A heuristic algorithm for large scale, highly heterogeneous, constrained container loading

***Abstract*— Our research aims to optimize container loading operations in logistics operations that use large trucks to transport goods between logistic centers. We adapted a heuristics-based algorithm to solve large, highly constrained bin-packing problems, to improve the volume efficiency while ensuring that a large number of constraints are met. We then use Particule Swarm Optimization to find the best combination of hyper-parameters in the heuristic algorithm. Measuring the performance of our heuristic in a set of scenarios comprising of real cargo data and restrictions in the logistics domain, the resulting algorithm achieved an average volume utilization and computing time, similar to other state-of-the-art solutions but handling far more constraints and a larger number of highly heterogeneous packages.**

***Index Terms*— Container loading problems, middle mile logistics, constrained optimisation, large trucks cargo, particle swarm optimization**

I. Introduction

Container Loading Problems (CLP), sometimes called 3D packing problems (3DPP), have been extensively used in the logistics arena to help the workforce carry out packaging that maximizes volume utilization considering the natural constraints present in loading processes. Bulk transportation logistics or middle-mile logistics impose specific conditions for the application of these techniques, as they are usually made using large containers (trucks), taking a very high number of packages with a large degree of heterogeneity in their sizes, with few stops between warehouses.

The fill rate of trucks is of paramount importance for both environmental and economic reasons. Loading the cargo into the truck by stacking/loose loading, which requires human intervention to introduce the packages and fit them properly, offers an average container fill rate of approximately 60-76% by volume. While using pallets with packages to fill the container is easier and quicker, it only scores at 50% fill rate on average.

In addition, there is a trade-off between volume usage and stacking complexity depending on the heterogeneity of the items to be packed. Some players consider approximately 25 different packaging dimensions (i.e., height = 10cm, width = 5cm, length = 20cm), but others have many more, such as Amazon. This fosters research into scenarios with a large degree of heterogeneity.

Inherently, with the purpose of this algorithm, our intention is to work on highly constrained realistic scenarios, as occurs in the middle mile, with large containers (trucks). The dimensions of the truck that we are going to use are ($c_W = 2.45m, c_H = 2.45m, c_L = 13.6m$), which makes a total volume of 81.6 cubic metres, and is an extreme scenario with one of the largest containers allowed by European regulation.

Although there are many approaches and solutions to this problem, most current approaches consider synthetic, small scenarios with a small degree of heterogeneity and use a reduced set of constraints. Our contribution is to solve the offline container loading problem by using a heuristic capable of handling complex situations in which a large set (at least 300 packets) of highly heterogeneous items must be packed to meet many constraints into a container to retrieve good volume usage results in a reasonable amount of time. In the formulation of our heuristic, we adapted several known approaches, modified some steps or techniques such as bounds for constraint satisfaction, randomization or sorting of packages before packing, and used Particle Swarm Optimization (PSO) to calculate the hyper-parameters identified in the general recursive and constructive strategy, to solve the packing challenges in highly constrained scenarios.

II. Literature review

The problem we attempt to solve is part of the family of combinatorial optimization problems called Container Loading Problems (CLP). More specifically, we focus on solving, in a 3D space, a mixture of the single large object placement problem and the single packet problem defined in [25]. It consists of two sets of items: one of the larger items $C = \{c_0, ..., c_m\},\ m \in \mathbb{N}$, the container or containers, and the other set for smaller items $P = \{p_0, ..., p_n\},\ n \in \mathbb{N}$, the packets. The objective is to maximize the volume utilization when packing a certain set of packets (P) in any container $c_i \in C,\ i \in \mathbb{N}$ satisfying a set of restrictions/constraints, $R = \{r_0, ..., r_o\},\ o \in \mathbb{N}$. Therefore, by definition, the problem has fixed limits in the result of its volume utilization, being $[0\%, 100\%]$ both the minimum and maximum values of volume utilization of a container. Furthermore, P must have a cumulative volume higher than the volume of the container so that the theoretical upper bound can be reached.

CLP is an NP-hard problem that is usually approached using heuristics and approximation algorithms. In this particular application of the problem, its complexity depends on how constrained it is, that is, how many of the restrictions in R are active, on the size of items in P for a given volume, and on the ratio between the cumulative volume of P and the volume of the container.

Importantly, heterogeneity is highly correlated with the problem complexity. We consider two packets to be different, even though they are cuboids, if their dimensions are different as defined by [25]. The heterogeneity of the set of packets increases the complexity and establishes a proportional upper limit in volume utilization, usually less than 100%, as the mismatch of the dimensions of a set of packets that must be stacked in a finite space introduces small cavities that most of the time cannot be filled with any other packet. When these remaining spaces are not well distributed and fail to ensure that there is no fall in the top-height allocated packets, the stability and compact properties of the solution in its transportation are inappropriate.

Restrictions are a key element in CLP. There is a subset $F \subset R$ of fundamental and necessary restrictions that any instance or CLP must satisfy. These are the physical constraints implied by the mutual respect of the physical dimensions between any pair of packets, thus not overlapping, as it should be physically impossible; any packet must be within the dimensions of the container in which it is packed. We can differentiate between hard and soft constraints [10], [14], [18], [21], [22]. The solution must meet the hard constraints, such as the fundamental ones F or (R1), (R2), (R3), and (R6), but not necessarily the soft ones.

Below is a summary of the constraints commonly considered in this context:

- Weight limit (R1): maximum tonnage allowed by regulation in the entire container and its slices.
- Weight distribution (R2): Weight distribution along different axes of container. We set specific limits as in [22], [21] and [4], in which the mass center must be located. In our case, the center of mass must be within a stricter range, that is $[0.3 * c_W, 0.7 * c_W]$, $[0.3 * c_L, 0.7 * c_L]$, $[0.0 * c_H, 0.5 * c_H]$ being c_W, c_L, c_H the width, length and height of the container, respectively.
- Loading priorities (R3): Some items have priorities, meaning that they must have a preference over others with lower priority if there is no space for all.
- Orientations (R4): packets can only be assigned to certain orientations or to all. For example, a TV set is not allowed to be oriented with the screen side aimed at the floor or roof of the container.
- Stacking (R5): Packets can support others, depending on their fragility and weight. We considered a regular packet able to support others with the same weight, and that is only able to support half its weight.
- Dangerous items (R6): Some items have special regulations because of their dangerous consideration. In this case, normative implies that they must be in specific positions near the rear of the container.
- Multi-drop situations (R7): Distribute packets along the cargo considering the distribution route.
- Stability (R8): Stacked packets must have a minimum part of their area supported. In our worst-case scenario we considered a minimum supporting area of 75% of the base.

We added two new restrictions not found in the literature to cope with packages containing part of an entire unit:

- Complete Shipment (R9): Packets may be part of a set that must be shipped together.
- Relative positioning (R10): Some items must be loaded close to others to improve the multi-drop situations.

As mentioned, CLPs have been in the air for many years. One of the most common approaches to solving CLPs is to use mixed-integer linear programming (MILP), such as in [3], [17] which restricts some or all decision variables to integer values or within specific bounds [11]. As in [8], [16], [19], these approaches do not achieve good results with more than the fundamental constraints simultaneously on instances with more than 20 items. In addition, besides the fact that some of these realistic constraints and their combinations are very difficult to formulate using an MILP approach. Others such as [22] and [21] take into account (R2), (R7) and (R8) formulated with MILP and added to the fundamental constraints on instances between 90 and 110 weakly heterogeneous elements.

Therefore, the literature shows that if we want to target weakly and strongly heterogeneous instances of average-low weight/volume packets using MILP, we will suppose an exponential complexity that significantly affects the time performance. In fact, there are some official software such as GUROBI to formulate this type of problem with MILP and other commercial solutions such as EasyPack that only consider (R1), (R5) and (R4).

In recent years, there has been a varied use of heuristics and metaheuristic implementations: [5] used Genetic Algorithms (GA) and reported to handle up to 100 packages of 10 different types; [2] used Greedy Randomized Adaptive Search Procedure (GRASP); [27] used Variable Neighborhood Search; [15] used PSO; [26] used Deep Reinforcement Learning, for example [28] focused on stability, considering no other restrictions, and validated the approach by means of a robot installation, reaching a range of 50 packages; and others, such as [10], used constructive heuristics. We found the largest set of packages handled in [12] that deals with the multiple bins balance problem and reports experiments up to 200 packages per bin; the stack the packages in 3 levels but in contrast to our proposal they only handle 3 types of different packages and do not consider other restrictions than the basic ones. The authors have described different approaches and steps in these sets of solutions; for example in [1], [20], [29] stated: stack building, horizontal layer building, block building, wall building, guillotine cutting, sequencing and improvement. An in-depth analysis of these solutions can be found in [1] and [7]. Most of the cited works generally consider constraints individually [8] or a small set of constraints [2], [18]. Recent contributions, such as [10] have begun to address a more extensive set of them simultaneously and implement them in real scenarios.

Our approach is designed to handle all the constraints mentioned before, all which are enforced as hard, except for (R7) where we allow a low number of unloading obstacles. Compared with the literature only [10] handles the same number of constraints. Therefore, our heuristic takes it as a framework, modifies it and combines some other approaches, making a new logic out of these modifications and using PSO [24] to improve the results.

III. THE ALGORITHM: STEPS AND RESTRICTIONS

Our method, inspired by [9], is named a ***Recursive Greedy Randomized Constructive Heuristic (RGRCH)***, which incorporates particle swarm optimization to fine-tune the heuristic parameters tailored to different scenarios.

This approach is called greedy because it uses multiple local fitness functions across various modules and stages. These functions, optimized using PSO, aim to minimize a global cost function that is inversely related to the efficiency with which the volume is utilized within the container. Randomization is used to shuffle packets after they have been sorted, which yields variability in the solutions. The process is constructive as it builds a solution incrementally for the given data set of packets, P.

The constructed solution, S, evolves through a sequence of steps, $S = s_0 \cup s_1 \cup ... \cup s_t$ where $t = |P_p| - 1$ and each step, s, marks the addition of a packet to the solution. Starting with an empty container, s_0, corresponds to the first packet being added, and each subsequent step represents the inclusion of an additional packet, cumulatively building the packed set.

The algorithm recursively calculates multiple solutions and selects the optimal one S_{best} that maximizes volume utilization while satisfying all imposed restrictions, R. This optimal solution is chosen from the set of iterative solutions, $AS = \{S_0,\ ...,\ S_n\}$, which can be feasible and infeasible solutions, as has been reported in other studies in the literature [2], [8], [22].

Figure 1 shows an overview of the stages, modules, and their corresponding interactions, referenced by numbers in order. We sequentially go over each of these modules.

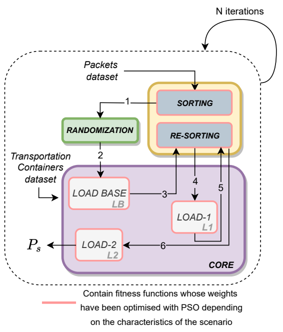
N iterations
Packets dataset
1
SORTING
RANDOMIZATION
RE-SORTING
2
Transportation Containers dataset
LOAD BASE
LB
3
4
5
LOAD-1
L1
P_s
LOAD-2
L2
6
CORE
Contain fitness functions whose weights have been optimised with PSO depending on the characteristics of the scenario

Fig. 1. RGRCH-PSO overview.

Part of the novelty of our heuristic lies in its ability to perform well in highly constrained scenarios, supporting all constraints. These constraints can occur simultaneously and affect volume usage, usually with an inverse relationship between the number of active constraints and their intensity, volume utilization, and percentage of feasible solutions. [8] demonstrated this, their work tests one by one all the practical constraints formulated with an integer linear programming approach, also stating that all practical constraints, except for multiple orientations (R4), reduce the volume when applied.

The starting data are a set of packets, P, which have at least the cumulative volume of the container and are sorted according to the scenario characteristics and constraints. The data set that must be packed is passed to the algorithm. As the algorithm builds the solution, the data set will split into two and will refer to the set of boxes which have already been sorted and randomized and are waiting to be packed P_q, and a set of packets already packed P_p, where P_q, P_p are disjoint, and its union forms the scenario data set $P = P_q \cup P_p$. The packing algorithm will be picking packets from P_q and introducing them into P_p with in the beginning as $P_p = \emptyset$.

a) **SORTING***:* Sorting has shown to be a crucial step in achieving constraint satisfaction and better packing solutions. It has been used through methods such as non-increasing volume sorting by [23] and criteria-based decreasing sorting by [10] focusing on priority and multi-drop constraints in warehouses.

Considering a set of packets P with properties $Priority_{max} \in \{0,\ 1\}$, W_{max}, V_{max} represent the maximum priority, weight, and volume of packets in P respectively, and $D = \{d_0, ...,\ d_{n-1}\}$, $n = |D|$ is the number of destinations. For each packet, p_i characterized by $(w_{p_i}, v_{p_i}, p_{p_i})$ as their weight, volume and priority respectively, and $dc_{p_i} \in \{0, ..., |D| - 1\}$ as the destination code inside the route, the fitness function is:

$$Fitness_{\text{sort}} = \left(\left(\frac{v_{i_l}}{V_{\max}} \cdot 0.5 + \frac{w_i}{W_{\max}} \cdot 0.5 \right) \cdot \alpha + \frac{p_i}{P_{\max}} \cdot \beta \right) + (d - dc_i) \quad (1)$$

Based on this function, P was sorted as $P' = \{p'_0, .., \ p'_n\}$ where p'_0 achieves the highest score and p'_n the lowest. The parameters (α, β), adjusted PSO balance the influence of the weight and volume. This holistic approach not only normalizes packet characteristics within the dataset but also aims to implement a last-in-first-out (LIFO) sorting strategy for efficient packet handling across multiple warehouses, thereby minimizing the separation time at destinations. The unified treatment of weight and volume through $alpha$ simplifies the PSO dimensionality by integrating previously isolated factors into a comprehensive fitness function.

The sorting is aware of multi-drop (R7) so the first packets in the queue correspond to the last customer on the route, good clustering of destinations minimizes the time spent moving packets from other destinations when the unloading of cargo from a certain destination, these are called "unloading objects/obstacles". The algorithm complies with relative positioning (R10) using a similar logic to the handling of (R7), but instead of destination codes with relative positioning codes.

b) **RANDOMIZATION***:* Research in [13], [2], and [10] proved the importance of randomization in iterative algorithms. Our implementation mirrors these approaches but imposes stricter constraints on the similarity between items being swapped. For a sorted set of items $P' = \{p'_0, ...,\ p'_n\}$, the randomization process consisted of three phases:

1) Volume swap: An item p'_j may swap with its consecutive p'_{j+1} with 50% chance if their volume ratio $v_{i_j}/v_{i_{j+1}} \in [0.85, 1.15]$ and have the same destination code.
2) Weight swap: this occurs between consecutive items with 50% chance if their weight ratio $w_{i_j}/w_{i_{j+1}} \in [0.85, 1.15]$ and have the same destination code.

3) Priority swap: Items may swap with a 50% chance if they have identical priorities, their weight ratio $w_{i_j}/w_{i_{j+1}} \in [0.85, 1.15]$, volume ratio $v_{i_j}/v_{i_{j+1}} \in [0.85, 1.15]$ and have the same destination code.

This selective swapping reduces unloading complications by maintaining localized item grouping per customer or warehouse. A priority-based swap is conditional and is applied only when the dataset contains priority items, reinforcing the need for adaptability in our recursive heuristic, which is crucial for evolving solutions across different iterations and algorithm instances.

c) General considerations in the loading stages: Regarding packet insertions Fig. 2 demonstrates the adaptation of the Extreme Points Strategy (EPS) from [6], which is commonly used in packing algorithms. This method involves initializing the first packet at the Base Left Front (BLF) of an empty container c at $(0, 0, 0)$ and subsequently inserting packets at extreme points created by earlier placements.

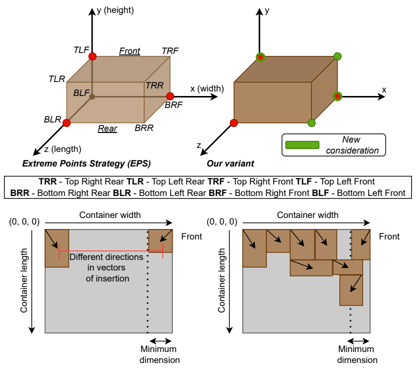
y (height)
TLF
Front
TRF
TLR
TRR
BLF
x (width)
BRF
BLR
Rear
BRR
z (length)
Extreme Points Strategy (EPS)
y
x
z
New consideration
Our variant
TRR - Top Right Rear TLR - Top Left Rear TRF - Top Right Front TLF - Top Left Front
BRR - Bottom Right Rear BLR - Bottom Left Rear BRF - Bottom Right Front BLF - Bottom Left Front
(0, 0, 0)
Container width
Front
Container length
Different directions in vectors of insertion
Minimum dimension
(0, 0, 0)
Container width
Front
Container length
Minimum dimension

Fig. 2. Extreme Points Modification.

In the original EPS, each new packet insertion at the BLF of the container generates three new PPs: Base Left Rear (BLR), Top Left Front (TLF), and Bottom Right Rear (BRR). If the BLR or BRR lacks direct support, these points are projected onto the nearest vertical surfaces. Our adaptation introduces the concept of an insertion vector, illustrated in Figure 2, which projects from the corner of a packet at an extreme point (EP) to its mass center. Our contributions are:

- **Positive to Negative Insertion Mode:** Insertion vector direction changes from positive $\langle +x, +y, +z \rangle$ to negative $\langle -x, +y, +z \rangle$ if the PP is located within a certain width range, from $Th_{W_{min}}$ to the container's maximum width (c_W), adjusting the packing strategy based on the packet's position relative to the container's dimensions.
- **Height Considerations:** If the PP is near the container's top (c_H), subsequent packet insertions may not generate feasible upper EPs as TLF or Top Right Front (TRF).
- **Width Adjustments:** When the PP is within $Th_{W_{min}}$ but not near the container's top, the next insertion might generate a Bottom Right Rear (BRR) instead of a Bottom Right Front (BRF), promoting configurations that facilitate negative mode insertions.

Handling during loading Stacking constraint (R5) requires packets to be stacked from the heaviest at the base to lighter ones above, regardless of the presence of "fragile" packets. The vertical center of mass of the packed solution P_p must not exceed half the container's height (R2), and the mass center on the transverse and longitudinal axes should stay within specified limits from the geometric center (detailed in Section II). The base area of each new packet must cover at least 80% of the top area of the underlying packets during the "Load-1" (L1) stage and 75% in the "Load-2" (L2) stage, with an average coverage of 90%.

Our heuristic addresses constraints (R9) and (R10), which have not been addressed in the literature. Packets with different "productId" may be a standalone products or belong to a larger within P, denoted by its "subgroupId". A subgroup there may contain one or many packets with different "productId"s. The feasibility of P_p depends on the inclusion of all packets from each subgroup, which complies with complete shipment constraints. In the re-sorting, L1, and L2 stages, priority is given to unpacked items from subgroups with several items already packed, following the initial sorting and "Load base" (LB) stage. The priority under constraint (R3) is binary, with packets marked as 1 for priority and 0 otherwise. By the end of the processing, P_p will include all prioritized packets.

d) Load Base: The Load Base (LB) phase is designed to establish a stable foundational layer of packets, known as the base, where packets are laid flat without stacking. This base is crucial as it sets the stage for meeting the constraints in the subsequent loading phases and ultimately in achieving the desired final configuration. Initially, this approach replaced three distinct phases that were previously aimed at maximizing stacking, which often led to the propagation of errors in highly constrained environments. This revised method aligns better with successful strategies noted in full construction models such as those in [10], particularly under stringent constraints.

One of the significant advantages of the LB phase is the effective exploration of all potential combinations of the initial horizontal layer of packets within a reasonable timeframe. This is facilitated by subdividing the container into subzones, which is essential for managing the weight distribution constraints ((R2)) and adhering to the weight limit constraints ((R1)). The configuration of these subzones varies depending on the mode of transport and nature of the load, typically distributing the total legal weight limit equally among the four subzones. Both (R1) and (R2) must be satisfied for solution feasibility.

The process begins by placing the first packet from P_q, p_{q_0}, into a selected extreme potential point from set E, which is updated dynamically with every packet insertion. The insertion point $e_i \in E$ used for p_{q_0} was then removed, and new potential points were generated as $E_{p_{q_0}}$. In this phase, the algorithm assesses the feasibility of each insertion and evaluates it by using a detailed fitness function. The procedure involves:

1) Select the nearest potential point pp_e to the front of the container or, in a tie, closest to the container's width extremes to minimize unutilizable small gaps by consolidating them into larger, more useful spaces.
2) A partial insertion test for each packet in P_q across all feasible orientations amounting to $\sum_{i=0}^{i=|P_q|} p_{q_i} * O_{p_{q_i}}$ potential insertion checks.
3) If a packet cannot be feasibly inserted in any orientation, it returns to P_p. If a feasible orientation was found, the

insertion was scored using a fitness function.

4) The fitness function, detailed in Equation 2, integrates several factors:

$$F_{base} = \frac{w_{p_{q_0}}}{W_{Pmax}} \cdot \epsilon + \left(1 - \frac{z_{mc_{p_{q_0}}}}{c_L}\right) \cdot \zeta + prio_{p_{q_0}} \cdot \eta \quad (2)$$

where $w_{p_{q_0}}$ is the weight of the packet being inserted, W_{Pmax} is the maximum packet weight in the set, $z_{mc_{p_{q_0}}}$ is the z-coordinate of the packet center of mass, c_L is the container length, and $prio_{p_{q_0}}$ is the priority of the packet. The coefficients ϵ, ζ, and η are determined through the PSO process, reflecting the relative importance of the packet weight, its proximity to the front of the container, and its priority, respectively.

The hard constraints are meticulously adhered to, with the multi-drop ((R7)) constraint explicitly addressed in this phase. For multi-destination datasets, the base area is allocated proportionally to ensure fairness and minimize the Mean Absolute Percentage Error (MAPE) between the theoretical and actual packed areas, aiming for an MAPE of less than 15%.

e) **RE-SORTING***:* This phase employs the same sorting function as the initial sorting phase, but with different coefficients (γ, δ) instead of (α, β). Re-sorting the packets in P_q, results in a new order that prioritizes packets from subgroups that must be shipped completely, complying with the complete shipment constraint. Packets from P_p that belong to these subgroups and have corresponding items in P_q are given higher priority if they are not already prioritized. This adjustment directly influences the ability of the algorithm to produce a greater number of feasible orientations, as shown in Eq.(1).

f) **LOAD-1***:* The Load-1 (L1) phase builds on the foundational setup completed in the Load Base (LB) phase. Here, P_q consists of packets not yet packed and resorted, while P_p includes those positioned on the container's floor, which have generated potential points divided by destination into $E = E_{d_0} + E_{d_1} + E_{d_2}$. In contrast to LB, L1 attempts to insert each packet from P_q into every applicable potential point within the same destination, resulting in a substantially increased number of placement evaluations: $\sum_{i=0}^{i=|D|-1} p_{q_{d_i}} * E_{d_i}$.

The insertion process assesses the fit of each packet against all potential points for all applicable constraints, as in LB. Successful insertions were evaluated by using a fitness function to determine the best placement. This process also generates new potential points using the extreme points projection technique, as illustrated in Figure 3.

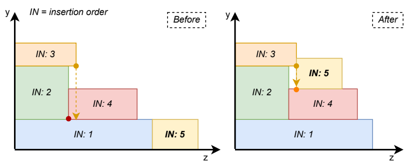

Fig. 3. Extreme Points Projection.

The fitness function for L1 optimizes the placement of each packet, defined as follows:

$$\begin{aligned} F_1 = {} & \left(1 - \frac{\|c_O - mc_{p_{q_0}}\|}{c_L}\right) \cdot \theta + S_{\text{cond}} \cdot \iota \\ & + \left(1 - \left|1 - \frac{A_{p_{q_0}}}{A_{p_{p_n}}}\right|\right) \cdot \kappa \\ & + \left(1 - \frac{\left(\frac{w_{p_{q_0}}}{W_{\text{Pmax}}} - 0.5\right)}{0.5} \cdot \frac{y_{mc_{p_{q_0}}}}{c_H}\right) \cdot \lambda \\ & + \text{prio}_{p_{q_0}} \cdot \mu \end{aligned} \quad (3)$$

- **Origin Attraction:** The first term prioritizes placing packets closer to the container's BLF, rather than just towards the front. The items are tightly packed in the most central and stable part of the container, enhancing stability and order, especially in cases where items vary greatly in size and shape.
- **Surrounding Condition:** S_{cond} measures the proximity of the top nearest items to $mc_{p_{q_0}}$, rewarding configurations in which these items belong to the same destination. This strategy minimizes the "unloading obstacles" by clustering items destined for the same location.
- **Area Condition:** This promotes stacking of packets that have similar top and bottom areas, optimizing the match between $A_{p_{q_0}}$ and $A_{p_{p_n}}$. A great match enhances the structural integrity, which is critical for the stacking stability as noted in [10].
- **Height-Weight Relation:** By adjusting the placement based on the weight of items relative to their height in the container, this term helps maintain a balanced weight distribution across the container's height, which is crucial for both safety and practical stacking strategies.
- **Priority Handling:** The final term ensures that high-priority items are placed preferentially.

After L1, most packets are packed, preparing for the next phase, Load-2 (L2), with the container's state updated and potential points set to continue refining the insertion process.

g) **LOAD-2***:* The Load-2 (L2) phase aims to maximize container volume utilization by placing previously discarded packets in unfilled spaces. These spaces often remain empty in the Load-1 (L1) phase because they do not explore all possible item orientations during the insertion process, unlike in the LB phase, owing to the computational scalability.

Before transitioning from L1 to L2, we transform the set of PPs based on the current state of the container and positions of the already loaded packets. This introduces a new set of PPs derived from items whose BRR falls below the threshold Th_{Wmin} (and their projections). This adjustment allows the consideration of new insertion possibilities that were not identified in L1, optimizing for potential orientations and placements that were previously overlooked owing to computational constraints in L1, which deals with a larger subset of packets in P_q.

In L2, the remaining packets in P_q, that have been re-sorted and rotated into feasible orientations, are considered for insertion. By exploring alternative orientations, L2 can effectively utilize spaces that L1 can not fill. The process in L2 follows the same workflow as L1 but adjusts the fitness function parameters $(\theta, \nu, \xi, \rho, \sigma)$ to better suit the different focus of this phase, which is to fill gaps left by the initial packing process as outlined in Formula 3.

IV. Validation

To train and test our algorithm, we used a data set of 20444 packets crawled from public webs of real-world companies (with 6727 unique dimensions and weight)[1], and containers of standard width $c_W = 2.45m$, height $c_H = 2.45m$, and length $c_L = 13.6m$). We created test scenarios by creating random partitions of the data set with a cumulative volume of approximately 110% of the container volume. These partitions ranged from 190 to 530 packets.

The design of our algorithm rendered 17 hyper-parameters to be tuned, which are the coefficients of the fitness equations described in the previous section. Searching for such a large-dimensional space makes an exhaustive search unfeasible. An analytical solution may not be possible because it will depend on scenario characteristics. Therefore we resorted to Particle Swarm Optimization [24] to obtain a set of parameters that maximize volume utilization in specific scenarios.

We configured the PSO with a population of 34 particles, initialized at random positions in the search space. The algorithm runs 200 iterations, far more than required as in our experiments we observed that the solution converges in the range of 100 to 150 iterations.

We defined four scenarios to run the optimization, with each set of packets that sum up the 115% of the container volume, that is, approximately 400 packets per scenario (70% unique). In addition, we ran the optimization in reduced versions of the scenarios, reducing the number of packets to 90% of the container volume. These scenarios are easier to compute because there are fewer packets, but the penalty for not inserting a packet is higher, as there are also fewer packages to substitute. We ran the optimization three times per scenario to avoid possible outliers.

Figure 4 shows the parameters computed for E1 scenario (others scenarios are similar), where each line represents one parameter set, and its color matches the quality of its solution. Base scenarios led to an average volume utilization of 79.29 %, with a std of 0.02. The reduced scenarios obtained worse results, as expected, with an average volume of 76.70 % and a std of 0.06.

The first insight obtained from this experiment is that small variations in parameters values do not have a strong impact on the solutions obtained (the algorithm does not seem to be very sensitive to the parameters). The green lines in the figure indicate that the parameter values may vary over a range and yield the same score. In addition, because the parameters are weights in the fitness functions and these functions are used to compare and rank the algorithm decisions, proportional sets of parameters will yield proportional fitness scores, and consequently, the solutions will be the same. Considering this fact, we compared the optimization results by measuring their cosine similarity, because that metric is not affected by the vectors' magnitude, only by their direction. The average cosine similarity in the case of the basic scenarios is 0.90, with std 0.06. Therefore it can be concluded that these sets of parameters are valid for general scenarios. The final values of the parameters are listed in Table I. However, comparing these solutions with those of the reduced scenarios, in some cases their similarity decreased to 0.73.

Sorting & Resorting				**Load 1 (L1)**				
α	β	γ	δ	θ	ι	κ	λ	μ
0.23	0.01	0.70	0.86	0.55	0.30	0.92	0.92	0.33
Base (LB)				**Load 2 (L2)**				
ϵ	ζ	η		ν	ξ	ρ	σ	τ
0.17	0.80	0.76		0.34	0.75	0.37	0.60	0.79

TABLE I
Obtained parameters

After the PSO findings, we performed 30 experiments with different characteristics using the weights in Table I. Their characteristics and results are presented in Table II, where the percentage of the container's used volume "% UV" achieves a mean of 74,06%. For all packets "P" the number of items packed is "nP" and those discarded are "nD". Although the instances are large, the computation time for 64 iterations (shown in the "t(s)" column) of the same instance averages 124s, which is in line with the requirements of logistics applications. "MD" and "mD", are the maximum and minimum physical dimensions in centimetres of the set of packets, with "MW" and "mW" showing the analogous for weights in grams. The percentage of over-volume of the instance with respect to the volume of the container is presented with "%oV", which means that for a certain scenario in which we have 80 cubic meters of container, the packages dataset will have a 96 cubic metres in total, showing how much the scenario is stressed. The number of destinations is "dst". In terms of heterogeneity, the percentage of unique packages that only consider dimensions is indicated in "%pU" with a mean value of 73%. Similarly, the percentages of subgroup and priority packages are indicated in "%pG" and "%pP", respectively and the number of unloading obstacles in "uO".

The importance of the constraints was noted in the volume utilization results. Our algorithm achieves 76,2% volume utilization with less than 1% of unloading obstacles in multi-drop scenarios, with all the soft and hard constraints satisfied if active. In addition, it can handle large and highly heterogeneous instances with a mean computation time per instance of 124 s on the machine mentioned above. Considering that this is an extreme scenario in terms of container size, our computation time is potentially feasible for the fast-paced operations of logistics companies.

As a brief analysis of the results, the decrease in volume utilization can be seen as the number of destinations increases because the algorithm deals with the unloading obstacles in a radical way. This behaviour was expected because the main focus area of the algorithm is transportation both inside the first and middle miles, and trucks usually do not make more than three stops. Taking this into account, if we only consider three destinations, the mean volume utilized is 77.8% and in the case of two destinations it is 79.6%.

This algorithm can compute highly constrained scenarios

[1]The dataset and script used to generate and personalise instances is available at(XXXXXXXX).

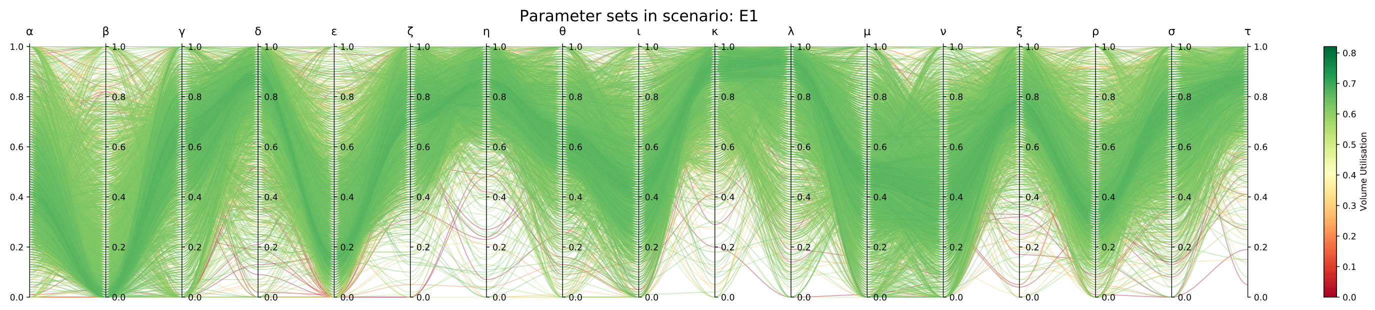

Fig. 4. Best parameter set of each PSO iteration for scenario E1.

i	%UV	P	nP	nD	t(s)	MD	mD	MW	mW	%oV	dst	%pG	%pU	%pP	uO
0	75.1	445	325	120	126	225	20	120	0.2	15	2	3	87	8	0
1	62	519	287	232	50	214	20	145	3.1	10	5	0	73	0	4
2	80.6	651	386	265	299	255	15	204	3.1	15	1	14	75	0	0
3	78.5	627	422	205	210	250	15	204	2.1	10	3	2	84	10	4
4	76.4	461	290	171	84	207	25	204	3	12	4	5	70	4	0
5	77.7	519	338	181	135	214	20	145	3.3	12	2	0	75	0	2
6	76.6	359	236	123	24	205	30	145	4.8	16	4	4	67	10	0
7	75.3	453	270	183	89	207	25	204	3.7	18	3	7	66	8	0
8	82.9	654	403	251	491	250	15	99	2	8	1	4	81	13	0
9	77.6	313	215	98	49	200	35	145	6.1	14	3	3	64	8	2
10	73.8	357	243	114	63	200	30	145	3.7	13	3	3	65	10	1
11	80.5	523	319	204	189	214	20	204	2.5	13	2	4	72	9	2
12	76.3	357	249	108	88	200	30	145	3.7	13	2	3	66	9	2
13	75.2	460	294	166	65	207	25	204	3.8	20	5	5	68	12	2
14	74.3	619	488	131	381	250	15	204	2.3	13	1	0	78	0	0
15	84.1	516	318	198	182	214	20	204	2.4	13	1	3	76	7	0
16	75	315	214	101	39	255	15	120	0.2	18	4	4	88	13	3
17	67.2	613	429	184	71	255	15	120	0.2	18	4	4	88	13	4
18	82.1	451	297	154	138	202	25	204	3	17	2	5	66	9	0
19	74.3	312	206	106	31	202	35	204	6.1	14	5	3	63	13	3
20	76.5	358	235	123	40	204	30	145	3.7	15	5	3	66	10	1
21	74.1	469	338	131	92	225	20	120	0.2	15	3	2	89	8	1
22	79.6	502	320	182	178	209	20	99	3.5	5	2	4	73	11	1
23	80.8	522	333	189	86	214	20	145	2.6	14	2	0	70	11	1
24	75	652	424	228	176	205	15	136	2.2	9	4	3	80	9	2
25	80.8	312	225	87	72	202	35	204	6.6	17	2	3	62	10	0
26	68.3	519	310	209	85	211	20	88	3.1	6	4	0	73	0	2
27	68.9	519	285	234	85	214	20	136	3.1	14	3	0	76	0	7
28	73.1	462	292	170	45	204	25	120	3.1	15	3	6	66	8	1
29	80.2	501	329	172	81	211	20	136	3.1	9	2	4	73	8	0

TABLE II
EXPERIMENTS RESULTS (YELLOW FOR HIGHEST VALUES AND BLUE FOR LOWEST VALUES IN COLUMN)

that no other than [10] have explored. Nevertheless, the size of the instances averages 79 packages with a maximum of 442 packages in their work whereas ours averages 478 packages and 651 as maximum. Regarding heterogeneity, in their case they explore instances with a maximum of 130 different types of packages whereas we have instances with up to 539 different package types.

However, we must consider that the weights of the fitness functions have not been specifically calculated for these scenarios, but rather for the best results for the full set of all scenarios presented in this article. In fact, some future lines of work would be to make matching classifications between weights and the main characteristics of scenarios, so that each type of scenario (i.e. having 200 vs 300 packages) has different weights in its fitness functions.

V. CONCLUSIONS

Our work was motivated by the existing opportunity to optimize cargo loading in complex scenarios. Our heuristic algorithm leverages and improves the previous proposals in the literature on CLP problems, and achieves good results in highly constrained scenarios, with a large number of packages and high heterogeneity on a large container, as in middle-mile logistics.

The structure, phases, and local functions exposed in this study propose a new way of optimizing volume while having up to ten types of constraints active. The inclusion of PSO enhances the rigor and precision of the local objective functions and connects them from a holistic perspective. Our experiments showed maximum volume utilization of a 84.1%

with an average of 76.70%.

The results obtained meet the objective of creating a heuristic capable of computing instances with more packages, especially with more unique items (high heterogeneity) than the state-of-the-art solutions in this field, while handling as many constraints as the best current work within an industry-feasible computing time, even with a more transparent and quantified intensity of the constraints present in the datasets.

References

[1] Sara Ali, António Galrão Ramos, Maria Antónia Carravilla, and José Fernando Oliveira. On-line three-dimensional packing problems: a review of off-line and on-line solution approaches. *Computers &amp; Industrial Engineering*, page 108122, March 2022.

[2] M. T. Alonso, R. Alvarez-Valdes, and F. Parreño. A GRASP algorithm for multi container loading problems with practical constraints. *4OR*, 18(1):49–72, January 2019.

[3] LJP Araújo and Plácido Rogério Pinheiro. Applying heuristics on integer linear programming for solving the container loading problem. In *42nd Brazilian Symposium of Operational Research*, pages 2088–2098, 2010.

[4] Mauro Maria Baldi, Guido Perboli, and Roberto Tadei. The three-dimensional knapsack problem with balancing constraints. *Applied Mathematics and Computation*, 218(19):9802–9818, June 2012.

[5] Tuğrul Bayraktar, Filiz Ersöz, and Cemalettin Kubat. Effects of memory and genetic operators on artificial bee colony algorithm for single container loading problem. *Applied Soft Computing*, 108:107462, September 2021.

[6] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Extreme point-based heuristics for three-dimensional bin packing. *INFORMS Journal on Computing*, 20(3):368–384, August 2008.

[7] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, November 2016.

[8] Oliviana Xavier do Nascimento, Thiago Alves de Queiroz, and Leonardo Junqueira. Practical constraints in the container loading problem: Comprehensive formulations and exact algorithm. *Computers &amp; Operations Research*, 128:105186, April 2021.

[9] Mikele Gajda, Alessio Trivella, Renata Mansini, and David Pisinger. An optimization approach for a complex real-life container loading problem. *SSRN*, 2020.

[10] Mikele Gajda, Alessio Trivella, Renata Mansini, and David Pisinger. An optimization approach for a complex real-life container loading problem. *Omega*, 107:102559, February 2022.

[11] Jr. H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[12] Y. Harrath. A three-stage layer-based heuristic to solve the 3d bin-packing problem under balancing constraint. *Journal of King Saud University-Computer and Information Sciences*, 34(8):6425–6431, 2022.

[13] Manuel Iori, Marco Locatelli, Mayron C. O. Moreira, and Tiago Silveira. Reactive GRASP-based algorithm for pallet building problem with visibility and contiguity constraints. In *Lecture Notes in Computer Science*, pages 651–665. Springer International Publishing, 2020.

[14] Rodolfo Ranck Júnior, Horacio Hideki Yanasse, Reinaldo Morabito, and Leonardo Junqueira. A hybrid approach for a multi-compartment container loading problem. *Expert Systems with Applications*, 137:471–492, December 2019.

[15] Sasithom Koonsintananan and Warangkhana Kimpan. Applied particle swarm optimization in solving container loading problem for logistics. In *2019 11th International Conference on Knowledge and Smart Technology (KST)*, pages 88–93. IEEE, January 2019.

[16] Deidson Vitorio Kurpel, Cassius Tadeu Scarpin, José Eduardo Pécora Junior, Cleder Marcos Schenekemberg, and Leandro C. Coelho. The exact solutions of several types of container loading problems. *European Journal of Operational Research*, 284(1):87–107, July 2020.

[17] Guido Perboli Mauro Maria Baldi and Roberto Tadei. Applying heuristics on integer linear programming for solving the container loading problem. *Applied Mathematics and Computation*, 4:9802–9818, 2012.

[18] Jonas Olsson, Torbjörn Larsson, and Nils-Hassan Quttineh. 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, February 2020.

[19] C. Paquay, M. Schyns, and S. Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23(1-2):187–213, July 2014.

[20] David Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, September 2002.

[21] António G. Ramos, Elsa Silva, and José F. Oliveira. A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, 266(3):1140–1152, May 2018.

[22] Elsa Silva, António G. Ramos, and José F. Oliveira. Load balance recovery for multi-drop distribution problems: A mixed integer linear programming approach. *Transportation Research Part B: Methodological*, 116:62–75, October 2018.

[23] Alessio Trivella and David Pisinger. The load-balanced multi-dimensional bin-packing problem. *Computers &amp; Operations Research*, 74:152–164, October 2016.

[24] H. Y. Tseng, P. H. Chu, H. C. Lu, and M. J. Tsai. Easy particle swarm optimization for nonlinear constrained optimization problems. *IEEE Access*, 9:124757–124767, 2021.

[25] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, December 2007.

[26] S. Yang, S. Song, S. Chu, R. Song, J. Cheng, Y. Li, and W. Zhang. Heuristics integrated deep reinforcement learning for online 3d bin packing. *IEEE Transactions on Automation Science and Engineering*, 2023.

[27] Defu Zhang, Yu Peng, and Stephen C.H. Leung. A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers & Operations Research*, 39(10):2267–2276, October 2012.

[28] H. Zhao, C. Zhu, X. Xu, H. Huang, and K. Xu. Learning practically feasible policies for online 3d bin packing. *Science China Information Sciences*, 65(1):112105, 2022.

[29] Xiaozhou Zhao, Julia A. Bennell, Tolga Bektaş, and Kath Dowsland. A comparative review of 3d container loading algorithms. *International Transactions in Operational Research*, 23(1-2):287–320, May 2014.