2004) by presenting a quicker implementation based on a balanced binary search tree and a priority heap with a doubly-linked list. In Leung et al. (2011), a complete set of techniques to deal with the 2D strip packing problem is presented. The authors use the so-called "skyline" approach in conjunction with greedy local search, a simulated annealing metaheuristic, and a multi-start diversification strategy. In terms of exact algorithms, the use of Combinatorial Benders decompositions has recently lead to good computational results for a number of 2D packing problems, as in, e.g., Côté et al. (2014, 2021) and Delorme et al. (2017).

Regarding 3D problems, Haessler and Talbot (1990) addressed a real CLP involving some practical constraints. They proposed an integer programming formulation and a heuristic algorithm. Bischoff and Ratcliff (1995) presented two heuristics for the CLP: the first one produces loading patterns with a high degree of stability; the second one considers a multi-drop situation in which a Last-In-First-Out constraint is imposed on the cargo. Terno et al. (2000) proposed the parallel generalized layer-wise loading approach (PGL-approach) for the CLP. Using a complex branch-and-bound algorithm, the authors obtained a certain degree of competitiveness with respect to classical solutions reported in the literature. Bortfeldt and Gehring (2001) proposed a hybrid genetic algorithm to solve the CLP with a single container and a strongly heterogeneous set of boxes, considering orientation, stability, stacking, weight, and balance. Egeblad et al. (2010) addressed the CLP for a single container (in the knapsack version), using irregular shaped items, and taking a stability constraint into account. They performed tests on randomly generated and real-world instances derived from a prominent European furniture producer. Józefowska et al. (2018) studied the CLP considering rotation, stackability, and stability constraints. They considered a case study arising from a household equipment factory, proposing a best-fit heuristic based on the idea of wall-building over available space. Kurpel et al. (2020) presented techniques to obtain bounds and exact approaches to solving input minimization and output maximization versions of the multiple CLP with rectangular boxes. They adapted and evaluated four discretization techniques from the literature and considered mathematical formulations to practical constraints such as rotation, vertical stability, and separation of the boxes. We also refer to a GRASP-based algorithm for the 3D bin packing problem proposed by Parreño et al. (2010). The related algorithm was developed by using a maximal-space heuristic for the CLP during the constructive phase, and several moves were combined in a Variable Neighborhood Descent approach in the improvement phase. For what concerns the use of 3D packing problems arising in freight transportation, we refer to the surveys by Iori and Martello (2010), Schmid et al. (2013) and Vidal et al. (2013).

The works that, in our opinion, most resemble ours are Alonso et al. (2016) and Ranck Júnior et al. (2019), which, however, do not consider families nor visibility and contiguity constraints. Ranck Júnior et al. (2019) addressed a real problem of a beverage company for packing boxes into a multi-compartment container and delivering over a predefined route, meeting the practical constraints of orientation, stability, load-bearing strength, and load balancing. The proposed heuristic is a hybrid method to create a solution based on layers. These are created via the First Fit and Best Fit heuristics; next, to generate a final solution, a mathematical model is solved, meeting the delivery constraints over the boxes. An iterated local search is carried out over individual layers as an improvement phase, in an attempt to generate new box dispositions.

Alonso et al. (2016) considered practical constraints through a real example originating from a logistics company required to load products into pallets (pallet building) and then load the created pallets into trucks (truck loading), by considering several practical constraints. For the pallet building, they incorporated orientation, support, priority, and stackability constraints. Regarding the truck loading, they adopted restrictions due to priority among pallets, stability, and stackability. They proposed a GRASP algorithm using a constructive phase, a randomized strategy to diversify the solutions, and an improvement phase. The efficiency of their GRASP was analyzed by comparing it with lower bounding procedures. The study was later extended in Alonso et al. (2019, 2017), who developed mathematical models for the case of multiple container loading, addressing several additional practical constraints such as vertical and horizontal stability, multi-drop, and load balance. A further extension was proposed by Alonso et al. (2020), which studied the advantages of using a metaheuristic algorithm when comparing to an integer programming formulation.

3 Problem description

We are given a set R of identical pallets. Each pallet has a two-dimensional loading surface of width $W \in \mathbb{Z}^*_+$ and length $L \in \mathbb{Z}^*_+$, which can be used to load items up to a maximum height $H \in \mathbb{Z}^*_+$. We are also given a set $I = \{1, 2, \ldots, n\}$ of 3D rectangular item types, where each item type $i \in I$ contains b^i identical items, each having width $w_i \in \mathbb{Z}^*_+$, length $l_i \in \mathbb{Z}^*_+$, and height $h_i \in \mathbb{Z}^*_+$, such that $w_i \leq W$, $l_i \leq L$, and $h_i \leq H$. When loading the items, 90-degree rotations are allowed.

In addition to item type, the concept of family is also essential to describe the PBP. Type only refers to a specific characteristic, while family is a more general concept, covering at least one characteristic, e.g., geometric dimensions, material type, purpose of use, and so on. More in detail, item types are partitioned into a set F of families as follows. Each item type i belongs to a given family $f \in F$, which, in the real instances we have addressed, is defined as a set of item types having similar height and weight. Note, however, that in other applications, families could also be defined differently, e.g., each family could be made up of products of the same customer.

Items can be used to form layers. Each layer is a 2D packing of items whose total width does not exceed W, and whose total length does not exceed L. When building a 2D packing, we suppose a layer occupies the positive quadrant of the Cartesian system, with width parallel to the x-axis, length parallel to the y-axis, and bottom-left corner located in the origin of the axes, i.e., we refer to the point (0,0) as the reference point of a layer. When packing an item j in a layer, we refer to the reference point of j as its bottom-left vertex (x_{1j}, y_{1j}). The three other item vertices can be directly determined as (x_{2j}, y_{1j}), (x_{1j}, y_{2j}), (x_{2j}, y_{2j}), where $x_{2j} = x_{1j} + w_j$ and $y_{2j} = y_{1j} + l_j$. If item j is rotated, then width and length of the item must be switched. Regarding the composition of the layers, we consider three types of layers:

- *single-item layers* are formed by a unique type of items;
- *single-family layers* are formed by different item types, but all belonging to the same family;
- *residual layers* are formed by a combination of items of different families or by items belonging to the same family in case these do not cover a minimum predefined area of the loading surface (see the fill factor constraint described below). A pallet can contain at most one residual layer, which has to be placed on top of all other layers.

About this classification, we remark that both single-item and single-family layers are required to meet a so called fill factor constraint, i.e., items on these layers should cover a minimum occupation area (see below for a formal description). Moreover, as previously mentioned, items belonging to the same family are required to have similar heights. Both requirements aim at guaranteeing that single-item and single-family layers can be used in a 3D packing to support other layers that are packed on top of them. Note that these requirements are not imposed for residual layers.

Similar to all packing problems, items cannot overlap. To this aim, we apply the *Separating Axis theorem* (see, e.g., Gottschalk (1996)) to find a feasible position for each item inside a layer. This theorem is applied to general shapes and states that two convex polytopes are disjoint if and only if there is an orthogonal axis, the separating axis, which separates either faces or edges of the two polytopes. Applying this theorem to our purpose, two rectangles do not overlap if and only if we can draw a straight horizontal line or straight vertical line or both between them.

Let us call a *group* a set of items having the same item type and being loaded in the same layer. Besides the overlap constraint, packings of items in a layer should fulfill two operational constraints that concern groups and are aimed at easing loading/unloading operations when the pallet is composed/delivered. These constraints are named *contiguity* and *visibility* constraints, and are defined as follows:

– *contiguity*: given two items of the same group, we introduce a maximum distance that can separate the two items in order to consider them as contiguous. Let $G \subseteq I$ be a generic subset of item types packed into a layer, we define parameter ξ as

$$0 \leq \xi \leq \min_{i \in G}(w_i, l_i),$$

i.e., ξ lies in the interval between 0 and the smallest edge length among all items in G. This way, two items j, q selected from this layer are considered as contiguous if the smallest distance between the edges of j and the edges of q is strictly lower than ξ. Roughly speaking, we guarantee that no other item can fit between j and q. The contiguity relation can be modeled by the overlapping relation. Indeed, we can view contiguity between two items as an overlap between the two items if their sizes are augmented by $\frac{\xi}{2}$ in each direction. Figure 1 illustrates this fact, considering the items j and q when the parameter ξ is fixed to the value $\min_{i \in G}(w_i, l_i)$. The notion of contiguity between items is employed to define the contiguity constraint for items of a given type i over a generic layer. Such constraint is met if, for each type $i \in G$, any item of type i is contiguous to at least another item of the same type (of course, provided that the number of items of that type is larger than one) and, moreover, there are no separated sub-groups of items of type i (i.e., subgroups whose distance is ξ or more). Stated in another way, the graph whose nodes are the items of type i and whose edges are defined by the contiguity relation, must be connected. Addressing a practical scenario, this constraint is very useful when a high number of different products (items) have to be placed in a container (layer). Indeed, contiguous item groups require less effort and time for load/unload operations.

– *visibility*: similarly to what we stated for the contiguity, we say that a group is visible from outside if, for at least one item in that group, the distance between its edges and one of the borders of the layer is strictly lower than ξ. In the same line, this constraint is very useful in a practical scenario to reduce the unload time because it allows to see from the outside which items compose the layer in a pallet. In general, this constraint is more commonly applied in small and medium companies due to their greater level of non robotized work in packaging logistics.

In Fig. 2a, we show an example where the contiguity constraint is not met. The fact that items of the same type are spread around within the layer slows down both the loading and the unloading operations. In Fig. 2b, we show an example where the visibility constraint is not met. In this case, we notice that the set of items lying completely in the interior of the layer

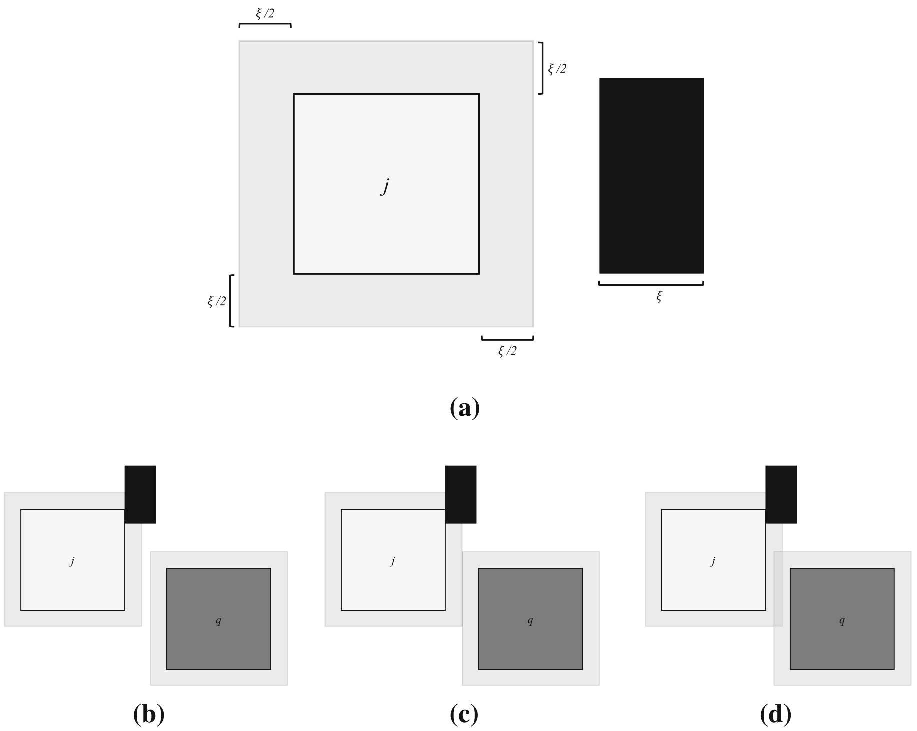

Fig. 1 Contiguity between items based on the overlapping between the same items with sizes enlarged by $\xi/2$ in each direction: given the enlarged items j and q, they will not be contiguous if their enlarged regions are either separated (1b) or, at most, share a portion of an edge (1c). Instead, they are contiguous if the enlarged regions overlap (1d), as it is impossible to insert another item between j and q

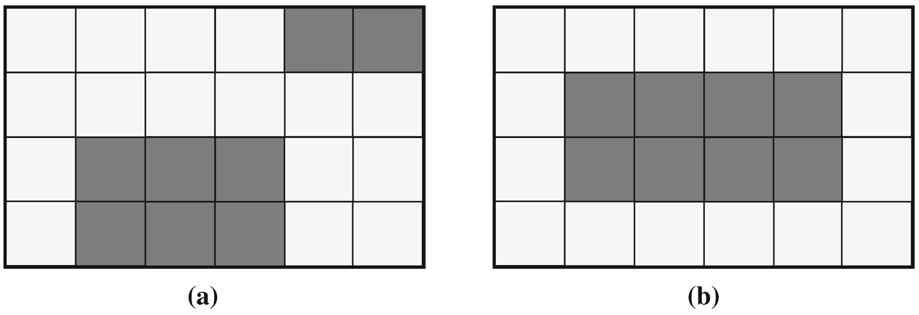

Fig. 2 An example where the contiguity constraint is not met (**a**) and an example where the visibility constraint is not met (**b**)

can only be loaded before and unloaded after the other items, thus forcing a precedence in the order with which the items are loaded and unloaded.

Considering layers in a pallet, the height of each layer corresponds to the height of the highest item in the layer (recall that in single-family layers the heights of the items are quite similar). The height cannot exceed H. Moreover, we impose two additional conditions:

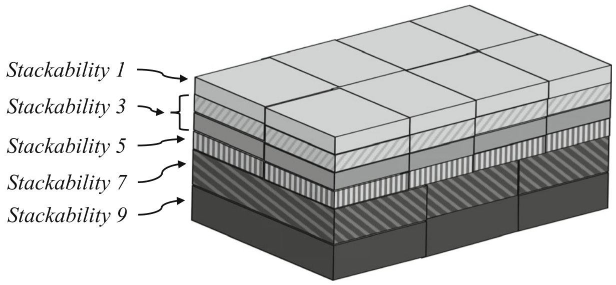

Fig. 3 Graphical illustration of the stackability constraint

- a *stackability constraint* imposes that resistant items cannot be on top of fragile ones. For a layer f with a set of items φ, its stackability ρ_f is set equal to the largest stackability value among all items in φ. Layer f cannot be put below layer g if $\rho_f < \rho_g$ (see Fig. 3).
- a layer can be used to support other layers only if its total area loaded with items reaches a minimum fraction α of the total loading surface WL. Parameter $0 < \alpha \leq 1$ is called *fill-factor*. A layer with a loaded area lower than αWL can still be used to build a pallet but can only be the topmost layer. We call this the *fill factor constraint*. The choice of the parameter α will be commented in Sect. 6.

We note that the stackability constraint above introduces a simplification of the real weight that an item has to bear in a load. The simplification is widely adopted in the literature as it works well in practice, see, e.g., Bortfeldt and Wäscher (2013). For a more elaborate formulation of load-bearing constraints, we refer, e.g., to de Queiroz and Miyazawa (2013, 2014).

The PBP that we face aims to load all items into the minimum number of pallets by ensuring that the following constraints are satisfied:

- all items are packed in layers by satisfying contiguity and visibility constraints;
- at most one residual layer can be used per pallet, and, in such a case, it is placed at the top of the pallet;
- single-item layers can be used to support any type of layers on top of them, as well as single-family layers can be used to support single-family and residual layers, as long as stackability and fill-factor constraints are satisfied;
- the total height of the layers in any pallet does not exceed H.

Figure 4 shows a toy instance of the described problem with a single pallet. Figure 4a represents family 1 with six item types. Figure 4b represents family 2 with one item type. Figure 4c represents family 3 with two item types. Figure 4d represents the pallet. First, layers are created as shown in Fig. 4e with, from top to bottom, a residual, four single-family and two single-item layers, respectively; next, the layers are packed inside the pallet for building the final solution as displayed in Fig. 4f.

4 A MILP formulation

While a mathematical programming model for the whole problem would be possible, its dimension would be quite large and impossible to solve within reasonable computing times. For this reason, we define two separate models, one for the creation of single layers from a selected set of items, and one for the creation of pallets once the layers are given.

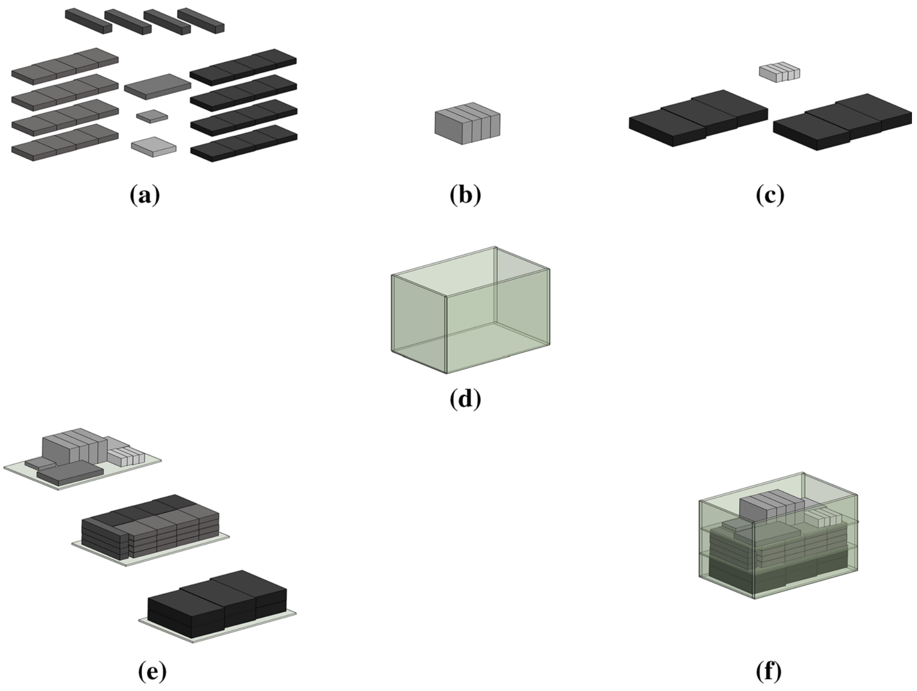

Fig. 4 PBP toy instance

4.1 MILP formulation to create layers

Let $\mathcal{G}$ be the set of items that can be inserted into a given layer. Let m be the number of different item types in $\mathcal{G}$ and, for each item type i, let b^i be the number of items of type i in $\mathcal{G}$. The proposed mathematical model finds a feasible disposition for items in $\mathcal{G}$ by meeting the following constraints.

Non overlap. Let us define $\Phi_{\mathcal{G}} = \{(i, p, j, q) : i, p = 1, \ldots, m, j = 1, \ldots, b^i, q = 1, \ldots, b^p, i > p \vee (i = p \wedge j > q)\}$. The problem of determining if the items of the set $\mathcal{G}$ can fit into a layer without overlapping can be modeled as follows:

$$\sum\nolimits_{u=1}^{4} o_{jqu}^{ip} \geq 1 \qquad (i, p, j, q) \in \Phi_{\mathcal{G}} \tag{1}$$

$$x_{2q}^{p} - x_{1j}^{i} \leq W(1 - o_{jq1}^{ip}) \qquad (i, p, j, q) \in \Phi_{\mathcal{G}} \tag{2}$$

$$x_{2j}^{i} - x_{1q}^{p} \leq W(1 - o_{jq2}^{ip}) \qquad (i, p, j, q) \in \Phi_{\mathcal{G}} \tag{3}$$

$$y_{2q}^{p} - y_{1j}^{i} \leq L(1 - o_{jq3}^{ip}) \qquad (i, p, j, q) \in \Phi_{\mathcal{G}} \tag{4}$$

$$y_{2j}^{i} - y_{1q}^{p} \leq L(1 - o_{jq4}^{ip}) \qquad (i, p, j, q) \in \Phi_{\mathcal{G}} \tag{5}$$

$$x_{2j}^{i} = x_{1j}^{i} + (1 - r_j^i)w^i + r_j^i l^i \qquad i = 1, \ldots, m, \ j = 1, \ldots, b^i \tag{6}$$

$$y_{2j}^{i} = y_{1j}^{i} + r_j^i w^i + (1 - r_j^i)l^i \qquad i = 1, \ldots, m, \ j = 1, \ldots, b^i \tag{7}$$

$$0 \leq x_{1j}^{i} \leq x_{2j}^{i} \leq W \qquad i = 1, \ldots, m, \ j = 1, \ldots, b^i \tag{8}$$

$$0 \le y^i_{1j} \le y^i_{2j} \le L \qquad i = 1, \dots, m,\ j = 1, \dots, b^i \tag{9}$$

$$r^i_j \in \{0, 1\} \qquad i = 1, \dots, m,\ j = 1, \dots, b^i \tag{10}$$

$$o^{ip}_{jqu} \in \{0, 1\} \qquad (i, p, j, q) \in \Phi_{\mathcal{G}}, u = 1, \dots, 4 \tag{11}$$

For item j of type i, variables x^i_{1j} and y^i_{1j} represent the minimum x and y coordinates, respectively, while x^i_{2j} and y^i_{2j} represent the maximum x and y coordinates. The binary variable r^i_j defines the orientation for item j of type i. It is equal to 1 when a 90-degree rotation is applied to the item, 0 otherwise. The binary variable o^{ip}_{jqu} is equal to 1 when there is a separation axis between item j of type i and item q of type p, more precisely, a vertical separation axis for $u \in \{1, 2\}$, and a horizontal separation axis for $u \in \{3, 4\}$. Constraints (1) ensure that between items j and q there is at least one separating axis. Constraints (2) ensure that there is a vertical separating axis between item j and item q if the binary variable o^{ip}_{jq1} is equal to 1 (in this case j lies at the right of q). Similarly for constraints (3) (in this case j lies at the left of q). Constraints (4) ensure that there is a horizontal separating axis between item j and item q if the binary variable o^{ip}_{jq3} is equal to 1 (in this case j lies above q). Similarly for constraints (5) (in this case j lies below q). Constraints (6) and (7) define the x_2 and y_2 coordinates of the items, taking into account their orientation. Constraints (8) and (9) ensure that the items lie inside the layer.

Grouping items. In order to meet the contiguity constraints, we first need to model the contiguity relation between items of the same type i. As previously seen in Sect. 3, contiguity is modeled as an overlap between enlarged items (enlarged by $\xi/2$ in each direction). Thus, we can model contiguity by turning the non overlapping constraints (2)-(5) into overlapping constraints between the enlarged items. More precisely, for two items j and q of the same type, contiguity is modeled as follows:

$$\left(x_{2q} + \frac{\xi}{2}\right) - \left(x_{1j} - \frac{\xi}{2}\right) \ge -W(1 - \bar{d}_{jq})$$
$$\left(x_{2j} + \frac{\xi}{2}\right) - \left(x_{1q} - \frac{\xi}{2}\right) \ge -W(1 - \bar{d}_{jq})$$
$$\left(y_{2q} + \frac{\xi}{2}\right) - \left(y_{1j} - \frac{\xi}{2}\right) \ge -L(1 - \bar{d}_{jq})$$
$$\left(y_{2j} + \frac{\xi}{2}\right) - \left(y_{1q} - \frac{\xi}{2}\right) \ge -L(1 - \bar{d}_{jq})$$
$$\bar{d}_{jq} \in \{0, 1\}.$$

The binary variable $\bar{d}_{jq}$ models contiguity since it can be equal to 1 only when there is an overlap between the enlarged items j and q. Now, let $\tau_{\mathcal{G}} = \{(i, j, q) : i = 1, \dots, m, 1 \le q < j \le b^i\}$. The problem of determining if item j is contiguous to item q of the same type i can be modeled as follows:

$$x^i_{2q} - x^i_{1j} + \xi \ge -W\ (1 - \bar{d}^i_{jq}) \qquad (i, j, q) \in \tau_{\mathcal{G}} \tag{12}$$

$$x^i_{2j} - x^i_{1q} + \xi \ge -W\ (1 - \bar{d}^i_{jq}) \qquad (i, j, q) \in \tau_{\mathcal{G}} \tag{13}$$

$$y^i_{2q} - y^i_{1j} + \xi \ge -L\ (1 - \bar{d}^i_{jq}) \qquad (i, j, q) \in \tau_{\mathcal{G}} \tag{14}$$

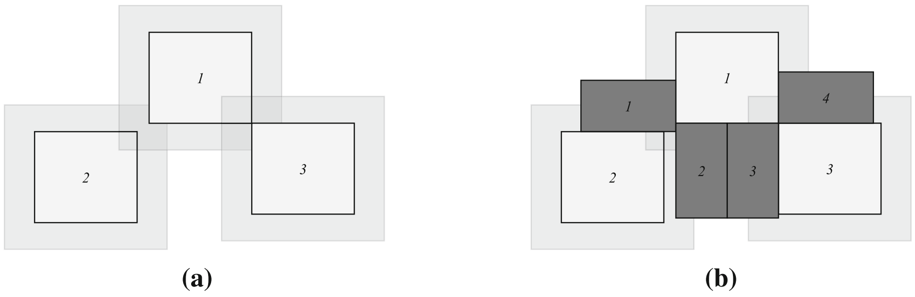

Fig. 5 Special case for the contiguity constraint: the contiguity constraint is always met when considering the group of light and dark items (**a** and **b**). For what concerns the light items, items 1 and 2 are contiguous and so are items 2 and 3, so that the contiguity relation defines a connected graph. The same holds for the dark items. However, in practice, when considering **b**, we notice that the group of dark items creates a physical barrier between the light ones

$$y^i_{2j} - y^i_{1q} + \xi \geq -L\,(1 - \bar{d}^i_{jq}) \qquad (i, j, q) \in \tau_{\mathcal{G}} \qquad (15)$$

$$\bar{d}^i_{jq} \in \{0, 1\} \qquad (i, j, q) \in \tau_{\mathcal{G}}. \qquad (16)$$

Remark 1 The notion of contiguity that we have introduced is a mild one. It may happen that items are contiguous according to this definition but there is no visible space between them. A simple example is detailed in Fig. 5.

The adoption of a suitable objective function which tends to group items of the same type makes unlikely the situation displayed in Fig. 5. However, we can also deal with this by strengthening the conditions under which contiguity holds. To this end, in addition to the previous constraints (12)-(16), we may also add the following ones, where $\psi_i = \min\{w_i, h_i\}$, i.e., ψ_i is the length of the shortest edge for items of type i:

$$\bar{d}^i_{jq} \leq d^i_{jq1} + d^i_{jq2} + d^i_{jq3} + d^i_{jq4} - 2 \qquad (i, j, q) \in \tau_{\mathcal{G}},$$

$$x^i_{2q} - x^i_{1j} \geq \frac{\psi_i}{2} - (W + \frac{\psi_i}{2})\,(1 - d^i_{jq1}) \qquad (i, j, q) \in \tau_{\mathcal{G}}$$

$$x^i_{2j} - x^i_{1q} \geq \frac{\psi_i}{2} - (W + \frac{\psi_i}{2})\,(1 - d^i_{jq2}) \qquad (i, j, q) \in \tau_{\mathcal{G}}$$

$$y^i_{2q} - y^i_{1j} \geq \frac{\psi_i}{2} - (L + \frac{\psi_i}{2})\,(1 - d^i_{jq3}) \qquad (i, j, q) \in \tau_{\mathcal{G}}$$

$$y^i_{2j} - y^i_{1q} \geq \frac{\psi_i}{2} - (L + \frac{\psi_i}{2})\,(1 - d^i_{jq4}) \qquad (i, j, q) \in \tau_{\mathcal{G}}$$

$$d^i_{jqu} \in \{0, 1\} \qquad (i, j, q) \in \tau_{\mathcal{G}}, u = 1, \ldots, 4.$$

According to these constraints, contiguity between items j and q occurs or, equivalently, $\bar{d}^i_{jq}$ can be equal to 1, if and only if three of the binary variables d^i_{jqu}, $u = 1, \ldots, 4$, are equal to 1. In this case the projections along the x-axis (or along the y-axis) of the horizontal sides (if $d^i_{jq1} = d^i_{jq2} = 1$) or vertical sides (if $d^i_{jq3} = d^i_{jq4} = 1$) of items j and q share a segment of length at least equal to $\frac{\psi_i}{2}$. For instance, if $d^i_{jqu} = 1$ for $u = 1, 2, 3$, then the upper horizontal side of an item and the lower horizontal side of the other item are at distance lower than ξ in view of constraints (12)-(16), and, moreover, their projections along the x-axis share a segment of length at least $\frac{\psi_i}{2}$. Note that two out of four of these binary variables, namely,

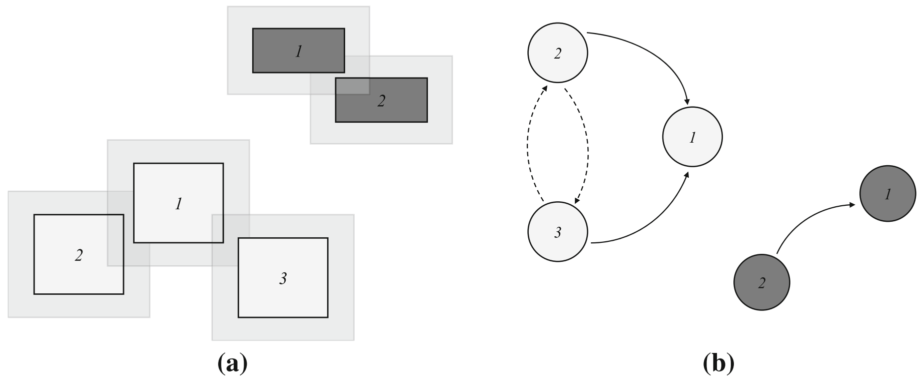

Fig. 6 Contiguous items by item type: **a** graphic and **b** graph representation

one for the horizontal sides ($u \in \{1, 2\}$) and one for the vertical sides ($u \in \{3, 4\}$), can always be set equal to 1. We point out that we do not include the last set of constraints in our computational experiments. Their addition is advised if the notion of contiguity needs to be strengthened.

Unique connected component. We can associate an oriented graph $F_i = (V_i, A_i)$ to each item type i as follows, where J_i denotes the set of all items of type i (note that $|J_i| = b^i$):

- a node j is associated to each item in J_i; one of these nodes (it does not matter which one) is selected as a representative for all items of type i, and we denote it with index 1;
- two additional nodes are introduced, a source S and a destination D;
- an arc from S to each node in J_i is introduced and its capacity is fixed to 1;
- a single oriented arc from node 1 to node D with infinite capacity is introduced;
- given $(i, j, q) \in \tau_{\mathcal{G}}, q \neq 1$, an arc between node $j \in J_i$ and node $q \in J_i$ and an arc between node $q \in J_i$ and node $j \in J_i$ are introduced both with capacity $b^i \cdot \bar{d}^i_{jq}$. Note, in particular, that if items j and q are not contiguous, i.e., if $\bar{d}^i_{jq} = 0$, then the capacity of the arcs is 0. In fact, arcs exiting from node 1 can be omitted.

Figures 6–8 illustrate the notions introduced above. In all cases the dotted and continuous arcs represent arcs with null capacity and with strictly positive capacity, respectively. For two distinct item types, Figure 6 illustrates the two graphs induced by the contiguity relations between the items, while the complete graph is represented in Fig. 7. Given graph F_i, whose arcs have capacity depending on the contiguity variables $\bar{d}^i_{jq}$, the contiguity constraint for items of type i is met if the restriction of the graph to the nodes in J_i, i.e., nodes corresponding to items of type i, is connected. This is equivalent to require that the maximum flow between node S and node D, which cannot be larger than b^i, is exactly equal to b^i.

Figure 7 shows the max-flow problem for the item disposition presented in Fig. 6a. In this case the contiguity constraint is met. On the other hand, Figure 8 shows a case where the contiguity constraint is not met.

Thus, we propose the following constraints to guarantee contiguity between items of the same type, based on the above max-flow formulation:

$$f^i_{jq} \geq 0 \qquad (i, j, q) \in \tau_{\mathcal{G}} \qquad (17)$$

$$f^i_{qj} \geq 0 \qquad (i, j, q) \in \tau_{\mathcal{G}}, q \neq 1 \qquad (18)$$

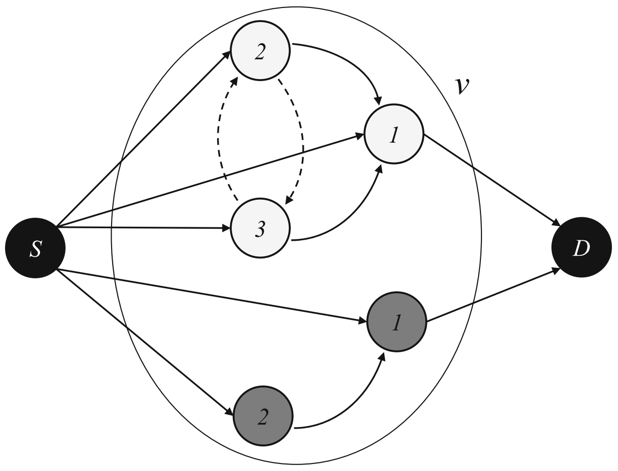

Fig. 7 Max-flow representation: light items 2 and 3 are not contiguous, but both are contiguous to light item 1, maintaining the overall connection of this group. The same holds for the dark items

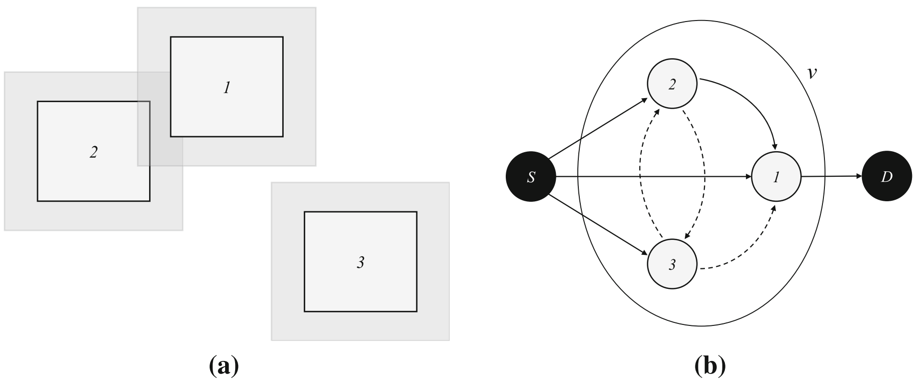

Fig. 8 Item group not fulfilling the contiguity constraint: for the item disposition in 8a, the max-flow in 8b shows that only items 1 and 2 are contiguous, resulting in a flow to D equal to 2, that is different from the number of items (3 in this case)

$$f^i_{jq} \leq \bar{d}^i_{jq} \,.\, b^i \qquad (i,j,q) \in \tau_{\mathcal{G}} \tag{19}$$

$$f^i_{qj} \leq \bar{d}^i_{jq} \,.\, b^i \qquad (i,j,q) \in \tau_{\mathcal{G}}, q \neq 1 \tag{20}$$

$$f^i_{Sj} \leq 1 \qquad i = 1,\ldots,m,\; j = 1,\ldots,b^i \tag{21}$$

$$f^i_{Sj} \geq 0 \qquad i = 1,\ldots,m,\; j = 1,\ldots,b^i \tag{22}$$

$$\sum\nolimits_{q=1}^{b^i} f^i_{jq} = \sum\nolimits_{q=2}^{b^i} f^i_{qj} + f^i_{Sj} \qquad i = 1,\ldots,m,\; j = 2,\ldots,b^i, q \neq j \tag{23}$$

$$f^i_{1D} = \sum\nolimits_{j=2}^{b^i} f^i_{j1} + f^i_{S1} \qquad i = 1,\ldots,m \tag{24}$$

$$f^i_{1D} = b^i \qquad i = 1,\ldots,m \tag{25}$$

Variable f^i_{jq} can be strictly positive only when j and q of the same type i are contiguous, otherwise its value is equal to 0. Constraints (17) and (18) ensure nonnegativity of the flow along arcs between nodes in J_i; (19) and (20) are the capacity constraints for the same arcs (note again that for arcs between nodes such that $\bar{d}^i_{jq} = 0$ the capacity is equal to 0); constraints (21) and (22) impose nonnegativity of the flow and the capacity, respectively, for arcs between node S and nodes in J_i; constraints (23) ensure flow conservation for all nodes in J_i except node 1; constraints (24) ensure flow conservation for node 1, and, thus, also defines the overall flow received by node D; finally, constraints (25) impose that the maximum flow between S and D is b^i, which ensures that items in J_i form a connected component and, thus, that the contiguity constraint for items of type i is met.

Item visibility. To meet the visibility constraint, each group of items of type $i \in G$ must have at least one item visible from the border. In simple terms, we apply the idea of grouping items considering the borders as generic items (top, bottom, right, left) that must be contiguous to a specific item of the group i (without loss of generality, we set this item equal to 1). More precisely, visibility is modeled by imposing the contiguity between item 1 of the given type, enlarged by $\frac{\xi}{2}$ in all directions, and the borders of the layer, all moved by $\frac{\xi}{2}$ towards the interior of the layer. That leads to the following constraints to model the group visibility:

$$\sum_{u=1}^{4} v^i_u \geq 1 \qquad i = 1, \ldots, m \tag{26}$$

$$x^i_{11} - \xi \leq W\,(1 - v^i_1) \qquad i = 1, \ldots, m \tag{27}$$

$$x^i_{21} + \xi \geq W\, v^i_2 \qquad i = 1, \ldots, m \tag{28}$$

$$y^i_{11} - \xi \leq L\,(1 - v^i_3) \qquad i = 1, \ldots, m \tag{29}$$

$$y^i_{21} + \xi \geq L\, v^i_4 \qquad i = 1, \ldots, m \tag{30}$$

$$v^i_u \in \{0, 1\} \qquad i = 1, \ldots, m, u = 1, \ldots, 4 \tag{31}$$

The binary variable v^i_u is equal to 1 when item 1 of type i is contiguous (i..e, within distance ξ) to one of the borders of the layer. In particular, taking into account constraints (27)-(30), $v^i_1 = 1$ ensures contiguity to the leftmost vertical side of the layer, $v^i_2 = 1$ ensures contiguity to the rightmost vertical side, $v^i_3 = 1$ ensures contiguity to the lowermost horizontal side, $v^i_4 = 1$ ensures contiguity to the uppermost horizontal side. Constraint (26) ensure that item 1 of type i is within distance ξ from at least one of the sides of the layer.

Bounding box. In the proposed approach, for a given set of items, we would like to establish whether there exists a way to place them over the layer so that all constraints are met. Under this respect the problem is a feasibility one. However, it is convenient to introduce an objective function which should favor compact arrangements for items of the same type. Here we use the concept of Bounding Box (rectangle convex hull) to group items of the same type. Therefore, we propose a mathematical model to minimize the sum of the (semi)perimeters of the bounding boxes that cover all items of each item type $i \in G$ inside the layer. The overall model to place items over the layer is the following:

$$\text{Minimize} \quad \sum_{i=1}^{m} [(X^i_2 - X^i_1) + (Y^i_2 - Y^i_1)] \tag{32}$$

$$\text{subject to} \quad X^i_1 \leq x^i_{1j} \qquad i = 1, \ldots, m, \; j = 1, \ldots, b^i \tag{33}$$

$$Y^i_1 \leq y^i_{1j} \qquad i = 1, \ldots, m, \; j = 1, \ldots, b^i \tag{34}$$

$$X^i_2 \geq x^i_{2j} \qquad i = 1, \ldots, m, \; j = 1, \ldots, b^i \tag{35}$$

$$Y^i_2 \geq y^i_{2j} \qquad i = 1, \ldots, m, \; j = 1, \ldots, b^i \tag{36}$$

$$(1) - (31).$$

Variables X_1^i and Y_1^i represent, respectively, the minimum x and y bounding box coordinates for items of type i, while X_2^i and Y_2^i represent the maximum x and y bounding box coordinates when considering item group of type i. Constraints (33)-(36) define the extreme points of a bounding box enclosing all items of type i, while the objective function is the sum of the (semi)perimeters of the bounding boxes for all the item types inside the layer, to be minimized.
As a final remark, we point out once again that the model is applied to a fixed set of items. Once the model has been solved, if a solution has been detected, then a new item is added to the set and the new model is solved. Instead, if no feasible solution is detected, the layer is declared complete and a new one is started.

4.2 Integer formulation for building pallets

The second mathematical model deals with the insertion of the generated layers into pallets in such a way that the number of pallets is minimized. In simple words, this mathematical model "sorts" the layers in pallets as described in Sect. 3: a pallet is made up by single-item layers in its base (if any), single-family layers in the middle (if any), and, possibly, by a residual layer in its top part. This practical constraint is defined to simplify the problem avoiding the addition of the stability constraint, since this structure generates more stable pallets when compared to the use of shuffled layer types.

For the sake of clarity, we present the following notation for data of the mathematical model: P stands for the set of pallets; R stands for the set of residual layers; I stands for the set of single-item layers; F stands for the set of single-family layers; $M = I \cup F$; $L = R \cup M$; H stands for the height of the pallet; h_t stands for the height of layer t; and S_t stands for the stackability of layer t. We make use of two families of binary variables:

$$x_{tp} : \text{it is equal to 1 if layer } t \in L \text{ is packed in pallet } p \in P,\ 0 \text{ otherwise;}$$
$$y_p : \text{it is equal to 1 if pallet } p \in P \text{ is used, 0 otherwise.}$$

Then, we obtain the following mathematical model:

$$\text{Minimize} \sum_{p \in P} y_p \tag{37}$$
$$\text{subject to} \sum_{t \in L} h_t x_{tp} \leq H y_p \qquad p \in P \tag{38}$$
$$\sum_{p \in P} x_{tp} = 1 \qquad t \in L \tag{39}$$
$$\sum_{t \in R} x_{tp} \leq y_p \qquad p \in P \tag{40}$$
$$\sum_{t' \in M : S_{t'} < S_t} x_{t'p} \leq |M|(1 - x_{tp}) \qquad t \in R,\ p \in P \tag{41}$$
$$\sum_{t' \in I : S_{t'} < S_t} x_{t'p} \leq |I|(1 - x_{tp}) \qquad t \in F,\ p \in P \tag{42}$$
$$x_{tp} \in \{0, 1\} \qquad t \in L,\ p \in P \tag{43}$$
$$y_p \in \{0, 1\} \qquad p \in P. \tag{44}$$

The objective function (37) aims at minimizing the number of used pallets. Constraints (38) ensure that the sum of the heights of all layers in p is not higher than H if pallet p is used. Constraints (39) ensure that each layer is placed in exactly one pallet. Constraints (40) ensure that at most a residual layer $t \in R$ can be packed in a pallet $p \in P$ (namely, at the top of the pallet). Constraints (41) ensure that, for each layer $t \in R$, the layers $t' \in M$ with stackability lower than t cannot be placed on the same pallet with t (recall that t is placed on top of all other layers and, thus, its stackability cannot be larger than that of the other layers in the pallet). Constraints (42) are similar to the previous ones, but we have the correspondence between single-item and single-family layers in the pallet, by requiring that all stackability values of the single-item layers are greater than the stackability values of the single-family layers (since the former are always placed below the latter). Note that stackability constraints should also be imposed between each pair of single-item or single-family layers. However, once we have established which single-item (single-family) layers are placed on the pallet, we only need to order them according to their stackability value. Constraints (43) and (44) are binary conditions for the variables.

As described in Sect. 3, we highlight that the height of each layer is the height of the highest item in the layer. Similarly, the stackability of a layer is computed as the maximum stackability value among the items in the layer.

5 Solution algorithms

In this section, we present the heuristic algorithms that we developed to solve the PBP. We first adopted a constructive heuristic (Section 5.1), then extended it to obtain a reactive GRASP metaheuristic (Sect. 5.2), and finally embedded the mathematical models of Sect. 4, obtaining a so-called matheuristic (Section 5.3). All these methods are based on the decomposition of the PBP into its two main components, first layer creation and then pallet building, already adopted in Sect. 4.

5.1 Constructive heuristic

The constructive heuristic that we developed is named Modified Extreme Points Heuristic (EPMH). It has been presented in detail in our preliminary work Iori et al. (2020b). We report here its main components to make the paper self-contained and because they are at the basis of the next GRASP algorithm.

Creating layers

Layer creation and classification. The heuristic tries in a first phase to create as many single-item and single-family layers as possible. Then, in a second phase, it takes care of creating the residual layers. This is because at most one residual layer can be inserted in a pallet, and so their number should be as small as possible.

In the first phase, families and their item types are sorted by a specific criterion. Using this order, the heuristic tries to generate single-item layers by packing items one at a time according to the concept of modified extreme points that we describe below. If the packing obtained with a single item type meets the fill-factor constraint, but also has free space to potentially accommodate more items, then the heuristic tries to add more items of the same family to generate a single-family layer. Once no more item fits the residual space, the layer is closed. When, instead, the current packing of a layer does not fulfill the minimum fill-factor, then the layer is destroyed and all items are inserted in a residual item list. As speed-up technique, when the sum of the areas of the items of a specific family is not enough to meet the fill-factor constraint, then the items are directly inserted in the residual item list.

At the end of the first phase, no more single-item and single-family layers can be built, so the heuristic focuses on the items in the residual list. The same criterion used in the first phase is also used in this second phase to select items one at a time and fill a layer as much as possible. The process is repeated, layer after layer, until all items have been packed in residual layers.

Item positioning. To pack an item in a layer, we propose an adaptation of the Extreme Points Heuristic (EPH) presented by Crainic et al. (2008). Originally, EPH was built for 3D packings, but we restrict here the discussion to the 2D case. EPH works with the concept of extreme points. An extreme point e is a point in the 2D space where an item can be packed by taking into account the partial solution built so far. For the sake of clarity, packing an item in an extreme point e means packing its bottom-left corner (reference point) in e.

In EPH, the items are packed one at a time in the layer, by considering a set E of available extreme points initialized with the origin point $(0, 0)$. Then, each time a new item is packed, E is updated by removing the point used for the packing and possibly inserting new extreme points. These new extreme points are obtained by computing the projection of the last item packed over the partial packing solution under construction, considering the axes x and y. For the x-axis, EPH horizontally projects the top edge of the item to its left, until the projection touches a previously packed item or the left border of the layer (i.e., the y-axis). This is the first extreme point that is possibly created. For the y-axis, instead, EPH considers the right edge of the item and vertically projects it towards the bottom until the projection reaches a previously packed item or the bottom border of the layer (i.e., the x-axis). This is the second extreme point possibly created. These two points are added to E if they were not already included in it. Figure 9a depicts an example with a set E formed by white and black points. The white point is selected for packing an item of type k and is thus removed from E. The dark gray points are the new extreme points added to E after the packing.

Note that, besides the projections from the last item packed, it is mandatory to verify all projections from previously packed items on the last item. This step, that we name past projections, is needed to find new extreme points not yet available. We execute it right after having computed the projections from the last item packed.

The original EPH solves a pure 2D packing problem, so we need to include a set of changes to be able to solve the more complex PBP with visibility and contiguity constraints. The first modification we apply consists in increasing the search space inside a layer. The EPH creates only two new extreme points at a time, whereas visibility and contiguity constraints narrow the search space. As a consequence, a layer could be closed even when its occupation is low because E does not contain any feasible extreme point. Figure 9b shows an example where this situation occurs because none of the current feasible extreme points fulfills the contiguity constraints.

To overcome this limitation, we included some new extreme points, considering two cases for contiguity and visibility. In the first case, the extreme points are included around the last item packed to allow the contiguity with the next items of the same type. Basically, four new points are added: the item top-left and bottom-right corners allow the top and right connections, respectively, and the points on the left and below the item enable the left and bottom connections, respectively. Figure 9c shows the new extreme points.

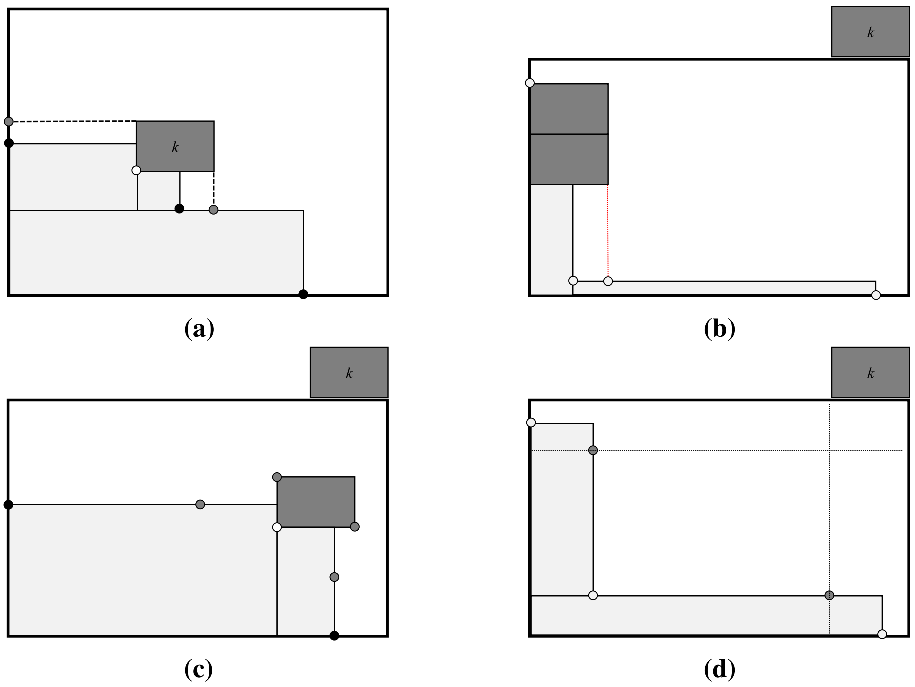

Fig. 9 Extreme points: (9a) Black and white points form the current set E; the white point is chosen to pack k and is then removed from E; dark gray points are added to E after the packing; (9b) Case where EPH generates only extreme points that are infeasible for the PBP with visibility and contiguity constraints; (9c) New extreme points created for contiguity: after packing k in the white point, four extreme points (represented by dark gray points) are included in E; (9d) New extreme points created for visibility: new dark gray points that meet visibility constraints are added to E

Extreme points regarding the visibility case are included close to the top and right layer borders, when the current item has a different type from the previously packed item and there is no point in E that allows the current item to meet all constraints. In this case, two new points are added: the first feasible point that meets the contiguity to the top layer border, proceeding from right to the left; and the first feasible point that meets the contiguity to the right layer border, proceeding from top to bottom. Figure 9d shows the new extreme points for this case.

The second modification is related to performance. Current extreme points are directly linked to visibility and contiguity constraints. These points can be either feasible or infeasible for the current item according to its type and to the partial solution (because of contiguity and visibility). Thus, we work with two extreme points sets: *feasible extreme points*, E_f, and *infeasible extreme points*, E_i, representing, respectively, points that meet or do not meet the visibility and contiguity constraints for the current item. Therefore, before adding an item to the partial solution, we first update these two sets, and only after this is done we check if the item overlaps with previously packed items. Note that these two sets are related to the current configuration, and after the addition of a new item, infeasible points may become feasible and vice-versa. The benefit of using these sets is that we avoid checking the overlapping constraint for all extreme points. An example of sets E_f and E_i is provided in Fig. 10.

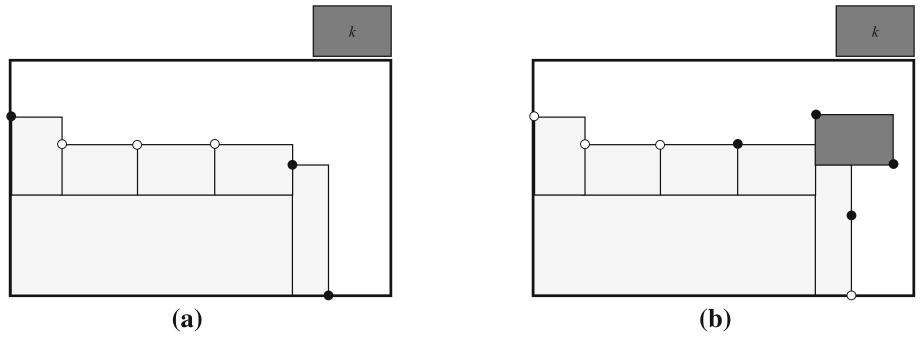

Fig. 10 Partial solutions and their feasible (black) and infeasible (light gray) extreme points for the next item of type k: representation of the extreme points that meet visibility (10a) and contiguity (10b) constraints

Evaluation Function. Given the current item and its set E_f, the point $e \in E_f$ that results in the best packing has to be chosen. For this purpose, the algorithm uses a specific fitness evaluation function that evaluates how good a packing is. The algorithm chooses the point $e \in E_f$ for which the resulting bounding box area is a minimum. When the current item is the first one in its group (recall a group is a subset of items of the same type packed in the same layer), its bounding box is calculated considering all items of the partial packing in the layer. When, instead, there is at least a previously packed item of the same group, then the fitness is calculated by considering only the items of this specific group. The use of the bounding box as evaluation function is motivated by extensive experiments in Iori et al. (2020b), where we tested several other functions (whose list is provided in the next section).

Building pallets

Once all layers have been generated, we need to stack them in the minimum number of pallets. To this aim, we developed a simple greedy constructive heuristic that works as follows. Recall the height of a layer is calculated as the height of the highest item in the layer, and, similarly, the stackability of a layer is computed as the maximum stackability among the items in the layer. Single-item, single-family and residual groups of layers are sorted according to non increasing stackability, breaking ties by non increasing height, and this order is maintained throughout all the algorithm.

As residual layers need to be packed in separated pallets, the heuristic starts by choosing the first residual layer, if any, in the order, and uses it to initialize a pallet. Then, it fills the current pallet with single-item layers, one at a time in the order above, by fulfilling stackability and maximum height constraints. If the current pallet has still unused height, but no more single-item layer can enter it, the heuristic attempts filling the pallet with single-family layers. In this case too, the layers are scanned according to the above order. The process is repeated until no more layer can enter the pallet. In such case the pallet is closed and a new pallet is considered. The process continues until all layers are packed into pallets, thus creating a feasible solution.

5.2 Reactive GRASP metaheuristic

The Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic originally developed by Feo and Resende (1989, 1995). Roughly speaking, the standard GRASP is an iterative technique that mixes greedy and randomized processes with local search through a two-phase approach: first, a random greedy procedure is used to explore the solution space in a construction phase; then, local search procedures are invoked to improve solutions. Considering alternative techniques for the GRASP construction phase (see Resende and Ribeiro (2019)), we highlight and use the so-called *reactive method*, firstly proposed by Prais and Ribeiro (2000). Basically, this is a probabilistic method that adapts the values taken by certain elements according to the quality of the solutions previously found.

The *Reactive GRASP with Extreme Points* (GREP) that we developed in our preliminary work Iori et al. (2020a) for the PBP is described in Algorithm 1. First, the required variables are initialized (line 2) and the item type set is sorted (line 3). Then, as long as the stopping condition is not met (i.e., a maximum execution time is not reached, line 4), the algorithm selects a fitness evaluation function (line 5) to construct a feasible solution to the problem (line 8). The solution is evaluated (line 9) according to a fitness function, to be maximized. We will use the fitness function (45), discussed later on in Sect. 5.2.3. The fitness of the solution is compared to the current reference value U (line 10): if the solution fitness is higher than U, then the algorithm tries to improve the solution by local search (line 12); otherwise, the local search is not performed and the reference value is decreased after a prefixed number of iterations without an improvement (lines 17-19). Lines 20-29 refer to the reactive method: the best (line 20), worst (line 23) and mean (lines 25-26) fitness values are possibly updated; next, after a prefixed number of iterations (line 27), all fitness values are re-evaluated (line 28) and the probability functions of the reactive method (which are used in the constructive phase, as discussed next) are updated (line 29).

Algorithm 1: REACTIVE GRASP WITH EXTREME POINTS (GREP)

```
Input  : $I$: item type set; $L$: pallet set; $D$: criteria set for selecting extreme points;
         $order\_family, order\_type$: criteria to sort families and item types; $\epsilon$:
         randomness percentage; $\varrho$: reactive parameter; $\beta$: upper bound decrease
         percentage; $\psi$: number of iterations to update probabilities; $\max_\kappa$:
         maximum number of rejected solutions.
Output: $Sol_{best}$: best solution found;
begin
    INITIALIZATION($numIter$, $V_{best}$, $V_{worst}$, $U$, $sum_d$, $mean_d$, $\kappa$, $n_d$, $P_d$);
    $S \leftarrow$ SORTITEMS($I$, $order\_family$, $order\_type$);
    while NOTSTOPCONDITION($time$) do
        $d \leftarrow$ CHOOSECRITERION($D$, $P_d$);
        $n_d = n_d + 1$;
        $numIter = numIter + 1$;
        $Sol \leftarrow$ CONSTRUCTIVE($S$, $L$, $d$, $\epsilon$);
        $V \leftarrow$ EVALUATESOLUTION($Sol$);
        if $V > U$ then
            $\kappa = 0$;
            $Sol \leftarrow$ IMPROVEMENT($S$, $L$, $d$, $\epsilon$);
            $V \leftarrow$ EVALUATESOLUTION($Sol$);
            $U \leftarrow V$;
        else
            $\kappa \leftarrow \kappa + 1$;
            if $\kappa > \max_\kappa$ then
                $U \leftarrow \beta U$;
                $\kappa = 0$;
        if $V > V_{best}$ then
            $V_{best} \leftarrow V$;
            $Sol_{best} \leftarrow Sol$;
        if $V < V_{worst}$ then
            $V_{worst} \leftarrow V$;
        $sum_d = sum_d + V$;
        $mean_d = \frac{sum_d}{n_d}$;
        if $mod(numIter, \psi) = 0$ then
            $eval_d \leftarrow (\frac{mean_d - V_{worst}}{V_{best} - V_{worst}})^\varrho$ , $\forall\ d \in D$;
            $P_d \leftarrow \frac{eval_d}{\sum_{d' \in D}(eval_{d'})}$ , $\forall\ d \in D$;
    return $Sol_{best}$;
```

In the remaining part of this section, we give full details on the procedures invoked by GREP and on the way parameters are used and updated.

5.2.1 Constructive phase

In this phase, we select one item at a time and pack it in an extreme point (if any), until a layer has been created. We reiterate until all items have been packed into layers, and then stack the layers into pallets.

Creating Layers. The choice of a new element is flexible in GRASP algorithms, since not necessarily the current best element, with respect to a given criterion, is selected, but rather a restricted candidate list of items is considered and the next element is randomly selected from such list. This concept is easily adapted to the deterministic method of selecting the next item described for the layer creation heuristic (Sect. 5.1) and leads to a greedy randomized heuristic that works as follows.

First, the algorithm establishes a specific order for the families and item types, assigning them to the data structure S. Then, it defines the selectable range based on a parameter ϵ. Let F_S be the sorted set of families representing the order of the families in S, and $G_f \subseteq S$ the sorted set of item types $s \in S$ representing the order of the item types of a family $f \in F_S$. A family f is selected randomly among the first $\epsilon|F_S|$ families, and then an item type $s \in G_f$ is selected among the first $\epsilon|G_f|$ item types.

Once an item of type $s \in S$ has been selected for packing, the heuristic determines in which extreme point it should be placed. This is again obtained by performing an adaptation in the greedy heuristic of Sect. 5.1 that takes into consideration ϵ. Let T be the set of all extreme points of E_f for the item type s. Each extreme point in T is evaluated according to a fitness function d. Then, the heuristic sorts all points in T by decreasing value of fitness, and randomly selects the extreme point where to pack the current item among the first $\epsilon|T|$ extreme points of T.

In this process, a key role is played by the fitness functions used to evaluate the quality of the assignment of an item to an extreme point. The reactive method that we adopted serves indeed to this aim. We use an overall set D, which is composed by different fitness evaluation functions, namely: Lower x; Lower y; Bounding Box; Bounding Square; Simple Contacts; Complex Contacts; Distance Sum; Center of Gravity (see Iori et al. (2020b) for details).

The reactive method works as follows: given the fitness evaluation function set D and the probability set P_d to choose in the current iteration the function $d \in D$ associated with the item type $s \in S$, the method initializes P_d to $1/|D|$ and updates these values after a given number ψ of iterations has been elapsed, considering the quality of the generated solutions. In particular, probabilities P_d are updated at line 29 of Algorithm 1. Note that the better the partial solution found by d is, the greater becomes the probability P_d to select d in the next iterations.

The rest of the heuristic works exactly as defined previously (Sect. 5.1), and the algorithm continues until all items have been packed into layers.

Building Pallets. After the layers have been created, GREP stacks them in the minimum number of pallets by using the constructive heuristic for pallet building presented in Sect. 5.1.

5.2.2 Improvement phase

The improvement phase is based on a local search that tries to improve the quality of the solution generated in the constructive phase by applying small changes to it. This is not always an easy task, since in view of the hard PBP constraints, any change in the solution may create infeasibility. For example, in the PBP it is difficult to change the position of individual items without violating contiguity and visibility constraints. Therefore, we opted to generate new solutions from scratch, by applying the small local search changes directly to the initial order of S provided to the constructive heuristic.

In detail, we first randomly select two families f_a and f_b (if any) and switch the order of their items in S. Then, for each family $f \in F$, we randomly select two item types i_c and i_d (if any) in f and switch the order of their items in S. These changes are applied according to a *first improvement strategy* (FIS): considering the original data structure S, a change is applied as soon as it improves the quality of the solution, otherwise it is disregarded and a new one is generated, until a maximum number of iterations is reached. It is worth highlighting that, if a change does not improve the solution, then the next change is applied to the original structure S, and not to the modified one.

Despite the improvement phase constitutes an essential part of the GREP, it is not always carried out, since low-quality starting solutions tend to produce low-quality final solutions. Thus, as shown at line 10 of Algorithm 1, we consider a threshold U to decide if it is worth looking for an improvement. The value of U is initialized with the value of the first solution created. Whenever a new solution has a fitness better than U, the improvement phase is performed and U is updated.

To avoid stagnation during the process, GREP controls U in the following way (lines 16-19 of Algorithm 1): if the fitness of a new solution does not reach U, then GREP increases an iteration counter κ that enumerates the number of consecutive solution rejections. Whenever κ gets larger than a maximum number of rejected solutions ($\max_\kappa$), U is decreased to βU ($\beta \in \mathbb{R}, 0 < \beta < 1$) and κ is reset to 0. Thus, the algorithm forces the improvement phase for solutions which are not necessarily the best ones but are of "good enough" quality. We also highlight that, as soon as a new solution with fitness higher than U is reached, U is set to this new fitness value and κ is reset to 0 (lines 11 and 14 of Algorithm 1).

5.2.3 Evaluating the fitness function of a solution

A basic strategy to compare solutions is to evaluate them on the basis of the number of pallets they produce. However, due to the complexity of the PBP with contiguity and visibility constraints, taking only this value into account may not be appropriate. Analyzing the structure of a solution, it is possible to extract more interesting information, and these peculiarities can help explore the search space. To this regard, let n_r be the number of residual layers, n_l the total number of layers, n_p the total number of pallets, and f_l the average fraction of the 2D occupation of the layers in a solution. Obviously, a good solution is a solution with small n_r and n_p values and with large f_l value. Thus, GREP calculates the fitness function of a solution as:

$$V = \frac{v_{residual} + v_{pallets} + f_l}{3}, \tag{45}$$

where $v_{residual}$ and $v_{pallets}$ are logarithmic functions that consider the proportion of, respectively, n_r and n_p over n_l, being defined as

$$\begin{aligned} v_{residual} &= -\log_{n_l+1} \frac{n_r+1}{n_l+1}, \\ v_{pallets} &= -\log_{n_l+1} \frac{n_p+1}{n_l+1}. \end{aligned}$$

All components of the summation in (45) are in the range between 0 and 1, and V is their average value, which we would like to maximize. The idea behind the use of logarithmic functions is to perform evaluations that are more sensitive to variations in the numbers n_r of residual layers and n_p of pallets when these numbers are close to 0. To clarify this point, we can consider Fig. 11. Such figure shows the behavior of $v_{residual}$ as a function of n_r, when considering $n_l = 70$ layers in total. We notice that the logarithmic curve decreases as n_r increases and has larger derivative at small values of n_r (value $v_{pallets}$ as a function of the number of pallets n_p behaves in a completely similar way).

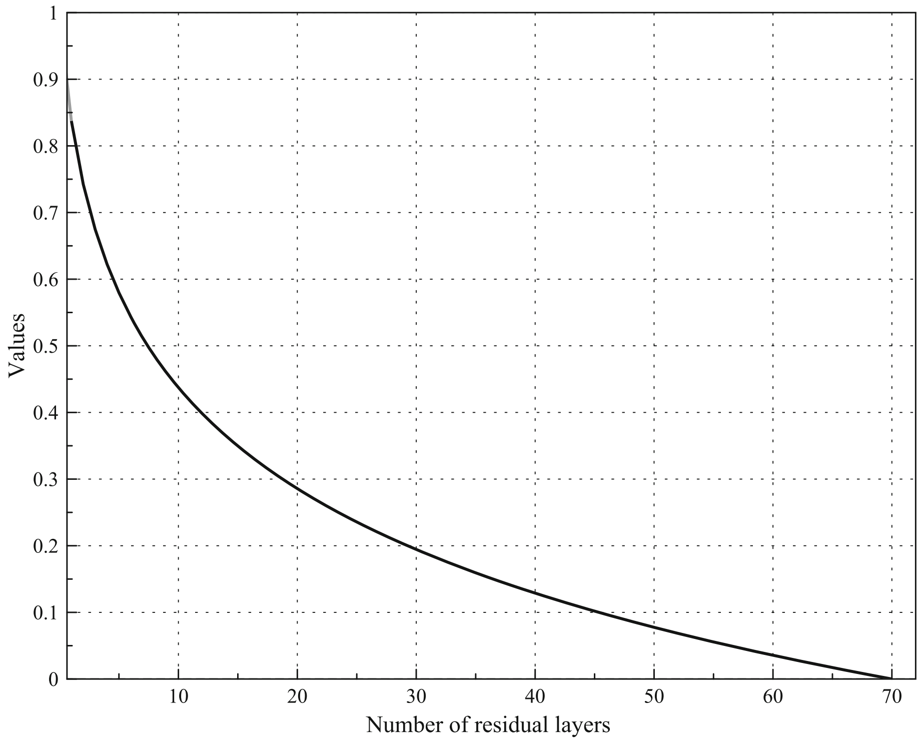

Fig. 11 Behavior of the values for the logarithmic function which represents the variable $v_{residual}$ when considering the increase/decrease in the number of residual layers for a total of 70 layers

5.3 Embedding the mathematical models in the heuristics

We created some variants of the aforementioned heuristics by making use of the mathematical models of Sect. 4.

For what concerns the creation of the pallets once the layers have been built, we created a variant of the EPMH heuristic, called *EPMH with Mathematical Model* (MEPMH), in which we first attempt to create the pallets through the quick greedy heuristic of Sect. 5.1. Then, we perform a second attempt in which we create a (hopefully better) solution by using the model of Sect. 4.2 with a short time limit, as carried out by Iori et al. (2021). Note that this model is relatively simple and is usually solved within a very short time. However, in some cases it takes more time and for this reason we imposed a time limit to its solution. To increase the execution speed of the solver, in the second attempt we set the initial model solution to the solution found by the heuristic during the first attempt. The main idea of this combination is to take advantage of the EPMH heuristic in order to create a solution in a very short time and then, check the possibility of reducing the final number of pallets by using a mathematical model, which, although more time consuming, usually leads to better solutions. The same idea has been used in the GREP algorithm of Sect. 5.2, leading to an alternative algorithm that we call *GREP with Mathematical Model* (MGREP). In this case, the GRASP metaheuristic is exploited to find solutions better than those returned by the EPMH heuristic.

For what concerns the creation of the layers, we did not find convenient using the model of Sect. 4.1 from scratch, as too time consuming, but we opted, instead, to use it to create denser layers than the ones produced by the heuristic of Sect. 5.1. In detail, after the heuristic has created a layer, we attempt to include other items in the layer, one at a time, while preserving feasibility. In simple terms, we insert in the set of items to be packed by the model a new item of an item type already contained in the layer: if the model returns a feasible packing, then we attempt inserting a new item of the same type (if any); if the model does not find a feasible solution, then we attempt inserting an item of a different type (if any). In a family layer, the process is executed considering the only family in the layer. For residual layers, instead, all families and item types are attempted. At each iteration, the model is allowed to run for a short time limit. The process is iterated until no more item can fit into the current layer or a maximum overall time limit is reached. We obtained in this way a modified version of MEPMH, that we call *MEPMH Full* (EPFULL), and a modified version of MGREP, that we call *MGREP Full* (GREPFULL). These strategies allow to create more compact layers and to detect better solutions. However, that comes at the cost of larger execution times, since the mathematical model is complex and its solution time consuming.

6 Computational results

All experiments have been conducted on a Virtual Machine VMware, Intel Xeon CPU E5-2640 v2 2.00GHz, 16GB RAM, Linux Ubuntu Server 18.04 Operating System. The algorithms have been implemented in Java and executed using Oracle JDK 11. We used the solver Gurobi Optimizer 9.1.0 to solve the mathematical models.

6.1 Test instances

For the tests, we adopted the 24 strongly heterogeneous instances originally presented by Iori et al. (2020b). These instances are separated into four groups, each containing six instances, being characterized by a different number of distinct items. Table 1 summarizes the details of the instances. Its columns are: ID number of the instance, number of item types (n), number of families, and total number of items ($\sum b^i$). The instances are of different difficulty. The larger the numbers of item types, families and items are, the more complex becomes the instance to be solved. In particular, the most challenging instances are those with more than 50 item types, 15 families, and 700 items. We report the database with the whole set of instances in https://github.com/silveira-tt/PBP_instances.

For all instances, the minimum required fill factor was set to 55%, and the container dimensions were set to 1500, 1250, and 1050 for height, width, and length, respectively. We allowed rotation of 90 degrees of the items. Note that the minimum fill factor value has been set equal to a relatively small value (55%) since we experimentally observed that larger values impose stronger constraints on the creation of layers and lead to poorer solutions (for more details, see Iori et al. (2021)). We also remark that this is just a minimum fill factor value, but, as we will see in the reported results, the average fill factor of the layers is considerably larger than such minimum. Of course, a layer with a small fill factor may cause instability if another layer is posed on the top of it. In this case the occupation of the layer can be increased by so called filler boxes, which allow to increase the occupation area and, thus, the stability.

We solved the instances by using all the algorithmic variants described in Sect. 5. For each variant, we carried out a simple test for parametric configuration. When GRASP is applied, we set $\epsilon = 0.15$, $\varrho = 10$, $\max_{\kappa} = 5$, $\beta = 0.98$, maximum number of repetitions for FIS equal to 25 and $\psi = 500$, as described by Iori et al. (2020a). For the variant GREPFULL,

Table 1 Instances settings (from Iori et al. (2020a))

ID	n	N. of families	$\sum b^i$
1	20	4	877
2		12	269
3	23	4	388
4		13	115
5	29	6	438
6		15	2103
7	34	7	698
8		17	1322
9	37	6	300
10		17	449
11	48	9	395
12		18	748
13	59	10	455
14		19	633
15	64	11	658
16		19	683
17	61	10	300
18		19	797
19	75	16	790
20		20	829
21	79	17	855
22		21	944
23	66	13	738
24		19	767

instead, we set the maximum number of repetitions for FIS to 1 and $\psi = 10$, due to the reduced number of iterations when using the mathematical models.

All variants involving GRASP have been run 15 times each, so as to obtain a robust evaluation of the performance. These algorithms use an initial structure S sorted by *random* order for families, and a non increasing order of *width* for item types. The deterministic algorithms (EPMH, MEPMH and EPFULL) use instead the Bounding Box function to evaluate the creation of item groups in a layer, as suggested by Iori et al. (2020b), and are run only once. After preliminary experiments, when considering the variants that use only the model for pallet creation (MEPMH and MGREP), we set the time limit for the solver to 30 seconds. When considering the variants that use both models (EPFULL and GREPFULL), we set the time limit to 1 second for the layer creation model and 3 seconds for the pallet creation model. Actually, we also tried larger time limits, but the increased computing times were not compensated by higher quality solutions. When considering GRASP variants, the whole time for the algorithms was set to 5 minutes, except for GREPFULL, where it was set to 15 minutes.

Due to the large number of tests that we performed, in the following we report only aggregate results. For the sake of clarity, we separated the results in different tables in which each row provides average results for a given algorithm on the 360 solutions obtained (15 for each of the 24 attempted instances) in case of randomized algorithms and 24 solutions (one

Table 2 Computational results for the proposed algorithms (the best results are highlighted in bold)

Algorithm	N. of pallets	N. of layers	Pallet filling (3D)			2D fill factor			Seconds
			Min	Max	Avg	Min	Max	Avg	
EPMH	11.29	41.88	0.22	0.84	0.59	0.36	0.96	0.83	0.03
GREP	10.80	42.02	0.26	0.85	0.62	0.49	0.95	0.84	300.00
MEPMH	10.33	41.88	0.32	0.86	0.66	0.36	0.96	0.83	3.69
MGREP	10.33	42.71	0.34	0.85	0.66	0.50	0.95	0.83	300.00
EPFULL	10.13	41.13	0.31	0.86	0.67	0.45	0.95	0.86	184.91
GREPFULL	**9.97**	**40.78**	0.33	0.86	0.68	0.54	0.95	0.86	900.00

per instance) in case of deterministic algorithms. For a more detailed comparison among the algorithms, we refer to https://github.com/silveira-tt/ANOR_PBP-Appendix, where we report full details of all the computational tests that we performed over each instance.

6.2 Evaluation

The average results are summarized in Table 2. In the first three columns the table reports, respectively, the name of the algorithm, the total number of pallets in the solution obtained and the total number of layers created. In the six successive columns, it reports the minimum, maximum and average pallet utilization and fill factor (2D space of all layers). Note that the minimum fill factor is below the imposed lower bound (55%), since layers on top of all the others (in particular, residual layers) are allowed to violate such bound. The last column reports the computational time (in seconds). To facilitate the analysis of the results, we highlight in bold the best results of pallets and layers obtained.

From Table 2, we can notice that the number of layers in EPMH is slightly lower than in GREP, but the number of pallets is larger. Thus, we can highlight that the minimization of the number of layers is less relevant than the way in which these are created, since the overall number of pallets depends on the disposition of items within the layers. This situation also occurs when comparing MEPMH and MGREP, but not when considering the algorithms with mathematical models EPFULL and GREPFULL. For what concerns 3D pallet filling and 2D fill factor, the best performance is, in general, obtained by model-based algorithms, but with the disadvantage of a larger computational time. The choice of the algorithm obviously depends on how much time we can dedicate to the whole operation. MEPMH represents a good compromise between quality of the solutions and computing times but, of course, if minutes rather than seconds are allowed to perform the computations, then EPFULL and GREPFULL appears to be the best choices.

Secondly, we present an analysis of layers type in the solution, summarized in Table 3. This table presents, for each algorithm, the average number of single-item, single-family and residual layers.

Considering layer type, we observe a tendency to create more homogeneous layers when using the GRASP metaheuristic, by increasing the number of single-item or single-family layers, and reducing the number of residual ones. About the quality metric that analyzes the worst case result, the reported value is significantly better when using GRASP-based algorithms in general. Also note that in the algorithms where layer creation is based on the solution of mathematical models, the number of single-family layers increases with respect

Table 3 Computational results per layer type

Algorithm	Single-item layers	Single-family layers	Residual layers
EPMH	23.08	15.08	3.71
GREP	23.79	15.00	3.24
MEPMH	23.08	15.08	3.71
MGREP	24.59	14.89	3.24
EPFULL	16.96	20.88	3.29
GREPFULL	16.59	21.09	3.09

Table 4 Computational results for GRASP-based algorithms

Algorithm	Objective function			Iterations	Local searches
	Min	Max	Avg		
GREP	0.5936	0.5952	0.5946	7888.28	1096.75
MGREP	0.5794	0.6045	0.5962	5573.50	806.56
GREPFULL	0.6031	0.6125	0.6087	8.62	1.82

to the number of single-item layers, in view of the higher ability of the mathematical model solvers in further adding items of the same family once all items of a given type have been inserted in the layer.

For a deeper analysis of the GRASP-based algorithms, we present Table 4. In this table, for each algorithm, we report, respectively, the minimum, maximum and average values of the fitness function (45), adopted to guide the search, the average number of iterations, and the average number of local searches performed.

For what concerns the fitness function values, GREP presents the most stable results, whereas MGREP presents a large variation among average and extreme values. The best results are obtained by GREPFULL, which also delivers the lowest number of pallets. This confirms that the fitness function provides a good representation of the search space.

For what concerns the update of U (line 18 of Algorithm 1), it is important to highlight the influence of the strategy to re-adapt this value. As the U value is related to the local search process, it is interesting to adjust it according to the quality and time metrics to find a new solution. For example, it is not interesting to spend more time on local searches than on global ones, since it would be more challenging to improve a solution after a local minimum was found. In particular, the rate of change of the U value is quite important: large and frequent variations of U tend to favor a larger number of local searches, while small and not frequent variations tend to reduce the number of local searches. Thus, we try to balance this trade-off using a small change (2%, i.e., $\beta = 0.98$) but fast ($max_k = 5$) decrease in U, which contributes to both exploitation and exploration of the search space. Considering the obtained results, we have a large reduction of the number of local searches in GREPFULL with respect to the other ones, but this is obviously due to the reduced number of iterations when using that algorithm.

Considering the previous results, we may highlight the most important parts of each algorithm. Algorithm EPMH obtains the lowest computing time, but it does not always find a solution of good quality. Hence, this strategy is suitable to be applied within metaheuristic or matheuristic approaches. For what concerns algorithm MEPMH, it provides a better compromise between computational effort and quality, since it brings an improvement in the solution quality without excessively increasing the execution time. When taking metaheuristic into account, we can notice a considerable increment in the execution time. However, in general, this extra time is used to perform a better exploration of the search space, which is obtained by the randomized greedy construction and local search improvement phases of Algorithm 1. For example, algorithm GREP brings that benefit, improving the solution quality while maintaining a good compromise between time and quality. However, for algorithm MGREP that does not happen, since the average quality of the solution is similar to algorithm MEPMH, besides increasing the execution time. Algorithms EPFULL and GREPFULL make full use of the mathematical models introduced in Sects. 4.1–4.2. Through a good mix of quick greedy construction of layers and pallets, better exploration of the solution space by means of randomization and local search, and search for improved solution by means of mathematical models invoked for short time limits, these approaches are able to return better quality solutions. They require computing times larger than the other approaches but still compatible with the needs of a company if the pallet creation is performed offline.

7 Conclusions

In this paper, we studied a pallet building problem with item rotations and practical constraints involving visibility and contiguity. We proposed heuristic algorithms and a mathematical formulation for the addressed problem, filling a gap in the existing literature. In general, our complete algorithm is based on a two-step heuristic approach, first creating layers and then pallets. We tailored each proposed strategy to GRASP metaheuristic with reactive method. Regarding to heuristic algorithms, these are an Extreme Points heuristic for creating 2D layers and a greedy heuristic for creating 3D pallets. Regarding to mathematical models, we embedded them in the previous heuristics, creating in this way variants to the two-step heuristic both to compact items into layers and to stack layers into pallets. Extensive computational experiments on real-world instances proved the effectiveness of the proposed algorithms.

Regarding the GRASP-based algorithms, it turns out that the upper bound updating strategy is essential to control the exploration and exploitation of the search space, and, consequently, to improve the solution quality. Besides, the robustness of the reactive method allows a more flexible search on the solution space when using a set of fitness evaluation functions in the Extreme Points heuristic to find a new place to pack items. The embedded mathematical models have shown a capacity of improvement of the quality of the solutions found, without increasing too much the computing time. In spite of the complexity of the real-world instances and the short execution times, interesting final results are reported, thus showing that the combination heuristics/mathematical models/metaheuristic is an efficient one for this type of problem.

As future work, we intend to study extensions of the GRASP metaheuristic, possibly considering the recently developed fixed set search metaheuristic (Jovanovic et al. (2019), Jovanovic and Voß (2019, 2020)), besides considering the most recent improvement strategies from the literature (Resende and Ribeiro (2019)), as for example the use of a learning mechanism. We are also interested in extending the concept of family to address characteristics beyond the geometric ones, and to study more complex problems involving vehicle routing aspects, and thus the need of sequencing items with layers and pallet so as to consider their unloading at the customers' sites. Finally, as an alternative to the approaches proposed in this paper, and, thus, a possible further topic for future research, we mention heuristics based on building blocks (see, e.g., Liu et al. (2011), Ren et al. (2011), Wang et al. (2008), Zhang et al. (2012)). Each block is made up by items of the same type, so that a block automatically satisfies contiguity between these items. However, basic building block heuristics should be adapted to meet also the visibility constraint.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10479-021-04349-w.

Funding Open access funding provided by Università degli Studi di Parma within the CRUI-CARE Agreement.

References

Alonso, M., Alvarez-Valdes, R., Iori, M., & Parreño, F. (2019). Mathematical models for multi container loading problems with practical constraints. *Computers & Industrial Engineering, 127,* 722–733.

Alonso, M., Alvarez-Valdes, R., Iori, M., Parreño, F., & Tamarit, J. (2017). Mathematical models for multi container loading problems. *OMEGA, 66,* 106–117.

Alonso, M., Alvarez-Valdes, R., & Parreño, F. (2020). A GRASP algorithm for multi container loading problems with practical constraints. 4OR-Q. *J Oper Res, 18,* 49–72.

Alonso, M., Alvarez-Valdes, R., Parreño, F., & Tamarit, J. (2016). Algorithms for pallet building and truck loading in an interdepot transportation problem. *Mathematical Problems in Engineering, 2016,* 1–11.

Alvarez-Valdes, R., Parreño, F., & Tamarit, J. (2008). Reactive GRASP for the strip-packing problem. *Computers & Operations Research, 35,* 1065–1083.

Bischoff, E., & Ratcliff, M. (1995). Issues in the development of approaches to container loading. *Omega, 23,* 377–390.

Bortfeldt, A., & Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research, 131,* 143–161.

Bortfeldt, A., & Wäscher, G. (2013). Constraints in container loading - a state-of-the-art review. *European Journal of Operational Research, 229,* 1–20.

Burke, E., Kendall, G., & Whitwell, G. (2004). A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research, 52,* 655–671.

Chazelle, B. (1983). The bottomn-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers C-32,* 697–707.

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

Crainic, T., Perboli, G., & Tadei, R. (2012). Recent advances in multi-dimensional packing problems. *In: New Technologies, chap., 5,* IntechOpen.

Côté, J. F., Dell'Amico, M., & Iori, M. (2014). Combinatorial benders' cuts for the strip packing problem. *Operations Research,62*, 643–661.

Côté, J. F., Haouari, M., & Iori, M. (2021). Combinatorial benders decomposition for the two-dimensional bin packing problem. *INFORMS J Comput* 1–16.

de Queiroz, T., & Miyazawa, F. (2013). Two-dimensional strip packing problem with load balancing, load bearing and multi-drop constraints. *International Journal of Production Economics, 145,* 511–530.

de Queiroz, T., & Miyazawa, F. (2014). Order and static stability into the strip packing problem. *Annals of Operations Research, 223,* 137–154.

Delorme, M., Iori, M., & Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research, 255,* 1–20.

Delorme, M., Iori, M., & Martello, S. (2017). Logic based benders decomposition for orthogonal stock cutting problems. *Computers & Operations Research, 78,* 290–298.

Egeblad, J., Garavelli, C., Lisi, S., & Pisinger, D. (2010). Heuristics for container loading of furniture. *European Journal of Operational Research, 200,* 881–892.

Feo, T., & Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters, 8,* 67–71.

Feo, T., & Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6,* 109–133.

Gilmore, P., & Gomory, R. (1965). Multistage cutting stock problems of two or more dimensions. *Operations Research, 13,* 94–120.

Gottschalk, S.: Separating axis theorem. (1996). Tech. rep., Technical Report TR96-024, Department of Computer Science. *UNC Chapel Hill.*

Haessler, R., & Talbot, F. (1990). Load planning for shipments of low density products. *European Journal of Operational Research, 44,* 289–299.

Imahori, S., & Yagiura, M. (2010). The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio. *Computers & Operations Research, 37,* 325–333.

Iori, M., de Lima, V., Martello, S., Miyazawa, F., Monaci, M.: Exact solution techniques for two-dimensional cutting and packing. European Journal of Operational Research **289**(2), 399–415 (2021).

Iori, M., Locatelli, M., Moreira, M., Silveira, T.: Reactive GRASP-based algorithm for pallet building problem with visibility and contiguity constraints. In: Lalla-Ruiz E., Mes M., Voß S. (eds) Computational Logistics. ICCL 2020. Lecture Notes in Computer Science, vol. 12433. Springer, Cham (2020a).

Iori, M., Locatelli, M., Moreira, M., Silveira, T.: Solution of a practical pallet building problem with visibility and contiguity constraints. In: International Conference on Enterprise Information Systems, vol. 1, pp. 327–338. SciTePress (2020b).

Iori, M., Locatelli, M., Moreira, M., & Silveira, T. (2021). A mixed approach for pallet building problem with practical constraints. In J. Filipe, M. Śmiałek, A. Brodsky, & S. Hammoudi (Eds.), *Enterprise Information Systems* (Vol. 417, pp. 122–139). Cham: Springer.

Iori, M., & Martello, S. (2010). Routing problems with loading constraints. *TOP, 18,* 4–27.

Jovanovic, R., Tuba, M., & Voß, S. (2019). *Fixed set search applied to the traveling salesman problem* (pp. 63–77). International Workshop on Hybrid Metaheuristics.

Jovanovic, R., Voß, S.: Fixed set search applied to the minimum weighted vertex cover problem. In: Analysis of Experimental Algorithms, *SEA 2019*, vol. 11544, pp. 490–504. Springer (2019)

Jovanovic, R., & Voß, S. (2020). *The fixed set search applied to the power dominating set problem. Expert Systems* (p. (p. e12559).)

Józefowska, J., Pawlak, G., Pesch, E., Morze, M., & Kowalski, D. (2018). Fast truck-packing of 3D boxes. *Engineering Management in Production and Services, 10,* 29–40.

Kurpel, D., Scarpin, C., Pécora Junior, J., Schenekemberg, C., & Coelho, L. (2020). The exact solutions of several types of container loading problems. *European Journal of Operational Research, 284,* 87–107.

Leung, S., Zhang, D., & Sim, K. (2011). A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *European Journal of Operational Research, 215,* 57–69.

Liu, J., Yue, Y., Dong, Z., Maple, C., & Keech, M. (2011). A novel hybrid tabu search approach to container loading. *Computers and Operations Research, 38,* 797–807.

Lodi, A., Martello, S., Monaci, M., & Vigo, D. (2014). *Two-Dimensional Bin Packing Problems* (pp. 107–129). John Wiley & Sons Ltd.

Parreño, F., Alvarez-Valdes, R., Oliveira, J., & Tamarit, J. (2010). A hybrid grasp/vnd algorithm for two- and three-dimensional bin packing. *Annals of Operations Research, 179,* 203–220.

Prais, M., & Ribeiro, C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing, 12*(3), 164–176.

Ranck Júnior, R., Yanasse, H., Morabito, R., & Junqueira, L. (2019). A hybrid approach for a multi-compartment container loading problem. *Expert Systems with Applications, 137,* 471–492.

Ren, J., Tian, Y., & Sawaragi, T. (2011). A tree search method for the container loading problem with shipment priority. *European Journal of Operational Research, 214,* 526–535.

Resende, M., & Ribeiro, C. (2019). *Greedy Randomized Adaptive Search Procedures: Advances and Extensions* (pp. 169–220). Cham: Springer International Publishing.

Scheithauer, G. (2018). *Introduction to cutting and packing optimization*. Newyork: Springer.

Schmid, V., Doerner, K., & Laporte, G. (2013). Rich routing problems arising in supply chain management. *European Journal of Operational Research, 224,* 435–448.

Silva, E., Oliveira, J., & Wäscher, G. (2016). The pallet loading problem: A review of solution methods and computational experiments. *International Transactions in Operational Research, 23,* 147–172.

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

Tsai, D. (1987). *Modeling and analysis of three-dimensional robotic palletizing systems for mixed carton sizes*. Iowa State University (**Ph.D. thesis**)

Vidal, T., Crainic, T., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research, 231,* 1–21.

Wang, Z., Li, K., & Levy, J. (2008). A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach. *European Journal of Operational Research, 191,* 86–99.

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

Zhang, D., Peng, Y., & Leung, S. (2012). A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers and Operations Research, 39,* 2267–2276.