Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

An approach for the pallet-building problem and subsequent loading in a heterogeneous fleet of vehicles with practical constraints

Daniel Cuellar-Usaquen[a], Guillermo A. Camacho-Muñoz[b], Camilo Quiroga-Gomez[a] and David Álvarez-Martínez[a*]

[a]Department of Industrial Engineering/University of Los Andes, Carrera 1E # 19A-40, Bogota, Colombia
[b]Department of Electrical and Electronical Engineering/ University of Valle, Calle 13 # 100-00, Cali, Colombia

C H R O N I C L E

Article history:
Received June 15 2020
Received in Revised Format November 23 2020
Accepted January 22 2021
Available online January, 22 2021

Keywords:
Pallet Packing
Container Loading Problem
GRASP

A B S T R A C T

This article presents a metaheuristic algorithm to solve the pallet-building problem and the loading of these in trucks. This approach is used to solve a real application of a Colombian logistics company. Several practical requirements of goods loading and unloading operations were modeled, such as the boxes' orientation, weight support limits associated with boxes, pallets and vehicles, and static stability constraints. The optimization algorithm consists of a two-phase approach, the first is responsible for the construction of pallets, and the second considers the optimal location of the pallets into the selected vehicles. Both phases present a search strategy type of GRASP. The proposed methodology was validated through the comparison of the performance of the solutions obtained for deliveries of the logistics company with the solutions obtained using a highly accepted commercial packing tool that uses two different algorithms. The proposed methodology was compared in similar conditions with the previous works that considered the same constraints of the entire problem or at least one of the phases separately. We used the sets of instances published in the literature for each of the previous works. The results allow concluding that the proposed algorithm has a better performance than the most known commercial tool for real cases. The proposed algorithm managed to match most of the test instances and outperformed some previous works that only involve decisions of one of the two problems. As future work, it is proposed to adapt this work to the legal restrictions of the European community.

1. Introduction

Moving products maximizing the utilization space at the lowest possible cost to the environment, and the company is one of the objectives of any carrier company. The complexity of this objective has increased with the preference of the volumetric weight pricing technique for commercial freight transport Viklund (2010). To reach this objective, logistics companies divide into two stages the packing operation. The first operation stage corresponds to the packing of a large assortment of three-dimensional goods (different sizes and weights) into pallets of the same size; this problem was named first by Ballew (2000) as the Three-Dimensional Distributor's Pallet Packing Problem (3D-DPPP). Formally in the improved typology of the cutting and packing problems proposed by Wäscher et al. (2007), this type of problem is known as the Three-Dimensional Single Stock Size Cutting Stock Problem (3D-SSSCSP). The second stage consists of the loading pallets into the available fleet of vehicles, and this problem belongs to the type of problem known as the Three-Dimensional Multiple Bin Size Bin Packing Problem (3D-MBSBPP). This article tackles the problem of packing operations inspired by the instance of a large logistics company located at Bogota-Colombia. On this company, goods already packed in boxes must be placed´ into identical pallets limited to a certain height that enabled the operators to accomplish the loading and unloading operations with the help of a

* Corresponding author
E-mail: d.alvarezm@uniandes.edu.co (D. Álvarez-Martínez)

2021 Growing Science Ltd.
doi: 10.5267/j.ijiec.2021.1.003

forklift. Depending on the number and the sizes of the pallets (heights) reached on the palletizing stage; it must be assigned a suitable set of vehicles of the available fleet to carry out the corresponding pallet-loading task. Pallet stacking can be performed if the vehicle capacity and cargo stability (static) constraints are met. Even though the size of the fleet servicing owned by this company is sufficient to perform its distribution tasks without failing to meet demands nor the need for outsourced vehicles, the packing operations are done following a resource optimization scheme. That is, they are not considered costs associated with the vehicles, and only the selection of the set of vehicles with the least capacity to carry the boxes demanded is required. That means finding the set of packing patterns (boxes into pallets then pallets into vehicles) that will minimize total unused space while respecting the practical constraints of the system. The constraints associated with the loading and unloading operations that the company includes: guarantee the allowed vertical orientations of the boxes, the weight support capacity for box and pallet stacking, the load-bearing limits of the vehicles, and static stability constraints.
The palletizing and subsequent vehicle loading has been studied for just a few previous works, despite its broad spectrum of applicability. This article addresses the problem with a metaheuristic approach due to three features: the intrinsic complexity of the integrated problem, the difficulty of including a large number of practical constraints, and the requirement to provide adequate solutions in real-time for more than 2000 boxes that must be attended daily. Besides that, the presented work pretends to contribute to the resolution of every single stage of the problem, specifically: Palletizing and Truck Loading. A benchmarking is performed against the best previous works to validate the proposed methodology and illustrating the advantages of this study.

The article is ordered as follows. The problem description is presented along with a formal definition of the constraints contemplated in this work in Section 2. In Section 3, a summary of existing related work is performed. The details of the optimization algorithms proposed for both stages of the problem are shown in Section 4; Section 5 presents the characteristics of the benchmark proposed to validate the algorithms. Section 6 presents the computational results along with their corresponding analysis. Lastly, the document finishes with the reached conclusions and the recommended future works.

2. Problem description

The problem lies in the delivery of an order consisting of a set of boxes. The boxes of this order must be packed on pallets, i.e., no loose boxes are allowed; because the manipulation operations are performed with a forklift. The company has enough pallets to dispatch the order. The use of pallets allows better maneuverability and stackability on (un)loading operations but requires considered space that becomes on unused space. To support daily deliveries, the company has a heterogeneous fleet of vehicles. The main purpose of the operation is to find the packing pattern of (1) boxes into pallets and (2) pallets into vehicles, in a way that unused space within the vehicles is minimized and the constraints of cargo are satisfied. The considered constraints are described on next. Let us consider a set of boxes to be packed as a list b_i of types of boxes (i = 1,2,...,N), for which certain characteristics such as length (l_i), width (w_i), height (h_i), weight (wg_i), number of items (q_i), allowed vertical orientations (vo_i^l, vo^w_i, vo^h_i) and stacking support limits for each allowed orientation (lbs^l_i, lbs^w_i and lbs^h_i) are known. Boxes are not grouped and the position within pallets is free and no box type constraints need to be satisfied. There is only one pallet type p where boxes can be positioned, meaning that pallet length (l_p), width (w_p), height (h_p), weight (wg_p) and load bearing limits (lbs_p) are known. The logistics company also establishes a maximum height limit for pallet building. For Palletizing, the following constraints must be considered:

(R1) Cargo height must not exceed the maximum established height. Cargo height is given by the distance between the upwards face of the highest box in the structure and the downwards face of the pallet (the height of an empty pallet is considered as part of the height of the cargo).

(R2) Each box must have full support on its base, and the virtual lateral wall of the pallet will not be exceeded. When placing an item, the face on its lower side must be fully supported (the whole area) by a support area (*SupArea*) that is composed of the upper faces of the boxes already located on the pallet, or top face of the pallet. That will satisfy the static stability of the boxes during palletizing, reducing risks for the operator and the cargo.

(R3) The sum of the weights of each box packed into an instance k of pallet p must not exceed the specified weight limit (i.e., $lbs_p \geq \sum_{i \in B_k} wg_i$; where B_k is the B set of boxes packed into the pallet k). The weight of palletized cargo A (pwg_A) is calculated as the sum of the weights of packed boxes and the weight of the pallet ($pwg_A = wg_p + \sum_{i \in B_A} wg_i$).

(R4) A box i can be layed down if there is a support area with a greater weight limit than the weight of the box ($\exists$ *SupArea* | $lbs_{SupArea} \geq wg_i$). A *SupArea* is the area conformed by the set $B_{TopArea}$ of adjacent boxes with tops at equal height in the pallet. The maximum weight limit that a support area can bear ($lbs_{SupArea}$) is given by the box in $B_{TopArea}$ with the lowest weight support limit ($lbsSupArea = min\ \{lbs_b^* | b \in BTopArea\ \}$).

(R5) Stacked boxes must comply, for each box i, with the constraints (R2) and (R4).

(R6) A 90° rotation is allowed for every proper vertical orientation of the boxes, preserving fragile constraints and avoiding damage to the packaged goods.

We must consider that the vehicle fleet V consists of a list v_j of available vehicles ($j = 1,2,...,M$) with known length (L_j), width (W_j), height (H_j), number of vehicles (Q_j), and maximum cargo weight (LBS_j). For the packing of palletized cargo into vehicles (Truck Loading), the following constraints are considered:

(R7) Any stack of pallets must not exceed the height of the vehicle. The stack of pallets' height is equal to the sum of the heights of the palletized cargo that conforms to the stack.

(R8) Other cargo can be placed on top of a palletized cargo A, as long as A has a total-upper-area TTA_A satisfying $TTA_A \geq l_p \cdot w_p \cdot 0.75$. Where 0.75 represents the minimum ratio required (75%) to maintain static stability on stacked palletized cargo. The total upper area of a palletized cargo A (TTA_A) is defined as the sum of the areas of the upper faces of the boxes found on the uppermost height level ($TTA_A = \sum_{i \in B_{TopArea}} l_i \cdot w_i$ | where l_i and w_i are the box width and length i for the given vertical orientation and $B_{TopArea}$ is the set of boxes located on the uppermost height level).

(R9) The total loaded weight (TWG) in v must not exceed the weight limit of the vehicle. Total loaded weight (TWG) on a vehicle v consists of the sum of the individual weights of palletized cargo assigned to the vehicle ($TWG_v = \sum_{I \in LP_v} pwg_I$ | where LP_v is the set LP of palletized cargo introduced into the vehicle v).

(R10) The limit of the maximum weight in a load in the superior area of a palletized cargo A (lbs_A) is defined as the minimum weight limit supported of the set of upper faces which make up the top area of the palletized load ($lbs_A = min\{lbs^*_{BSupTopArea}$ | $BSupTopArea$ is the set of top faces of the palletized cargo $A\}$). A palletized load B can be stacked on a palletized load A, if the maximum load weight limit of the upper area of the palletized cargo A (lbs_A) is greater or equal than the weight of the palletized load B ($lbs_A \geq pwg_B$).

(R11) Palletized cargo can be stacked together. Generally, B can be stacked on top of A if (R8) is valid for A and (R10) is valid for both A and B.

(R12) Palletized cargo can be rotated 90° around its vertical axis; no other rotations are allowed to prevent damage to merchandise.

In terms of the constraints classifications proposed by Bortfeldt and Wäscher (2013), the reference company seeks to guarantee: (R1 and R7) height limits for pallets and vehicles, (R2 and R8) static stability of the boxes inside the vehicles, (R3 and R9) maximum weight supported by pallets and vehicles, (R4 and R10) load-bearing strength for each item and palletized cargo, (R5 and R11) rules for stacking boxes and pallets, and (R6 and R12) the permissible orientations for items and pallets. The Palletizing purpose is to find a solution that minimizes the number of pallets for a specified maximum height (which is the same as maximizing volume utilization), satisfying the constraints (R1 - R6). The Truck Loading seeks for a solution maximizing vehicle volume utilization, satisfying the constraints (R7 - R12). In this way, on the solution of the logistics company problem, the set of constraints (R1 - R12) must be satisfied while pursuing the objective of maximizing vehicle volume utilization.

3. Literature Review

In this section is exposed the works related to the packing problem in two stages. Besides of that is reviewed the contributions made over each problem studied individually.

3.1. Packing Problem in two stages

Just a few previous works have tackled the two-staged packing problem (Palletizing and Truck Loading). Morabito et al. (2000) presents a 2D variation of the problem, which requires packing uniform boxes into similar pallets and loading them afterward into a homogeneous fleet of vehicles. The authors use the heuristic algorithm of five blocks presented in Morabito and Morales (1998) to reduce the number of vehicles required.
The problem of multiple containers and pallets with different sizes is tackled by Takahara (2005), using heuristic procedures to generate the original sequence of items for packing. In this, permutations are made between the set of containers and available pallets, thus obtaining the final packing pattern — the objective is to reduce the number of bins and pallets needed.
The authors in Zachariadis et al. (2012) consider the Pallet Packing Vehicle Routing Problem with Time Windows (PPVRPTW). In the first stage, the goods of the customers are loaded into identical pallets, adding the constraint (R13), which alludes to the full delivery of the load, this restriction requires all the boxes that a client demands stacked on the same pallet. In the second stage, the set of pallets is loaded into identical vehicles with defined capacity. Having said this, it is not necessary

to generate packing patterns; the loading process is created as the routes are built to visit each customer. The objective is to make lower-cost routes for a set of customers, who demand a collection of rectangular objects. The authors solve the problem of employing a Local Search and Tabu Search algorithm. Besides, benchmarking is presented with the ideal solution for the packing phase heeding the load's geometric constraints. Thus, in Alonso et al. (2017), a two-staged packing problem is solved using an integer linear programming, aiming to reduce the number of containers. The proposed model started from a basic formulation, and the restrictions are summed gradually. Stage 1 is reduced to a 1D variation of the problem, according to the building layers rules imposed by the company. Additionally, they add the restriction (R14) according to the maximum weight limit that each axle of the truck can support, based on the calculation of the center of gravity of the palletized load. Furthermore, Alonso et al. (2016) implements a GRASP algorithm to solve the problem introduced in Alonso et al. (2017). They used a reactive and randomized constructive phase to define the load pattern within the pallet and the location of the pallet within the truck. Moreover, the improvement phase uses three different movements: removing and refilling boxes from a container, swapping boxes between containers, and insertion of boxes, trying to maximize the volume occupied inside the trucks. In contrast to the present work, Moura and Bortfeldt (2017) added the constraints (R15) and (R16); related with the packing order for palletizing and truck loading, guaranteeing the sequence of delivery to the customers, these constraints are also known as multi-drop constraint Martínez et al. (2015). The authors solve the first stage of the problem using the constructive algorithm presented in Moura and Oliveira (2005). For the second stage, the problem is reduced to a 1D variation, and a tree-search algorithm is applied. The aim is to minimize the number of trucks required.

Alonso et al. (2019) attempted the problem and restrictions mentioned in Alonso et al. (2016) and additionally add constraints (R17) and (R18) related to the dynamic stability for palletizing and truck loading. Bortfeldt and Wäscher (2013) and Martínez et al. (2018a) present a detailed description of (R17) and (R18). The authors solve the problem using an integer linear programming, so the number of containers is reduced. The proposed model starts with a basic formulation, and the constraints are summed gradually. The authors focus on the computational flexibility and complexity of the raised model.
In Alonso et al. (2020), a GRASP algorithm is implemented to solve the problem introduced in Alonso et al. (2019). Once again, the constructive phase is reactive and randomized; it defines the load pattern within the pallet, considering three different types of pallets and locates the pallets within the truck. The improvement stage seeks to reduce the height and the total weight of the generated layers, attempting to minimize the ones that were not loaded into the pallet.

Table 1
Summary of constraints addressed on the Two-Staged Packing Problems

Author	First Stage									Second Stage								
	R1	R2	R3	R4	R5	R6	R13	R15	R17	R7	R8	R9	R10	R11	R12	R14	R16	R18
Morabito et al. (2000)		√				√									√			
Takahara (2005)	√	√				√				√	√				√			
Zachariadis et al. (2012)	√	√				√	√					√						
Alonso et al. (2016)	√	√	√	√	√	√				√	√	√	√	√	√	√		
Alonso et al. (2017)	√	√	√	√	√	√				√	√	√	√	√	√	√		
Moura and Bortfeldt (2017)	√	√	√	√	√	√	√	√		√	√	√	√	√	√		√	
Alonso et al. (2019)	√	√	√	√	√	√			√	√	√	√	√	√	√	√		√
Alonso et al. (2020)	√	√	√	√	√	√			√	√	√	√	√	√	√	√		√
Olsson et al. (2020)	√	√	√	√	√	√				√	√	√	√	√	√	√		
Present work	√	√	√	√	√	√				√	√	√	√	√	√			

Finally, Olsson et al. (2020) approach a two-staged problem: place smaller packages into consolidation pallets, then, place larger boxes and consolidation pallets in a container. They report the use of seven of the ten constraints published by Bortfeldt and Wäscher (2013): weight limits, weight distribution, orientation, complexity, positioning, stability, and stacking with a particular case where the items present a higher pressure on the edges than in the center of its top surface. Their solution is based on a two-level metaheuristic approach: (1) a constructive greedy-type placement heuristic, (2) a genetic algorithm that adjusts a weighted scored function to assess the output of the constructive heuristic. Table 1 presents a summary of the constraints contemplated by each author compared with the present work. From this Table, it is identified that the majority of works differ in the type of constraints considered, reducing the importance of contemplating a higher number of constraints and focusing on modeling the functional conditions of the real application in particular. The resolution of different variants of the two-staged packing problem contributes directly to logistics companies and the general public since it allows them to identify the proposed solution methodologies that best suit their packing process. However, from this summary, it is essential to highlight that most works consider a sequential solution approach except for the work proposed in Olsson et al. (2020), this due to the design complexity involved.

3.2. Palletizing

The task of loading a heterogeneous set of boxes into the pallet is found in the literature by the names like Distributor's Pallet Packing Problem (DPPP) and Single Stock Size Cutting Stock Problem (SSSCSP). The SSSCSP requires packing a set of rectangular and weakly heterogeneous items into identical pallets while maximizing pallet volume utilization while minimizing the number of pallets necessary. An updated description of the problem can be found in Silva et al. (2016).

The SSSCSP is NP-hard in the strict sense; most solution methods in recent years are heuristic and metaheuristic in nature. Following a timeline, we first found an exact algorithm to pack boxes into a single container presented by Martello et al. (2000); they proposed a Branch & Bound algorithm that combines different approximation algorithms. It was able to solve test instances with almost 100 boxes. Later, a greedy search heuristic is presented in De Castro Silva et al. (2003), where the objective is to find the packing pattern that minimizes the number of used containers, guaranteeing the static stability constraint. Then, Lim and Zhang (2005) contemplated single and multiple-sized container loading problems, proposing the use of dynamic prioritization to manage non-fitting box types, through an adaptive greedy algorithm. In this one, the kind of box with the highest priority will be loaded into the lowest surfaces of the container or previous containers. In Crainic et al. (2008) was presented the concept of extreme point and a new extreme rule based on points to load items into a container, new extreme point rule is used to obtain new constructive heuristics. An approximation algorithm is presented in Miyazawaa and Wakabayashi (2009), with asymptotic performance limits of 2.64. They allowed 90 degrees rotations over the items. In Amossen and Pisinger (2010) is presented the problem of determining if a set of multidimensional rectangular boxes can be orthogonality loaded into a rectangular container, all while considering guillotine cut constraints. Epstein and Levy (2010) is regarded as the dynamic packing of squares and rectangles into unit squares and the dynamic packing of cubes and boxes into unit cubes, presenting a dynamic version of the Next Fit Decreasing Height algorithm for the out-of-line problem. Burke et al. (2012) introduced a genetic programming system to automatically generate a quality heuristic focusing on the knapsack problem and load packing problems in one, two, or three dimensions. In Hifi et al. (2014) is proposed a solution by using a simple strategy based on an ILP heuristic, without any improvement based on metaheuristics. Lastly, Zudio et al. (2018) presented a variable neighborhood descent (VND) algorithm adapted with a biased random-key genetic algorithm (BRKGA). The authors seek to balance the population according to the average quality of the random keys and the solutions with ordered sub-sequences.

3.3. Truck Loading

Therefore, after loading the boxes onto pallets, the pallets must be loaded into a fleet of vehicles with heterogeneous dimensions; another given name for this type of problem is the multiple container loading problem. The MBSBPP requires the packing of a heterogeneous set of rectangular pieces into a strongly heterogeneous set of containers, maximizing container utilization volume, or minimizing container utilization cost. This problem is frequently addressed as the Bin Packing Problem (BPP), but the dimensions of the containers are different, Delorme et al. (2016) presents a detailed description of this problem. The MBSBPP is also considered NP-hard strictly, for this reason, most solution methods published are heuristic and metaheuristic algorithms. The authors in Jin et al. (2003) proposed a compound algorithm, trying to solve the 3D-BPP with several practical constraints, designed for a general case with mixed containers and evaluating they are lower bound over the set of instances. A tree-search algorithm to solve large-sized instances of the 3D-BPP is presented in Brunetta and Grégoire (2005) to maximize the average volume utilization of the containers. They implicitly explore the solution space, combining classical heuristic for the Pallet Loading and the Container Loading Problem to solve each sub-problem created. In De Almeida and Figueiredo (2010) is presented a Container Loading Problem motivated by a real-world application. The authors perform an alternative; non-linear formulation for the 3D-BPP version, as well as new heuristic algorithms (CPBOX and BOXCP) designed to approximate an optimal solution involving heterogeneous boxes and containers. A variant of the routing-packing problem is considered in Ceschia and Schaerf (2013). A real-world industrial application inspires the problem. The goal of the work is to pack a set of boxes in different containers and, at the same time, build the routes to visit the customers. It includes several characteristics of great importance for many practical situations. They proposed a solution technique based on local search metaheuristics. In Alvarez-Valdés et al. (2013) is proposed a GRASP algorithm to solve 2D and 3D variants of the MBSBPP. The algorithm is composed of a constructive procedure, a post-processing stage, and some improvement moves. The best solutions are combined using a path-relinking procedure with three developed versions: static, dynamic, and evolving. The purpose is minimizing of cost of containers required. Finally, in Paquay et al. (2018) is solved the MBSBPP for a real case of an air cargo company, using a tailored two-phase constructive heuristic. The first phase provides a loading pattern for a given set of boxes and ULD type, and in the second phase, it is an improvement phase with different movements. The objective is to minimize the total used cost. After reviewing the literature, it is essential to highlight that this work presents several methodological and technical contributions. We extended a heuristic to calculate a lower bound that takes into account weight and volume limits. We adapted two algorithms that consider the remaining load as a criterion of the decision in the packing process. We proposed a new coding scheme where several maximal spaces are opened simultaneously. Besides, we solved a new variant of the problem of two-staged packing proving useful for logistics companies whose packing process is similar.

4. Proposed Algorithms

The methodology approach is presented in two stages: (1) Palletizing - packing the boxes into the smallest set of pallets, (2) Truck Loading - loading the pallets into a heterogeneous fleet of vehicles, seeking to maximize volume utilization. The first stage uses a GRASP algorithm based on the encoding of maximal spaces. Its constructive phase and local search procedures share the same structure, differing only in the degree of randomness of their functions. It is important to note that this algorithm differs from previous works on three key aspects. The first occurs by initializing the constructive phase with *np* maximal initial empty spaces instead of a single free space (being *np* the minimum number of pallets required to accommodate the cargo). The second is the dynamic control of the size of the Restricted Candidate List based on the autotuning of the alpha parameter using a training phase. The last one is the random selection of the criterion for choosing the arrangement of items to be placed on a maximal space. GRASP algorithm of stage one is adapted to solve the problem of Truck Loading; for this, a new criterion was included that allows to select the order of the vehicles to be loaded.

Algorithm 1 Lower Bound Heuristic

```
Input: List B: list of boxes; v: pallet volume; w: pallet's maximum weight limit
Output: lb: number of pallets
1: $lb \leftarrow 1$
2: $v^{res} \leftarrow v$
3: $w^{res} \leftarrow w$
4: for $i \leftarrow 1$ To $|B|$ do
5:     if $B_i.volume \leq v^{res}$ then
6:         if $B_i.weight \leq w^{res}$ then
7:             $v^{res} \leftarrow v^{res} - B_i.volume$
8:             $w^{res} \leftarrow w^{res} - B_i.weight$
9:         else
10:            $lb \leftarrow lb + 1$
11:            $w^{aux} \leftarrow B_i.weight - w^{res}$
12:            $w^{res} \leftarrow w - w^{aux}$
13:            $v^{res} \leftarrow v$
14:        end if
15:    else
16:        if $B_i.weight \leq w^{res}$ then
17:            $w^{res} \leftarrow w$
18:        else
19:            $w^{aux} \leftarrow B_i.weight - w^{res}$
20:            $w^{res} \leftarrow w - w^{aux}$
21:        end if
22:        $lb \leftarrow lb + 1$
23:        $v^{aux} \leftarrow B_i.volume - v^{res}$
24:        $v^{res} \leftarrow v - v^{aux}$
25:    end if
26: end for
27: return $lb$
28: end
```

4.1. Stage 1 - Palletizing

To solve this stage, we must compute the number of maximum empty spaces *np*. Lower-Bound heuristic proposed by Clautiaux et al. (2014) was adapted to include the available-space in the pallet to load the remaining items while considering their volumes and weights.

This adaptation is presented on Algorithm 1. The lower bound heuristic consists of using the volume and weight limits available for the pallets while the boxes are introduced unidimensionally. When the boxes exceed some of the available capacities (volume or weight), they are fractioned, allocating as much to the current pallet and conserving the remaining for a future pallet. In this, let *B* be a list of boxes with their volume and weight defined. Let us consider a pallet *p* described by its available-volume *v* and its maximum weight limit *w*.

The heuristic starts by opening a pallet (*lb*) and initializing the residual volume and weight (v^{res} and w^{res}) with the original pallet volume and maximum weight supported (lines 1-3). Iteratively, the list of boxes is checked to see if the box fits in the volume and remaining weight. Four possible situations should be checked. In the first case, if the box fits by volume and weight (lines 5-6), only the volume and residual weight are updated (lines 7-8). In the second case, if the box fits by volume but not by weight (line 9), a new pallet must be opened, the remaining weight is updated according to the surplus weight of the current box, and the remaining volume goes back to the original pallet volume (lines 10-13). In the third case, if the box does not fit by volume but by weight (line 16), a new pallet must be opened (lines 22), the remaining volume is updated according to the surplus volume of the current box (lines 23-24), and the remaining weight goes back to be the maximum weight supported (line 17). Finally, if the box does not fit because of volume or weight (line 18), a new pallet must be opened (line 22), and the remaining volume and weight updated according to the volume and weight surplus of the current box (lines 19-20 and 23-24). The algorithm returns *lb* the lower bound of the number of pallets needed if only the volume and weight to pack the boxes was considered. It is important to note that lower bound heuristic is not susceptible to the order in which boxes enter this procedure.

Algorithm 2 GRASP algorithm for Palletizing

```
Parameters: totalIterations, K%: percentage of boxes to remove, A: set of alphas
Input: List B: list of boxes; Pallet p;
Output: List Incumbent: list of pallet packing patterns;
1: for $i \leftarrow 1$ to $totalIterations * 2$ do
2:     Bool $flag \leftarrow False$
3:     if $localsearch = False$ then
4:         List $CurrentPalletPatterns \leftarrow \emptyset$
5:         $np \leftarrow LowerBound(B, p.volume, p.weight)$
6:         List $E \leftarrow CreateMaximalSpaces(p, np)$
7:     end if
8:     while ($flag = False$) do
9:         while $B \neq \emptyset$ and $E \neq \emptyset$ do
10:            $e \leftarrow SelectMaximalSpace(E)$
11:            List $Layers \leftarrow GenerateLayersList(e, B)$
12:            if $localsearch = False$ then
13:                List $RCL \leftarrow BuildRCL(Layers, i)$
14:                Layer $L \leftarrow SelectLayerRandomly(RCL)$
15:            else
16:                Layer $L \leftarrow SelectLayerByBestFit(Layers)$
17:            end if
18:            Packing Pattern $P \leftarrow LocateLayerOnSpace(L, e)$
19:            $E \leftarrow UpdateListOfMaximalSpaces(P, e, E)$
20:            $B \leftarrow UpdateListOfRemainingBoxes(L)$
21:            $CurrentPalletPatterns \leftarrow CurrentPalletPatterns \cup P$
22:        end while
23:        if $B \neq \emptyset$ then
24:            $np \leftarrow LowerBound(B, p.volume, p.weight)$
25:            List $E \leftarrow CreateMaximalSpaces(p, np)$
26:        else
27:            $flag \leftarrow True$
28:        end if
29:    end while
30:    Update Incumbent if CurrentPalletPatterns is better
31:    $localsearch \leftarrow Not(localsearch)$
32:    if localsearch then
33:        $E, B, CurrentPalletPatterns \leftarrow RemoveK\%(CurrentPalletPatterns, K\%)$
34:    end if
35: end for
36: return Incumbent
37: end
```

The procedure that solves the Palletizing is presented in the Algorithm 2. Only one algorithm is defined to describe constructive and local search procedures. The algorithm alternates between construction and improvement in each iteration because both use almost the same steps. The flag *localsearch* is used to call distinctive functions of each phase. In the construction iteration, the packing pattern is generated from zero, and the maximal spaces are initialized using the Algorithm 1 (lines 4-6). The construction will only be completed until all boxes are palletized. For this, the algorithm must first select an available space from the list *E*, choosing the free space furthest from the pallet base (line 10), ties are broken by selecting the space with the highest volume. Then, all the box layers that fit within the selected space are generated (line 11). The layers are built with the types of boxes that have several copies; a layer consists of a rectangular array of boxes positioned in rows or columns, six different layers (*XY,XZ,Y X,Y Z,ZX* and *ZY*) can be obtained by combining the three axes. For all possible rotations of the box, its different layers must be built. Types of items with only one copy are also considered as layers. Having the layer list (*Layers*) the selection of the best ones must be made (lines 13-14). In this work, a dynamic selection process was established that allows adjusting the number of candidate layers based on the performance achieved for different alpha values. This way, a balance can be made between small RCL sizes, which may result in excessively greedy behavior, and large RCL sizes, which can rapidly compromise efficiency. In the construction, the selection of the winning layer is made choosing randomly one layer from the RCL (line 14), different from the improvement phase, where the layer that best fits in the space is selected from the original list of layers (line 16). This layer must be placed within the space to build the packing pattern so that the list of remaining boxes and the available maximal spaces can be updated (lines 18-21). The maximal spaces can be exhausted, and there are still items to be packed, so it will be necessary to create new spaces (pallets) by calling the Algorithm 1 again (lines 23-25). The solution built is compared with the incumbent and is updated if it is better than the current one (line 30). In the local search iteration, the packing pattern starts from the built solution after removing a percentage of boxes. The removal consists of sorting the constructed pallets from the emptiest to the fullest and on this order, eliminate item after item until reaching a K% of boxes eliminated (lines 32-34). In this way, empty pallets are eliminated from the solution, while half-empty pallets, together with the removed items, must enter again the packing process described above. It is essential to clarify that for this iteration that lines 3 to 7 will not be executed. The constructive and improvement phases are executed a *totalIterations* number of times (line 1). The algorithm returns the *Incumbent*, which records all the palletized cargo.

Algorithm 3 describes the reactive feature of the proposed GRASP. Set A contains a pre-specified number of coefficients ($0 < \alpha_j \leq 1; j = 1, ..., |A|$), each of which defines one possible size for the RCL. For instance, in Algorithm 2, an $\alpha_j = 0.3$ would lead to an RCL comprised of the 30% most promising layers. The objective of the reactive mechanism is to intensify the use of those α_j that yield better results. This is done by dynamically adjusting the probability of choosing each α_j as a function of its cumulative performance. All coefficients start with no record of performance ($S_\alpha = 0, \forall\alpha \in A$) and equal probability of being chosen ($P_\alpha \leftarrow 1/|A|, \forall\alpha \in A$). Then, whenever a GRASP iteration uses a specific α^*, the objective function value $q_\alpha*$ is used to update its cumulative performance $S_\alpha*$ (line 17). After updating performance, the probabilities of choosing coefficients are normalized to fit a proper discrete probability distribution (line 18). The core of this strategy (lines 2-6) is to select one α_j based on the iteratively performance-adjusted probability distribution (p_α), except for the first *nTraining* iterations, which uses the initial uniform distribution instead (P_0), seeking to have information for all the α_j. The selection of the criteria to choose the most promising boxes is made randomly (line 7). There are two criteria, the first one consists of the layer that fits better in the selected space (line 8), and the second one is the layer with the highest volume (line 10). The list of layers is ordered using the selected criterion. The restricted candidate list (RCL) is constructed, passing the first α^* elements from the ordered list of layers. Finally, the RCL is returned.

Algorithm 3 Build RCL

Parameters: $A = \{\alpha_1, \alpha_2, ..., \alpha_m\}$: set of alphas, *nTraining:* number of iterations for alpha training
Input: List *Layers*: List of all layers built; *i*: current iteration;
Output: List *RCL*: restricted candidates list of layers;

```
1: $P_0 \leftarrow 1/|A|, \forall\alpha \in A$
2: if ($i \leq nTraining$) then
3:     select randomly $\alpha^*$ from $A$ with a probabilty $P_0$
4: else
5:     select randomly $\alpha^*$ from $A$ with a probability $p_\alpha$
6: end if
7: if ($rand() \leq 0.5$) then
8:     sort Layers by best-fit criteria
9: else
10:    sort Layers by max volume criteria
11: end if
12: for $i \leftarrow 1$ to $|Layers|$ do
13:     if $i \leq |Layers| * \alpha^*$ then
14:         $RCL \leftarrow RCL \cup Layers_i$
15:     end if
16: end for
17: $S_\alpha* \leftarrow S_\alpha* + q_\alpha*$, where $q_\alpha*$ is the objective function obtained with $\alpha^*$
18: $pa \leftarrow Sa/Pmi=1\ Si, \forall\alpha \in A$
19: return RCL
20: end
```

Algorithm 4 GRASP algorithm for Truck Loading

Parameters: *graspIterations, K*%: percentage of pallets to remove, *A*: set of alphas
Input: List *LP*: palletized cargo list; **List** *V*: list of vehicles;
Output: List *LoadingTruck*: loading plan for vehicles;

```
1: while $LP \neq \emptyset$ or $V \neq \emptyset$ do
2:     $BestVehiclePattern \leftarrow \emptyset$
3:     $v^{max} \leftarrow max(LP_i.volume); \forall i \in LP$
4:     $w^{max} \leftarrow max(LP_i.weight); \forall i \in LP$
5:     List $C_i^{max} \leftarrow min(\left\lfloor \frac{V_i.volume}{v^{max}} \right\rfloor, \left\lfloor \frac{V_i.weightlimit}{w^{max}} \right\rfloor); \forall i \in V$
6:     $ve \leftarrow arg_i min(abs(C_i^{max} - |LP|))$
7:     for $i = 1$ to $graspIterations * 2$ do
8:         if $localsearch = False$ then
9:             List $E \leftarrow CreateMaximalSpaces(V, ve)$
10:        end if
11:        while $LP \neq \emptyset$ and $E \neq \emptyset$ do
12:            $e \leftarrow SelectMaximalSpace(E)$
13:            List $Layers \leftarrow GeneratePalletLayerList(e, LP)$
14:            if $localsearch = False$ then
15:                List $RCL \leftarrow BuildRCL(Layers, i)$
16:                Pallet $sp \leftarrow SelectLayer(RCL)$
17:            else
18:                Pallet $sp \leftarrow SelectLayerByBestFit(Layers)$
19:            end if
20:            Packing Pattern $P \leftarrow LocatePalletLayerOnSpace(sp,e)$
21:            $E \leftarrow UpdateListOfMaximalSpaces(P,e,E)$
22:            $LP \leftarrow UpdateListOfRemainingPallets(sp)$
23:            $CurrentVehiclePattern \leftarrow CurrentVehiclePattern \cup P$
24:        end while
25:        Update $BestVehiclePattern$ if $CurrentVehiclePattern$ is better
26:        $localsearch \leftarrow Not(localsearch)$
27:        if $LP \neq \emptyset$ then
28:            if $localsearch$ then
29:                $E, LP, CurrentVehiclePattern \leftarrow RemoveK\%(CurrentVehiclePattern, K\%)$
30:            end if
31:        end if
32:    end for
33:    $LoadingTruck \leftarrow LoadingTruck \cup BestVehiclePattern$
34: end while
35: if $LP \neq \emptyset$ then
36:     display "Load exceeds the fleet capacity"
37: else
38:     return $LoadingTruck$
39: end if
40: end
```

4.2. Stage 2 - Truck Loading

The Algorithm 4 is designed to receive as parameters the palletized cargo in the *LP* list and the available vehicle fleet (*V* list). The vehicles best suitable for the characteristics of the remaining palletized cargo are selected in an iterative process. The selected vehicle and the remaining cargo are sent to a GRASP algorithm (lines 7-32), which generates the packing pattern, trying to maximize the vehicle's occupation. If the vehicle fails to pack the entire load, a new vehicle must be selected from the fleet, repeating the process. In detail, the algorithm starts by identifying the largest volume and weight of the palletized loads (lines 3-4). Then, for each available vehicle, an estimate of the maximum number of pallets possible to be packed inside it is calculated (line 5). The vehicle that presents the least difference between the estimate of pallets to be packed and the number of remaining pallets is selected (line 6). The GRASP algorithm is called to try to load the remaining pallets within the chosen vehicle (lines 7-32). This GRASP is a modification of the one presented in Algorithm 2. In this, the construction iteration begins by creating a single maximal space corresponding to the selected vehicle (line 9). The loading process selects the free space closest to one of the lower corners of the vehicle, looking to load the vehicle from the corners to the central zone (line 12); ties are broken by selecting the space with the highest volume. The list of pallet layers to be loaded in this space is generated (line 13), taking into account that only rotating the pallet on its vertical axis is permitted (R12 constraint) and the top face of the palletized load must have a minimum support area in order to be able to stack pallets (as described in R8 constraint). The function *GeneratePalletLayerList* must consider these conditions to generate the pallet rows or columns. As in Algorithm 2, construction of a restricted list of pallet layers and selection of one of these is performed (lines 15-16), as well as the updating of remaining free spaces in the vehicle, remaining pallet cargo to be loaded and the current vehicle loading plan (20-23). The best loading plan for this vehicle is updated if the current plan is better (line 25). If the pallet load has not been fully loaded (line 27), an attempt will be made to improve this loading plan by removing *K*% of the loaded pallets. The removal consists of eliminating pallets per pallet until *K*% of the pallets are completed, unloading the pallets furthest from the back of the vehicle in an orderly manner (line 29).

After this, the list of pallets still to be loaded along with the removed pallets, the modified loading plan, and the remaining spaces created after removal enter the local search iteration. In the local search iteration, the loading vehicle pattern starts from the built solution after removing a percentage of pallets. In this iteration, the only thing different from the construction phase is that the selected layer is the layer of pallets that best fits in the space (line 18). The constructive and improvement phases are executed a *graspIterations* number of times (line 7), seeking to maximize the volume of utilization of the vehicle. From this process, the best vehicle-loading pattern is recorded on the truck loading planning (line 33). This process is repeated as long as there is palletized cargo to be packed, or the vehicles in the fleet have not been exhausted (lines 1-34). This heuristic can end up with remaining loads, which would imply that the available fleet is insufficient for transporting all the cargo (lines 35-38). The algorithm returns the packing patterns for loading the vehicles if the total demanded cargo has been successful. It is important to note that for this stage and comparison purposes, the algorithm is modified to minimize each vehicle's utilization cost. For this, post-processing takes into account the cost of containers, which can be not proportional to its volume. The cargo is re-assigned to a cheaper vehicle as long as the bounding-box of the cargo fits inside.

5. Computational experiment and analysis results

5.1. Solving a real-world case and validating the proposal

We have used two sets of data: instances from the real world and reference instances. The first set motivated the design of the algorithms and was used to validate the results of our approach to the two-staged problem. In contrast, the second was used to validate the performance of our tool when compared to the best results found in the literature that contemplates at least one of the problems considered in this work. The real-world instances were provided by an industrial company and consisted of 2235 boxes of 165 different sizes. The boxes must be transported in a sub-set of the vehicles shown in Table 2.

Table 2
Available Vehicles

Vehicle ID	Internal Dimensions (cm)			Maximum Load (kg)
	Length	Width	Height	
T1	190	150	180	1000
T2	300	170	170	2500
TT	450	220	220	4500
ST	650	240	240	9000
DT	850	240	240	17000
TR	1200	230	220	33500
MC	1200	250	250	31600
P	1200	240	270	17000
MF	1200	250	265	31600

The cost of each vehicle is proportional to its volume. The number of vehicles and pallets are both unlimited, and each pallet has a size of 120cmx100cmx175cm length, width, and height, respectively. The instances are publicly available for future research purposes on http://www.opentechs.co/instances or can be requested from the corresponding author. For the reference instances, three groups of instances were selected according to three criteria: solutions reported in the literature for the DPPP or MBSBPP cases, performances measured and published for these solutions and access to download the test instances. To validate our algorithm regarding the DPPP, we selected the benchmark instances presented in Zachariadis et al. (2012) under the names: (1) PPVRP, (2) Tight PPVRP. The first group comprises four categories: 3, 5, 10, and 20 types of box, each with 14 instances. The second group was generated under tighter loading constraints conditions and is divided into two categories: three and five types of boxes, each one made of seven instances. The goal of these instances is to reduce the required number of vehicles. Those instances were first detailed in Zachariadis et al. (2012) and are available to download from Zachariadis (2017) with the name 'Pallet Packing Vehicle Routing Problem.' To validate our algorithm regarding the MBSBPP, we selected two groups of instances: the first group of test cases were developed by Ceschia and Schaerf (2013), formed by 13 problems using two different types of containers with limited availability. The goal is to load completely the demanded boxes into containers of several sizes, minimizing the required number of containers. Instances are available in Ceschia and Stutzle (2017). The second group of cases was developed by Alvarez-Valdés et al. (2013) and is formed by 320 problems divided into eight classes of 40 instances each. Every class contains four groups of 10 instances, and each has several different boxes to be loaded (50, 100, 150, and 200 boxes). The entire cargo must be load into containers of different sizes and costs, minimizing total packing cost. The instances are available on Pisinger (2016).

5.2. Details of the experiments

All Algorithms were coded into a prototype software named GrasPacking, using C++. The software generates a report containing the performance indicators of the computed solution. Meanwhile, the PackageCargo software presented in Martínez et al. (2018b) was used to visualize the calculated solution.

For real-world instances, it is not possible to make a fair comparison with the previous works (See Table 1). For this reason, we use the specialized commercial software Esko (2016) (CapePack^{TM} version 16.01), which the company uses in its loading operations because it satisfies the constraints (R1 - R12), that enables a valid comparison. This software allows the selection between two heuristics to solve the problem: an algorithm of construction by either rows or columns and adapt the necessary constraints. CapePack^{TM} also generates a figure with the solution and delivers performance indicators. Both software tools were installed and run in a personal computer with an Intel Core^{TM} i7-4790 processor clocked at 3.60GHz and 8GB of RAM running a 64-bit version of Windows 7®.The proposed methodology requires the tuning of the parameters: the number of total iterations, the percentage of boxes to be removed, the set of alphas and the number of iterations for training the alphas. These parameters are called *totalIterations*, *K%*, *A* and *nTraining* respectively. Meanwhile, for the Truck Loading the number of total iterations is mentioned in the document as *graspIterations*. The values of the parameters used for the methodology were calibrated through an experimental design, being *totalIterations* = 10 × *ItemTypes* × *np*, where *ItemTypes* is the number of types of boxes to be packed and *np* the number of pallets obtained by the Algorithm 1 for the original demand of boxes. *K%* = 30%, obtaining again the reference value proposed in Martínez et al. (2015). *A* = {10%, 20%, 30%, 40%, 50%}, being a reduced set of values, which introduce a controlled degree of randomness on the selection candidate function. *nTraining* = 35%, guaranteeing a large enough number of iterations for all alpha values to be tested and performance information obtained. Finally, *graspIterations* = 10 × *ItemTypes*, where *ItemTypes* is the number of remaining pallets types to be packed. To validate our performance regarding the DPPP, we used the results presented on Zachariadis et al. (2012), mainly focusing on the results obtained by using the packing proposed strategy, that performs a local search and implements a Tabu search algorithm. The reason for this selection is that it provides the best results in the literature for this particular group of instances.

$$Utilization\ (\%) = \frac{\sum_{i \in PB} V_{C_i}}{VC} \times 100 \tag{1}$$

$$Cubic\ Waste(m^3) = \sum_{j \in UV} (1 - Utilizationj) * vmj \tag{2}$$

$$GAP = \frac{UB - LB}{UB} \tag{3}$$

For validation regarding the MBSBPP, we used the results presented in Alvarez-Valdés et al. (2013), centered on the results produced by their GRASP algorithm using the dynamic path relinking improvement procedure, as this has the best global performance. We also used the lower bounds obtained using the linear relaxation of an integer formulation of the problem, presented on Alvarez-Valdés et al. (2015). For the MBSBPP, we used the information provided by Ceschia and Schaerf (2013), which uses a local search approach based on a compound strategy that interlaces the simulated annealing algorithm with large-scale neighborhood search.

5.3. Performance indicators

For real-world instances, we measured the quality of the solution by the percentage of utilization defined by Equation 1, where VC is the total volume of the container and V_{C_i} is the volume of the ith item packed (being PB the set of packed boxes). Furthermore, cubic waste, as defined in Equation 2, was measured for stage two, as vehicles had different sizes. Being vm_j the volume in m^3 of the vehicle j and $Utilization_j$ the percentage of volume utilization of the vehicle j ($\forall j \in UV$), UV being the set of used vehicles to pack all of the palletized cargo. The computing time is measured with the clock functions inserted on the GrasPacking prototype. Static stability is determined through the reproduction of the solutions in a robotic packing cell, as is implemented in Camacho-Munoz et al. (2018).

For cases regarding the DPPP, the performance indicators measured in Zachariadis et al. (2012) were used: *Tp* is the total number of pallets required to load the entire shipment and *v* is the total number of vehicles necessary to load all of the pallets, for these instances the vehicles have a predefined pallet capacity.

On the MBSBPP, *v'* was used to measure the required number of the vehicles needed to load the entire cargo, and ϕ^0 is the average percentage of vehicle utilization. The computing time needed to reach the solution is also recorded. On the other hand, comparing against Alvarez-Valdés et al. (2013) has measured the GAP, which is defined in Alvarez-Valdés et al. (2015) and presented on the Equation 3, where *UB* is the solution for each algorithm, and *LB* is the lower bound.

Table 3
Results for real instances – Palletizing

	CapePack^{TM}		GrasPacking	LB
	Rows	Columns		
Utilization (%)	77.6	80.6	91.5	-
Number of Pallets (units)	28	27	24	23
Computing Time (sec)	5.4	174	47	<1

Table 4
Results for real instances - Truck Loading

	CapePack^{TM}				GrasPacking	
	Rows		Columns			
Vehicles[*]	DT	ST	DT	ST	DT	TT
Utilization (%)	64.3	49.9	60.7	56.4	62.8	70.3
Computing Time (sec)	1	<1	7	1	<1	<1
Cubic Waste (m^3)	17.5	18.8	19.2	16.3	18.23	6.5
Total Cubic Waste (m^3)	36.2		35.6		24.7	

[*] DT stands for single unit 3-axle truck, ST is a small truck, and TT is a cargo van (details in Table 2)

5.4. Results for real-world instances

Results for the first stage are presented in Table 3. CapePack^{TM} column displays the results obtained by the two heuristics of this software (Rows Heuristic and Columns Heuristic), GrasPacking column refers to the results obtained by the GRASP algorithm proposed for Palletizing (Algorithm 2) and the column LB displays the lower bound obtained by the Algorithm 1 for the original demand of the boxes. All solutions were reached in reasonable computing time. The percentage of use is in favor of GrasPacking, as fewer pallets are required to load all of the items: four pallets less than the Rows Heuristic (28 pallets) and three pallets less than the Columns Heuristic (27 pallets). Moreover, the solution obtained by the GrasPacking tool is very similar to the one found by the Algorithm 1 (23 pallets). Results for stage two are presented on Table 4. CapePack^{TM} column displays the results obtained by the two heuristics of this software for Truck Loading purposes (Rows Heuristic and Columns Heuristic), GrasPacking column refers to the results obtained by the GRASP algorithm proposed for Truck Loading (Algorithm 4). All three algorithms produce a solution using two vehicles. Computing times remained reasonable. Analyzing the results of the software CapePack^{TM} it can be seen that both use the same vehicles, it is noted that the Rows Heuristic has the best arrangement for DT vehicle, because it loads the tallest pallets. GrasPacking used the fleet of vehicles with the smallest volume (DT and TT), achieving an adequate optimization of the load on the TT vehicle by arranging the pallets on the base of the vehicle in a non-guillotine pattern, resulting in a cubic waste less than the solution obtained by CapePack^{TM}.

5.5. Results for reference instances

Results obtained for the DPPP are presented in Table 5. The tool used by Zachariadis et al. (2012) was implemented in Visual C# and executed on a computer using a single-core E6600 processor clocked at 2,4 GHz. Analyzing the results produced by our algorithm against those obtained by Zachariadis et al. (2012), it can be observed that our algorithm can solve instances that contemplate 3, 5, and 20 types of boxes. For those instances, GrasPacking computes a solution that corresponds with the minimum number of pallets and vehicles reported by Zachariadis et al. (2012). When solving test cases involving ten types of boxes, our algorithm cannot reach the solutions because maximal spaces of different stacks are not combined, making it impossible to stack large items on top of them. Those above motivated the authors to include a routine of joining maximal spaces, solving this issue. Still, the computational cost was high and, for this reason, was not included inside the algorithm. Results obtained by GrasPacking and LS-TS both present a marked difference concerning the results produced by the Lower Bound. The authors consider that this behavior remarks on the effects of the geometric constraint and complete shipment constraint on the minimization of required pallets and vehicles.

For the analysis of the MBSBPP results, a non-parametric test was used to relate the measures, using the Wilcoxon test for paired samples, as mentioned in Gehan (1965). The results of the proposed methodology are presented in the Table 6 in comparison with Ceschia and Schaerf (2013), which used a 2.66 GHz quad-core machine with 4 GB of RAM, coded in C++ and ran on Ubuntu Linux x86 64. It can be observed that GrasPacking has better utilization of the vehicles, with a 66% on average; meanwhile, Ceschia and Schaerf (2013) has 59%, this difference is reflected by analyzing the total sum of vehicles needed to pack the boxes of all the test cases, Ceschia and Schaerf (2013) uses 191 vehicles while the proposed GRASP uses 14 fewer vehicles. It should also be noted that in cases such as SD-CSS5, although both algorithms use two vehicles, the proposed GRASP has a better occupancy because it selects vehicles of lower capacity. When performing the test of Wilcoxon, this mentions that the difference between the two methodologies is not significant for the vehicles. Otherwise, analyzing the percentage of use (ϕ^0) and the number of iterations, the Wilcoxon test shows significant differences between these two indicators between the two methodologies. That means our algorithm presents a higher performance maximizing the volume of utilization of the container having a difference in the average of 7% and finding these solutions in a lower number of iterations. The Table 7 shows the comparison between the results provided by our metaheuristic and the best performance algorithm presented in Alvarez-Valdés et al. (2013), who uses a Pentium Mobile machine at 1500 MHz with 512Mbytes of RAM and a C++ encoding. Using the Wilcoxon test, can be concluded that there is a significant difference when using the post-processing after executing the Algorithm 4, achieving a better performance in the gaps obtained for each class (column 4 and 5 of the Table 7). In comparison with Alvarez-Valdés et al. (2013), we can see that our metaheuristic produces slightly better solutions in four of the eight classes using post-processing, but when performing the Wilcoxon test, it is obtained that there is no significant difference between these two solution approaches. It can also be seen that the average number of iterations required by our algorithm exceeds the amount reported for Alvarez-Valdes et al. (2013). This difference allows us to see that our algorithm considers a higher number of solutions in its search procedure, almost five times more than the GRASP/PR.

Table 5
Results of the instances referring to the DPPP. Compared against the LS-TS algorithm of Zachariadis et al. (2012)

Class	Types of Boxes	Avg Boxes	Number of Intances	LS-TS		GrasPacking		LB	
				v	Tp	v	Tp	v	Tp
1	3	9920	7	87	348	87	348	75	299
2	5	11364	7	99	396	99	396	84	333
3	3	1022	14	188	784	188	784	136	542
4	5	1120	14	181	735	181	735	137	544
5	10	2997	14	**197**	**802**	201	806	136	541
6	20	8276	14	200	815	200	815	136	541
	TOTAL		70	952	3880	956	3884	704	2800

Table 6
Results of the instances referring to the MBSBPP. Compared against the SA-LNS algorithm of Ceschia and Schaerf (2013)

Instance	SA-LNS*				GrasPacking				LB
	v'	φ'	sec	$iterations$	v'	φ'	sec	$iterations$	v'
SD-CSS1	4	0.56	16.1	110000	4	0.56	1.34	5600	3
SD-CSS2	13	0.60	0.33	250000	13	0.60	0.28	7100	8
SD-CSS3	**22**	0.49	0.79	330000	23	0.47	0.10	7800	11
SD-CSS4	**11**	0.62	3.98	370000	12	0.57	0.27	8800	7
SD-CSS5	2	0.33	228.53	820000	2	0.34	7.19	4700	2
SD-CSS6	19	0.53	2749.62	430000	**13**	0.77	51.28	45100	10
SD-CSS7	10	0.63	1.1	900000	10	0.63	0.2	5100	7
SD-CSS8	21	0.64	1769.38	960000	**16**	0.81	22.76	44400	13
SD-CSS9	17	0.65	1079.92	560000	**15**	0.74	5.65	27200	12
SD-CSS10	7	0.67	1023.35	600000	**6**	0.78	2.28	9600	6
SD-CSS11	9	0.63	513.38	1840000	**7**	0.80	2.72	11700	6
SD-CSS12	**37**	0.73	17.58	1290000	42	0.64	0.43	32700	27
SD.CSS13	19	0.63	1328.95	1290000	**14**	0.86	20.04	38600	12
Average		0.59				**0.66**			
TOTAL	191		671.77	9750000	**177**		**8.81**	**2484000**	124

*sec stands for computing time in seconds required for the instance and $iterations$ is the number of iterations evaluated.

Table 7
Results of the instances referring to the MBSBPP compared against the GRASP/PR algorithm of Alvarez-Valdes et al. (2013)

Class	Results reported at Alvarez-Valdes et al. (2013)		GrasPacking*	
	LB	GRASP/PR (Gap)	Original (Gap)	W/Post-pro (Gap)
1	226551.4	18.89	18.76	**18.51**
2	253805.6	**12.10**	13.39	12.53
3	236982.1	16.87	14.68	**14.31**
4	559487.55	2.90	3.54	**2.56**
5	134126.88	**16.45**	18.59	16.90
6	188948.25	**13.07**	16.96	15.02
7	95387.75	**24.32**	28.26	26.81
8	15343.45	18.94	18.95	**17.48**
Overall	231090.50	**15.44**	16.64	15.51
Overall Iterations		**4369.40**	19668.53	

*W/Post-pro stands for the results reached after applying post-processing (reassigning the palletized cargo to a cheaper vehicle), and Original is the result without post-processing.

6. Conclusions and remarks

We have presented a heuristic approach to solve a two-stage real-life packing problem: the pallet-building problem and subsequent loading in a heterogeneous fleet of vehicles considering practical constraints. Our heuristic is based on a GRASP optimization scheme and maximal spaces representation. This combination has shown significant flexibility. Specifically, creating and managing several initial maximal spaces allow the packing process to fill several pallets simultaneously. We improved the optimization scheme using a reactive GRASP that adjusts the size of the restricted candidate list as a function of performance (considering a training phase) and alternating the item selection criteria randomly. We have found difficulties in reaching a fair comparison between our algorithm and previous works on the two-stage real-life packing problems. That is because the earlier works were inspired by different real applications, which results in particular variants of the problem. Our revision allowed us to register the practically relevant constraints reported on those problems. To the best of our knowledge, there is no complete registration of constraints in the two-staged real-life packing problem than the one presented in our paper. To handle the difficulty in comparison, we defined two strategies: (1) use instances of the real world, and (2) use benchmark instances that contemplate at least one of the problems considered in our work. The first strategy took as a baseline comparison, a commercial software of high acceptance, which allows us to parameterize the enabled constraints. The results show successful performance in reasonable computing times, outperforming the quality of solutions against the commercial software, and matching the solutions of other algorithms on the literature. The heuristic approach presented in this study has shown to be a suitable and efficient solution method for two-staged packing problems. Therefore, its applications must be expanded by considering different real-world constraints, like load balance constraint of the European community for land transport loading plans or the dynamic stability constraint that tries to ensure integrability of the cargo and safety of the loading/unloading operators. Furthermore, the heuristic could be hybridized with other optimization techniques based on the formulations presented in Alonso et al. (2019a), to obtain better solutions in satisfactory times.

References

Alonso, M. T., 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. T., Alvarez-Valdes, R., Iori, M., Parreño, F., & Tamarit, J. M. (2017). Mathematical models for multicontainer loading problems. *Omega*, *66*, 106-117.

Alonso, M. T., Alvarez-Valdes, R., & Parreño, F. (2020). A GRASP algorithm for multi container loading problems with practical constraints. *4OR*, *18*(1), 49-72.

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

Alvarez-Valdés, R., Parreño, F., & Tamarit, J. M. (2013). A GRASP/Path relinking algorithm for two-and three-dimensional multiple bin-size bin packing problems. *Computers & Operations Research*, *40*(12), 3081-3090.

Alvarez-Valdés, R., Parreño, F., & Tamarit, J. M. (2015). Lower bounds for three-dimensional multiple-bin-size bin packing problems. *OR spectrum*, *37*(1), 49-74.

Amossen, R. R., & Pisinger, D. (2010). Multi-dimensional bin packing problems with guillotine constraints. *Computers & operations research*, *37*(11), 1999-2006.

Ballew, B. P. (2000). *The distributor's three-dimensional pallet-packing problem: a mathematical formulation and heuristic solution approach* (No. AFIT/GOR/ENS/00M-02). AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING.

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

Brunetta, L., & Grégoire, P. (2005). A general-purpose algorithm for three-dimensional packing. *INFORMS Journal on Computing*, *17*(3), 328-338.

Burke, E. K., Hyde, M. R., Kendall, G., & Woodward, J. (2012). Automating the packing heuristic design process with genetic programming. *Evolutionary computation*, *20*(1), 63-89.

Camacho-Munoz, G., Rodriguez, C., Alvarez-Martinez, D., (2018, June). Modelling the kinematic properties of an industrial manipulator in packing applications. In *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, IEEE, pp. 1028–1033.

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

Ceschia, S., Stutzle, T., (2017). http://www.diegm.uniud.it/ceschia/index.php?page=vrclp.

Clautiaux, F., Dell'Amico, M., Iori, M., & Khanafer, A. (2014). Lower and upper bounds for the Bin Packing Problem with Fragile Objects. *Discrete applied mathematics*, *163*, 73-86.

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

de Almeida, A., & Figueiredo, M. B. (2010). A particular approach for the Three-dimensional Packing Problem with additional constraints. *Computers & Operations Research*, *37*(11), 1968-1976.

de Castro Silva, J. L., Soma, N. Y., & Maculan, N. (2003). A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions in Operational Research*, *10*(2), 141-153.

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), 1-20.

Epstein, L., & Levy, M. (2010). Dynamic multi-dimensional bin packing. *Journal of Discrete Algorithms*, *8*(4), 356-372.

Esko, (2016). https://www.esko.com/es/products/cape-pack.

Gehan, E.A., (1965). A generalized wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika* 52, 203–224.

Hifi, M., Negre, S., & Wu, L. (2014). Hybrid greedy heuristics based on linear programming for the three-dimensional single bin-size bin packing problem. *International Transactions in Operational Research*, *21*(1), 59-79.

Jin, Z., Ito, T., & Ohno, K. (2003). The three-dimensional bin packing problem and its practical algorithm. *JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing*, *46*(1), 60-66.

Lim, A., & Zhang, X. (2005, March). The container loading problem. In *Proceedings of the 2005 ACM symposium on Applied computing* (pp. 913-917).

Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations research*, *48*, 256-267.

Martínez, D. A., Alvarez-Valdes, R., & Parreño, F. (2015). A grasp algorithm for the container loading problem with multi-drop constraints. *Pesquisa Operacional*, *35*(1), 1-24.

Martínez, J. C., Cuellar, D., & Martínez, D. (2018a). Review of Dynamic Stability Metrics and a Mechanical Model Integrated with Open Source Tools for the Container Loading Problem. *Electronic Notes in Discrete Mathematics*, *69*, 325-332.

Martínez, J.C., Cuellar, D., Cespedes, E., Martínez, D., (2018b). Packagecargo - open source tool based on optimization and simulation for the container loading problem with dynamic stability. In *2018 International Conference on Industrial Engineering and Operations Management*, IEOM, pp. 2363–2370.

Miyazawa, F. K., & Wakabayashi, Y. (2009). Three-dimensional packings with rotations. *Computers & Operations Research*, *36*(10), 2801-2815.

Morabito, R., & Morales, S. (1998). A simple and effective recursive procedure for the manufacturer's pallet loading problem. *Journal of the Operational Research Society*, *49*(8), 819-828.

Morabito, R., Morales, S. R., & Widmer, J. A. (2000). Loading optimization of palletized products on trucks. *Transportation Research Part E: Logistics and Transportation Review*, *36*(4), 285-296.

Moura, A., & Bortfeldt, A. (2017). A two-stage packing problem procedure. *International Transactions in Operational Research*, *24*(1-2), 43-58.

Moura, A., & Oliveira, J. (2005). A GRASP approach to the container-loading problem. *IEEE Intelligent Systems*, *20*, 50-57.

Olsson, J., Larsson, T., & Quttineh, N. H. (2020). Automating the planning of container loading for Atlas Copco: Coping with real-life stacking and stability constraints. *European Journal of Operational Research*, *280*(3), 1018-1034.

Paquay, C., Limbourg, S., & Schyns, M. (2018). A tailored two-phase constructive heuristic for the three-dimensional Multiple Bin Size Bin Packing Problem with transportation constraints. *European Journal of Operational Research*, *267*(1), 52-64.

Pisinger, D., (2016). http://hjemmesider.diku.dk/~pisinger/codes.html.

Silva, E., Oliveira, J. F., & Wäscher, G. (2016). The pallet loading problem: a review of solution methods and computational experiments. *International Transactions in Operational Research*, *23*(1-2), 147-172.

Takahara, S. (2005, July). Loading problem in multiple containers and pallets using strategic search method. In *International Conference on Modeling Decisions for Artificial Intelligence* (pp. 448-456). Springer, Berlin, Heidelberg.

Viklund, A., (2010). http:http://metrology.burtini.ca/dimwt.php.

Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European journal of operational research*, *183*(3), 1109-1130.

Zachariadis, E.E., (2017). http://users.ntua.gr/ezach/. Online; accessed 13/12/2018.

Zachariadis, E. E., Tarantilis, C. D., & Kiranoudis, C. T. (2012). The pallet-packing vehicle routing problem. *Transportation Science*, *46*(3), 341-358.

Zudio, A., da Silva Costa, D. H., Masquio, B. P., Coelho, I. M., & Pinto, P. E. D. (2018). BRKGA/VND hybrid algorithm for the classic three-dimensional bin packing problem. *Electronic Notes in Discrete Mathematics*, *66*, 175-182.