Computers & Industrial Engineering 209 (2025) 111450

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A computational intelligence method to solve the problem of packing variable-length fresh products into fixed-size bins

Yanjie Zhou, Xiaojin Wang, Hang Wang, Jiang Xu *

School of Management, Zhengzhou University, China

ARTICLE INFO

Keywords:
Bin packing
Computational intelligence
Fresh product packing

ABSTRACT

Fresh production packing is very essential in green logistics. A fresh product usually can be parceled up into different three-dimensional sizes. This paper studies the hybrid variable of fresh product packing into fixed-size bins, where part of the product can be parceled into different three-dimensional sizes. A nonlinear mathematical model is proposed to formulate the studied problem, and a piecewise linearization approximation method is used to linearize it. A three-phase computational intelligence framework integrated with variable neighbourhood search is proposed to solve large-scale studied problems. The proposed computational intelligence framework can readily adopt any computational intelligence method. This paper adopts a genetic algorithm and harmony search to implement the proposed framework. This paper conducts various experiments to verify the performance of the proposed solution framework. The experimental results showed the effectiveness and efficiency of the proposed solution framework.

1. Introduction

Fresh products, encompassing perishable goods such as recently harvested fruits, vegetables, and freshly processed meats, require specialized packaging solutions to maintain quality and integrity. Common packaging materials, including plastic bags, corrugated cardboard boxes, molded foam trays, cling films, and vacuum-sealed containers, protect against physical damage and environmental factors during handling, transportation, and storage. Effective packaging design plays a critical role in preserving product freshness. For instance, fragile items like eggs necessitate customized solutions such as compartmentalized pulp cartons that provide individual cushioning to prevent shell fractures. The geometric challenges of packaging irregular polyhedral-shaped produce (e.g., broccoli, star-fruits) require flexible three-dimensional packaging configurations that adapt to product morphology while maintaining structural stability.

Fig. 1 demonstrates the dimensional variability in egg packaging through different carton designs. This illustrates how identical quantities of products can be accommodated in varying spatial arrangements through strategic container selection. Such flexibility in packaging geometry enables optimized space utilization during logistics operations while ensuring product protection. The choice of appropriate packaging configurations must balance protection requirements with transportation efficiency and material sustainability considerations.

This paper studies the hybrid variable of the fresh product packing into fixed-size bins problem. In the studied problem, there is a set of fresh and non-fresh products. Only parts of the fresh product can be parceled into variable three-dimensional sizes. The remainder of the products are parceled up into fixed three-dimensional sizes. All the products need to be packed into a set of three-dimensional bins. The goal of the studied problem is to maximize the utilization of three-dimensional bins by stratifying the constraints of freshness. The contributions of this paper are summarized as follows.

(1) This paper first studied the hybrid variable of the fresh product packing into fixed-size bins problem that allows part of the fresh product to adjust its packing size. This paper adopts living fish packing as a case study. A nonlinear integer mathematical model is proposed to formulate the fresh fish packing problem, considering the oxygen requirement of living fish and activity space during transportation.
(2) The nonlinear mathematical model is complex to solve by both commercial and non-commercial mixed integer programming solvers, including Cplex, Gurobi, and SCIP. A piecewise linearization approximation method makes the nonlinear mathematical

* Corresponding author.
E-mail addresses: ieyjzhou@zzu.edu.cn (Y. Zhou), 1970591837@qq.com (X. Wang), 1277574781@qq.com (H. Wang), xujiang@zzu.edu.cn (J. Xu).

https://doi.org/10.1016/j.cie.2025.111450
Received 31 March 2025; Received in revised form 10 June 2025; Accepted 3 August 2025
Available online 9 August 2025

model solvable. The developed linear mixed integer programming model provides optimal solutions for small-scale problems.

(3) To solve large-scale problems, this paper decomposes the studied problem into three subproblems. A three-phase computational intelligence framework is proposed by integrating variable neighborhood search methods into the three subproblems. This paper adopts genetic algorithms and harmony search algorithms as examples of computational intelligence methods to implement the proposed framework. This paper conducts various experiments to verify the proposed solution framework. The experiment results show the outperformance of the proposed solution framework compared with the mathematical models

The remainder of this paper is organized as follows. Section 2 shows the literature review. The proposed mathematical models are shown in Section 3. Section 4 presents the proposed solution method. The experimental results are shown in Section 5. Finally, the conclusions are given.

2. Literature review

The Bin packing problem is a classic NP-hard combinatorial optimization problem (Coffman et al., 1980), which is widely used in logistics, transportation systems, and production systems. The mixed bin packing problem of three-dimensional variable-size goods studied in this paper is a three-dimensional bin packing problem(3D-BPP). The bin packing problem explores how to place a set of strongly or weakly heterogeneous goods into one or a group of strongly or weakly heterogeneous large containers to maximize the loaded items (or value) or minimize the number of containers used. In general, the research can be divided into two levels, one of which is the extension of the application of the bin packing problem, and the other is the research on the algorithms of the three-dimensional bin packing problem.

2.1. Application research on the three-dimensional bin packing problem

Many scholars have proposed different detailed loading scenarios for the Loading-related constraints based on the actual cargo loading process. Alonso et al., 2019a propose a two-stage pallet loading problem that considers multiple constraints such as geometry, weight, and prohibition of gaps between pallets (to prevent goods from moving). Considering the above, stability constraints and periodic constraints for loading are considered (Alonso et al., 2019b). Sheng et al. (2017) and Ekici (2021) explore different forms of full loading constraints. According to the machine loading scenario, Zhao et al. (2021) study the online 3D-BPP. The above researches provide a comprehensive theoretical framework for different actual loading scenarios, focusing on the industrial field and route arrangement, simplifying cargo loading planning.

Regarding the goods-related constraints, Ren et al. (2011) considered the cargo loading priority. A 3D-BPP with time windows is studied by Liu et al. (2021). A complex 3D-BPP involving multiple goods is proposed by Ceschia and Schaerf (2011). Zhao et al. (2022) take into account the effect of deformation of compressible cargo, but it is only considered a weak constraint in the loading process. Regarding the packaging and loading of a single type of goods, Kilincci and Medinoglu (2021) study the packaging and loading of a single type of cargo. However, few studies focus on the impact of changing cargo size on improving loading rate and the mixed loading of variable-size and fixed-size goods.

Regarding container-related constraints, Paquay et al. (2014) propose a loading problem for boxes with special shapes based on air transportation to meet the special needs of air transportation. Que et al. (2023) discuss the 3D-BPP with adjustable container height, providing new ideas for packing optimization in scenarios with variable container sizes. Ananno and Ribeiro (2024) focus on the 3D-BPP of multiple containers, take into account constraints such as container-full-shipment to minimize the total number of containers used, and finally achieve the complete palletizing of the order. However, no in-depth research has been conducted on the compatibility of containers of different sizes with various types of goods.

Regarding the study of multi-objective 3D-BPP, Gendreau et al. (2006) were the first to combine the 3D-BPP and path planning problems. Ceschia and Schaerf (2013) propose a multi-objective function with four parts and assigned weights. Erbayrak et al. (2021) study the multi-objective 3D-BPP, aiming to minimize the number of containers used and maximize the family unity ratio, providing a new direction for the study of cargo correlation in the packing problem. However, there is a lack of targeted research on the needs of special fields, such as fresh food, in terms of multi-objective trade-offs.

2.2. Algorithm of the three-dimensional bin packing problem

In the literature on using exact algorithms to solve the 3D-BPP,

Fig. 1. An example of fresh egg packing into different types of cartons.

usually, a mathematical model for the 3D-BPP is built first, and then the solver (e. g., CPLEX, LINGO, etc.) will be used to solve the model. Junqueira et al. (2012) construct a mixed integer linear programming model and solve it with the CPLEX solver. Chen et al. (1995) propose a zero-one mixed integer programming model. Nascimento et al. (2021) model common constraints. Fleszar (2022) proposes the branch and bound method. Zhang et al. (2020) introduce the branch and price algorithm. Hifi et al. (2010) improve the model relaxation lower bound through effective inequalities. However, the exact algorithm is inefficient for solving large-scale problems, and it isn't easy to fully meet the strict assumptions of the model in actual application scenarios.

The constructive heuristic algorithm can quickly provide an approximate solution to the problem through the designed placement rules. The placement rules are designed with concepts such as "wall-building"(George & Robinson, 1980; Kang et al., 2024), "layer-building" (Saraiva et al., 2015), "block-building" (Zhu et al., 2012), and "tower-building". There are also extreme point methods (Crainic et al., 2008) and Deep Bottom Left with Fill strategy (Korhan & Mustafa, 2004). The construction heuristic algorithm can quickly provide approximate solutions, improving the packing efficiency and solution quality, but its adaptability to complex constraints is limited.

Deep reinforcement learning algorithms (Jiang et al., 2021; Que et al., 2023; Wang et al., 2025) and local search algorithms (T. et al., 2003; Faroe et al., 2003) are common *meta*-heuristic algorithms. The calculation time for *meta*-heuristic algorithms is relatively long, and it is often combined with a construction heuristic algorithm to form a hybrid heuristic algorithm to improve the overall performance and increase the convergence speed and performance of the algorithm. Moon et al. (2013) combine multiple algorithms to solve the 3D-BPP, considering load balance, effectively combining the convergence advantage of the greedy algorithm with the optimization advantage of the ant colony algorithm. Zhang et al. (2024) propose a method combining a generative adversarial network and a genetic algorithm to solve the 3D-BPP, which proves to generate high-quality solutions and has excellent robustness and effectiveness in 3D-BPP solving, but the method relies upon the quality of the initial solution.

This paper takes the packaging and loading of live fish in fresh cold chain logistics as a scenario and proposes a three-dimensional variable-size mixed cargo packing problem. A linear mixed integer programming model is constructed, and a variable-size optimization algorithm is designed. The algorithm contains three stages: construction, search, and improvement. The article is dedicated to providing logistics companies with reasonable solutions to improve container loading rates and promote digital development.

3. Mathematical model

This section introduces the problem description and mathematical model.

3.1. Problem descriptions

Given a set of cargo I and $I = \left\{\widetilde{I}, \bar{I}\right\}$, $\widetilde{I}$ denotes the set of cargo with variable size and $\bar{I}$ denotes the set of cargo with fixed size, where $|I| = k$, $\left|\widetilde{I}\right| = n$ and $|\bar{I}| = k - n$. (w_i, h_i, d_i) represents the width, height, and height of the cargo i, respectively. v_i denotes the volume of cargo i. If $i \in \bar{I}$, w_i, h_i and d_i are constant value. If $i \in \widetilde{I}$, h_i and d_i are variable with the constraints that $h_i^{LB} \le h_i \le h_i^{UB}$ and $d_i^{LB} \le d_i \le d_i^{UB}$ are hold. All the cargo will be packed into a container C with a length of W, a width of H, a height of D. VC denotes the volume of container C. This paper aims to maximize the utilization of container space, taking into account many constraints in the actual loading process, and aims to provide logistics companies with an optimized cargo transportation packaging and container loading solution. The parameters and decision variables are shown in Table 1.

The assumptions used in this paper are summarized as follows:

1) All cargo to be loaded is packed as regular cuboids. The shape of the cargo is divided into two types: regular and irregular. Living fish were originally irregularly shaped cargo, but they became regular blocks after packaging. A subset of cargo is variable.
2) In this paper, the packing for transporting live fish was adjusted to increase the container loading rate, without considering the impact of different transport packaging schemes on the survival rate of live fish. Therefore, it is assumed that if the live fish have a certain amount of space to move around in the bag, the impact of the transportation packaging scheme on the survival rate of the fish is not considered in this paper.
3) In the actual packaging process of live fish, the size of the packaging bag is usually chosen according to the length of the fish. Therefore, the length of the transport package changes less than the width and height. Thus, when adjusting the transport package of live fish, this paper assumes that the length of the goods is fixed, and only changes the height and width of the cargo. Fig. 2 shows the Living fish packing process.
4) The container and the cargo both have a certain weight-bearing capacity. This article assumes that the total weight of the cargo placed in the container is within its bearing range and that no squeezing or deformation occurs between the cargo. Regardless of the weight limit constraints, cargo weight-bearing constraints during loading are not considered.
5) The cost of customized packaging and packaging materials is not considered. Due to the scale effect, this article assumes that the logistics company's live fish packing cost is lower than that of the shipper; the cost of customized packaging is not considered.

Table 1
Symbols and decision variables.

Type	Symbol	The meaning of the parameters
Parameter	C	Container
	I	A set of cargo I, where$I = \left\{\widetilde{I}, \bar{I}\right\}$ and $\|I\| = k$
	$\widetilde{I}$	The set of cargo with variable size, where $\left\|\widetilde{I}\right\| = n$
	$\bar{I}$	The set of cargo with fixed size, where $\|\bar{I}\| = k - n$
	i	Index of cargo, where $i \in I$
	j	Index of cargo, where $j \in I$
	(W,H,D)	The length, width and height of the container
	(w_i, h_i, d_i)	The length, width and height of the cargo i
	VC	The volume of container C
	v_i	The volume of the cargo i
	n	Number of variable-size goods to be loaded
	k	Total quantity of goods to be loaded
	h_i^{LB}, h_i^{UB}	The lower and upper limits of the height dimensional changes of cargo i
	d_i^{LB}, d_i^{UB}	The lower and upper limits of the width dimensional changes of cargo i
	m	The number of function segment intervals
Decision variables	(x_i, y_i, z_i)	The coordinates of the lower left rear corner of cargo i in the coordinate system
	h_i'	Width after the size of cargo i changes
	d_i'	Height after the size of cargo i changes
	s_i	1 If cargo i is placed in the container, then it is 1; 0 otherwise.
	l_{ij}	1 if cargo i is to the left of cargo j; 0 otherwise
	r_{ij}	1 if cargo i is to the right of cargo j; 0 otherwise
	u_{ij}	1 if cargo i is below cargo j; 0 otherwise
	o_{ij}	1 if cargo i is above cargo j; 0 otherwise
	b_{ij}	1 if cargo i is behind cargo j; 0 otherwise
	f_{ij}	1 if cargo i is in front of cargo j; 0 otherwise
	α_{ij}	The jth breakpoint of the continuous variable p_i
	β_{ij}	The jth breakpoint of the continuous variable q_i

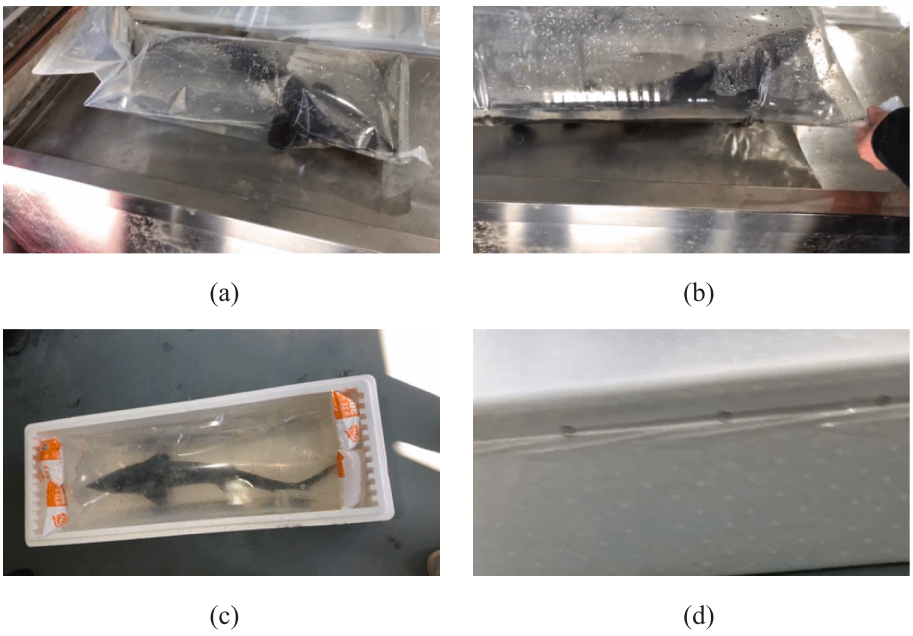

Fig. 2. Living fish packing process.

3.2. Nonlinear mixed integer programming model

This paper aims to maximize the utilization of container space, and the corresponding objective function can be expressed as:

$$P_N Max \frac{\sum_{i=1}^{n} v_i s_i}{VC} \tag{1}$$

3.2.1. Cargo geometric constraints

Obviously, the cargo cannot exceed or overlap with the container, so the cargo location coordinates must satisfy the following constraints:

$$\begin{cases} 0 \le x_i \le W - w_i \\ 0 \le y_i \le H - h_i \\ 0 \le z_i \le D - d_i. \end{cases} \tag{2}$$

3.2.2. Constraints on the non-overlapping of cargo

Here, six binary decision variables represent the positional relationships between the goods: left, right, top, bottom, front, and back, where $i < j$. To ensure that cargo i and cargo j placed in a container do not overlap or intersect, the following relationship must exist (Egeblad and Pisinger, 2009).

$$l_{ij}+r_{ij}+u_{ij}+o_{ij}+b_{ij}+f_{ij} \ge 1 \tag{3}$$

This constraint states that when both cargo i as well as cargo j are placed in a container($s_i = s_j = 1$), the positional relationship between them must be front or rear, left or right, top or bottom, and the positional relation between the goods must include one or more items. At the same time, the coordinates of goods i and j must satisfy the following inequalities:

$$\begin{array}{l} l_{ij} = 1 \Rightarrow x_i + w_i \le x_j, r_{ij} = 1 \Rightarrow x_j + w_j \le x_i \\ u_{ij} = 1 \Rightarrow y_i + h_i \le y_j, o_{ij} = 1 \Rightarrow y_j + h_j \le y_i \\ b_{ij} = 1 \Rightarrow z_i + d_i \le z_j, f_{ij} = 1 \Rightarrow z_j + d_j \le z_i \end{array} \tag{4}$$

3.2.3. Constraints on the constant volume of cargo

h_i', d_i' are the width and height of the cargo i after the size change. To ensure that the volume of the cargo remains unchanged after the size change, the height and width after the change must meet the following requirements:

$$\frac{v_i}{w_i} = h_i' \times d_i' = h_i \times d_i \tag{5}$$

In the model assumptions, this paper assumes that the length of the variable-size cargo is fixed, and the width and height are variable. Here, it is necessary to constrain the range of changes in the size of the goods to ensure that the live fish have a certain space to move. First, the width and height range depend on the size of the side of the cargo. The cargo size cannot exceed the side area, which determines the upper limit of the contents. At the same time, it is also, to some extent, related to the size of the live fish. The changed packaging must completely wrap the goods, which determines the lower limit of the size change. Fig. 3 is a simple schematic diagram of size change.

To facilitate calculation, it is assumed that the length and width of fish i are $h_i/2$ and $d_i/2$. Therefore, the lower limit of h_i' and d_i' are $h_i/2$ and $h_i/2$. Accordingly, the upper limit is the side area, so the range of cargo

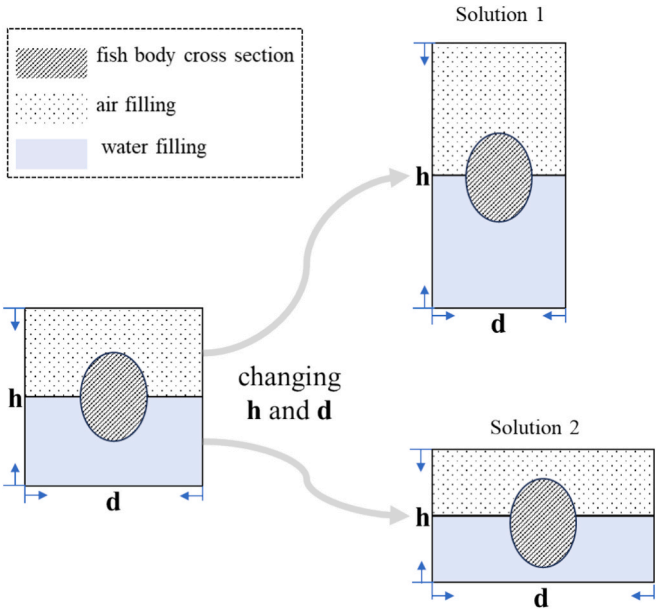

Fig. 3. Schematic diagram of dimensional change.

size variation can be defined as:

$$h_i^{LB} = h_i/2, h_i^{UB} = h_i \times d_i \tag{6}$$

$$d_i^{LB} = h_i/2, d_i^{UB} = h_i \times d_i$$

The mixed integer programming model proposed in this paper is an extended model of the traditional three-dimensional container model. The model introduces the constraint of constant volume of cargo packaging (5) to ensure that the volume of the box remains constant after changes. Eq. (5) is a nonlinear constraint, which cannot be directly modelled and solved by either commercial mixed integer programming solvers (such as CPLEX) or open source solvers (such as CBC), the model needs to use a linearization method to transform the nonlinear constraints into linear constraints before the mixed integer programming solvers can perform modelling and solving.

3.3. Linear mixed integer programming model

As mentioned before, P_N is a nonlinear mixed integer programming model. This subsection adopts a piecewise linearization approximation (PLA) method to linearize it. In this subsection, we introduce two auxiliary continuous variables, p_i and q_i to transform the constraints.

Let $p_i = (h_i' + d_i')/2, q_i = (h_i' - d_i')/2$, then $p_i^2 - q_i^2 = h_i' \times d_i'$, where $h_i' = (p_i + q_i), d_i' = (p_i - q_i)$. $h_i' \in (h_i^{LB}, h_i^{UB})$ and $d_i' \in (d_i^{LB}, d_i^{UB})$ are hold. The upper and lower limits of p_i and q_i are shown as follows.

$$\frac{1}{2}(h_i^{LB} + d_i^{LB}) \le p_i \le \frac{1}{2}(h_i^{UB} + d_i^{UB}) \tag{7}$$

$$\frac{1}{2}(h_i^{LB} - d_i^{UB}) \le q_i \le \frac{1}{2}(h_i^{UB} - d_i^{LB}) \tag{8}$$

The PLA method is then used to linearize the nonlinear constraints: p_i^2 is segmented, that is, along the p_i-axis, the interval is divided into m intervals $[\alpha_{i0}, \cdots \alpha_{im}]$ of equal length. $p_i \in \left((h_i^{LB} + d_i^{LB})/2, (h_i^{UB} + d_i^{UB})/2\right)$, there are $\alpha_{i0} = (h_i^{LB} + d_i^{LB})/2$ and $\alpha_{im} = (h_i^{UB} + d_i^{UB})/2$. Let $\theta_{\alpha i} = \left((h_i^{UB} + d_i^{UB})/2 - (h_i^{LB} + d_i^{LB})/2\right)/m$, then α_{it} and β_{it} can be defined as follow.

$$\alpha_{it} = \alpha_{i0} + (t-1) \times \theta_{\alpha i}, \forall t = 1, \cdots, m \tag{9}$$

$$\beta_{it} = \beta_{i0} + (t-1) \times \theta_{\beta i}, \forall t = 1, \cdots, m \tag{10}$$

Then the nonlinear constraints (5) can be linearized as follows.

$$\sum_{t=0}^{m} \alpha_{it}\varepsilon_{it} = p_i \forall i = 1, \cdots, n \tag{11}$$

$$\sum_{t=0}^{m} \varepsilon_{it} = 1 \forall i = 1, \cdots, n \tag{12}$$

$$\varepsilon_{i0} \le \omega_{i0} \forall i = 1, \cdots, n \tag{13}$$

$$\varepsilon_{it} \le \omega_{i,t-1} + \omega_{i,t} \forall t = 1, \cdots, m-1 \tag{14}$$

$$\varepsilon_{i,m} \le \omega_{i,m-1} \forall i = 1, \cdots, n; t = 1, \cdots, m \tag{15}$$

$$\sum_{t=0}^{m-1} \omega_{it} = 1 \tag{16}$$

$$\sum_{t=0}^{m} \beta_{it}\delta_{it} = q_i \forall i = 1, \cdots, n \tag{17}$$

$$\sum_{t=0}^{m} \delta_{it} = 1 \forall i = 1, \cdots, n \tag{18}$$

$$\delta_{i0} \le \varphi_{i0} \forall i = 1, \cdots, n \tag{19}$$

$$\delta_{it} \le \varphi_{i,t-1} + \varphi_{i,t} \forall t = 1, \cdots, m-1 \tag{20}$$

$$\delta_{i,m} \le \varphi_{i,m-1} \forall i = 1, \cdots, n \tag{21}$$

$$\sum_{t=0}^{m-1} \varphi_{it} = 1 \forall i = 1, \cdots, n \tag{22}$$

$$\sum_{t=0}^{m} \alpha_{it}^2\varepsilon_{it} - \sum_{t=0}^{m} \beta_{it}^2\delta_{it} = h_i \times d_i \forall i = 1, \cdots, n \tag{23}$$

$$\delta_{it}, \varepsilon_{it} \ge 0, \varphi_{it}, \omega_{it} \in \{0,1\} \forall t = 0, \cdots, n-1 \tag{24}$$

Finally, the linearized mixed integer programming model could be obtained as follows.

$$Max \frac{\sum_{i=1}^{k} v_i s_i}{V_c} \tag{25}$$

$$l_{ij} + r_{ij} + u_{ij} + o_{ij} + b_{ij} + f_{ij} \ge s_i + s_j - 1 \qquad \forall i, j = 1, \cdots, k \tag{26}$$

$$x_i - x_j + Wl_{ij} \le W - w_i \forall i, j = 1, \cdots, k \tag{27}$$

$$x_j - x_i + Wr_{ij} \le W - w_j \forall i, j = 1, \cdots, k \tag{28}$$

$$y_i - y_j + Hu_{ij} \le H - (p_i + q_i) \forall i, j = 1, \cdots, k \tag{29}$$

$$y_j - y_i + Ho_{ij} \le H - \left(p_j + q_j\right) \forall i, j = 1, \cdots, k \tag{30}$$

$$z_i - z_j + Db_{ij} \le D - (p_i - q_i) \forall i, j = 1, \cdots, k \tag{31}$$

$$z_j - z_i + Df_{ij} \le D - \left(p_j - q_j\right) \forall i, j = 1, \cdots, k \tag{32}$$

$$0 \le x_i \le W - w_i \forall i = 1, \cdots, k \tag{33}$$

$$0 \le y_i \le H - (p_i + q_i) \forall i = 1, \cdots, k \tag{34}$$

$$0 \le z_i \le D - (p_i - q_i) \forall i = 1, \cdots, k \tag{35}$$

$$p_i = (h_i + d_i) \forall i = n+1, ..k \tag{36}$$

$$q_i = (h_i - d_i) \forall i = n+1, ..k \tag{37}$$

$$\sum_{t=0}^{m} \alpha_{it}\varepsilon_{it} = p_i \forall i = 1, \cdots, n \tag{38}$$

$$\sum_{t=0}^{m} \varepsilon_{it} = 1 \forall i = 1, \cdots, n \tag{39}$$

$$\varepsilon_{i0} \le \omega_{i0} \forall i = 1, \cdots, n \tag{40}$$

$$\varepsilon_{it} \le \omega_{i,t-1} + \omega_{i,t} \forall t = 1, \cdots, m-1 \tag{41}$$

$$\varepsilon_{i,m} \le \omega_{i,m-1} \forall i = 1, \cdots, n \tag{42}$$

$$\sum_{t=0}^{m-1} \omega_{it} = 1 \forall i = 1, \cdots, n \tag{43}$$

$$\sum_{t=0}^{m} \beta_{it}\delta_{it} = q_i \forall i = 1, \cdots, n \tag{44}$$

$$\sum_{t=0}^{m} \delta_{it} = 1 \forall i = 1, \cdots, n \tag{45}$$

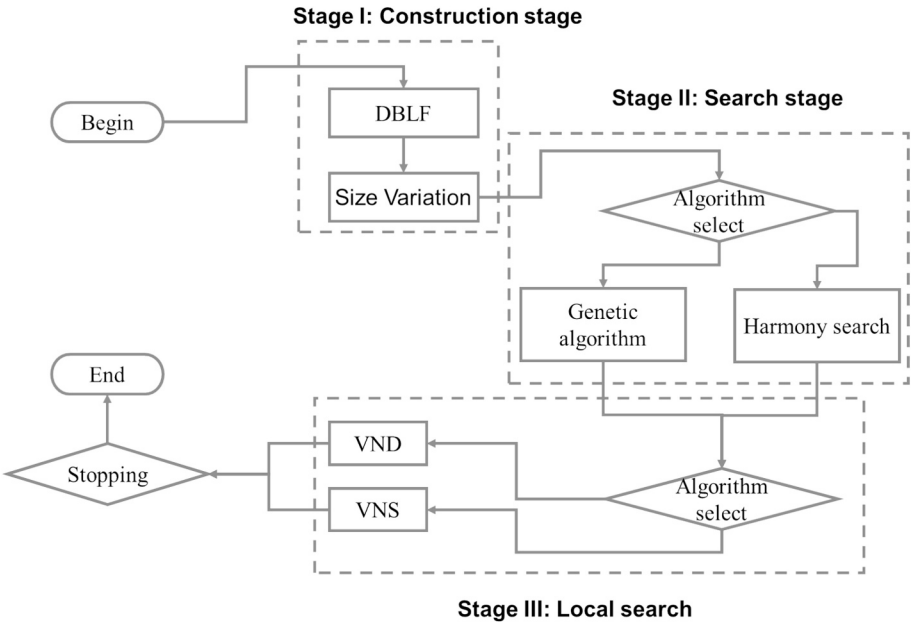

Fig. 4. The proposed solution framework.

$$\delta_{i0} \le \varphi_{i0} \forall i = 1, \cdots, n \tag{46}$$

$$\delta_{it} \le \varphi_{i,t-1} + \varphi_{i,t} \forall t = 1, \cdots, m-1 \tag{47}$$

$$\delta_{i,m} \le \varphi_{i,m-1} \forall i = 1, \cdots, n \tag{48}$$

$$\sum_{t=0}^{m-1} \varphi_{it} = 1 \forall i = 1, \cdots, n \tag{49}$$

$$\sum_{t=0}^{m} \alpha_{it}^2 \varepsilon_{it} - \sum_{t=0}^{m} \beta_{it}^2 \delta_{it} = h_i \times d_i \forall i = 1, \cdots, n \tag{50}$$

$$\frac{1}{2}(h_i^{LB} + d_i^{LB}) \le p_i \le \frac{1}{2}(h_i^{UB} + d_i^{UB}) \forall i = 1, \cdots, n \tag{51}$$

$$\frac{1}{2}(h_i^{LB} - d_i^{UB}) \le q_i \le \frac{1}{2}(h_i^{UB} - d_i^{LB}) \forall i = 1, \cdots, n \tag{52}$$

$$l_{ij}, r_{ij}, u_{ij}, o_{ij}, b_{ij}, f_{ij} \in \{0,1\} \forall i, j = 1, \cdots, k \tag{53}$$

$$s_i \in \{0,1\} \forall i = 1, \cdots, k \tag{54}$$

$$\delta_{it}, \varepsilon_{it} \ge 0, \varphi_{it}, \omega_{it} \in \{0,1\} \forall t = 0, \cdots, n-1 \tag{55}$$

$$x_i, y_i, z_i \ge 0 \forall i = 1, \cdots, n \tag{56}$$

In the model, the formula (25) is the objective function, maximizing the sum of the volumes of goods placed in the container. The formula (26) ~ (32) means that there is no overlap between the goods placed in the container; (33)–(35) indicates that the goods are entirely placed in the container and there is no overlapping with the container; Equations (36)–(52) are a linearized representation of the constraint (5), which introduces $2mn$ binary decision variables and $2\ n(m+1)$ continuous variables(ε_{it}, δ_{it}) to ensure that the cargo volume after the change is constant; The formulas (53)–(56) represent decision variable constraints.

The model is a mixed integer programming model whose complexity is defined by the quantity of goods, the quantity of goods with variable sizes, and the number of segments in the linearization process. There are $6k^2+k+4mn$ binary decision variables and $3k+2mn$ continuous variables.

4. Solution framework

The above proposed linear mixed integer programming model could be solved by solvers for small-scale problems. For large-scale problems, a more effective method should be designed. Many previous studies have used computational intelligence methods to solve complex combinatorial problems (Zhou et al., 2025). The problem studied is more complex to solve than the previous 3D-BPP. To solve the studied problem, this paper decomposes it into three subproblems: the construction stage, the search stage, and the local search stage. The following content introduces the proposed solution framework (see Fig. 4).

4.1. Construction stage

In the construction stage, the cargo loading sequence is given. This stage aims to load the cargo into the given bin according to the given sequence. In this paper, there are two types of cargoes. We first load the fixed-size cargoes. Then, we load the variable-sized cargoes. Deepest Bottom Left with Fill (DBLF), which was proposed by Korhan and Mustafa (2004), is adopted for the fixed-size cargoes. After loading the fixed-size cargoes, we will collect the cargoes that have been placed (***IP***). The set of potential positions (**List *PP***) and a collection of remaining space (**List *MS***). The set of variable-size goods that have not yet been placed is ***I***, where $|\boldsymbol{I}| = n$. The following algorithm introduces the method for loading variable-sized cargoes.

Algorithm 1:Size change algorithm

Input: IP, List PP *and* List *MS*
for $i = 1$ **to** n **do**
for $j = 1$ **to** $|MS|$ **do**
if $max\_v_j > v_i$ and $max\_v_j < v_{fit}$ **then**
$V_{fit} = max\_v_j$
$fitness\_point = j$
$w_{max} = max\_w_j$
$h_{max} = max\_h_j$
$d_{max} = max\_d_j$
end if
end for
if $h_i > h_{max}$ **then**
$h_i = h_{max}$
$d_i = \frac{v_i}{w_i h_i}$
end if

(*continued on next page*)

(*continued*)

Algorithm 1:Size change algorithm

15: **if** $d_i > d_{max} d_i > d_{max}$ **then**
16: $d_i = d_{max}$
17: $h_i = \frac{v_i}{w_i d_i}$
18: **end if**
19: **if** cargo i can be placed at point j **then**
20: place cargo i at point j
21: update **List *PP*** and **List *MS*** by **DBLF**
22: **end if**
23: **end for**
24: **end**

In the above algorithm, V_{fit} is the volume of the maximum remaining space at the optimal placement point. We assume that $V_{fit} = M$, where M is the maximum value. max_v_j represents the volume of the remaining space, where $max_{v_j} = max_{w_j} \times max_{h_j} \times max\_d_j$.

A construction heuristic algorithm based on size change is designed to combine the above-mentioned size change scheme with the DBLF loading strategy. First, the goods are loaded in sequence according to the given loading order using the DBLF loading strategy. After the first round of packing is completed, the second round of packing is carried out, that is, the size of the unloaded variable-length goods is changed in turn and then packed. The algorithm pseudo code is shown in algorithm 2.

Algorithm 2:Packing Strategy Based on Size Variation

Input: List *PP*
1: **for** $i = 1$ to n **do**
2: **for** $j = 1$ **to** $|PP|$
3: **if** goods i can be placed at potential placement point j **then**
4: the goods i will be placed at the potential place j
5: update List PP with **DBLF**
6: **break**
7: **end if**
8: **end for**
9: **end for**
10: update the collection of goods with variable sizes that are not included ***I***
11: **for** $i = 1$ **to** n **do**
12: **if** cargo i is a variable-size cargo and is not placed in container C **then**
13: the size will be changed with **algorithm 1**
14: **break**
15: **end if**
16: **end for**
17: **end**

4.2. Search stage

In the construction stage, we will generate a loading configuration by using the given loading sequence. By using the configuration generated by the construction, we can calculate the objective function. Thus, we can evaluate the fitness of any loading sequence. In this subsection, two computational intelligence methods are introduced, which are shown below.

4.2.1. Genetic algorithm

The following contents introduce the core operation in genetic algorithm design.

4.2.1.1. Encoding and decoding. This paper uses a vector of integers to denote the loading sequences, which is the encoding of the genetic algorithm design. Algorithm 2 is adopted as the decoding method to calculate the objective function.

4.2.1.2. Hybrid population initialization. This paper adopts the random and probability-based methods to initialize the population. Random initialization is a pure random method to generate an integer vector, and probability-based initialization methods are also random initialization methods, in which the cargo volume is adopted as the probability for generating the loading sequence.

4.2.1.3. Genetic operation. In parent selection, the roulette wheel method is adopted. In crossover operation, one-point and two-point crossover are adopted (see Fig. 5).

Gene transposition is adopted as the mutation operation. Repair operation is shown in Fig. 6.

4.2.2. Harmony search

We adopt a random key encoding and decoding in the harmony search algorithm design to represent the harmony memory. Fig. 7 shows an example of the harmony encoding and decoding method.

In the random key encoding and decoding method adopted in this paper, the element of harmony memory is a float number. *HM* can be defined as follows.

$$HM = \begin{bmatrix} X^1 \\ X^2 \\ \vdots \\ X^{HMS} \end{bmatrix} = \begin{bmatrix} x_1^1 x_2^1 \cdots x_k^1 | f(X^1) \\ x_1^2 x_2^2 \cdots x_k^2 | f(X^2) \\ \vdots \\ x_1^{HMS} x_2^{HMS} \cdots x_k^{HMS} | f(X^{HMS}) \end{bmatrix} \tag{57}$$

Thus, we could use the following two equations to generate a new element of the harmony memory for both x_i^{new} from and not from the current *HM*, respectively.

$$x_i^{new} \leftarrow \begin{cases} x_i^{new} \in \{x_i^1, x_i^2, \ldots, x_i^{HMS}\} \\ x_i^{new} \in (x_i^L, x_i^U) \end{cases}, HMCR \tag{58}$$

$$x_i^{new} \leftarrow \begin{cases} x_i^{new} \pm bw^*rand \\ x_i^{new} \end{cases}, PAR, 1 - \mathrm{PAR} \tag{59}$$

rand is a random floating number generator between 0 and 1. $x_i^{new} = x_i^{new} + bw^*rand$ when $rand < 0.5$, and $x_i^{new} = x_i^{new} - bw^*rand$ when $rand \geq 0.5$. The fine-turned variable x_i^{new} still needs to satisfy $x_i^{new} \in (x_i^L, x_i^U)$.

4.3. Local search stage

In the local search stage, this paper designs variable neighbourhood search methods as a local search to improve a given solution.

4.3.1. Neighbourhood design

Based on the neighbourhood action, we need to design the neighbourhood structure N_k of a given solution, where k denotes the k^{th} neighborhood. In this paper, the maximum value of k is 4. The following contents introduce the N_1, N_2, N_3 and N_4.

(1) Neighbourhood structure N_1: N_1 is generated by a two-point swap operation, which is similar to the two-point swap mutation operation.
(2) Neighbourhood structure N_2: N_2 is generated by the basis of fragment flipping, which is similar to the inversion mutation in genetic algorithms. A subset of the loading sequence is chosen, and then we invert the entire string in the subset.
(3) Neighbourhood structure N_3: N_3 is generated by a random insertion operation, in which a randomly selected loading sequence is selected and randomly inserted into a random position.
(4) Neighbourhood structure N_4: N_4 is generated by rearrangement operation. In the rearrangement operation, we first select several piece loading sequences and insert them into the head of the loading sequence.

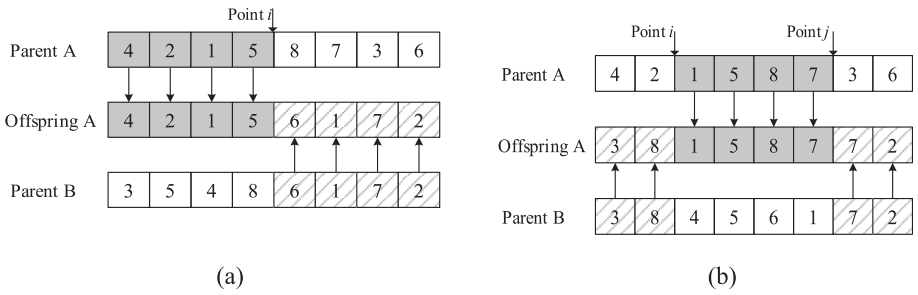

Fig. 5. One-point and two-point operation.

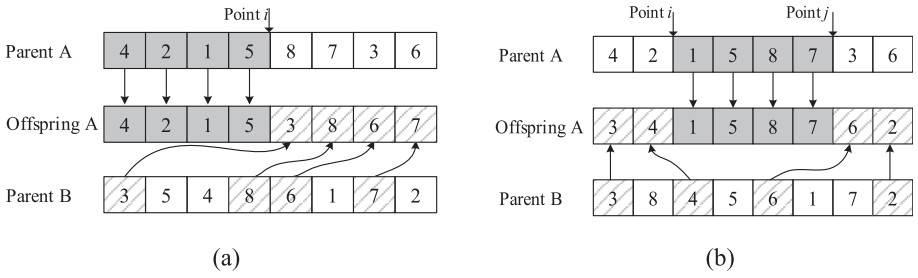

Fig. 6. Repair solutions for one-point and two-point intersections.

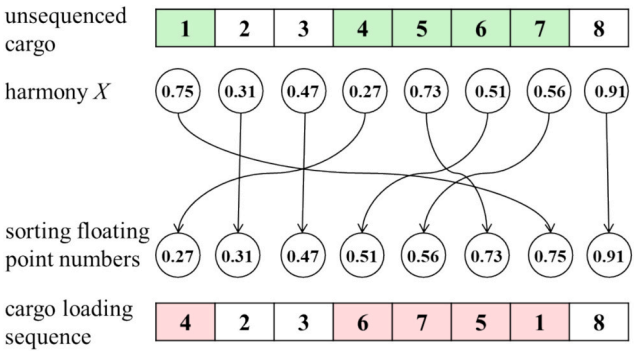

Fig. 7. Using floating point numbers to obtain cargo loading order.

Fig. 8 shows four examples of N_1, N_2, N_3 and N_4.

4.3.2. Search strategies

In this paper, two search strategies are adopted, which are variable neighborhood descent (VND) and variable neighbourhood search (VNS).

5. Experimental results

5.1. Experimental environment

The experimental environments are as follows: CPU is Intel (R) Core (TM) i5-12500H @2.50 GHz, memory is 32 GB, and the operating system is Windows 11. In this paper, IBM ILOG CPLEX 12.6 is used to solve the mixed integer programming model. The hybrid heuristic algorithm is written in C++, and the code is compiled through Dev C++ 5.11 with the GCC compiler.

5.2. Benchmark data set

Through the investigation of the actual transportation process of live fish in a logistics enterprise, it is found that live fish are often packed in water bags and transported in cartons. According to the classic test data set of the 3D packing problem, two groups of example data of different sizes are set up. In the small-scale test case, 10 cargoes of different sizes are randomly generated to test the performance of the mixed integer programming model and the heuristic algorithm. The dimensions of container C in the case are $W = 569, H = 213, D = 218$, W $= 569$, H $= 213$, D $= 218$W $= 569$, H $= 213$, D $= 218$. Here, the parameter generation rules for cargoes in the small-scale example are shown in Table 2:

Here, the generation rules of cargo size for a small-scale example

Table 2
Rules for generating parameters of cargo size in small-scale examples.

Range of the cargo length	Range of the cargo width	Range of the cargo height	Type of goods
$[1, \frac{1}{2}W]$	$[\frac{2}{3}H, H]$	$[\frac{2}{3}D, D]$	$[0, 1]$

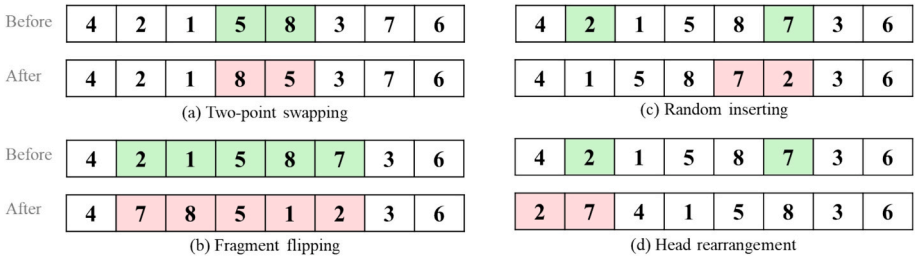

Fig. 8. Examples of N_1, N_2, N_3 and N_4.

come from Martello et al. (2000), whose literature proposed a variety of rules for generating cargo data sets for 3D packing, which are widely used as benchmark data sets. Since the generated data sets are all fixed-size cargoes, and to distinguish between variable-size and fixed-size cargoes, this paper randomly generates the cargo type through the rand function, where 1 represents variable-size cargoes and 0 represents fixed-size cargoes.

Large-scale experiments are designed to test the performance of the algorithm in solving large-scale calculation examples. The large-scale calculation examples come from the widely used classical test data sets BR 1-BR 7, which were proposed by Bischoff and Ratcliff (1995). The calculation examples in large-scale testing are randomly selected from BR1 ~ BR7. The number of goods included in each calculation example is between 80 and 140, and the heterogeneity of goods in the examples, BR1 − BR7, is gradually enhanced. Similarly, BR does not distinguish between cargo types and uses the rand function to randomly generate the type of cargo to distinguish variable-size cargo from fixed-size cargo.

5.3. Parameter tuning

To determine the parameters of the proposed genetic algorithm and harmony search, the Taguchi method is adopted, which can effectively determine the parameters of metaheuristics in the algorithm and has been used in many previous studies (Xin et al. 2024; Zhou and Lee 2020).

5.3.1. Genetic algorithm

The Taguchi method is adopted to determine the parameters of the genetic algorithm, including population size, crossover probability, mutation probability, and maximum iterations. The optimal tuning parameters are shown in Table 3. The details of the Taguchi method for determining the parameters of the genetic algorithm are shown in Appendix A.

5.3.2. Harmony search

In harmony search, we need to determine four parameters: harmony memory size, harmony memory considering rate, pitch adjusting rate, and maximum number of iterations. The details of the Taguchi method for determining parameters of harmony search are shown in Appendix B. The determined parameters of the harmony search algorithm are shown in Table 4.

5.3.3. Perturbation factors

This paper conducted the following experiments to determine the perturbation factor for variable neighbor search. GA represents the genetic algorithm. HS represents the harmony search algorithm. M1 represents the variable neighbourhood search algorithm, of which neighbourhood action 1 is the perturbation strategy (See Table 5).

5.4. Comparison between mathematical model and computational intelligence methods

This subsection compares the solutions obtained by the mathematical model and computational intelligence methods.

Table 3
Parameter setting of genetic algorithm.

Parameters of the genetic algorithm	Level
Population size (***P***)	30
Crossover probability (***P*** $_c$)	0.8
Mutation probability (***P*** $_m$)	0.25
Iterations (***Iter***)	300

Table 4
Parameter setting of harmony search algorithm.

Parameters of the harmony search algorithm	level
Harmony Memory Size (HMS)	50
Harmony Memory Considering Rate (*HMCR*)	0.85
Pitch Adjusting Rate (*PAR*)	0.25
Maximum number of Iterations (*MaxIter*)	220

5.4.1. Small-scale problems

Based on the characteristics of the exact algorithm, which applies to small-scale examples and can solve them accurately, small-scale example experiments are designed to verify the effectiveness of the two hybrid algorithms and whether they can reach the global optimum. Therefore, the experimental results of the two algorithms are compared with the results of the mixed integer programming model solved by the CPLEX solver to verify the effectiveness of the algorithm.

In the small-scale experiment, based on 10 examples, the CPLEX solver was used to solve the mixed integer programming model. The hybrid genetic algorithm and the hybrid harmony search algorithm are both implemented in C++. The calculation example was run independently 10 times, each time to obtain the average loading rate and average running time. Since the purpose of this experiment is to verify whether the two hybrid heuristic algorithms can obtain the global optimal solution or not, the *Gap* values between algorithms and models, and between the algorithms are calculated through formula (60) and compared, and Gap_1 & Gap_2 are regarded as the Gap values of loading rate and the average operation time, respectively.

$$Gap = \frac{l_1 - l_2}{l_1} \times 100\% \tag{2}$$

The l in the formula represents the algorithm's loading rate or computation time.

The experimental results of the model and algorithm are shown in Table 6, which includes the loading rate and running time of the mixed integer programming model, as well as the optimal, worst, average loading rate, and average running time of the hybrid genetic algorithm and the hybrid harmony search algorithm.

According to the experimental results, the Hybrid integer programming model, hybrid genetic algorithm and hybrid harmony search algorithm are compared pairwise, obtaining the *Gap* value of loading rate and operation time, respectively. The results are shown in Table 7.

The experimental results show that the hybrid genetic algorithm (HGA) can get the global optimal solution every time, proving the stability of HGA. As can be seen from Table 7, the medians of the three comparisons are all 0, which shows that the hybrid integer programming model constructed in this paper and the variable size optimization algorithm designed can obtain the global optimal solution, and the correctness and effectiveness of the model and algorithm are verified. At the same time, the average running times of the model and the two algorithms are 3.83 s, 1.49 s, and 0.47 s, respectively. And the value of the hybrid integer programming model Vs the hybrid genetic algorithm is 60.32 %, the value of the hybrid integer programming model Vs the hybrid harmony search algorithms is 87.37 %, and the value of the hybrid genetic algorithm Vs the hybrid harmony search algorithm is 65.78 %. Compared with the mixed integer programming model, both of the two variable-size optimization algorithms can obtain the optimal solution in a shorter time, with good convergence, and obtain the ideal loading plan in a shorter time.

In a word, in the small-scale example experiments, the model and algorithms designed in this paper are able to obtain the global optimal solution within a certain period of time, verifying the correctness of the model and algorithm. From the perspective of running time, the hybrid genetic algorithm and the hybrid harmony search algorithm can be used to obtain the optimal solution in a relatively short time, which shows the effectiveness of the two algorithms.

Table 5
Experimental results of perturbation strategy.

Methods		Example 1	Example 2	Example 3	Example 4	Example 5	Average loading rate (%)	Running time (s)
		Loading rate (%)	Loading rate (%)	Loading rate (%)	Loading rate (%)	Loading rate (%)		
GA	M1	86.54	**86.19**	**80.54**	87.52	80.92	**84.34**	94.49
	M2	86.13	86.11	80.15	87.41	80.59	84.08	104.05
	M3	**86.80**	85.94	79.77	87.66	**81.05**	84.25	**90.17**
	M4	86.13	85.77	79.35	**87.81**	80.55	83.92	98.17
HS	M1	79.47	83.46	**76.46**	81.53	77.24	79.63	**193.98**
	M2	**79.85**	**83.88**	76.34	82.57	76.62	**79.85**	231.52
	M3	78.39	**83.88**	74.94	**82.85**	77.34	79.48	227.21
	M4	77.60	**83.88**	75.55	80.77	**79.97**	79.55	327.18

Table 6
Experimental results of a small-scale example.

Example	Hybrid integer programming model		Hybrid genetic algorithm				Hybrid harmony search algorithm			
	Loading rate/%	Running time/s	Loading rate/%			Running time/s	Loading rate/%			Running time/s
			Minimum	Maximum	Average		Minimum	Maximum	Average	
test_1	90.85	2.92	90.85	90.85	90.85	1.39	84.08	90.85	84.92	0.54
test_2	74.52	4.71	74.52	74.52	74.52	1.35	70.77	74.52	72.75	0.55
test_3	72.73	4.58	72.73	72.73	72.73	1.04	67.17	72.73	69.77	0.48
test_4	83.63	4.56	83.63	83.63	83.63	1.50	79.21	83.63	80.10	0.38

Example	Hybrid integer programming model		Hybrid genetic algorithm				Hybrid harmony search algorithm			
	Loading rate/%	Running time/s	Loading rate/%			Running time/s	Loading rate/%			Running time/s
			Minimum	Maximum	Average		Minimum	Maximum	Average	
test_5	84.35	3.32	84.35	84.35	84.35	1.34	84.35	84.35	84.35	0.46
test_6	79.02	3.22	79.02	79.02	79.02	2.05	79.02	79.02	79.02	0.43
test_7	76.58	3.44	76.58	76.58	76.58	1.08	72.65	76.58	74.69	0.39
test_8	80.27	4.55	80.27	80.27	80.27	2.13	80.27	80.27	80.27	0.40
test_9	84.12	4.09	84.12	84.12	84.12	2.18	84.12	84.12	84.12	0.57
test_10	83.17	2.86	83.17	83.17	83.17	0.84	83.17	83.17	83.17	0.46
Average	**80.93**	**3.83**	**80.93**	**80.93**	**80.93**	**1.49**	**72.69**	**80.93**	**79.32**	**0.47**

Table 7
Gap value comparison.

Example	Hybrid integer programming model VsHybrid genetic algorithm		Hybrid integer programming model VsHybrid harmony search algorithms		Hybrid Inheritance Algorithm VsHybrid harmony search algorithms	
	Gap(%)		Gap(%)		Gap(%)	
	Gap_1	Gap_2	Gap_1	Gap_2	Gap_1	Gap_2
test_1	0	52.40	0	81.51	0	61.15
test_2	0	71.34	0	88.32	0	59.26
test_3	0	77.29	0	89.52	0	53.85
test_4	0	67.11	0	91.67	0	74.67
test_5	0	59.64	0	86.14	0	65.67
test_6	0	36.34	0	86.65	0	79.02
test_7	0	68.60	0	88.66	0	63.89
test_8	0	53.19	0	91.21	0	81.22
test_9	0	46.70	0	86.06	0	73.85
test_10	0	70.63	0	83.92	0	45.24
average	**0**	**60.32**	**0**	**87.37**	**0**	**65.78**

5.4.2. Large-scale problems

In large-scale examples, with the increase of the number of goods, the difficulty of the exact algorithm, i.e., the mixed integer programming model, increases, and the running time increases exponentially, resulting in the model's optimal solution not being obtained within a specific period of time. Therefore, in large-scale comparative experiments, this paper did not use CPLEX to solve the hybrid integer programming model as a contrast, but only compared the hybrid harmony search algorithm with the hybrid genetic algorithm. Similarly, due to the increase in the number of goods, to ensure the stability of the heuristic algorithm, this paper independently runs 12 groups of large-scale examples 10 times each time, takes the average of the loading rate and running time, and then compares the results of the heuristic algorithm with the results of the hybrid integer programming model, compares the differences between the two algorithms in average loading rate and average running time to calculate the results between the two algorithms. The results of the large-scale experiment are shown in Table 8.

The table above lists the calculation results of 12 large-scale examples calculated by the hybrid genetic algorithm and the hybrid harmony search algorithm, mainly including the optimal loading rate, the worst loading rate, the average loading rate, and the average running time. Gap_1 and Gap_2 come out through calculation. We can see that the average loading rates of the hybrid harmony search algorithm and the hybrid genetic algorithm are 76.65 % and 81.89 %, respectively. The hybrid genetic algorithm has a higher solving ability than the hybrid harmony search algorithm, with an average value of −6.84 %, which verifies the convergence and effectiveness of the hybrid genetic algorithm. From the comparison of running time, the average running time of the hybrid harmony search algorithm is 235.43 s, while the average running time of the hybrid genetic algorithm is 127.81 s, with an average value of 45.71 %. Through the above comparison, it is found that compared with the hybrid harmony search algorithm, the hybrid genetic algorithm significantly shortens the solution time and can obtain a better solution in a shorter time. It has good convergence and can quickly provide customers with an ideal loading solution.

The iterative curve of the two algorithms is plotted to compare the convergence of the hybrid genetic algorithm and the hybrid harmony algorithm. The two algorithms have different internal designs, so the maximum number of iterations set by the algorithm cannot be used as the independent variable. Here, the fitness evaluation times represent the independent variable, and the iteration curve, as shown in Fig. 9, is drawn, with the target value representing the dependent variable. Due to limited space, this article will show the algorithm iteration curves of

Table 8
Large-scale example experimental results.

Calculation example	Hybrid harmony search algorithm				Hybrid genetic algorithm				*Gap*/%	
	Loading rate/%			Running time/s	Loading rate/%			Gap_1		Gap_2
	Minimum	Maximum	Average		Minimum	Maximum	Average			
BR1-1	76.82	79.42	77.98	292.19	85.83	87.12	86.83	155.35	−11.35	46.83
BR1-2	82.21	84.71	82.96	459.36	85.94	87.61	86.15	175.54	−3.84	61.79
BR2-1	74.45	77.42	75.47	362.16	78.39	82.32	80.83	39.38	−7.10	89.13
BR2-2	80.63	81.72	80.98	264.95	87.29	87.71	87.49	114.50	−8.04	56.78
BR3-1	75.99	78.48	76.80	92.26	80.06	84.79	81.84	90.85	−6.56	1.53
BR3-2	74.59	79.49	77.50	123.00	81.72	86.17	83.67	112.90	−7.96	8.21
BR4-1	75.39	76.20	75.84	139.26	80.20	82.01	81.00	78.35	−6.81	43.74
BR4-2	73.50	76.19	74.34	193.02	79.19	82.64	80.87	152.10	−8.79	21.20
BR5-1	74.39	76.86	75.64	97.71	76.26	79.31	77.46	70.48	−2.41	27.87
BR5-2	72.07	75.06	73.69	232.23	76.30	82.70	79.28	207.26	−7.59	10.75
BR6-1	73.52	75.53	74.57	285.54	76.88	77.57	77.06	165.24	−3.33	42.13
BR7-1	73.79	74.26	74.04	283.42	77.38	82.55	80.22	171.73	−8.35	39.41
average	**75.61**	**77.95**	**76.65**	**235.43**	**80.45**	**83.54**	**81.89**	**127.81**	**−6.84**	**45.71**

the following five examples.

5.5. Ablation experiment

In order to verify the impact of the variable neighbourhood search algorithm on the performance of the genetic algorithm, an ablation experiment was designed to compare the performance changes of the hybrid genetic algorithm before and after the combination of the genetic algorithm and the variable neighbourhood search algorithm, and the impact of VND on the algorithm. VND and VNS are combined with the genetic algorithm separately and form two algorithms, GA + VNS and GA + VND. Through the experiments, the algorithm iteration curve diagram is obtained after the genetic algorithm is combined with VNS and VND. As is shown in Fig. 10, the three curves represent the iteration curves of the GA, GA + VND, and GA + VNS algorithms, respectively. The vertical axis represents the target value of the optimal individual in the population, and the horizontal axis represents the fitness evaluation times. Due to limited space, this paper will show the algorithm iteration curve diagrams of the following five examples.

It can be found that the fitness evaluation times of the genetic algorithm without a local search strategy are far less than those of the other two algorithms. Although the convergence of the algorithm is good, it is easy to fall into a local optimum and not obtain the global optimal solution. Due to the lack of a perturbation strategy, the GA + VND algorithm also fell into the local optimum early, and the convergence speed is not as fast as the genetic algorithm; As for the GA + VNS algorithm combined with VNS, its fitness evaluation times are far more than that of the other two algorithms, and the quality of its solution is significantly improved, reflecting the optimization of the algorithm. Experiments have proved that the hybrid genetic algorithm combined with the variable neighbourhood search algorithm can not only improve the algorithm's solving ability through multiple neighbourhood structures, but also effectively prevent the algorithm from falling into the local optimum through the perturbation strategy. Therefore, the variable neighbourhood search algorithm designed in this paper has a significant impact on improving the solution ability of the genetic algorithm.

5.6. Management insights

This paper proposed a solution method for packing variable-sized fresh products into a fixed-sized container. This solution method could help the decision makers to pack variable-sized fresh products to reduce the reliance on human judgment, which can be subjective and time-consuming, especially when dealing with a large variety of product sizes and bin configurations.

The method's ability to handle the variability in fresh-product lengths is a key strength. Fresh produce comes in diverse shapes and sizes, and packing it efficiently has always been a headache for logistics managers. The new approach takes into account these variations and provides optimized packing solutions, ensuring that the available bin space is utilized to its maximum potential. This leads to improved space utilization rates, which can be a game-changer in an industry where transportation and storage costs are major cost drivers.

For the logistics managers, in actual logistics work scenarios, they may get optimized cargo loading plans via user-friendly software interfaces developed based on this framework, by inputting information such as the size, quantity, whether the goods are variable in size, and the specifications of the container, ultimately improving the loading rate of containers and reducing transportation costs. The method can be used as a tool to enforce sustainable packaging regulations for the policymakers as well. For an example, the policymakers could encourage and supervise related enterprises to follow the methods in the framework to design better packaging plans while meeting the needs of product preservation, avoiding materials waste.

6. Conclusions

Fresh production is crucial for the agricultural and food industries, bolstering the supply chain and generating income for farmers. In the domain of green logistics, the packing of fresh produce is of utmost importance. Fresh products typically come in a variety of three-dimensional dimensions for packaging. This study focuses on the problem of packing fresh products, some of which have diverse three-dimensional sizes, into fixed-size bins.

To address this, we first formulate the problem using a nonlinear mathematical model. Subsequently, a piecewise linearization approximation approach is employed to linearize the model. For solving large-scale instances of the problem, we propose a three-phase computational intelligence framework integrated with variable neighbourhood search. One of the advantages of this framework is its flexibility, as it can incorporate any computational intelligence method. In this study, we implement the proposed framework using a genetic algorithm and harmony search. Extensive experiments are carried out to validate the performance of the proposed solution framework. The experimental results demonstrate the effectiveness and efficiency of the proposed approach, providing valuable insights for optimizing fresh product packing in green logistics.

Despite the advantages of the computational intelligence method proposed for packing variable-length fresh products into fixed-size bins, several limitations must be acknowledged to provide a comprehensive understanding of its practical application and potential drawbacks. (1) This paper only considered one fixed-size bin. More bins could be considered. (2) In the fresh-produce industry, obtaining precise and up-

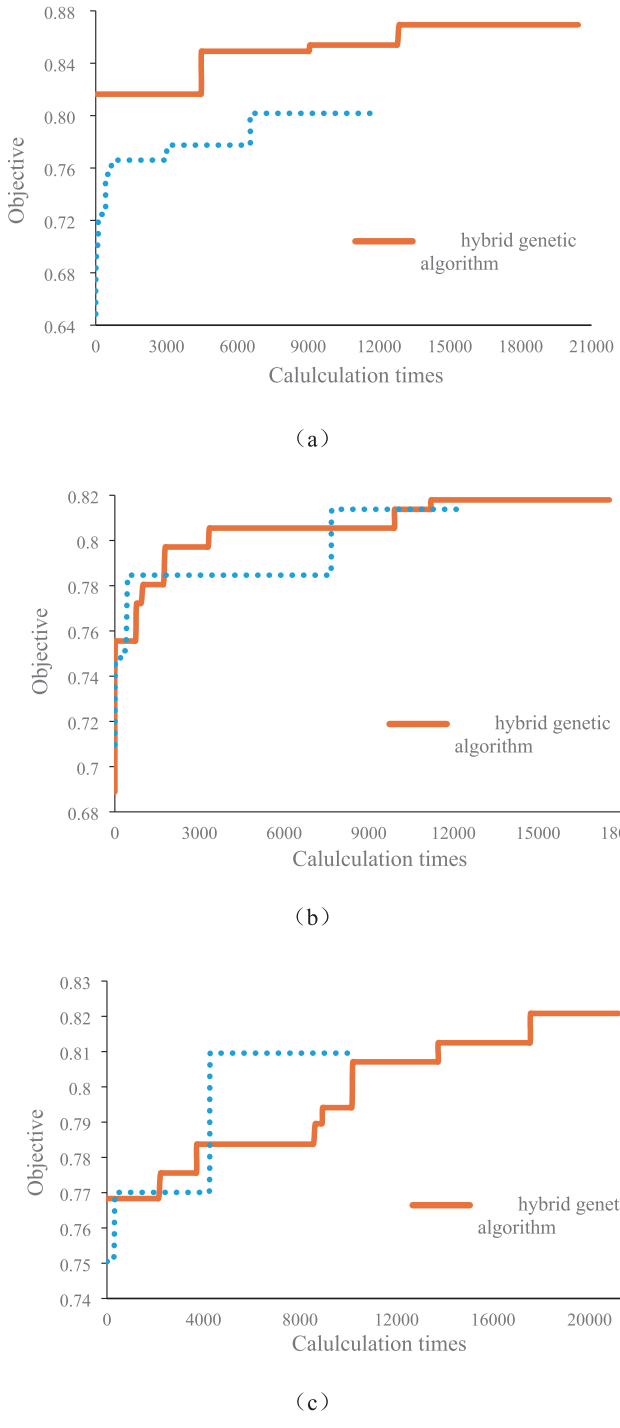

Fig. 9. Iterative curve of the hybrid genetic algorithm and the hybrid harmony algorithm.

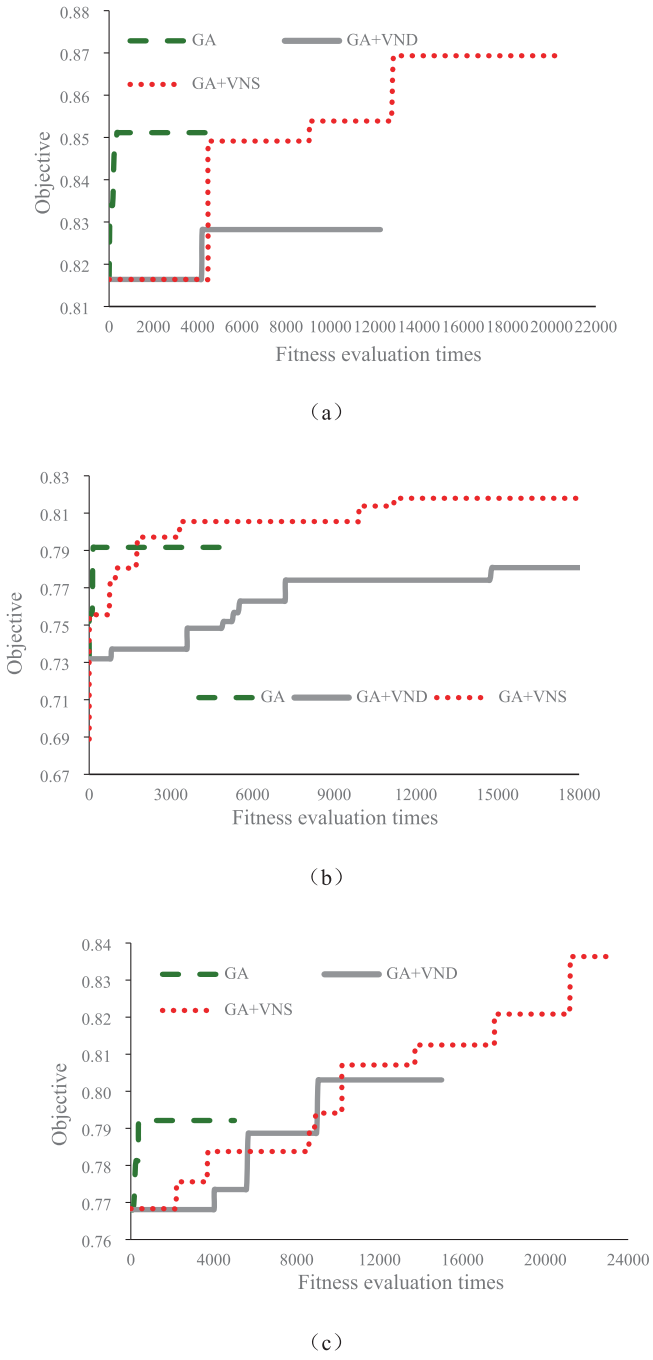

Fig. 10. Iteration curve of algorithms.

to-date data on product sizes, shapes, and weights can be challenging. More constraints could be considered for fresh products.

Future studies could consider the following aspects: (1) Reinforcement learning could be adopted for solving larger case problems. (2) More application of fresh products could be explored.

CRediT authorship contribution statement

Yanjie Zhou: Validation, Methodology, Writing – original draft, Project administration, Writing – review & editing, Supervision, Conceptualization. **Xiaojin Wang:** Writing – original draft, Investigation, Methodology, Visualization. **Hang Wang:** Software, Formal analysis, Conceptualization, Methodology. **Jiang Xu:** Writing – review & editing, Methodology, Supervision, Conceptualization, Writing – original draft.

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.

Acknowledgments

This research was supported by the National Natural Science

Foundation of China (72201252).

Appendix

Appendix A:. Nonlinear mixed integer programming model

The nonlinear mixed integer programming model could be obtained as follows.

$$P_N Max \frac{\sum_{i=1}^{n} v_i s_i}{VC} \tag{A1}$$

$$\begin{cases} 0 \le x_i \le W - w_i \\ 0 \le y_i \le H - h_i \\ 0 \le z_i \le D - d_i. \end{cases} \tag{A2}$$

$$l_{ij}+r_{ij}+u_{ij}+o_{ij}+b_{ij}+f_{ij} \ge 1 \tag{A3}$$

$$l_{ij} = 1 \Rightarrow \mathrm{x}_i + w_i \le x_j, r_{ij} \ = \ 1 \Rightarrow \mathrm{x}_j + w_j \le x_i \tag{A4}$$

$$u_{ij} = 1 \Rightarrow \mathrm{y}_i + h_i \le y_j, o_{ij} \ = \ 1 \Rightarrow \mathrm{y}_j + h_j \le y_i$$

$$b_{ij} = 1 \Rightarrow \mathrm{z}_i + d_i \le z_j, f_{ij} \ = \ 1 \Rightarrow \mathrm{z}_j + d_j \le z_i$$

$$\frac{v_i}{w_i} = h_i^{'} \times d_i^{'} = h_i \times d_i \tag{A5}$$

$$h_i^{LB} = h_i/2, h_i^{UB} = h_i \times d_i \tag{A6}$$

$$d_i^{LB} = h_i/2, d_i^{UB} = h_i \times d_i$$

Appendix B:. Genetic algorithm design

Parameter selection of genetic algorithm

The main parameters to be debugged in the genetic algorithm include population size (*P*), cross probability (*Pc*), mutation variation (*Pm*), and maximum number of algorithm iterations (*Iter*). Through prior algorithmic testing, five different levels of parameters, considered to be potentially optimal levels, have been designed. The overall parameter level design is shown in Table 9.

Table 9
Genetic algorithm parameter levels.

Serial number	Parameters of the genetic algorithm			
	P	*Pc*	*Pm*	*Iter*
1	20	0.5	0.05	140
2	30	0.6	0.1	180
3	40	0.7	0.15	220
4	50	0.8	0.2	260
5	60	0.9	0.25	300

Based on the above genetic algorithm parameter level table, this paper uses the Taguchi method to set up an orthogonal matrix table to design an experiment. As shown in Table 10.

Table 10
Experimental design of genetic algorithm.

Programme	Parameters of the genetic algorithm			
	P	*Pc*	*Pm*	*Iter*
1	20	0.5	0.05	140
2	20	0.6	0.1	180
3	20	0.7	0.15	220
4	20	0.8	0.2	260
5	20	0.9	0.25	300
6	30	0.5	0.1	220
7	30	0.6	0.15	260
8	30	0.7	0.2	300
9	30	0.8	0.25	140
10	30	0.9	0.05	180

(*continued on next page*)

Table 10 (*continued*)

Programme	Parameters of the genetic algorithm			
	P	*Pc*	*Pm*	*Iter*
11	40	0.5	0.15	300
12	40	0.6	0.2	140
13	40	0.7	0.25	180
14	40	0.8	0.05	220
15	40	0.9	0.1	260
16	50	0.5	0.2	180
17	50	0.6	0.25	220
18	50	0.7	0.05	260
19	50	0.8	0.1	300
20	50	0.9	0.15	140
21	60	0.5	0.25	260
22	60	0.6	0.05	300
23	60	0.7	0.1	140
24	60	0.8	0.15	180
25	60	0.9	0.2	220

In the orthogonal experiment, five large-scale examples were tested. The algorithm ran each calculation example 10 times independently, and the average loading rate and average run time came out. The calculation results are shown in Table 11:

Table 11
Results of genetic algorithm orthogonal experiment.

Programme	BR1-1	BR1-2	BR2-1	BR2-2	BR3-1	Average loading rate (%)	Running time (s)
	Loading rate (%)	Loading rate (%)	Loading rate (%)	Loading rate (%)	Loading rate (%)		
1	84.71	85.36	75.47	87.36	78.41	82.26	**0.75**
2	84.10	85.48	75.79	87.51	80.35	82.65	1.14
3	84.81	85.86	78.88	87.60	79.82	83.39	1.53
4	84.61	85.86	76.98	87.68	80.79	83.18	1.88
5	85.07	85.99	78.92	**87.70**	79.81	83.50	2.10
6	85.31	85.44	77.71	87.52	79.10	83.02	1.94
7	85.42	86.07	76.31	87.61	79.74	83.03	1.37
8	85.63	86.03	**79.13**	87.69	**80.82**	**83.86**	1.81
9	84.59	85.53	76.53	87.42	79.11	82.64	2.53
10	85.35	85.78	77.42	87.57	78.09	82.84	2.60
11	85.48	86.11	78.31	87.60	79.85	83.47	0.94
12	83.28	85.23	74.55	87.48	77.48	81.60	1.33
13	83.78	85.52	76.08	87.55	79.96	82.58	1.97
14	84.95	85.73	77.38	87.62	79.19	82.97	2.34
15	**86.31**	85.90	77.14	87.54	79.30	83.24	2.64
16	82.94	85.44	75.32	87.42	78.09	81.84	1.12
17	84.22	85.69	76.86	87.57	78.83	82.63	1.65
18	84.35	85.36	77.85	87.46	78.73	82.75	2.18
19	85.77	**86.28**	77.90	87.59	79.38	83.38	2.54
20	83.26	85.27	74.87	87.36	77.05	81.56	2.82
21	84.55	86.03	76.95	87.44	77.95	82.58	1.10
22	84.53	85.69	77.13	87.60	78.37	82.66	1.78
23	82.06	85.19	74.53	87.36	77.35	81.30	2.28
24	83.97	85.78	75.45	87.36	78.60	82.23	2.75
25	83.27	85.44	75.54	87.41	77.88	81.91	2.90

In order to determine the optimal parameter level configuration of genetic algorithm, Taguchi method is used to analyse the large-the-better S/N ratio with the formula and main effect diagram of signal-to-noise ratio (SNR) are shown in (61).

$$S/N = \log_{10}\left(\frac{\sum UF^2}{n}\right) \tag{61}$$

In the formula (61), *UF* represents the objective function, where the average load.
rate is represented, and *n* represents the number of times each calculation example runs separately.

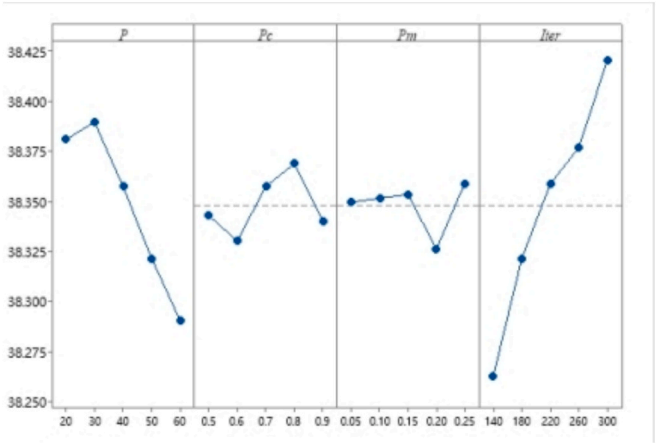

Fig. 11. Main effect diagram of signal-to-noise ratio (SNR).

Appendix C:. Harmony search design

Parameter selection of harmony search algorithm

Similarly, the Taguchi method is also used to determine the parameters of the harmonic search algorithm. The main parameters involved in the harmony search algorithm are the harmonic memory size (*HMS*), harmony memory considering rate (*HMCR*), pitch adjusting rate (*PAR*), and maximum number of iterations (*MaxIter*). First, based on previous algorithm tests, five levels of parameters were designed as potential optimal level parameters. The overall parameters are shown in Table 12.

Table 12
Parameters level of harmony search algorithm.

Serial number	Parameters of the harmonic search algorithm			
	HMS	*HMCR*	*PAR*	*MaxIter*
1	20	0.55	0.05	140
2	30	0.65	0.1	160
3	40	0.75	0.15	180
4	50	0.85	0.2	200
5	60	0.95	0.25	220

Based on the above parameter level table of the harmonic search algorithm, an orthogonal matrix table is set to design the experiment, as shown in Table 13.

Table 13
Experimental design of harmony search algorithm.

	Parameters of the harmonic search algorithm			
	HMS	*HMCR*	*PAR*	*MaxIter*
1	20	0.55	0.05	140
2	20	0.65	0.1	160
3	20	0.75	0.15	180
4	20	0.85	0.2	200
5	20	0.95	0.25	220
6	30	0.55	0.1	180
7	30	0.65	0.15	200
8	30	0.75	0.2	220
9	30	0.85	0.25	140
10	30	0.95	0.05	160
11	40	0.55	0.15	220
12	40	0.65	0.2	140
13	40	0.75	0.25	160
14	40	0.85	0.05	180
15	40	0.95	0.1	200
16	50	0.55	0.2	160
17	50	0.65	0.25	180
18	50	0.75	0.05	200
19	50	0.85	0.1	220
20	50	0.95	0.15	140
21	60	0.55	0.25	200

(*continued on next page*)

Table 13 (*continued*)

	Parameters of the harmonic search algorithm			
	HMS	*HMCR*	*PAR*	*MaxIter*
22	60	0.65	0.05	220
23	60	0.75	0.1	140
24	60	0.85	0.15	160
25	60	0.95	0.2	180

In the orthogonal experiment, five large-scale examples were tested. The algorithm ran each example 10 times independently and took its loading rate and running time. The calculation results are shown in Table 14.

Table 14
Orthogonal experiment results of harmony search algorithm.

	BR1-1	BR1-2	BR2-1	BR2-2	BR3-1	Average loading rate (%)	Running time (s)
	Loading rate (%)	Loading rate (%)	Loading rate (%)	Loading rate (%)	Loading rate (%)		
1	76.12	76.38	73.57	79.22	74.01	75.86	**4.05**
2	77.66	82.25	74.46	80.24	75.40	78.00	10.02
3	78.02	83.25	75.87	80.73	75.29	78.63	9.01
4	77.53	82.92	75.28	80.17	75.39	78.26	11.48
5	77.49	83.09	75.41	80.84	74.65	78.30	15.87
6	76.84	81.71	75.01	80.10	74.25	77.58	11.95
7	77.76	82.42	75.61	80.19	75.46	78.29	14.54
8	78.11	83.13	76.12	81.31	76.27	78.99	18.21
9	78.39	83.59	74.83	81.31	75.79	78.78	13.04
10	77.80	82.63	75.89	80.46	75.59	78.47	14.35
11	77.69	82.25	75.05	80.71	75.14	78.17	20.04
12	78.07	82.05	74.45	79.43	74.64	77.73	11.73
13	78.32	82.25	75.01	80.16	76.09	78.37	12.60
14	79.03	83.50	75.61	81.22	76.72	79.22	21.75
15	79.06	83.83	76.35	81.14	75.24	79.12	33.31
16	77.65	82.21	75.73	80.24	74.89	78.14	19.50
17	**79.24**	82.84	**77.86**	80.89	**77.36**	**79.64**	26.10
18	79.24	83.09	76.46	80.84	76.53	79.23	23.05
19	78.75	83.21	76.34	**81.76**	76.76	79.36	24.47
20	77.58	82.59	74.94	80.31	76.02	78.29	13.87
21	78.16	82.75	75.55	80.50	75.90	78.57	24.89
22	78.68	83.38	77.42	80.86	75.16	79.10	28.84
23	78.55	82.46	75.97	80.55	76.27	78.76	21.30
24	78.66	**83.88**	76.47	81.13	75.43	79.11	28.57
25	77.53	83.79	76.64	81.20	75.05	78.84	28.85

The signal–noise ratio is calculated according to formula (61), and draw the main effect diagram of the SNR is plotted. The results are shown in Fig. 12 Main effect diagram of SNR for harmony search algorithm.

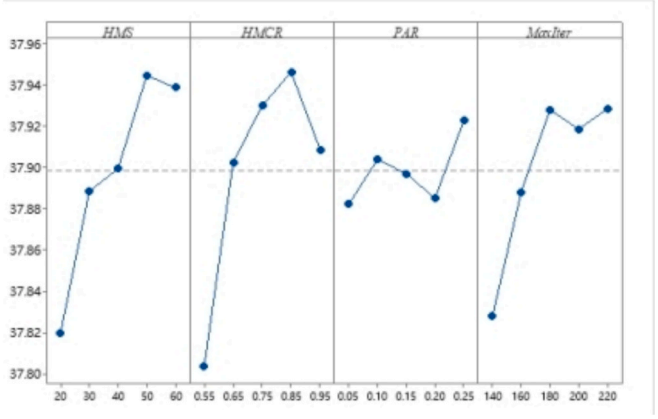

Fig. 12. Main effect diagram of SNR for harmony search algorithm.

Data availability

Data will be made available on request.

References

Alonso, M. T., Alvarez-Valdes, R., Iori, M., & Parreño, F. (2019). Mathematical models for multi container loading problems with practical constraints. *Computers & Industrial Engineering, 127*, 722–733.
Alonso, M. T., Alvarez-Valdes, R., & Parreño, F. (2019). A GRASP algorithm for multi container loading problems with practical constraints. *4OR, 18*(1), 49–72.

Ananno, A. A., & Ribeiro, L. (2024). A multi-heuristic algorithm for multi-container 3-d bin packing problem optimization using real world constraints. *IEEE Access, 12*, 42105–42130.
Bischoff, E. E., & Ratcliff, M. S. W. (1995). Issues in the development of approaches to container loading. *Omega, 23*(4), 377–390.
Ceschia, S., & Schaerf, A. (2011). Local search for a multi-drop multi-container loading problem. *Journal of Heuristics, 19*(2), 275–294.
Chen, C. S., Lee, S. M., & Shen, Q. S. (1995). An analytical model for the container loading problem. *European Journal of Operational Research, 80*(1), 68–76.
Coffman, E. G., Jr., Garey, M. R., Johnson, D. S., & Tarjan, R. E. (1980). Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing, 9*(4), 808–826.
Crainic, T. G., Perboli, G., & Tadei, R. (2008). Extreme point-based heuristics for three-dimensional bin packing. *INFORMS Journal on Computing, 20*(3), 368–384.
Egeblad, J., & Pisinger, D. (2009). Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers & Operations Research, 36*(4), 1026–1049.
Ekici, A. (2021). Bin packing problem with conflicts and item fragmentation. *Computers & Operations Research, 126*, Article 105113.
Erbayrak, S., Özkır, V., & Mahir Yıldırım, U. (2021). Multi-objective 3D bin packing problem with load balance and product family concerns. *Computers & Industrial Engineering, 159*, Article 107518.
Faroe, O., Pisinger, D., & Zachariasen, M. (2003). Guided local search for the three-dimensional bin-packing problem. *INFORMS Journal on Computing, 15*(3), 267–283.
Fleszar, K. (2022). A branch-and-bound algorithm for the quadratic multiple knapsack problem. *European Journal of Operational Research, 298*(1), 89–98.
Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2006). A tabu search algorithm for a routing and container loading problem. *Transportation Science, 40*(3), 342–350.
George, J. A., & Robinson, D. F. (1980). A heuristic for packing boxes into a container. *Computers & Operations Research, 7*(3), 147–156.
Hifi, M., Kacem, I., Negre, S., & Wu, L. (2010). Heuristics algorithms based on a linear programming for the three-dimensional bin-packing problem. *IFAC Proceedings Volumes, 43*(8), 72–76.
Jiang, Y., Cao, Z., & Zhang, J. (2021). Learning to solve 3-d bin packing problem via deep reinforcement learning and constraint programming. *IEEE Transactions on Cybernetics, 53*(5), 1–12.
Junqueira, L., Morabito, R., & Sato Yamashita, D. (2012). Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research, 39*(1), 74–85.
Kang, Z., Guan, Y., Wang, J., & Chen, P. (2024). Research on genetic algorithm optimization with fusion tabu search strategy and its application in solving three-dimensional packing problems. *Symmetry, 16*(4), 449.
Kilincci, O., & Medinoglu, E. (2021). An efficient method for the three-dimensional container loading problem by forming box sizes. *Engineering Optimization, 54*(6), 1–16.
Karabulut, K., & İnceoğlu, M. M. (2004). A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. *Lecture Notes in Computer Science, 3261*, 441–450.
Liu, Q., Cheng, H., Tian, T., Wang, Y., Leng, J., Zhao, R., Zhang, H., & Wei, L. (2021). Algorithms for the variable-sized bin packing problem with time windows. *Computers & Industrial Engineering, 155*, Article 107175.
Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations Research, 48*(2), 256–267.
Moon, I., & Nguyen, T. V. (2013). Container packing problem with balance constraints. *OR Spectrum, 36*(4), 837–878.
Nascimento, O. X. D., Alves de Queiroz, T., & Junqueira, L. (2021). Practical constraints in the container loading problem: Comprehensive formulations and exact algorithm. *Computers & Operations Research, 128*, Article 105186.
Paquay, C., Schyns, M., & Limbourg, S. (2014). 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.
Que, Q., Yang, F., & Zhang, D. (2023). Solving 3D packing problem using Transformer network and reinforcement learning. *Expert Systems with Applications, 214*, Article 119153.
Ren, J., Tian, Y., & Sawaragi, T. (2011). A priority-considering approach for the multiple container loading problem. *International Journal of Metaheuristics, 1*(4), 298.
Saraiva, R. D., Nepomuceno, N., & Pinheiro, P. R. (2015). A layer-building algorithm for the three-dimensional multiple bin packing problem: A case study in an automotive company. *IFAC-PapersOnLine, 48*(3), 490–495.
Sheng, L., Xiuqin, S., Changjian, C., Hongxia, Z., Dayong, S., & Feiyue, W. (2017). Heuristic algorithm for the container loading problem with multiple constraints. *Computers & Industrial Engineering, 108*, 149–164.
OsogamiH, T., & Okano. (2003). Local search algorithms for the bin packing problem and their relationships to various construction heuristics. *Journal of Heuristics, 9*(1), 29–49.
Wang, B., Lin, Z., Kong, W., & Dong, H. (2025). Bin packing optimization via deep reinforcement learning. *IEEE Robotics And Automation Letters, 10*(3), 2542–2549.
Xin, J., Yuan, Q., D'Ariano, A., Guo, G., Liu, Y., & Zhou, Y. (2024). Dynamic unbalanced task allocation of warehouse AGVs using integrated adaptive large neighbourhood search and Kuhn–Munkres algorithm. *Computers & Industrial Engineering, 195*, Article 110410.
Zhang, B., Yao, Y., Kan, H. K., & Luo, W. (2024). A GAN-based genetic algorithm for solving the 3D bin packing problem. *Scientific Reports, 14*(1), 7775.
Zhang, Z., Denton, B. T., & Xie, X. (2020). Branch and price for chance-constrained bin packing. *Informs Journal on Computing, 32*(3), 547–564.
Zhao, H., She, Q., Zhu, C., Yang, Y., & Xu, K. (2021). Online 3D bin packing with constrained deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence, 35*(1), 741–749.
Zhao, J., Zhou, L., Wang, F., Liu, H., & Yang, J. (2022). Multibox three-dimensional packing problems for heterogeneous extrudable items. *Mathematical Problems in Engineering, 2022*, 1–12.
Zhou, Y., He, Z., Liu, C., Zhang, J., Li, Y., & Wang, Y. (2025). Less-than-container cargo scheduling for China railway express along belt and road initiative routes. *Transportation Research Part E: Logistics and Transportation Review, 197*, Article 104066.
Zhou, Y., & Lee, G. M. (2020). A bi-objective medical relief shelter location problem considering coverage ratios. *International Journal of Industrial Engineering: Theory, Applications and Practice, 27*(6), 971–988.
Zhu, W., Oon, W.-C., Lim, A., & Weng, Y. (2012). The six elements to block-building approaches for the single container loading problem. *Applied Intelligence, 37*(3), 431–445.