EURO Journal on Transportation and Logistics 14 (2025) 100167

Contents lists available at ScienceDirect

EURO Journal on Transportation and Logistics

journal homepage: www.sciencedirect.com/journal/euro-journal-on-transportation-and-logistics

A three-dimensional container loading algorithm for solving logistics packing problem

Ying Ma, Yu Zhou *, Qiwei Fang, Shengwei Xia, Wei Chen

School of Electronic, Electrical Engineering and Physics, Fujian University of Technology, Fuzhou, Fujian, 350118, China

ARTICLE INFO

Keywords:
Improved genetic algorithm
Placeable point strategy
Heuristic algorithm
Strongly heterogeneous cargos
Three-dimensional container loading algorithm

ABSTRACT

In addressing the challenge of large-scale packing of strongly heterogeneous cargo in logistics loading, this paper proposes a three-dimensional container loading algorithm that integrates a block-building heuristic, an optimized placement strategy, and an enhanced genetic algorithm. The proposed approach accounts for six practical constraints inherent in logistics packaging and establishes an optimization model aimed at maximizing container space utilization while adhering to permissible center of gravity deviations. First, a block construction heuristic is employed to preprocess the cargo by aggregating items into larger blocks, thereby significantly reducing problem size and computational complexity. Subsequently, an improved placement strategy, combined with a genetic algorithm, is applied to identify an initial feasible packing layout. Finally, a simulated annealing operator is introduced to further refine the solution through local optimization, thereby obtaining an enhanced loading solutions with improved loading efficiency. Through comparative experiments with other leading algorithms using public datasets, the results demonstrate that the proposed algorithm performs on par with existing methods in solving the three-dimensional packing problem for strongly heterogeneous items while satisfying multiple constraints. It is capable of providing high-quality loading solutions for logistics enterprises and offers valuable insights for addressing more realistic container loading problems in future research.

1. Introduction

Cargo loading constitutes a critical stage in the logistics process, directly influencing the efficiency and cost of transportation operations. Through rational loading algorithms and layout planning, it is possible to maximize space utilization, reduce the empty-load rate, and furthermore, contribute to lower vehicle carbon emissions, thereby supporting sustainable development objectives. Essentially, logistics loading can be formulated as a three-dimensional container loading problem (3D-CLP). Numerous scholars have conducted extensive research in this field, exploring various algorithmic approaches to address container loading challenges. For example, Ramos AG et al. (Ramos et al., 2018) and Davies Ap et al. (Davies and Bischoff, 1999) pursued load balancing of loaded containers, Harrath Youssef (2021) pursued the minimum number of containers used, and Goldberg N et al. (Goldberg and Karhi, 2019) studied the problem of online packing.

After decades of research, numerous optimization algorithms have been developed to address the three-dimensional container loading problem (3D-CLP). These approaches can be broadly categorized into three groups: exact methods, traditional heuristics, and meta-heuristics. Exact methods aim to compute the theoretically optimal solution for a given problem instance (Delorme and Wagenaar, 2024; Nascimento et al., 2021). However, due to the NP-Hard nature of 3D-CLP, the computational complexity of exact algorithms increases exponentially with problem size. Traditional heuristic methods rely on human-designed rules and prior knowledge to maximize container space utilization (Araya et al., 2017). In contrast, meta-heuristic algorithms incorporate stochastic processes into their search strategies to enhance convergence, thereby achieving satisfactory solutions within acceptable time frames (Bayraktar et al., 2021; Dereli and Das, 2011). Recent studies have indicated that hybrid approaches can further improve algorithmic performance. Consequently, state-of-the-art packing algorithms increasingly adopt hybrid meta-heuristics or combined strategies. For instance, Zhang et al. (2022) proposed a multi-strategy hybrid heuristic algorithm that employs a probabilistic model to select between greedy stacking and ant colony stacking methods, applied to the single-container weakly heterogeneous three-dimensional packing problem. Similarly, Yang et al. (2024) developed a two-layer heuristic

* Corresponding author.
E-mail address: 2231905020@smail.fjut.edu.cn (Y. Zhou).

https://doi.org/10.1016/j.ejtl.2025.100167
Received 15 June 2025; Received in revised form 14 September 2025; Accepted 3 November 2025
Available online 4 November 2025

algorithm consisting of an outer heuristic framework for determining container types and an inner constructive heuristic for generating efficient solutions to 3D-CLP.

In addition to the algorithms described above, numerous scholars have developed high-performance three-dimensional packing algorithms tailored to the practical characteristics of the problem. Examples include: Parreno et al. (Parreño et al., 2010) developed a novel heuristic algorithm based on variable neighborhood search (VNS) to address the container loading problem. The fundamental principle of VNS involves the systematic alteration of predefined neighborhood structures during the search process to escape local optima. A major contribution and key innovation of their work lies in the design of a set of sophisticated and effective neighborhood structures specifically tailored for the CLP, which are crucial to the high efficiency of the VNS algorithm. Experimental results demonstrate that the proposed algorithm exhibits strong performance in solving the CLP. Junqueira et al. (2012a) proposed a mixed-integer linear programming model for the container loading problem that incorporates both vertical and horizontal cargo stability constraints, as well as load-bearing strength considerations, including fragility of items. The study primarily designed an algorithmic framework integrating layered decomposition and mathematical programming. This framework employs a MIP solver to address complex combinatorial optimization and constraint satisfaction problems, placing greater emphasis on obtaining high-quality, physically feasible solutions rather than pursuing pure mathematical optimality. The proposed model helps motivate future research exploring decomposition methods, relaxation techniques, heuristic approaches, and other strategies to tackle more realistic container loading problems. Junqueira et al. (2012b) present approaches based on a mixed integer linear programming model (MIP) for the problem of packing rectangular boxes into a container or truck, considering multi-drop constraints. The study primarily introduces a decomposition-based recursive solution framework. This framework employs a strategy of sequential inward loading (by drop-off point) starting from the container door, effectively decomposing a complex global problem into a series of more manageable subproblems, thereby skillfully mitigating the computational complexity. Although rooted in mathematical programming, the decomposition strategy enables the generation of high-quality, strictly feasible solutions—i.e., those fully compliant with LIFO constraints—for real-world scale instances. The proposed models and methods can provide impetus for future research addressing integrated vehicle routing and container loading problems.

Instances of 3D-CLP may be composed of many types of boxes (known as strongly heterogeneous instances), which is a common scenario for courier services; a few types of boxes (weakly heterogeneous), such as when batches of cargo from a warehouse are sent to retailers; or entirely of one type of box (homogeneous), a situation that is prevalent for cargo coming off an assembly line (Zhu and Lim, 2012). Under the improved typology proposed by Wäscher et al. (2007), the weakly heterogeneous CLP is classified as a three-dimensional rectangular single large object placement problem (3D- SLOPP), while the strongly heterogeneous CLP is classified as a single knapsack problem (3D-SKP). This article is based on logistics loading, which usually involves a wide variety of cargos, and different cargos have their own placement restrictions (fragile cargo, etc.), vehicle transportation safety, etc., so what we want to solve is the 3D-SKP of strongly heterogeneous cargo with a variety of practical constraints. Although there are already a number of packing algorithms that report high container volume utilization, achieving satisfactory loading results under multiple constraints is still a challenging task. Existing research on the 3D-CLP has predominantly focused on fundamental constraints such as weight, volume, overlap, stability, and placement orientation, while relatively limited attention has been given to center-of-gravity deviation and cargo load-bearing constraints. The incorporation of these additional constraints significantly increases the computational complexity of the algorithm and presents considerable challenges in algorithmic design. Although a subset of studies has attempted to integrate these two constraints—namely, center-of-gravity deviation and load-bearing capacity—the performance of the resulting algorithms remains inferior compared to those considering only basic constraints, particularly when handling strongly heterogeneous cargo types. Therefore, this study aims to address the following problem: in the context of logistics loading, with a diverse range of cargo types and under multiple practical constraints—including center-of-gravity deviation and load-bearing limitations—to design an efficient three-dimensional container loading algorithm capable of generating high-quality loading solutions.

Based on the aforementioned limitations in current research on the 3D-CLP, the optimization objectives of the algorithm proposed in this paper are to improve space utilization of loading plans and enhance computational efficiency, while ensuring vehicle stability and cargo safety. The primary contribution of this work lies in the development of a three-dimensional container loading algorithm that integrates and optimizes multiple heuristic loading strategies within a metaheuristic framework. This algorithm is capable of effectively solving the three-dimensional strongly heterogeneous knapsack problem (3D-SKP) in logistics under all practical constraints, while achieving computational performance comparable to that of three-dimensional container loading algorithms considering only basic constraints. Specifically, improvements have been made to existing block generation and placeable point strategies by incorporating additional evaluation criteria to facilitate more effective cargo combination and loading. Furthermore, the genetic algorithm has been enhanced through the introduction of strategies such as penalty functions and elitist retention, in addition to a simulated annealing operator that refines the initial solutions generated by the genetic algorithm to achieve higher-quality results. The solution process of the algorithm is as follows: first, a block generation heuristic is employed to reduce the scale of the cargo to be loaded, thereby effectively decreasing computational complexity; then, an improved placeable point strategy is combined with the enhanced genetic algorithm to produce a high-quality initial loading plan, which is further optimized locally using the simulated annealing operator to obtain a superior solution.

2. Three-dimensional container loading algorithm

2.1. 3D-CLP model and constraints

2.1.1. Problem model

The 3D-CLP for logistics loading is described as follows:

First, the three-dimensional container loading problem (3D-CLP) involves the optimal arrangement of a set of three-dimensional items within one or more three-dimensional containers—typically boxes or shipping containers—with the objective of minimizing the number of containers used or maximizing space utilization. This article focuses on the single-container loading scenario, wherein the aim is to maximize the space utilization rate within an individual container.

Secondly, the issue of vehicle stability following loading operations remains a critical consideration in logistics loading. Thus, it is essential to ensure that the vehicle's center of gravity remains within a safe range after cargo placement.

In order to reduce the complexity of the packing calculation, this paper treats both the regular cargo and the wagon as a three-dimensional cuboid, so the packing model can be simplified to: specify the container size as L, W, H,the dimensions of the cargo as l, w, h. The ultimate goal is to maximize the use of space in the cabin and minimize the shift of the center of gravity. The left-front-bottom corner point of the carriage is the origin, the bottom surface is XY plane, and the vertical bottom is the Z axis to establish a coordinate system. The specific model is shown in Fig. 1.

Accounting for all real-world constraints would render the packing problem intractable for modeling and solution. To balance computational feasibility with practical requirements, this study adopts the

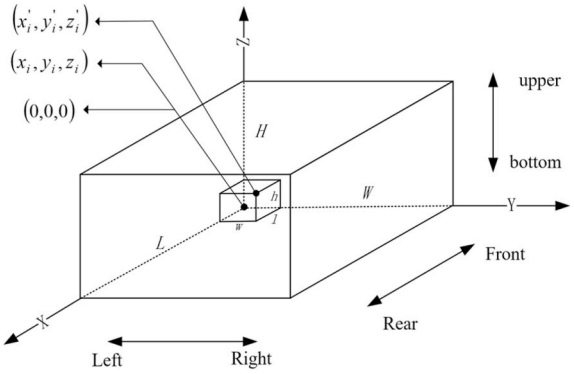

Fig. 1. Schematic diagram of the carriage model.

following assumptions based on common logistical scenarios.

① The regular cargo and container are cuboidal, and the cargo is smaller than the container;
② The mass of the cargo is evenly distributed, and the center of gravity of the cargos is the geometric center;
③ The cargos are allowed to be stacked within the load-bearing range;
④ The cargos are not deformed;
⑤ The cargos all arrive at the same destination.

η_i is a 0/1 variable, the cargos are loaded into the carriage $\eta_i = 1$, and the cargos are not loaded $\eta_i = 0$;

l_i, w_i, h_i, v_i, m_i are the length, width, height, volume and mass of cargo i;

$(x_i, y_i, z_i), (x_i', y_i', z_i')$ are the coordinates of the left-front-bottom corner point and the right-rear-upper corner point of cargo i respectively; (x_i, y_i, z_i) is also used to denote the coordinate position of cargo i in the container.

V、M are the volume and load capacity of the carriage respectively.

In general, a regular cargo can be put into the container with 6 different postures, as shown in Fig. 2. We can use 0 and 1 to mark whether the length, width and height of the cargos can be placed vertically. Mark 1 indicates that the corresponding dimensions of the cargo are allowed to be placed vertically inside the container, and mark 0 does the opposite. Table 1 shows the placeable posture of the cargo in different situations. We stipulate that $p(p = 1 \ldots 6)$ is used to represent the postures of the cargos, $t(t = 1 \ldots 7)$ is used to represent 7 different situations, and P represents the collection of the placeable postures of the cargos under different situations.

Table 1
The placeable posture of the cargos in different situations.

t	l	w	h	P
1	0	0	1	1,2
2	0	1	0	3,4
3	0	1	1	1,2,3,4
4	1	0	0	5,6
5	1	0	1	1,2,5,6
6	1	1	0	3,4,5,6
7	1	1	1	1,2,3,4,5,6

2.1.2. Constraints

In the real situation, the logistics packing problem also needs to consider the following practical constraints.

1) Volume and mass constraints. The total volume and mass of the cargo shall not exceed the maximum volume and load that the container can hold.

$$\sum_{i=1}^{n} m_i \eta_i \le M \tag{1}$$

$$\sum_{i=1}^{n} v_i \eta_i \le V \tag{2}$$

2) Cargo Attitude Constraints. Considering the diversity of cargos in reality, different cargo items will have different restrictions on the way they are placed, such as not being upside down, not being placed sideways, etc. Where $P_{i,t}(t = 1 \ldots 7)$ represents the placeable posture set of cargo i, and the different values of t correspond to different sets of placeable posture (see Table 1). For each cargo i, six variables $r_{i,p}(p = 1,2, \ldots,6)$ are introduced, corresponding to the six postures in Fig. 2, which represent the placement posture of its choice. Each variable $r_{i,p}$ can be valued as 0 or 1, and equation (3) indicates that each cargo must and can only choose one direction. Equation (4) means that all directions in the posture set P_{it} can be selected. Equation (5) represents the specific dimensions of the cargos in the three directions of X-axis, Y-axis and Z-axis under different attitudes. Take posture 1 as the initial posture, and the corresponding dimensions are l_i, w_i, h_i.

$$\sum_{p=1}^{6} r_{i,p} = 1 \tag{3}$$

$$\sum_{p \in P_{i,t}} r_{i,p} = 1 \tag{4}$$

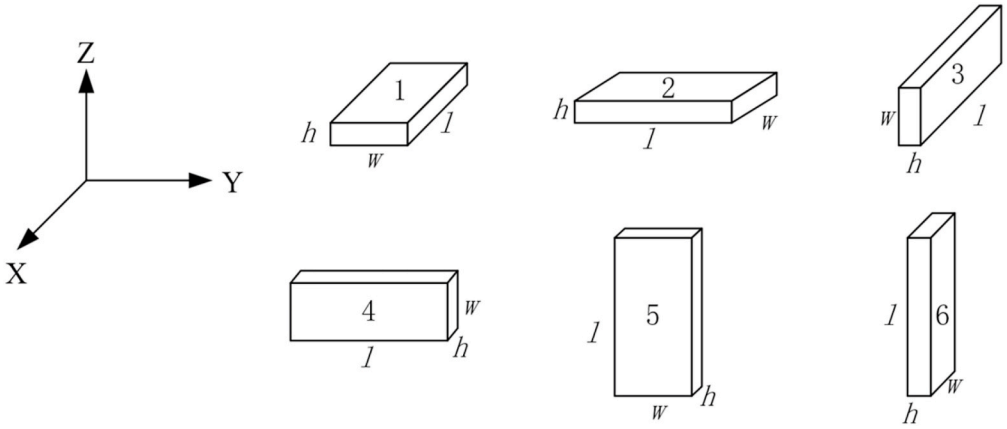

Fig. 2. Six postures for the placement of cargos.

$$\begin{cases} d_{ix} = l_i(r_{i,1} + r_{i,3}) + w_i(r_{i,2} + r_{i,6}) + h_i(r_{i,4} + r_{i,5}) \\ d_{iy} = l_i(r_{i,2} + r_{i,4}) + w_i(r_{i,1} + r_{i,5}) + h_i(r_{i,3} + r_{i,6}) \\ d_{iz} = l_i(r_{i,5} + r_{i,6}) + w_i(r_{i,3} + r_{i,4}) + h_i(r_{i,1} + r_{i,2}) \end{cases} \tag{5}$$

3) Cargo overlap constraints. No two cargo items in a stack can overlap each other. For any two cargo i and j, if they do not overlap on at least one axis, it can be proved that the two cargos do not overlap each other. (x_i, y_i, z_i), (x_j, y_j, z_j) represent the coordinates of cargo i and j in the container, respectively. (d_{ix}, d_{iy}, d_{iz}), (d_{jx}, d_{jy}, d_{jz}) represent the dimensions of the two cargos.

$$\begin{aligned} &\forall i \neq j, \left[(x_i + d_{ix} \leq x_j) \vee (x_j + d_{jx} \leq x_i)\right] \\ &\vee \left[\left(y_i + d_{iy} \leq y_j\right) \vee \left(y_j + d_{jy} \leq y_i\right)\right] \\ &\vee \left[(z_i + d_{iz} \leq z_j) \vee (z_j + d_{jz} \leq z_i)\right] \end{aligned} \tag{6}$$

4) Stability constraints. In order to ensure the stability of the stacking, cargo cannot be placed in the air, and more than half of the bottom surface of the upper cargo needs to be supported. If the current cargo is located at the bottom of the carriage, there is no need to consider stability. When two cargos are placed next to each other, the support stability needs to be calculated. In equation (7), c represents the cargo to be placed, c' is any of the cargo in the set of cargo that have been placed, A_c is the bottom area of cargo c, and S_z represents the collection of cargo located below the height z_c and may come into contact with c. θ represents the threshold ratio of the support area, with a value range of (0,1), which can be adjusted at any time according to actual needs.

$$\begin{cases} z_c = 0 \vee \sum_{c' \in S_z} \text{OverlapArea}(c, c') \geq \theta \cdot A_c \\ S_z = \{c' | z_{c'} + d_{c'z} = z_c\} \\ \text{OverlapArea}(c, c') = \max(0, \min(x_c + d_{cx}, x_{c'} + d_{c'x}) - \max(x_c, x_{c'})) \times \\ \max\left(0, \min\left(y_c + d_{cy}, y_{c'} + d_{c'y}\right) - \max(y_c, y_{c'})\right) \end{cases} \tag{7}$$

5) Load-bearing capacity constraints. Each cargo has its own load-bearing capacity, and it is important to consider the load-bearing capacity limit when stacking and placing it to avoid damage to the cargos. D_i represents the total mass of all cargos above cargo i, and R_i represents the maximum load-bearing capacity of cargo i. C_i is the set of cargo placed directly above cargo i, and m_j is the mass of cargo j.

$$\begin{cases} D_i \leq R_i, \forall i \\ D_i = \sum_{j \in C_i} (m_j + D_j) \end{cases} \tag{8}$$

6) Center of gravity constraints. In order to ensure the safety of the vehicle and the cargos during transportation, the center of gravity of the cargos after placement should be within the specified safety range. $[conx_1, conx_2]$ 、 $[cony_1, cony_2]$ are the safety intervals of the center of gravity in the direction of X and Y, respectively; $conz_1$ represents the maximum allowable height of the center of gravity in the Z-axis direction. $(gx_i$、gy_i、$gz_i)$ represents the coordinates of the geometric center point of the cargo i;

$$\begin{cases} \text{con}x_1 \leq \dfrac{\sum_{i=1}^{n} (m_i \eta_i g x_i)}{\sum_{i=1}^{n} m_i \eta_i} \leq \text{con}x_2 \\ \text{con}y_1 \leq \dfrac{\sum_{i=1}^{n} (m_i \eta_i g y_i)}{\sum_{i=1}^{n} m_i \eta_i} \leq \text{con}y_2 \\ \dfrac{\sum_{i=1}^{n} (m_i \eta_i g z_i)}{\sum_{i=1}^{n} m_i \eta_i} \leq \text{con}z_1 \end{cases} \tag{9}$$

A balanced center of gravity enhances both the safety and efficiency of vehicular transportation while also contributing to reduced fuel consumption. Consequently, optimizing the position of the vehicle's center of gravity after loading constitutes a key objective within the algorithm proposed in this study. Given that determining the actual vehicle's center of gravity involves numerous complex parameters, the present work focuses primarily on the center of gravity of the cargo compartment as a representative approximation.

We regard the unloaded wagon as a uniform density cuboid, and we can use equation (10) to calculate the position of the center of gravity of the carriage when it is empty, and set it as the ideal center of gravity position of the carriage, and use (*conx, cony, conz*) to represent the coordinates of the ideal center of gravity position.

$$\begin{cases} conx = (x_0 + L/2) \\ cony = (y_0 + W/2) \\ conz = (z_0 + H/2) \end{cases} \tag{10}$$

Where (x_0, y_0, z_0) is the origin coordinate (0,0,0), generally the left-front-bottom corner point of the container as the origin point (see Fig. 1).

2.2. Block generation heuristics

The block generation method is a constructive strategy commonly employed in three-dimensional packing problems to handle cargo combination. It enhances loading efficiency by consolidating multiple small-sized items into larger composite blocks. This approach is particularly suitable for scenarios involving multiple cargo sizes, as it effectively reduces the complexity of the packing task and lowers computational demands. Other block generation approaches include the algorithms developed by (Bortfeldt et al., 2003; Eley, 2002). There have also been wall-generation approaches (Bortfeldt and Gehring, 2001; Pisinger, 2002), where a container is filled using vertical layers (called walls), and layer-generation approaches (Bischoff and Ratcliff, 1995; Terno et al., 2000), where the container is filled from the bottom up using horizontal layers. Both a wall and a horizontal layer can be seen as special cases of a block, so these approaches can also be considered block generation approaches. Zhu et al. (2012) summarizes six key elements based on the common characteristics of all block generation methods:

K1: How to represent free space in a container;

K2: How to generate a list of blocks;

K3: How to choose free space;

K4: How to select blocks;

K5: How to put the selected block into the selected space and update the list of free spaces;

K6: What is an overall search strategy.

For K1, K3, K4, and K5, we adopt the improved placeable point strategy (see section 2.3). For K2, we use the method employed by Fanslau et al. (Fanslau and Bortfeldt, 2010), first combining cargo of the same size into simple blocks, and then combining these simple blocks into general blocks of the same size and general blocks of similar sizes.

For K6, we use the improved genetic algorithm (see section 2.4). Therefore, the main contribution of this paper is to combine the improved placeable point strategy and the improved genetic algorithm into the block generation method to solve the 3D-CLP. It will be simpler and more efficient than most existing algorithms.

Existing studies typically classify blocks into two categories: "simple blocks" and "generic blocks". A simple block is composed of cargo items of identical size and orientation, as illustrated in Fig. 3(a). In contrast, a generic block may incorporate items of varying sizes and different orientations, as shown in Fig. 3(b). Furthermore, generic blocks permit minor deviations in cargo dimensions; however, instantiation is permitted only when the ratio of the total actual volume of the cargo within the block to the volume of the block itself reaches a predefined threshold. This threshold is consistent with the value established in (Fanslau and Bortfeldt, 2010), set at 98 %. That is, a generic block is considered valid only if the cargo occupies at least 98 % of the block's volume. Notably, any simple block can also be regarded as a special case of a generic block.

2.2.1. Rules for the construction of blocks

The block construction process in this algorithm can be broadly categorized into three main stages: simple block generation, general block generation for items of identical size, and general block generation for items of similar sizes. The detailed procedure is described as follows.

(1) Initialize a cargo list named "cargos." Subsequently, all items to be loaded are grouped according to their dimensions, such that each group contains cargo items of identical size. Finally, all cargo groups are placed into the list "cargos" for further processing.
(2) If the quantity of cargos in the current group is greater than or equal to 2, (3) is executed. Otherwise, the cargos in the current group will be added to the new to-be list "combine_cargos".
(3) Simple blocks are constructed as follows. First, all cargo items within the current group are oriented identically along the X-, Y-, and Z-axes, and column-like structures are attempted in each of the three orthogonal directions, as illustrated in Fig. 4 (a), (b), and (c). If the dimensions of a column in any direction exceed those of the container, the number of items in that column is progressively reduced until the assembly fits within the container boundaries. This process continues until no further combinations of identical items are possible. The distances between each candidate column assembly and the corresponding container boundaries are then computed along each axis. The direction yielding the smallest gap is selected for the final column formation, thereby maximizing space utilization within the container. Finally, the combined items are added to the "combine_cargos", and the procedure is repeated until all cargo groups have been processed.
(4) The combination process for general blocks of identical sizes proceeds as follows: All simple blocks of identical dimensions (including individual cargo units) within "combine_cargos" are utilized to construct general blocks, enabling the previously formed "columns" to be further assembled into "layers". Three distinct layer construction methods are illustrated in Fig. 5 (a), (b), and (c). The process begins by designating the first item in "combine_cargos" as the 'current cargo'. Subsequently, a search is conducted among the remaining items in "combine_cargos" for cargo units whose length matches any two edges of the 'current cargo'. If the dimensions of the combined cargo assembly exceed the container size, the combination is considered unsuccessful, and the search continues with other items. Otherwise, the combination is deemed successful. The combined cargo units, along with any uncombinable items, are then added to the updated cargo list "final_cargos".
(5) Finally, general blocks are constructed by combining cargo of similar dimensions (within permissible deviations). The construction process is analogous to that described in (4), with the key distinction that the dimensions of the compared cargo and the current cargo need not be identical. Instead, a tolerance range for similar sizes is defined. Cargo meeting this similarity criterion are eligible for combination attempts with the current cargo. A combination is considered successful if the actual volume ratio of the merged cargo reaches 98 %, and the resulting assembly is added to the "new_cargos" list. If a combination attempt fails, the algorithm continues to evaluate the remaining cargo until all possible consolidation methods have been exhausted. Cargo that cannot be combined are also added to the "new_cargos" list. Finally, all items in "new_cargos" are sorted in descending order of volume, resulting in the final list of cargo to be loaded.

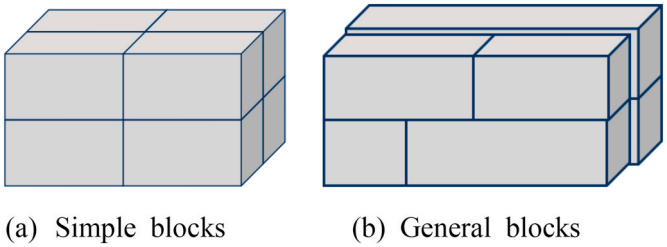

Fig. 3. Classification of blocks.

Fig. 6 (a), (b), and (c) illustrate the pseudocode for the three steps in the block generation algorithm, respectively.

2.3. Improved placeable point strategy

In the 3D packing algorithm, the placeable point strategy serves as a method for efficiently positioning cargo. It assists in identifying suitable placement locations by maintaining and dynamically updating a set of placeable points, adjusting both their positions and quantity in real time. The enhancement proposed in this article involves the introduction of a scoring rule for placeable points, enabling the selection of optimal placement positions for each cargo item. It is defined that the left-front-bottom corner point of each cargo serves as its placement point, representing the coordinate position of the cargo within the container. The origin point, located at the left-front-bottom corner of the container, serves as the initial placeable point (0, 0, 0). Suppose cargo i has coordinates of (0, 0, 0) and dimensions of (l, w, h). When cargo i is successfully placed in the container, three new placeable points $(l,0,0)$, $(0,w,0)$, $(0,0,h)$ are generated in the three directions X、Y、Z. If cargo i' continues to be placed on point $(0,w,0)$, assuming that cargo i' has coordinates of $(0,w,0)$ and size (w',h',d'), three new placeable points $(l',w,0)$, $(0,w+w',0)$, $(0,w,h')$ will continue to be generated. Because point $(0,w,0)$ is used, it is removed from the list of placeable points. Therefore, the current list of placeable points contains 5 placeable points: $(l,0,0)$, $(0,0,h)$, $(l',w,0)$, $(0,w+w',0)$, $(0,w,h')$. Fig. 7 is a schematic diagram of the placeable point strategy.

2.3.1. The placeable points are moved down

When employing the placeable point strategy during loading, a situation may arise wherein two cargo items are stacked such that the bottom area of the upper cargo exceeds that of the lower one, resulting in the generation of dangling placeable points by the upper cargo. Since cargo cannot be placed in mid-air, these suspended points cannot be used directly and must be shifted downward. Each dangling placeable point is projected vertically downward until it contacts either the top surface of another cargo or the bottom of the container, thereby generating a new, valid placeable point. The original suspended points are subsequently removed from the list, and the newly generated points

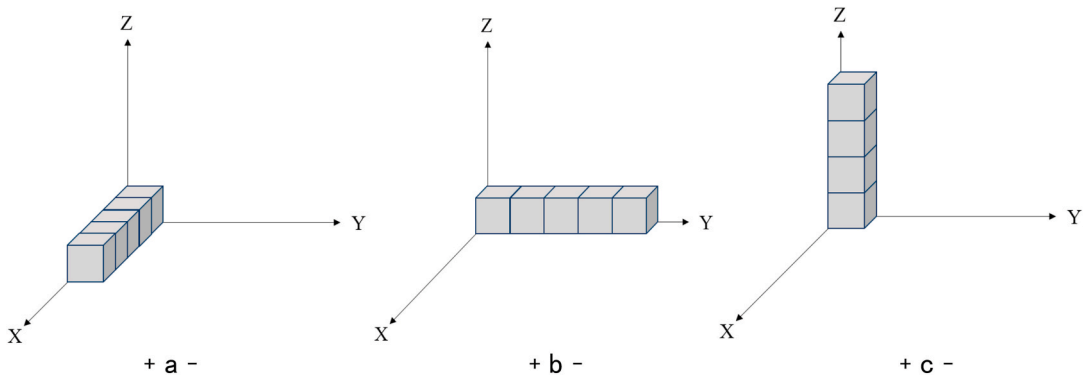

Fig. 4. There are three ways in which columns are constructed.

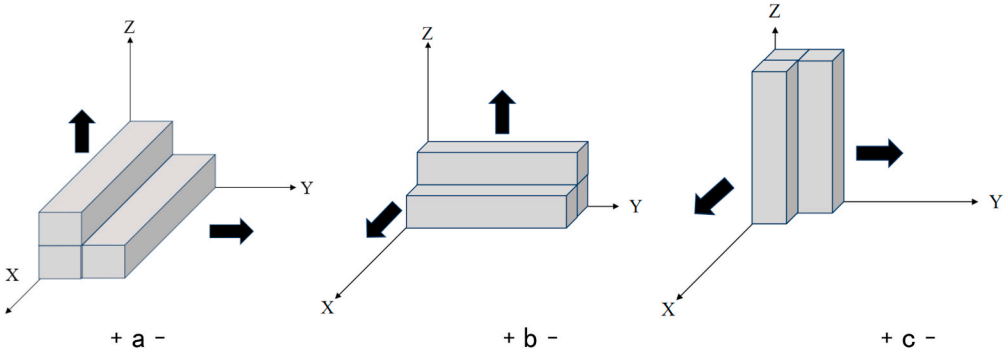

Fig. 5. There are three types of layers that are constructed (a) Simple block generation.

are incorporated. This process of downward projection is illustrated in Fig. 8.

2.3.2. *Placeable point scoring strategies*

In the placeable point strategy, a cargo item can often satisfy loading constraints at multiple placeable points simultaneously, meaning it can be placed at any of these feasible locations. However, randomly selecting one from numerous eligible placeable points does not guarantee placement optimality. Therefore, a scoring rule for placeable points is introduced. When multiple feasible points are available for a cargo, a placement score is computed for each point according to this rule. The point with the highest score is subsequently selected for placement. This approach ensures that the cargo is positioned at the optimal placeable point each time, thereby promoting more effective utilization of the remaining container space and enhancing overall space utilization.

The scoring rule is specifically based on the "cargo surface coverage ratio", which is implemented as follows: when a cargo item satisfies placement constraints at multiple placeable points simultaneously, it should be positioned at a point where it achieves greater contact with other cargo items or the container walls. A two-dimensional loading illustration in Fig. 9 demonstrates the placement of cargo a at different candidate points. As shown, there are four placeable points initially available for cargo a, though placement at point 4 is infeasible due to container size constraints. When placed at point 1, cargo a integrates more effectively into the remaining container space and exhibits a larger contact area with adjacent cargo and the container walls. In contrast, placement at points 2 and 3 results in reduced contact area and leads to fragmented gaps between cargo a and already placed items, which may hinder subsequent placements and reduce overall space utilization. Thus, placeable point 1 represents the optimal placement choice for cargo a.

In summary, for each candidate point, the total contact area between the cargo and its surroundings—including other cargo and the container walls—is computed. This value is then divided by the total surface area of the cargo itself to obtain a ratio, which serves as the score for that placeable point. The scoring function is defined as follows:

$$S(C,P)=\frac{A_{\text{contact}}(C,P)}{A_{\text{surface}}(C)} \tag{11}$$

Where $A_{\text{contact}}(C,P)$ is the total contact area of cargo C with the container boundary and other cargos at point P. $A_{\text{surface}}(C)$ is the total surface area of the cargo.

The specific placement of cargo is determined by the score assigned to each feasible placeable point. In cases where multiple points share the same score, the following tie-breaking rules are applied: preference is given to the point with the smallest x-coordinate value; if multiple points have the same x-coordinate, the point with the largest z-coordinate is selected. Prioritizing points with smaller x-coordinates encourages cargo to be stacked toward the front of the container, while selecting points with larger z-coordinates promotes the utilization of upper container space. The advantage of this rule is that it facilitates the formation of a more regular cargo stack at the front section of the container, thereby preserving the integrity of the remaining space in the rear. This approach helps ensure that subsequent cargo can be loaded more efficiently.

2.3.3. *Corner-first strategy*

The interior corners of a container are often challenging to fill due to constraints imposed by two adjacent walls. As a result, corner spaces frequently form gaps during the loading process that are difficult to utilize, leading to inefficient space usage. To address this issue, a "Corner-first" strategy is adopted, wherein the four bottom corners of the container are prioritized for loading. This approach ensures that corner spaces are utilized effectively from the outset. Preferential filling of container corners offers several advantages.

Algorithm 1 Block generation algorithm (Simple block)

```
Input: Initial cargo list cargos
Output: List of cargo after combination combined_cargos
Initialize the combined_cargos, cargo_groups, best_combination
Group cargos by size into cargo_groups
for each group in cargo_groups do
    count = size of group
    if count = 1 then
        Add cargo to combined_cargos
    else
        while count >= 2 do
            for factor from count down to 2 do
                for direction x, y, z do
                    Calculate combined dimensions for the direction
                    Check if combined dimensions fit in container
                    if valid and difference with container is minimal then
                        Update best_combination
                    end if
                end for
            end for
            if best_combination exists then
                Add best combination to combined_cargos
                Update count by subtracting factor
            else
                Break
            end if
        end while
        if count = 1 then
            Add remaining_cargo to combined_cargos
        end if
    end if
end for
```

(a) Simple block generation

Algorithm 2 Block generation algorithm (General block same size)

```
Input: Simple block cargo list combined_cargos
Output: List of cargo after combination final_cargos
Initialize the final_cargos, current_cargo, other_cargo
while combined_cargos is not empty do
    Set current_cargo as the first element of combined_cargos
    matched = True
    while matched do
        matched = False
        for each other_cargo in combined_cargos do
            for each placeable posture of other_cargo do
                Check if two dimensions match with current_cargo
                if valid combination then
                    Update current_cargo
                    Remove other_cargo from combined_cargos
                    matched = True
                    Break
                end if
            end for
            if matched then
                Break
            end if
        end for
    end while
    Add current_cargo to final_cargos
end while
Sort final cargos by volume in descending order
```

(b) General blocks of the same size are generated

Algorithm 3 Block generation algorithm (General block similar size)

```
Input: General block cargo list final_cargos
Output: List of cargo after combination new_cargos
Initialize the new_cargos, current_cargo, other_cargo
while final_cargos is not empty do
    Set current_cargo as the first element of final_cargos
    matched = False
    for each other_cargo in final_cargos do
        for each placeable posture of other_cargo do
            Check if two dimensions match with current_cargo
            if valid combination then
                Update current_cargo
                Remove other_cargo from final_cargos
                matched = True
                Break
            end if
        end for
        if matched then
            Break
        end if
    end for
    Add current_cargo to new_cargos
end while
Sort new_cargos by volume in descending order
```

(c) General blocks of the similar size are generated

Fig. 6. Block generation algorithm pseudocode.

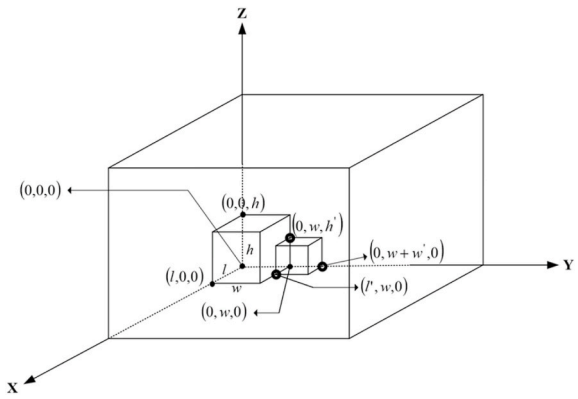

Fig. 7. Placeable point strategy.

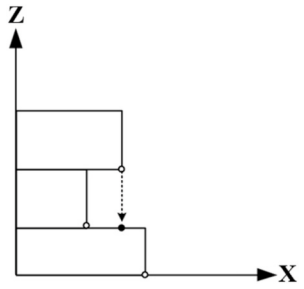

Fig. 8. The placeable points are moved down.

(1) It enhances the load-bearing capacity of the container base, reducing the risk of unstable or tilted cargo stacks.
(2) It encourages remaining free space to consolidate toward the central region of the container, thereby promoting spatial continuity and minimizing fragmentation.
(3) For subsequent loading steps, corner placements provide stable reference points, facilitating the construction of a more regular and tightly packed cargo arrangement, which ultimately improves overall space utilization.

The specific implementation of the corner-first strategy in the algorithm proceeds as follows: all cargo items in the list to be loaded are iterated sequentially, with priority given to placing them at the corner placeable points within the container. Corner loading follows a "first matching strategy", meaning that once a cargo can be successfully placed at one of the current corner points, it is placed immediately without further evaluation of alternative points. This approach enhances computational efficiency. Furthermore, placeable point scoring is omitted during corner loading, as the objective is solely to occupy the four corner spaces, rendering score-based filtering unnecessary. After the four bottom corners of the container have been filled, the placeable points are updated based on the dimensions and coordinates of the successfully loaded cargo. Subsequent cargo items are then loaded sequentially according to the standard placeable point placement strategy.

In this article, the coordinate point is set to the left-front-bottom corner of the cargos, so the placeable points in the corners of the container need to be dynamically changed according to the dimensions of each cargo. We need to calculate the corresponding placeable points according to the dimensions of the current cargo, and then use these placeable points to place the current cargo. We can name the placeable points corresponding to the four corners of the bottom of the container as the left front point, the left back point, the right front point, and the right back point. The left front point $P_{\text{left-front}}$ is the initial placeable point (0,0,0). Assuming the dimensions of cargo i are (l_i, w_i, h_i), the

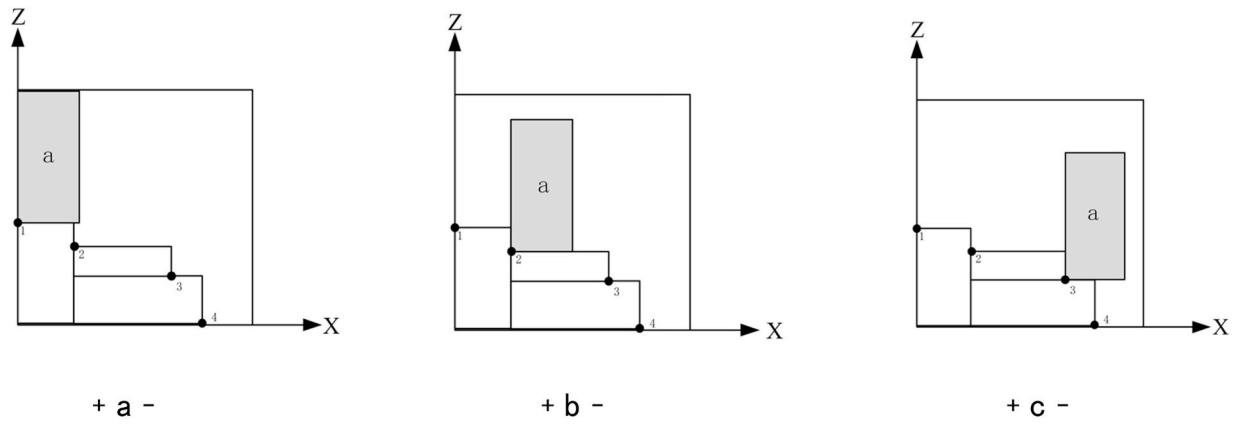

Fig. 9. Different placements of cargo *a*.

coordinates of the remaining corner points resulting from the dimensions of cargo *i* are: $P_{\text{left-rear}} = (L - l_i, 0, 0)$, $P_{\text{right-front}} = (l_i, W - w_i, 0)$, $P_{\text{right-rear}} = (L - l_i, W - w_i, 0)$,*L, W, H* represent the length, width, and height of the container. Fig. 10 below is a schematic diagram of the corner placeable point

2.4. Improving the genetic algorithm

Genetic algorithm (GA) is a search algorithm inspired by natural evolution (Goldberg and Holland, 1988) and although it was proposed decades ago, it is still used in many studies until now because it performs very well in many optimization problems (Lee, 2018). The genetic algorithm exhibits rapid and stochastic global search capabilities, enabling it to identify high-quality packing solutions within a short timeframe. In logistics loading scenarios involving large-scale, strongly heterogeneous cargo, the genetic algorithm can compute a near-optimal solution that meets practical requirements within a specified duration. However, due to the inherently stochastic nature of its search process, the algorithm is prone to issues such as slow convergence and a tendency to become trapped in local optima, which limits further refinement of solution quality. To address these limitations, we introduce the "elite retention strategy", "penalty function", and "simulated annealing operator" for algorithmic enhancement.

2.4.1. Encoding and decoding

In the packing algorithm, it is generally necessary to consider three key factors: placement order, placement posture, and placement position. Since logistics loading typically constitutes an offline loading scenario—meaning all cargo information is known prior to loading—the cargo items to be loaded are sorted in descending order of volume. This approach aligns with practical loading logic, which favors placing larger items first to secure more contiguous space within the container, followed by smaller items to fill residual gaps. Prioritizing smaller items could result in larger ones failing to fit due to insufficient available space, thereby reducing overall space utilization. Moreover, loading larger items first can provide a stable foundation for subsequently loaded items, enhancing the stability of the overall loading arrangement. Although volume-based sorting does not guarantee an optimal loading solution, this strategy generally yields satisfactory results while simplifying the algorithmic logic and improving computational efficiency.

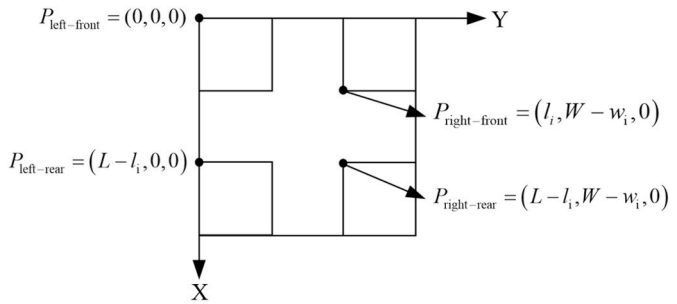

Fig. 10. Dynamically placeable points in the corners of containers.

The final list "new_cargos", computed in Section 2.2 (Block Generation Heuristics), serves as the list of cargo to be loaded. It includes both composite cargo blocks and non-combinable cargo items, all sorted in descending order by volume. During encoding, only placement posture is considered, utilizing real-number encoding where the integers 1 through 6 represent the six possible placement postures of the cargo, as illustrated in Fig. 2. Each individual is denoted by $S = (s_1, s_2, \ldots, s_n)$. Where $s_1 \ldots s_n$ represents the posture of cargo 1 to cargo *n* in the list "new_cargos", a new individual is formed each time by randomly assigning the permissive posture to all the cargo in the list "new_cargos". For example, individual *i* might be S_i = (1,3,5,4,2,6, …,2,3,6), individual *j* might be S_j = (4,6,2,5,2,6, …,1,3,5). The placement of the cargo is processed at the time of decoding, and the actual packing operation is carried out according to the improved placeable point strategy.

2.4.2. Evaluation function and penalty function

The fitness value directly reflects the quality of the current solution. Given that the primary objective of the algorithm proposed in this paper is to maximize container space utilization, the total volume of the loaded cargo in the current loading scheme is employed as the main component of the fitness value. This component is subsequently integrated with a penalty function to compute the final fitness. The v_i in equation (12) represents the volume of the cargo *i* and *E* represents the total volume of cargo loaded inside the container.

$$E = \sum_{i=1}^{n} \eta_i v_i \tag{12}$$

The role of the penalty function is to impose penalties on solutions that violate constraints by incorporating a penalty term into the fitness value, thereby guiding the algorithm toward feasible regions of the solution space. In this algorithm, the relevant penalty functions are primarily designed to address the center of gravity constraint, load-bearing capacity constraint, and stability constraint. *B* in each of the following equations represents the penalty function. Eq. (13) is the center of gravity shift penalty, Eq. (14) is the load-bearing capacity penalty, Eq. (15) is the stability penalty, and Eq. (16) is the evaluation function.

$$\begin{cases} B_1 = \left| \dfrac{\sum_{i=1}^{n} m_i \eta_i g x_i}{\sum_{i=1}^{n} m_i \eta_i} - conx \right| \\ B_2 = \left| \dfrac{\sum_{i=1}^{n} m_i \eta_i g y_i}{\sum_{i=1}^{n} m_i \eta_i} - cony \right| \\ B_3 = \dfrac{\sum_{i=1}^{n} m_i \eta_i g z_i}{\sum_{i=1}^{n} m_i \eta_i} - conz \end{cases} \tag{13}$$

$$B_4 = \begin{cases} 1 & D_i \le R_i \\ 0 & otherwise \end{cases} \tag{14}$$

$$B_5 = \begin{cases} 1 & \sum_{c' \in S_z} \text{OverlapArea}(c, c') \ge 0.5 \cdot A_c \\ 0 & otherwise \end{cases} \tag{15}$$

The final evaluation function is:

$$F_i' = (E_i - (B_1 + B_2 + B_3)) \cdot B_4 \cdot B_5 \tag{16}$$

2.4.3. GA manipulation process

(1) Selection

The selection phase employs a combination of roulette wheel selection and an elite retention strategy. Roulette Wheel Selection is a probability-based selection mechanism widely utilized in the individual selection process of Genetic Algorithms (GA). Its core principle is that the probability of an individual being selected is directly proportional to its fitness value—that is, individuals with higher fitness have a greater chance of selection, while those with lower fitness retain a non-zero probability of being chosen. Equation (17) represents the probability G_i that individual i is selected, where F_i' is the fitness value of the individual i, N is the total number of individuals in the population, and $\sum_{j=1}^{N} F_j'$ is the sum of the fitness of the population.

$$G_i = \frac{F_i'}{\sum_{j=1}^{N} F_j'} \tag{17}$$

Additionally, due to the stochastic nature of genetic algorithms, there is a risk that high-quality individuals may not be preserved. To prevent the loss of the current optimal solution and to enhance the convergence and stability of the algorithm, an elite retention strategy is adopted to emulate the principle of "survival of the fittest." This ensures that the best individuals from each generation are retained and directly propagated to the next generation.

(2) Crossover

The crossover operation employs a multi-point crossover strategy. This involves selecting multiple crossover points from two parent individuals and generating two offspring by exchanging segments of their genetic material. Two integers are randomly generated within the range [1, n] to serve as crossover points, where n denotes the length of the parent individual. By swapping the gene segments between these two points, Child 1 and Child 2 are produced. The algorithm assigns a 70 % probability for the crossover operation to be executed; if crossover does not occur, the parent individuals are retained unchanged. Fig. 11 provides a schematic illustration of this process, with points A and B indicating the crossover locations.

(3) Mutation

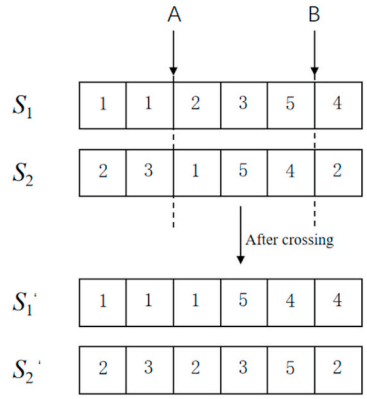

Fig. 11. Schematic diagram of cross-operation.

The mutation phase employs a strategy incorporating random mutation and a dynamic mutation rate. Random mutation involves stochastically determining both the number and positions of mutations within an individual. At each selected gene position, a new posture is randomly chosen from the allowable cargo postures (1–6), with the possibility of retaining the original posture. The dynamic mutation rate strategy is implemented as follows: if individual fitness shows no improvement over an extended period, the mutation rate is gradually increased, up to a maximum of 50 %; conversely, when fitness improvement occurs, the mutation rate is reduced, with a minimum threshold of 5 %. Fig. 12 provides a schematic illustration of the mutation process.

(4) Simulated Annealing Operator

Simulated Annealing (SA) is a probabilistic global optimization algorithm inspired by the thermodynamic annealing process in metals. It was initially proposed by Kirkpatrick et al. (1983) in 1983, drawing from the physical principle whereby metals attain thermodynamic equilibrium through heating and controlled cooling. By incorporating a structured stochastic search and a gradual cooling mechanism, the Simulated Annealing algorithm efficiently explores solutions in the vicinity of the current state, exhibits a strong capability to escape local optima, and maintains a balance between exploration and exploitation by accepting suboptimal solutions based on a probabilistic criterion.

In this paper, Simulated Annealing is employed as a local optimization operator, primarily aimed at refining the current optimal solution obtained from the Genetic Algorithm through localized search. To preserve the feasibility of the initial solution, a stochastic generation method is used to produce neighborhood solutions—specifically, by randomly altering the placement posture of a single cargo within the individual. This approach ensures minimal perturbations in the solution space, facilitating exploration through limited random variations to further optimize the initial solution. Fig. 13 presents the pseudocode for the Simulated Annealing-related procedures.

To validate the effectiveness of simulated annealing within the algorithm, a comparative experiment was conducted between a genetic

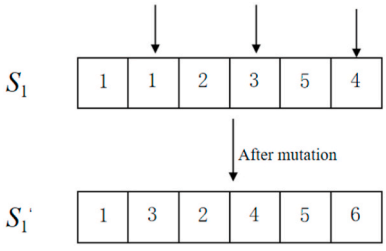

Fig. 12. Schematic diagram of the mutation operation.

Algorithm 3 Simulated Annealing for 3D-CLP

```
Input: Cargos, Container, Initial_Solution, Initial_Temperature, Cooling_Rate, Min_Temperature
Output: best_solution, best_fitness
Initialize:
current_solution ← Initial_Solution
best_solution ← Calculate_Fitness(current_solution)
best_solution ← current_solution
best_fitness ← current_fitness
temperature ← Initial_Temperature
while temperature > Min_Temperature do
    neighbor ← Generate_Neighbor(current_solution)
    neighbor_fitness ← Calculate_Fitness(neighbor)
    △← (neighbor_fitness - current_fitness)
    if △>0 then
        current_solution ← neighbor
        current_fitness ← neighbor_fitness
    else if Random(0,1) < exp(△/temperature) then
        current_solution ← neighbor
        current_fitness ← neighbor_fitness
    end if
    if current_fitness > best_fitness then
        best_solution ← current_solution
        best_fitness ← current_fitness
    end if
        temperature ← temperature×Cooling_Rate
end while
return (best_solution, best_fitness)
```

Fig. 13. Simulated annealing algorithm pseudocode.

algorithm enhanced with a simulated annealing operator and a baseline genetic algorithm without it. The test data were selected from the BR instances (BR10 to BR15) in reference (Bischoff and Ratcliff, 1995), which comprise six distinct heterogeneity types. For each category, the first ten sets of goods within the group were utilized for experimentation. The parameter configurations for both algorithms were consistent with the settings described in Section 3(Experimental results and analysis)below. The final results for each group were obtained by averaging the outcomes from ten independent computational runs.

Table 2 presents the detailed comparative results. As can be observed, the genetic algorithm enhanced with the simulated annealing operator consistently outperforms its counterpart across all experimental groups. This improvement can be attributed to the fact that the simulated annealing operator in this study further refines the initial optimal solution obtained by the genetic algorithm. Consequently, given a sufficient number of iterations, simulated annealing generally yields superior results. These findings demonstrate the significant role of simulated annealing within the genetic algorithm proposed in this paper.

2.4.4. The specific process of 3D container loading algorithm

Fig. 14 shows the overall process of the 3D container loading algorithm in this paper, and the specific steps are as follows.

Step 1: Enter basic information such as the size of the vehicle and the cargo into the program, and set the relevant parameters of the algorithm.

Table 2
Algorithm comparison results 1.

Test set	GA	GA + SA
BR10	91.12 %	91.55 %
BR11	91.04 %	91.38 %
BR12	90.23 %	90.58 %
BR13	90.11 %	90.35 %
BR14	89.27 %	89.98 %
BR15	88.59 %	89.78 %
average	90.06 %	90.60 %

Step 2: The cargo to be loaded are combined by using the block generation heuristic algorithm, and the combined cargos are added back to the list to be loaded and sorted according to the volume of the cargos.
Step 3: Decode according to an improved placeable point strategy and perform cargo packing operations.
Step 4 Judges whether the constraints such as quality, volume, stability of cargos and non-overlap are met according to formula (1) ~ formula (7), and if satisfied, turn step 5. Otherwise, go to step 9.
Step 5 Calculates the fitness function with the goal of maximizing the space utilization and minimizing the center of gravity offset.
Step 6 The simulated annealing operator was used to optimize the optimal individuals in the current initial population.
Step 7: Make a selection using the roulette strategy.
Step 8: If the individual is good enough, it will be directly copied to the next generation. On the contrary, multi-point crossover and random mutation are used to achieve crossover and mutation.
Step 9 Combining elite retention strategies with solutions from improved genetic algorithms to generate a new generation of populations
Step 10 performs the next cycle to determine whether the termination condition is met. If satisfied, the best loading scheme will be output. If not, return to Step 6.

Fig. 15 illustrates the relevant pseudocode for the genetic algorithm.

3. Experimental results and analysis

The experimental scenarios in this paper are based on the following hardware and software environments: AMD Ryzen 7 7735HS with Radeon Graphics 3.20 GHz. Operating system: Windows 11. Algorithm implementation language: Python 3.9. The setting of genetic algorithm parameters is more determined by the experience of predecessors and the actual calculation situation. The parameters in this paper are derived from multiple experimental attempts, and a general parameter range is established based on various objectives. Finally, the optimal parameter combination is selected, allowing the algorithm to compute a solution that meets the requirements within a reasonable time. The parameters are set as follows: the number of populations of genetic algorithm is 100, the number of iterations is 100, the crossover probability is 0.7, the variation probability is 0.05–0.5, the initial temperature of simulated annealing is 100 °C, the minimum temperature is 1 °C, and the cooling rate is 0.9.

Table 3 shows the time consumption of the block combination method in the overall algorithm. The test cases are selected from the first set of goods in each group of BR1 to BR15 from the BR standard instances (Bischoff and Ratcliff, 1995). The average running time is obtained after executing each case 10 times. Here, combination time represents the average time consumed by the block combination method over 10 computations, while total time denotes the overall average time consumed by the entire algorithm after 10 executions. The number refers to the total number of cargos in the current group.

As can be observed from Table 3, the total time of the algorithm generally increases with the growing heterogeneity of the cargos. In contrast, the combination time by the block combination method is very short, averaging less than 0.1s, which is almost negligible in the total computational time and remains largely unaffected by the heterogeneity of the cargos.

3.1. Algorithm comparison

To evaluate the performance of the proposed algorithm, experiments were conducted using the Bischoff & Ratcliff standard instances provided in (Bischoff and Ratcliff, 1995). This dataset comprises 15 categories, designated BR1 to BR15, each containing 100 distinct problems. The number of cargo types per category ranges from 3 to 100. Since the

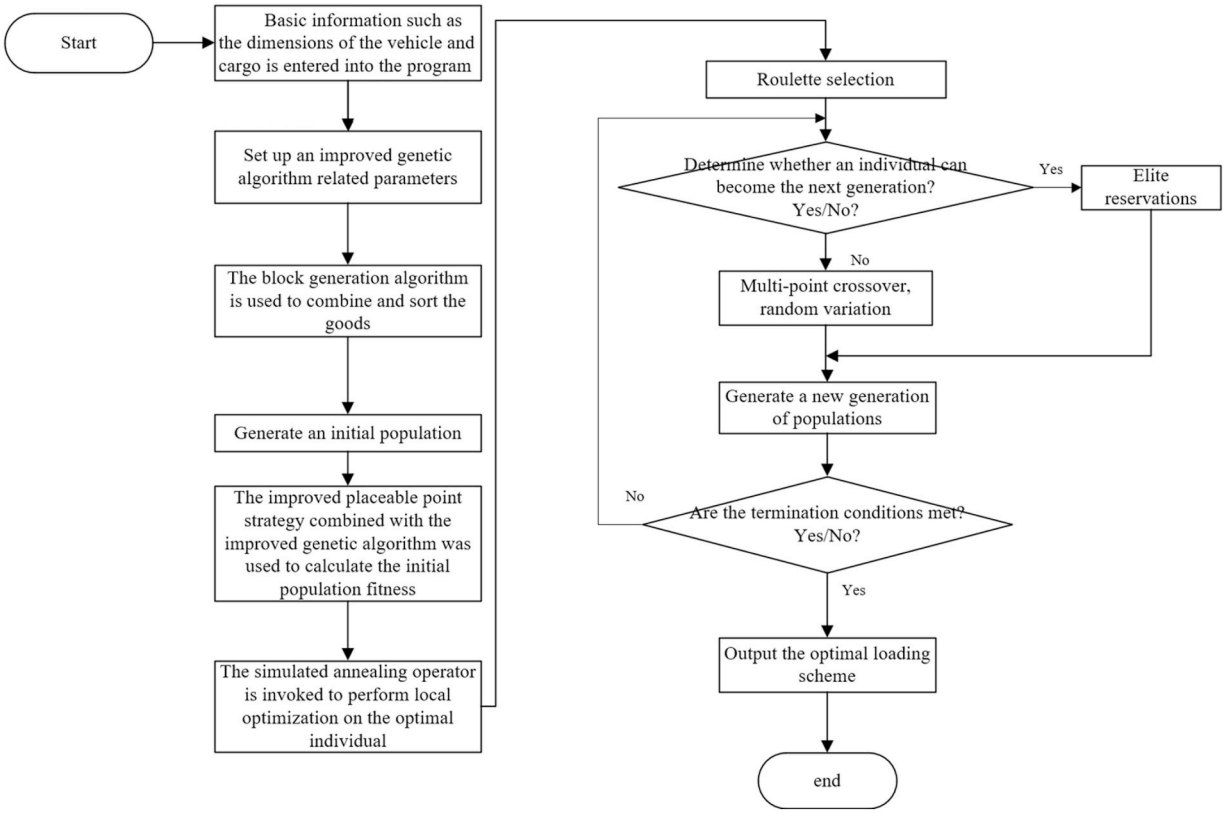

Fig. 14. Flow diagram of the 3D packing algorithm.

proposed algorithm incorporates load-bearing capacity constraints and center of gravity constraints, each cargo item is assigned a randomly generated mass between 10 kg and 40 kg. The performance of the proposed method is compared against several algorithms reported in the literature that have demonstrated strong results in this domain. Detailed comparisons are presented in Table 4. The computational results of the algorithm proposed in this paper are derived from the averaged outcomes across all 100 data sets for each BR type. Considering that the comparison algorithms have all incorporated fully supported constraints, the proposed algorithm also achieves full support by setting $\theta = 1$ within the stability constraints (refer to Section 2.1.2). ID-GLTS (Zhu and Lim, 2012) was implemented in Java and run on a server with Intel Xeon E5520 QuadCore CPUs clocked at 2.27 GHz with 8 GB RAM using CentOS Linux. The CLTRS (Fanslau and Bortfeldt, 2010) was implemented in C. and run on a PC with a 2.6 GHz Intel processor. BRKGA (Gonçalves and Resende, 2013) was implemented in C++ on an computer with an Intel 2.66 GHz Xeon Quadcore CPU using CentOS Linux. The BSG (Araya and Riff, 2014) were performed on a server PowerEdge T420, with 2 quad-processors Intel Xeon, 2.20 GHz and 8 GB RAM, running Ubuntu Linux. The codes were implemented in C++ and compiled using gcc.

ID-GLTS: Iterative-doubling Greedy–Lookahead tree search algorithm (Zhu and Lim, 2012).

CLTRS: Container Loading by Tree Search (Fanslau and Bortfeldt, 2010).

BRKGA: Biased random key genetic algorithm (Gonçalves and Resende, 2013).

BSG: Beam-Search-based algorithm that uses a Greedy-based function (Araya and Riff, 2014).

As shown in Table 4, a direct comparison of running times across all algorithms is infeasible due to differences in their operating environments; these values should therefore be considered only as reference. In terms of space utilization, the proposed algorithm slightly underperforms ID-GLTS, CLTRS, and BRKGA when handling weakly heterogeneous cargo types (BR1–BR7), but demonstrates superior performance with strongly heterogeneous types (BR8–BR15), achieving improvements in average space utilization of 0.62 %, 1.3 %, and 0.95 %, respectively. However, compared to BSG, which represents the state-of-the-art method for solving 3D-SKP, the proposed algorithm exhibits a 0.44 % decrease in average space utilization. This reduction may be attributed to the incorporation of two additional practical constraints—the center of gravity constraint and the cargo load-bearing constraint—which could restrict the solution search space and slightly compromise packing density. Nevertheless, the proposed algorithm offers broader applicability in real-world scenarios. In summary, the algorithm developed in this study is capable of generating high-quality loading plans while satisfying multiple practical constraints, thereby effectively addressing the three-dimensional knapsack problem (3D-SKP). Fig. 16 illustrates the loading layout for the first instance in each of the BR12–BR15 categories, with space utilization rates of 90.23 %, 90.04 %, 89.23 %, and 88.56 %, respectively.

In the aforementioned comparative experiments, to the best of our knowledge, apart from this paper, none of the other four compared algorithms consider the center-of-gravity offset constraint and the cargo load-bearing constraint (all other constraints are the same). The center-of-gravity offset constraint is a global constraint, as the placement position of any item affects the center of gravity of the entire container loading scheme. Consequently, the algorithm cannot evaluate the placement decision of a single item in isolation but must continuously perform global assessments either during or after the entire loading process. The cargo load-bearing constraint is typically a local-global hybrid constraint. The stability of an item depends on the items directly beneath it that provide support (local), yet the strength of this support structure influences the stability of the entire stack through force transmission paths (global). This requires the algorithm, when placing an item, to not only consider its spatial position but also evaluate its support area and the compressive strength of the items below. Therefore, introducing both center-of-gravity and load-bearing

Algorithm 4 Genetic Algorithm for 3D-CLP

```
Input: Cargos, Container, pop_size, gen_count
Output: best_individual, best_fitness
function generate_initial_population()
    create individuals with valid poses for each cargo
    return population
function calculate_fitness(individual)
    rest container
    place cargos according to poses
    evaluate placement (volume, weight, contact area, space fragmentation)
    return fitness score
function crossover(parent1，parent2)
    with probability 0.9
    select two crossover points
    exchange segments between parents
    return two children
function mutate(individual)
    adjust mutation rate dynamically
    with probability mutation_rate
    select one cargo position
    replace its pose with another valid pose
    return mutated individual
function roulette_wheel_selection(population, fitnesses)
    remove individuals with -inf fitness
    compute cumulative fitness distribution
    spin roulette to select pop_size individuals
    return selected individuals
function maintain_elite(population, fitnesses, percentage=0.1)
    sort individuals by fitness
    keep top percentage as elites
    return elites
function run()
    population ← generate_initial_population()
    best_fitness ← -inf
    best_individual ← None
    no_improvement ← 0
    early_stop_threshold ← max(50, gen_count×0.2)
    for generation = 1 to gen_count do
        fitnesses ← calculate_fitness for each individual
        update best_individual if improved
        if no improvement for early_stop_threshold generations then
            stop early
        every 5 generations apply simulated_annealing on best_individual
        update best if improved
        elites ← matain_elites(population, fitnesses)
        selected ← roulette_wheel_selection(population, fitnesses)
        next_generation ← elites
        while size(next_generation) < pop_size do
            parent1, parent2 ← choose from selected
            children ← crossover(parent1, parent2)
            foreach child in children do
                next_generation.add(mutate(child))
        population ← next_generation
    return best_individual, best_fitness
```

Fig. 15. Genetic Algorithm pseudocode.

Table 3
Comparison of algorithm execution time.

Test set	number	combination time/s	total time/s
BR1	112	0.04	18.80
BR2	81	0.04	14.29
BR3	94	0.04	15.65
BR4	106	0.08	22.35
BR5	98	0.07	23.29
BR6	129	0.09	35.07
BR7	110	0.09	29.45
BR8	142	0.08	96.31
BR9	146	0.10	132.01
BR10	136	0.12	130.74
BR11	128	0.10	173.32
BR12	136	0.11	205.71
BR13	126	0.11	227.74
BR14	118	0.09	248.15
BR15	121	0.11	268.63
average	119	0.08	109.43

Table 4
Algorithm comparison results 2.

Test set	ID-GLTS	CLTRS	BRKGA	BSG	this paper
BR1	94.40 %	94.51 %	94.34 %	94.50 %	94.27 %
BR2	94.85 %	94.73 %	94.88 %	95.03 %	94.69 %
BR3	95.10 %	94.74 %	95.05 %	95.17 %	94.25 %
BR4	94.81 %	94.41 %	94.75 %	94.97 %	93.83 %
BR5	94.52 %	94.13 %	94.58 %	94.80 %	93.60 %
BR6	94.33 %	93.85 %	94.39 %	94.65 %	93.42 %
BR7	93.59 %	93.20 %	93.74 %	94.09 %	93.04 %
BR8	92.65 %	92.26 %	92.65 %	93.15 %	92.78 %
BR9	92.11 %	91.48 %	91.90 %	92.53 %	92.56 %
BR10	91.60 %	90.86 %	91.28 %	92.04 %	91.88 %
BR11	90.64 %	90.11 %	90.39 %	91.40 %	91.53 %
BR12	90.35 %	89.51 %	89.91 %	90.92 %	90.77 %
BR13	89.69 %	88.98 %	89.27	90.51 %	90.35 %
BR14	89.07 %	88.26 %	88.57 %	89.93 %	89.98 %
BR15	88.36 %	87.57 %	87.96 %	89.33 %	89.55 %
average(1–7)	94.51 %	94.20 %	94.53 %	94.74 %	93.87 %
average(8–15)	90.56 %	89.88 %	90.23 %	91.23 %	91.18 %
average(1–15)	92.40 %	91.90 %	92.24 %	92.87 %	92.43 %
Time/s	150	320	232	150	110

constraints into the three-dimensional container loading algorithm significantly increases the complexity of the problem and the computational difficulty. It prevents the algorithm from focusing solely on geometric packing, thereby substantially raising the challenge of finding high-quality solutions. However, since the research context of this paper is logistical loading, and these two constraints cannot be overlooked in real-world logistical scenarios, achieving a well-performing solution while satisfying these additional constraints is one of the objectives of the algorithm proposed in this paper. Thus, the decision to not adopt the same conditions as the other algorithms in the comparative experiments above was made precisely to demonstrate the effectiveness of the proposed algorithm under these more stringent and realistic requirements. Comparative experiments under the same constraints were also conducted in the following text.

Alonso et al. (2014) also discussed cargo load-bearing constraints, so the algorithm proposed in this paper was compared with their proposed Reactive-GRASP. The Reactive-GRASP was coded in C++ and has been run on an Intel Core Duo T6500 with 2.1 Ghz and 4 GB of RAM. A total of 1500 sets of data from BR1 to BR15 in the BR example are still used for experiments, still employing full support constraints. The weight of the randomly assigned cargo is proportional to the load-bearing capacity, and the data structure used to calculate the load-bearing capacity is consistent with the literature (Alonso et al., 2014). The specific comparison results are shown in Table 5.

As shown in Table 5, regarding space utilization, the algorithm proposed in this paper performs better than the Reactive-GRASP algorithm in each BR instance, achieving a 5.65 % improvement in total average utilization (BR1-BR15). Regarding the running time, a fair comparison cannot be made due to the differences in specific parameters and operating environments of the two algorithms. The running times in Table 5 are for reference only.

In addition, to evaluate the applicability of the proposed algorithm across different scenarios, comparative experiments were conducted using the instances introduced by Loh and Nee in (Loh and Nee, 1992). The LN instances comprise 15 groups of data, with the number of cargo types ranging from 6 to 10, representing a weakly heterogeneous benchmark. Each data set was executed 30 times, and the best result from these runs was selected for comparison. The comparative outcomes with three other algorithms known to perform effectively on LN instances are presented in Table 6. Among them, the TDHD algorithm runs on an Intel(R) Core(TM) i7-3770 CPU @ 3.4 GHz computer with 16 GB of RAM running Ubuntu Linux 12.04 LTS1. The LNS algorithm is implemented in the Visual Basic Application and CLP Spreadsheet Solver, the experimental environment is a computer with an Intel R Core TM CPU of 1.60 GHz–1.80 GHz, an i5-8265U, and 16 GB of main

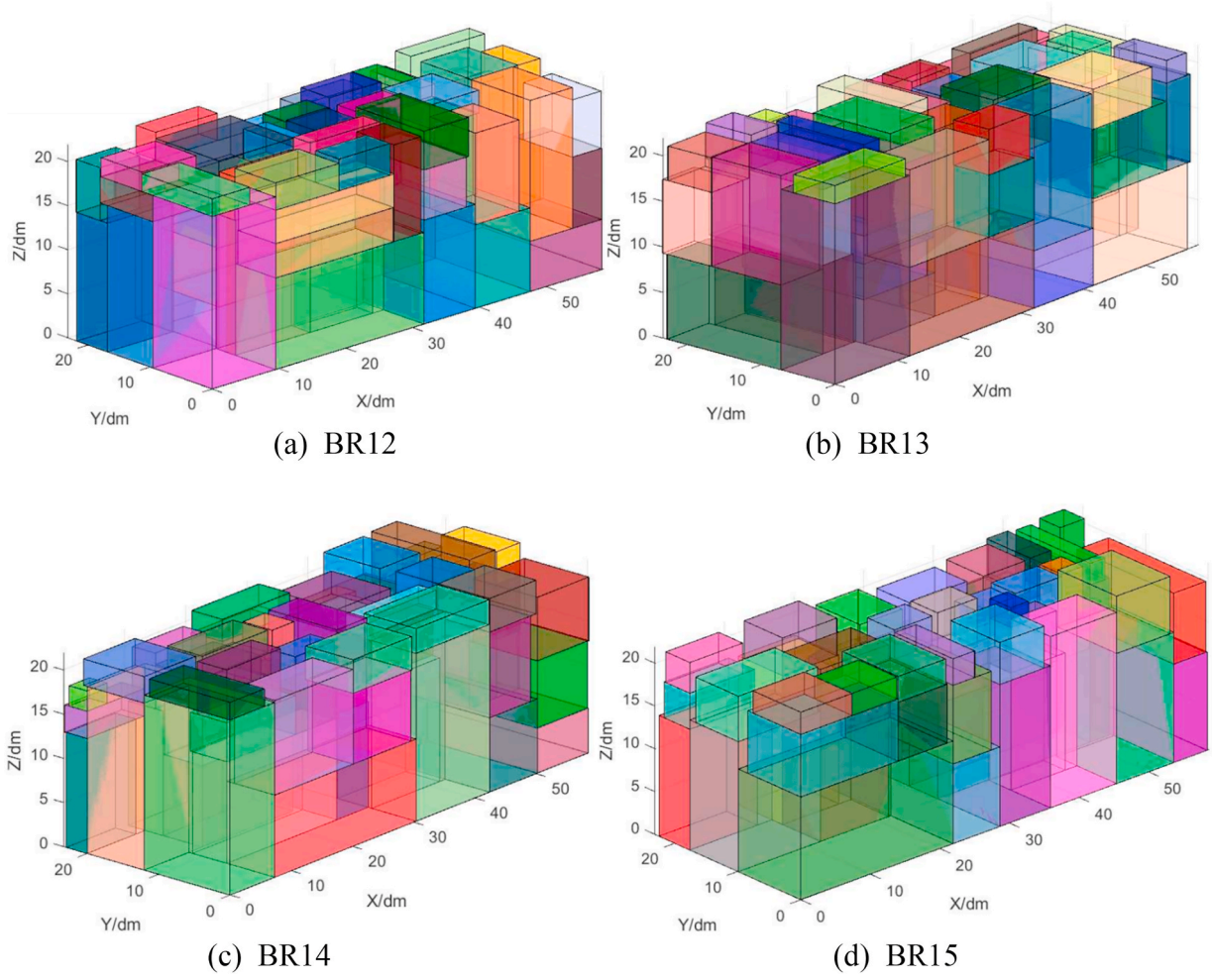

Fig. 16. BR12-BR15 loading renderings.

Table 5
Algorithm comparison results 3.

Test set	Reactive-GRASP	this paper
BR1	81.40 %	87.20 %
BR2	85.70 %	91.49 %
BR3	87.30 %	93.24 %
BR4	86.90 %	92.27 %
BR5	86.60 %	91.02 %
BR6	86.30 %	91.48 %
BR7	85.70 %	91.20 %
BR8	85.50 %	90.09 %
BR9	84.80 %	89.85 %
BR10	84.00 %	88.05 %
BR11	82.80 %	88.74 %
BR12	81.30 %	87.60 %
BR13	80.20 %	86.50 %
BR14	78.90 %	86.10 %
BR15	78.00 %	85.27 %
average(1–7)	85.70 %	91.13 %
average(8–15)	81.94 %	87.78 %
average(1–15)	83.69 %	89.34 %
Time/s	39.8	98

Table 6
Algorithm comparison results 3.

Test set	TDHDA	LNS	BBCP	this paper
LN1	62.5 %	62.5 %	62.5 %	62.5 %
LN2	93.2 %	94.6 %	95.9 %	95.2 %
LN3	53.4 %	53.4 %	53.4 %	53.4 %
LN4	55.0 %	55.0 %	55.0 %	55.0 %
LN5	77.2 %	77.2 %	77.2 %	77.2 %
LN6	93.0 %	92.6 %	94.6 %	93.3 %
LN7	84.7 %	84.7 %	84.7 %	84.7 %
LN8	59.4 %	59.4 %	59.4 %	59.4 %
LN9	61.9 %	61.9 %	61.9 %	61.9 %
LN10	67.3 %	67.3 %	67.3 %	67.3 %
LN11	62.2 %	62.2 %	62.2 %	62.2 %
LN12	78.5 %	78.5 %	78.5 %	78.5 %
LN13	85.6 %	85.6 %	85.6 %	85.6 %
LN14	62.8 %	62.8 %	62.8 %	62.8 %
LN15	59.5 %	59.5 %	59.5 %	59.5 %
average(1–15)	70.4 %	70.5 %	70.7 %	70.6 %
Time/s	/	/	513.8	24.5

memory. BBCP are conducted on the Spartan HPC system with an Intel (R) Xeon(R) Gold 6448H CPU processor, and constraint programming models are implemented in Python and solved using the Google OR-Tools CP-SAT Solver (v9.10.4067).

TDHDA: A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem (Toffolo et al., 2017).

LNS: A Large Neighborhood Search Algorithm for Solving Container Loading Problems (Şafak and Erdoğan, 2023).

BBCP: A block-building constraint programming model for the container loading problem (Liu et al., 2025).

Since the LN instances represent a relatively simple weakly heterogeneous benchmark, most data sets admit a theoretical optimal solution. As shown in Table 6, the algorithm proposed in this paper achieves the

best results in 13 of the data sets. However, for LN2 and LN6, its performance is slightly lower than that of the exact algorithm BBCP, which attained values of 95.9 % and 94.6 %, respectively. This discrepancy can be attributed to the generally stronger performance of exact algorithms in handling weakly heterogeneous cargo. As noted earlier, due to differences in algorithmic parameters and experimental environments, the average computation times provided in the table should be considered for reference only. Among them, the TDHDA and LNS algorithms did not report specific computation time data in the literature. For the BBCP algorithm, since it failed to obtain the optimal solution within the allotted time for instances LN2 and LN6, the computation time was set to 3600 s as reported in the respective study, resulting in a comparatively higher average computation time. The algorithm in this paper significantly shortens the processing time for weakly heterogeneous types of cargos compared to strongly heterogeneous types. Although there are still some shortcomings, the comparative experiments mentioned above have demonstrated that the algorithm proposed in this paper performs well in handling weakly heterogeneous cargo.

4. Conclusion

This paper mainly introduces a three-dimensional container loading algorithm based on a variety of heuristic algorithms and combined with improved genetic algorithm in the context of express logistics, which is used to solve the actual logistics packing problem. The algorithm aims to maximize the space utilization rate of the carriage and minimize the offset of the center of gravity of the vehicle, and considers six typical realistic constraints. In this paper, the block generation algorithm is used to reduce the computational complexity, and then the improved placeable point strategy is combined with the improved genetic algorithm and the simulated annealing operator to calculate the optimal loading scheme. A comparative evaluation against other state-of-the-art algorithms was conducted using a public dataset, demonstrating that the proposed algorithm maintains competitive performance even when additional constraints—namely, cargo load-bearing and center of gravity shift constraints—are taken into account. The results indicate that the proposed algorithm is capable of computing high-quality solutions for packing strongly heterogeneous items under multiple constraints, thereby offering valuable insights for addressing the three-dimensional single knapsack problem (3D-SKP).

The following are potential research directions that could be further explored based on the work presented in this paper.

(1) More diverse strategies for block generation can be explored, as different methods of combination can lead to significant differences in the sizes of goods to be processed, thus affecting the subsequent loading outcomes.
(2) Additional evaluation metrics beyond contact area comparison could be incorporated during the loading process. This would help ensure each item is placed in an optimal position, ultimately improving space utilization.
(3) Given the variety of cargo shapes in logistics scenarios—which include not only regular-shaped items but also numerous irregular-shaped ones—future studies could incorporate more diverse geometric forms into the loading calculations.

CRediT authorship contribution statement

Ying Ma: Writing – review & editing, Supervision, Resources, Project administration, Investigation, Funding acquisition, Conceptualization. **Yu Zhou:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Data curation. **Qiwei Fang:** Visualization, Validation, Supervision. **Shengwei Xia:** Supervision, Investigation. **Wei Chen:** Project administration, Methodology.

Funding

This work was supported by Fujian Provincial Science and Technology Plan Project (Innovation Fund Project) under Grant 2024C0045.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Alonso, M.T., Alvarez-Valdes, R., Tamarit, M.J., Parreño, F., 2014. A reactive GRASP algorithm for the container loading problem with load-bearing constraints. Eur. J. Ind. Eng. 8 (5).
Araya, I., Riff, M.-C., 2014. A beam search approach to the container loading problem. Comput. Oper. Res. 43, 100–107. https://doi.org/10.1016/j.cor.2013.09.003.
Araya, I., Guerrero, K., Nuñez, E., 2017. VCS: a new heuristic function for selecting boxes in the single container loading problem. Comput. Oper. Res. 82, 27–35. https://doi.org/10.1016/j.cor.2017.01.002.
Bayraktar, T., Ersöz, F., Kubat, C., 2021. Effects of memory and genetic operators on Artificial Bee Colony algorithm for Single Container Loading problem. Appl. Soft Comput. 108, 107462. https://doi.org/10.1016/j.asoc.2021.107462.
Bischoff, E.E., Ratcliff, M.S.W., 1995. Issues in the development of approaches to container loading. Omega 23 (4), 377–390.
Bortfeldt, A., Gehring, H., 2001. A hybrid genetic algorithm for the container loading problem. Eur. J. Oper. Res. 131 (1), 143–161. https://doi.org/10.1016/S0377-2217(00)00055-2.
Bortfeldt, A., Gehring, H., Mack, D., 2003. A parallel tabu search algorithm for solving the container loading problem. Parallel Comput. 29 (5), 641–662. https://doi.org/10.1016/S0167-8191(03)00047-4.
Davies, A.P., Bischoff, E.E., 1999. Weight distribution considerations in container loading. Eur. J. Oper. Res. 144 (3), 509–527.
Delorme, M., Wagenaar, J., 2024. Exact decomposition approaches for a single container loading problem with stacking constraints and medium-sized weakly heterogeneous items. Omega 125, 103039. https://doi.org/10.1016/j.omega.2024.103039.
Dereli, T., Das, G., 2011. A hybrid 'bee(s) algorithm' for solving container loading problems. Appl. Soft Comput. 11 (2), 2854–2862. https://doi.org/10.1016/j.asoc.2010.11.017.
Eley, M., 2002. Solving container loading problems by block arrangement. Eur. J. Oper. Res. 141 (2), 393–409. https://doi.org/10.1016/S0377-2217(02)00133-9.
Fanslau, T., Bortfeldt, A., 2010. A tree search Algorithm for solving the container loading problem. Inf. J. Comput. 22 (2), 222–235. https://doi.org/10.1287/ijoc.1090.0338.
Goldberg, D.E., Holland, J.H., 1988. Genetic algorithms and machine learning. Mach. Learn. 3, 95–99.
Goldberg, N., Karhi, S., 2019. Online packing of arbitrary sized items into designated and multipurpose bins. Eur. J. Oper. Res. 279 (1), 54–67. https://doi.org/10.1016/j.ejor.2019.05.029.
Gonçalves, J.F., Resende, M.G.C., 2013. A biased random key genetic algorithm for 2D and 3D bin packing problems. Int. J. Prod. Econ. 145 (2), 500–510. https://doi.org/10.1016/j.ijpe.2013.04.019.
Junqueira, L., Morabito, R., Yamashita, D.S., 2012a. Three-dimensional container loading models with cargo stability and load bearing constraints. Comput. Oper. Res. 39 (1), 74–85. https://doi.org/10.1016/j.cor.2010.07.017.
Junqueira, L., Morabito, R., Sato Yamashita, D., 2012b. MIP-based approaches for the container loading problem with multi-drop constraints. Ann. Oper. Res. 199, 51–75. https://doi.org/10.1007/s10479-011-0942-z. OCT.
Kirkpatrick, S., Gelatt, C.D., Vecchi, A.M.P., 1983. Optimization by simulated annealing. Science. https://doi.org/10.1126/science.220.4598.671.
Lee, C.K.H., 2018. A review of applications of genetic algorithms in operations management. Eng. Appl. Artif. Intell. 76, 1–12.
Liu, C., Smith-Miles, K., Wauters, T., Costa, A.M., 2025. A block-building constraint programming model for the container loading problem. Comput. Oper. Res. 182, 107111. https://doi.org/10.1016/j.cor.2025.107111.
Loh, H.T., Nee, A.Y.C., 1992. A Packing Algorithm for Hexahedral Boxes.
Nascimento, O., Alves de Queiroz, T., Junqueira, L., 2021. Practical constraints in the container loading problem: comprehensive formulations and exact algorithm. Comput. Oper. Res. 128, 105186. https://doi.org/10.1016/j.cor.2020.105186.
Parreño, F., Alvarez-Valdes, R., Oliveira, J.F., et al., 2010. Neighborhood structures for the container loading problem: a VNS implementation. J. Heuristics 16 (1), 1–22. https://doi.org/10.1007/s10732-008-9081-3.
Pisinger, D., 2002. Heuristics for the container loading problem. Eur. J. Oper. Res. 141 (2), 382–392. https://doi.org/10.1016/S0377-2217(02)00132-7.
Ramos, A.G., Silva, E., Oliveira, J.F., 2018. A new load balance methodology for container loading problem in road transportation. Eur. J. Oper. Res. 266, 1140–1152.
Şafak, Ö., Erdoğan, G., 2023. A large neighbourhood search algorithm for solving container loading problems. Comput. Oper. Res. 154, 106199. https://doi.org/10.1016/j.cor.2023.106199.

Terno, J., Scheithauer, G., Sommerweiß, U., Riehme, J., 2000. An efficient approach for the multi-pallet loading problem. Eur. J. Oper. Res. 123 (2), 372–381. https://doi.org/10.1016/S0377-2217(99)00263-5.

Toffolo, T.A.M., Esprit, E., Wauters, T., Vanden, Berghe G., 2017. A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem. Eur. J. Oper. Res. 257 (2), 526–538. https://doi.org/10.1016/j.ejor.2016.07.033.

Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. Eur. J. Oper. Res. 183 (3), 1109–1130. https://doi.org/10.1016/j.ejor.2005.12.047.

Yang, Y., Wu, Z., Hao, X., Liu, H., Qi, M., 2024. Two-layer heuristic for the three-dimensional bin design and packing problem. Eng. Optim. 56 (10), 1601–1638. https://doi.org/10.1080/0305215X.2023.2269868.

Youssef, Harrath, 2021. A three-stage layer-based heuristic to solve the 3D bin-packing problem under balancing constraint. J. King Saud Univ. Comput. Inf. Sci. 34 (8), 6425–6431.

Zhang, D., Gu, C., Fang, H., Ji, C., Zhang, X., 2022. Multi-strategy hybrid heuristic algorithm for single container weakly heterogeneous loading problem. Comput. Ind. Eng. 170, 108302. https://doi.org/10.1016/j.cie.2022.108302.

Zhu, W., Lim, A., 2012. A new iterative-doubling Greedy–Lookahead algorithm for the single container loading problem. Eur. J. Oper. Res. 222 (3), 408–417. https://doi.org/10.1016/j.ejor.2012.04.036.

Zhu, W., Oon, W.-C., Lim, A., Weng, Y., 2012. The six elements to block-building approaches for the single container loading problem. Appl. Intell. 37 (3), 431–445. https://doi.org/10.1007/s10489-012-0337-0.