Journal of Industrial and Management Optimization

https://www.aimspress.com/journal/jimo

JIMO, 22 (1): 310–335.
DOI: 10.3934/jimo.2026012
Received: 25 May 2025
Revised: 02 November 2025
Accepted: 02 November 2025
Published: 28 November 2025

Research article

Optimization of a parallel replenishment input sequence for a clustered storage "parts-to-picker" order picking system

Xin Wang and Yaohua Wu*

School of Control Science and Engineering, Shandong University, Jinan 250061, China

* **Correspondence:** Email: yaohua.wu@sdu.edu.cn.

Abstract: The clustered storage strategy can significantly improve the picking efficiency of the "parts-to-picker" order picking system, which has been widely used. The parallel warehousing strategy of multiple stock keeping units (SKU) can reduce the number of times of replenishment in and out of the same storage unit and save the system operation time. For the parallel replenishment optimization problem of the "parts-to-picker" order picking system under the clustered storage strategy, we established a linear programming model, calculated the closeness centrality of SKUs and the correlation between SKUs, and proposed a greedy algorithm based on SKU centrality and correlation (GASCC) and a simulated annealing algorithm based on SKU centrality and correlation (SASCC). Through comparison experiments, we concluded that the linear programming model can solve the small-scale problem. GASCC and SASCC have the effectiveness in solving the large-scale problem, where GASCC solves fast and SASCC can obtain approximate optimal solutions, which are better than other algorithms. For problems of different sizes, SASCC can optimize solutions by 5% to 25% compared to Random, GASCC, the simulated annealing algorithm (SA), genetic algorithm (GA), and particle swarm optimization (PSO). In addition, we analyzed the influence of algorithm parameters and system parameters on the optimization effect of the algorithm and the smaller the number of SKU types stored in a single storage unit, and the higher the number of buffer locations, the more obvious the optimization effect is. Finally, the comparison with other traditional metaheuristic algorithms proves the effectiveness of the proposed GASCC and SASCC.

Keywords: "parts-to-picker" order picking system; clustered storage; parallel replenishment; linear programming; simulated annealing
Mathematics Subject Classification: 90C57, 90C11

1. Introduction

"Parts-to-picker" order picking systems have been widely used in all parts of the material supply chain, and these systems have not only improved the operational efficiency of warehousing and picking, but also made it feasible to use a variety of flexible and efficient storage strategies. Storage allocation strategy determines how the goods are allocated to the storage location, which have an important impact on order picking efficiency. Traditional storage strategies include random storage strategies, class-based (ABC) storage strategies, and full-turnover-based (FTB) strategies. ABC and FTB strategies assign frequently requested products to locations close to the warehouse, reducing the distance and time for picking and searching for products to be moved, but each of their storage units can generally store only one stock keeping unit (SKU), and do not fully consider the correlation between products, i.e., how often two products are ordered together. The clustered storage strategy takes into account SKU correlation and is widely used in "parts-to-picker" order picking systems. As shown in Figures 1–2, in the robotic mobile fulfillment system (RMFS) and automated storage and retrieval system (AS/RS), by storing multiple SKUs in the same storage unit, the storage unit can fulfill multiple SKUs in an order each time the storage unit is out of the warehouse, which significantly reduces the number of times of outgoing of the storage unit and improves the picking efficiency. Therefore, clustered storage strategy considering the correlation of materials has been gradually applied in recent years. Here, the storage unit is the smallest movable container, and the storage unit for robotic mobile fulfillment system (RMFS) is a movable shelf, while the storage unit in the automated storage and retrieval system (AS/RS) is a pallet that can store multiple SKUs. Figure 3 illustrates the storage units of an autonomous vehicle storage and retrieval system (SBS/R) or a carousel system employing a clustered storage strategy. Figure 3(a) shows each storage unit divided into 2, 3, or 4 sub-bins. The storage unit in Figure 3(b) is divided into 9 sub-bins, with each sub-bin holding different quantities of goods with the same SKU. A cluster of SKUs obtained through the clustering storage strategy can be placed in different sub-bins within the same storage unit.

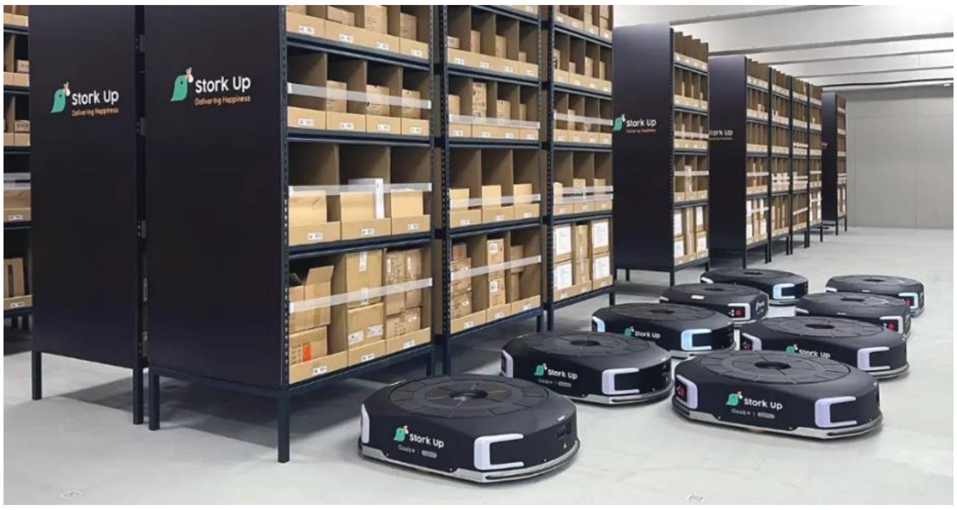


Figure 1. Robotic mobile fulfillment system.

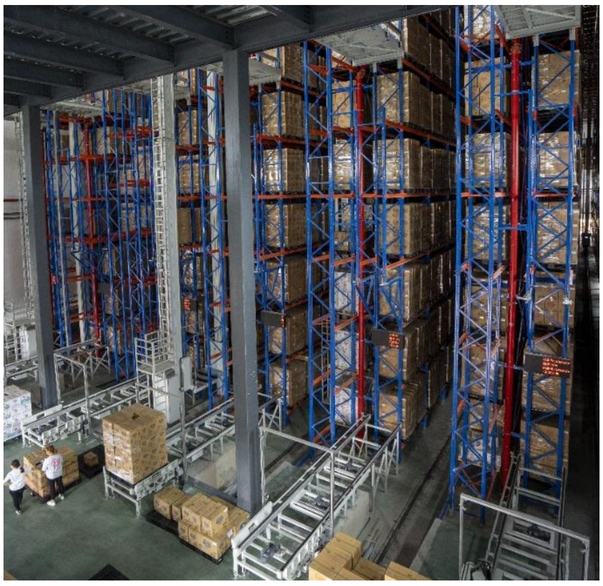
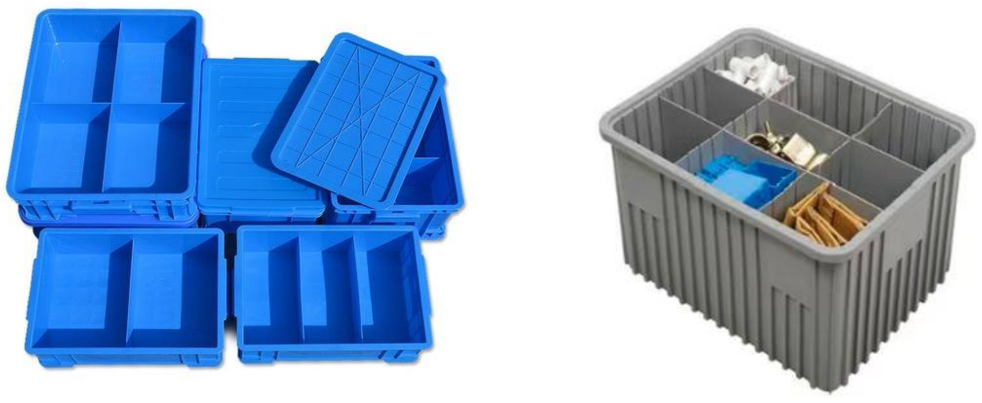


Figure 2. Automated storage and retrieval system.



(a) Empty storage units. **(b)** A storage unit with sub-bins.

Figure 3. Storage units that can be clustered.

In a traditional storage strategy where each storage unit corresponds to a single SKU, replenishment simply involves transporting the required storage units to the inbound port based on the SKU in the replenishment pallet. The order in which these storage units are outbound has little impact. However, in a clustering storage strategy, a single storage unit may contain multiple SKU requirements. If only one SKU can be replenished per outbound operation, the storage unit must undergo multiple outbound operations to replenish all SKUs. For example, a storage unit containing shampoo and body wash would need to exit and enter the storage area twice to replenish both items. However, if the replenishment buffer locations simultaneously hold both shampoo and body wash, the storage unit would need to only exit and enter the storage area once to replenish both items. Therefore, such systems must inevitably set up multiple replenishment buffer locations to improve replenishment efficiency. We refer to this replenishment model as parallel replenishment. Specifically, parallel replenishment refers to a system with multiple replenishment buffer locations, where goods simultaneously present at each buffer location can be replenished into a storage unit. In other words, if goods from multiple replenishment buffer locations are all required by a storage unit, they can be replenished together to that storage unit, similar to parallel operations involving multiple goods. However, the parallel replenishment strategy introduces another issue: Under the constraint of the limited buffer location, which SKUs can be grouped together for parallel replenishment to minimize the number of storage units out of the storage area? In other words, if SKUs that can be replenished in parallel are treated as a SKU group, how to group SKUs for replenishment is the issue that we need to explore. The optimization of parallel replenishment and warehousing sequences in “parts-to-picker” order picking systems using a clustered storage strategy is a relatively novel issue. Researchers have not addressed similar research. Therefore, we review the literature on clustered storage strategy, replenishment, and operations in “parts-to-picker” order picking systems. Figure 4 presents a schematic diagram of a “parts-to-picker” order picking system, labeling storage area, storage units, inbound port, and buffer position. The system comprises an AS/RS and an SBS/R. The AS/RS stores pallets as its storage units, while the SBS/R stores bins. The SBS/R features two inbound ports, each functioning as a workstation managed by one operator and corresponding to two buffer locations, all of which are loaded with full replenishment pallets.

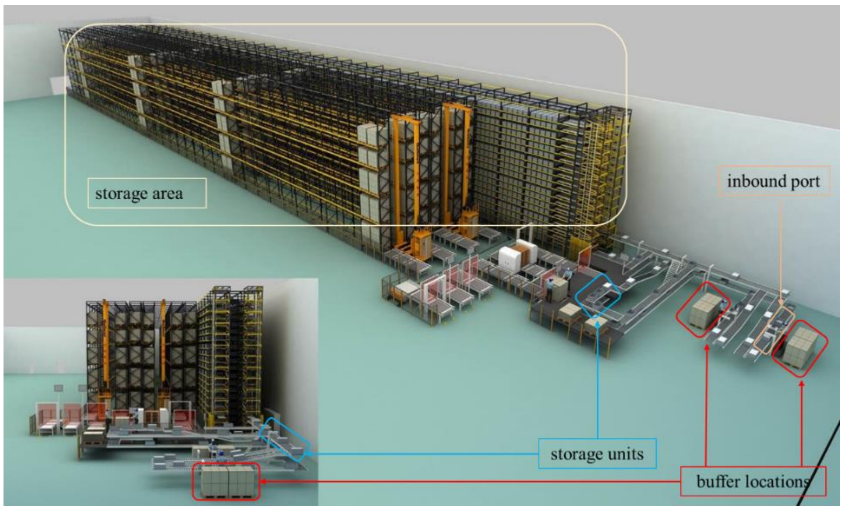


Figure 4. Rendering of the “parts-to-picker” order picking system.

In recent years, more and more “parts-to-picker” order picking systems have adopted a clustered storage strategy, and related research has increased greatly. However, most of these researchers focus on how to achieve better SKU clustering. They use algorithms to calculate the connections or correlations between SKUs and form different storage requirements based on this. Therefore, some researchers also refer to this as a correlation-based storage strategy. Zhang et al. [1] studied the demand correlation pattern of the “parts-to-picker” order picking system, which assigns the ordered products together to storage locations and solves the storage location assignment problem. Wang et al. [2] proposed a storage location assignment model, analyzed and utilized the order data, and proposed a local search heuristic algorithm to improve the picking efficiency. Li et al. [3] proposed a greedy genetic algorithm to solve the dynamic storage allocation problem considering product correlation to maximize the correlation between the total products in each zone. Zhang et al. [4] designed a mixed-integer program for a dynamic slotting problem to minimize the order completion time by finding the probability of order related items in the same picking route in the same time period. Foroughi et al. [5] designed a dynamic model of picking routes and a heuristic rule for product assignment based on related orders. The robotic mobile fulfillment systems (RMFS) are suitable for clustered storage strategy due to their product characteristics. Yuan et al. [6] proposed different mathematical models and heuristic algorithms to optimize the storage allocation in RMFS by dividing the problem into two stages: Product allocation and pod allocation. Keung et al. [7] implemented a data-driven approach to achieve region clustering and storage allocation classification in RMFS by performing a TO-BE analysis of a cloud-based cyber-physical system framework for robotic process automation. Yang [8] developed a policy evaluation model to measure the cost of order picking by using item and order similarity in RMFS. This paper is based on the premise of cluster storage strategies and does not explore storage strategies. Our optimization focuses on the inbound process, enabling the system to implement cluster storage strategies more effectively. Researchers have mostly investigated optimizing storage location arrangements based on correlation analysis. To our knowledge, no similar research on cluster storage strategies has been found in the literature.

In terms of research on replenishment problems, very few researchers have considered replenishment at the operational level, and most researchers have looked at how to determine the timing and quantity of replenishment to avoid stock-outs or delays during order picking [9]. Takano et al. [10] studied the replenishment problem with simultaneous replenishment and order picking. De Vries et al. [11] considered unit-load replenishment under a per-item replenishment strategy and proposed different heuristic replenishment strategies based on different assumptions. Kim et al. [12] studied the case where replenishment and order picking are performed consecutively, focusing on unit-load replenishment in a specific layout where the forward and reserve zones for each item are in the same area. Haverhals [13] proposed a method to find an optimal replenishment strategy to minimize the total expected order picking, batch picking and replenishment costs. There are also some studies on replenishment strategies from the perspective of time series forecasting, using deep learning models to predict future sales of goods and using the predicted volume as a basis for the replenishment process. Wang et al. [14] proposed a model that fuses GAN and XGBoost models for predicting the sales volume of each item in a store, creating an accurate sales-based ordering strategy for merchants. Chandriah et al. [15] proposed an RNN and an improved Adam optimizer to predict the demand for automotive parts to minimize cost and prevent stock-outs. Joseph et al. [16] constructed a CNN-BiLSTM framework coupling CNN with BiLSTM to accurately predict the merchandise demand for store items. In summary, most of these researchers, regardless of the algorithm used, focus on calculating replenishment quantities and timing. We focus on optimizing operational processes to make replenishment faster, which differs somewhat from previous studies.

In addition, research on “parts-to-picker” order picking systems revolves around the analysis and optimization of single systems, where optimization can be divided into system design optimization and operational optimization. The major objectives of operational optimization include reducing order processing time, delivery time, and improving resource utilization. Decision-making content includes vehicle allocation strategies, storage location allocation, workstation allocation rules, storage location allocation, order batching, picking path planning, and picking task allocation, among others. This study falls under the category of operational optimization problems. Yuan et al. [17] investigated an optimization problem on the inbound warehouse process by coordinating multiple resources in a type of automated warehouse system. Hui et al. [18] presented an integrated framework that utilizes swarm intelligence algorithms and collaborative scheduling strategies to optimize inbound/outbound operations. Zhen et al. [19] proposed an integrated optimization model for scheduling the operations in AMR-assisted picker-to-parts warehouse systems. Mokarrari et al. [20] developed mathematical programming models based on two approaches to assign orders to pickers and determine pickers' routes. Wu et al. [21] introduced a five-stage mixed-integer programming model designed to optimize the robotic mobile fulfillment system (RMFS) by minimizing the longest completion time, a critical metric in warehouse efficiency. Barnhart et al. [22] supported robotic parts-to-picker operations in warehousing by optimizing order-workstation assignments, item-pod assignments, and the schedule of order fulfillment at workstations. Zarinchang et al. [23] introduced an artificial intelligence algorithm that seeks to find an acceptable solution to storage location assignment problem deciding where to store products associated with incoming orders. These warehouse operations optimizations are based on research conducted in specific scenarios, and no research has been found on parallel replenishment and inbound order optimization. In addition, the algorithms used in those papers are designed for specific problems and are not generalizable. Nevertheless, we reference these literature resources to obtain algorithmic methods and frameworks from the field of operations research optimization.

Table 1. Comparison of relevant studies in the literature.

Refs.	Storage location assignment	Correlation clustered storage	Inbound outbound order picking operations	Approach
Zhang [1]	✓	✓		heuristic; simulated annealing method
Wang et al. [2]	✓	✓		greedy genetic algorithm;
Zhang et al. [4]	✓	✓		mixed-integer program; Two heuristics
Foroughi et al. [5]	✓	✓		exact; heuristic
Yuan et al. [6]	✓	✓		greedy algorithm; simulated annealing
Keung et al. [7]	✓	✓		A* algorithm; classification algorithms
Yuan et al. [17]			✓	mixed-integer programming; particle swarm optimization
Hui et al. [18]			✓	swarm intelligence algorithms
Zhen et al. [19]			✓	column- and row-generation algorithm
Mokarrari et al. [20]			✓	mathematical programming
Wu et al. [21]			✓	mixed-integer programming; variable neighborhood search
Barnhart et al. [22]	✓		✓	neighborhood search
Zarinchang et al. [23]	✓	✓		simulated-annealing algorithms
This paper	✓	✓	✓	linear programming; greedy algorithm; simulated annealing

Table 1 presents a comparison between this paper and most related studies. It shows that researchers investigating category or SKU correlation primarily focused on storage location optimization. Few researchers looking at inbound/outbound and order picking operations have integrated correlation analysis. Additionally, while storage locations have considered clustering strategies, we use clustering-based storage or SKU correlation merely as background research for optimizing inbound operations. We reference these studies and selected the greedy algorithm and simulated annealing algorithm to address the problem in this paper. In summary, our aim is to optimize the replenishment operations based on the latest practical applications and to determine the grouping and parallel replenishment order of replenished SKUs based on the established SKU storage locations under the cluster storage strategy, which has not been found in this area. Thus, the major contributions of this paper can be summarized as follows:

• We propose a novel parallel replenishment optimization problem of the "parts-to-picker" order picking system, and the solution of this problem can significantly reduce the replenishment time of the system.

• The problem is modeled as a linear programming model, which can solve small-scale problems.

• We present a greedy algorithm based on SKU centrality and correlation (GASCC) and a simulated annealing algorithm based on SKU centrality and correlation (SASCC) to solve the problem.

The remainder of this paper is organized as follows. We give problem description and methods in Section 2. In Sections 3 and 4, we describe the algorithms GASCC and SASCC, respectively. In Section 5, we present computational experiments to evaluate the efficiency of the proposed GASCC and SASCC and to investigate the influence of algorithm parameters on the computational performance and effectiveness of all rules. In Section 6, we conclude the paper by highlighting the major contributions.

2. Problem definition and mathematical model

In this paper, we study the problem of parallel replenishment optimization for "parts-to-picker" order picking systems under the clustered storage strategy. Clustered storage means that after clustering of SKUs, multiple SKUs with correlation are stored in one storage unit. This order picking system contains a storage area, an inbound port, and a limited number of buffer locations. The smallest unit of storage area is the storage unit, which is placed in a storage area (see Figure 5 for a schematic diagram). When replenishing, the system moves the replenishment pallets from other warehouses or factories to the buffer location. All the SKUs in the buffer locations can be replenished at the same time, which is parallel replenishment. SKUs that can be replenished in parallel constitute an SKU group. As shown in Figure 5 (a), A, B, C, and D constitute the initial SKU group. Then, according to the SKUs in the buffer locations, the system transports the storage unit requiring these SKUs to the inbound port, and the workers replenish goods from buffer locations to storage units. After the storage unit completes replenish, the system transports it back to the storage area. The storage units marked with red circles in Figure 5(b) are sequentially transported from the storage area to the inbound port, and after replenishing, they return to the storage area. As can be seen in Figure 5(b), the B and C SKUs in the first storage unit can complete the replenishment task in a single outbound process, and the same applies to the A and D SKUs in the third storage unit, etc. The letters marked in yellow in the storage unit in the picture indicate that replenishment has been completed. If all replenishment work is completed on a pallet in a buffer location, the worker will move the next pallet of SKUs to that buffer location. Repeat the above steps until all SKUs have been replenished. Figure 5(c) shows that C has been replenished at the third buffer location in Figure 5(b), and then the worker replaced B. Figure 5(d) shows that B has completed replenishment at the second buffer location in Figure 5(c), and then the worker replaced F. At this point, storage units that require both F and D can be replenished.

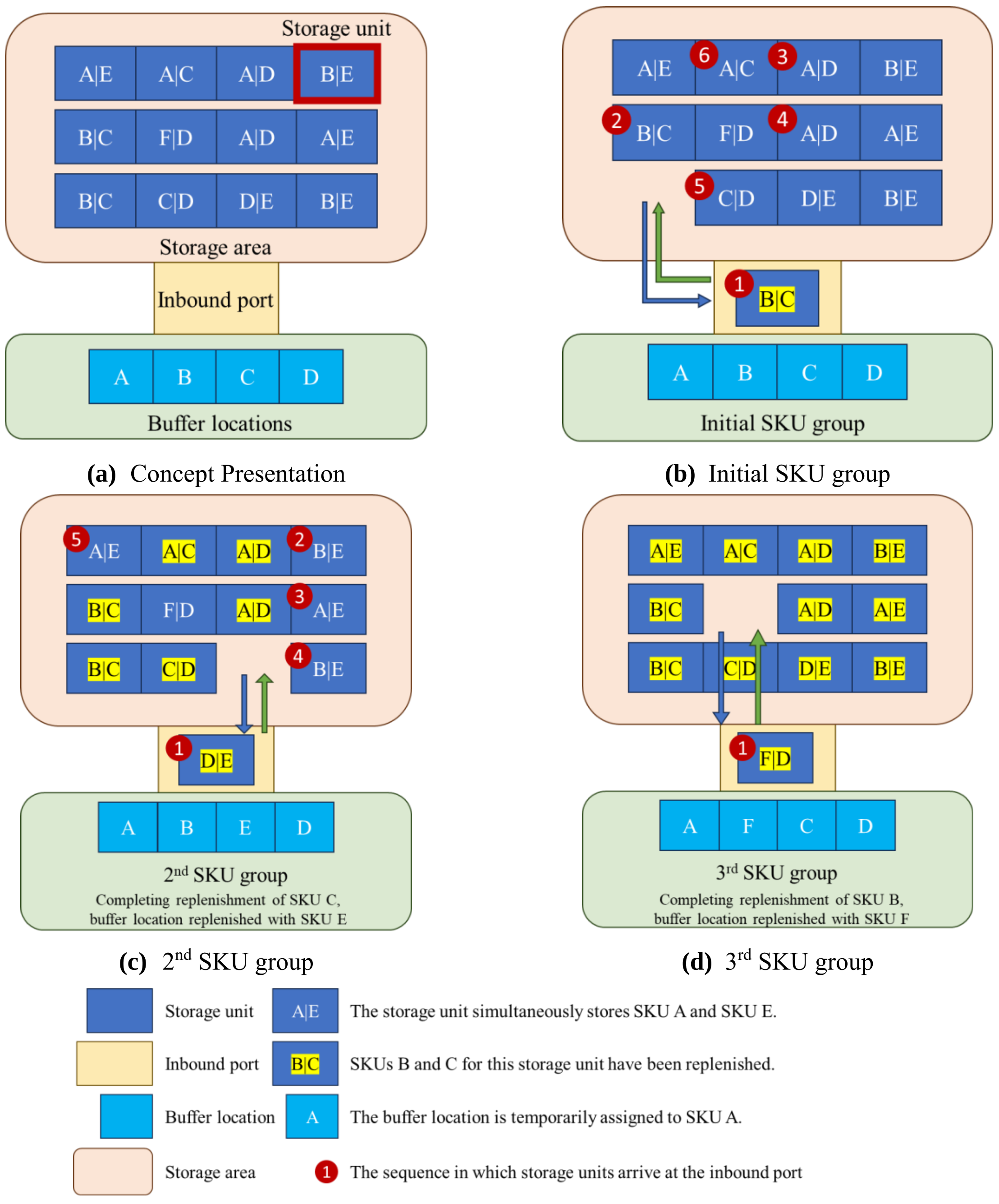

Storage unit
A|E A|C A|D B|E
B|C F|D A|D A|E
B|C C|D D|E B|E
Storage area
Inbound port
A B C D
Buffer locations
(a) Concept Presentation
Initial SKU group
(b) Initial SKU group
2nd SKU group
Completing replenishment of SKU C, buffer location replenished with SKU E
(c) 2nd SKU group
3rd SKU group
Completing replenishment of SKU B, buffer location replenished with SKU F
(d) 3rd SKU group
Storage unit
Inbound port
Buffer location
Storage area
The storage unit simultaneously stores SKU A and SKU E.
SKUs B and C for this storage unit have been replenished.
The buffer location is temporarily assigned to SKU A.
The sequence in which storage units arrive at the inbound port

Figure 5. Parallel replenishment diagram.

Additionally, for the convenience of research, we make the following assumptions based on actual conditions and provided explanations. Assumption 1: The replenishment process for each SKU is non-preemptive. Switching to the next SKU before the replenishment of a SKU in a buffer location is complete would incur switching time costs and management costs. Assumption 2: The quantity of goods on the replenishment pallet exactly meets the replenishment requirements. The replenishment quantity for each SKU can be pre-calculated, and workers simply place the corresponding quantity of goods onto the replenishment pallet. Therefore, once the replenishment of a particular SKU is complete, the replenishment pallet has no remaining goods and can be switched to the next SKU for replenishment. Assumption 3: The number of buffer locations is less than the number of SKU types requiring replenishment. If the number of buffer locations is greater than or equal to the number of SKU types requiring replenishment, all replenishment SKUs can be placed in the buffer locations, which lacks research value. Assumption 4: In the initial state, all buffer locations have been placed with the SKUs. In reality, replenishment pallets are placed in the buffer locations one by one starting from empty buffer locations. This sequential placement process may complete part of the replenishment. This process has little impact on overall replenishment efficiency, but it causes significant problems for model construction, so we ignore the initial group formation process.

Assume that the storage area has n storage units storing m SKUs and the replenishment area has s buffer locations. The entire replenishment process generates ordered K groups of SKUs, according to the replenishment pallet entry order. Only one SKU changes in the two SKU groups before and after. Here, even if multiple buffer locations are simultaneously replenished and completed, they are divided into ordered multiple SKU groups. Therefore, the number of SKU groups is known, i.e., $K = m - s + 1$. The objective of the problem is to minimize the number of times all storage units are out of storage area by adjusting the order in which SKU pallets come in to form ordered groups of SKUs. Let the 0-1 variable x_j^i indicate whether the ith storage unit stores the jth SKU, $i = 1,2,\dots,n$; $j = 1,2,\dots,m$. It is specified that a storage unit stores at least one SKU and stores up to B SKUs, so that $B \geq \sum_{j=1}^{m} x_j^i \geq 1$. Here, B is a setting defined by the system, representing the maximum number of SKU types within a storage unit in the system. In this paper, we propose the 0-1 variable y_j^k as the decision variable, $y_j^k = 1$ indicates that the kth SKU group contains the jth SKU, $k = 1,2,\dots,K$; $j = 1,2,\dots,m$, otherwise $y_j^k = 0$, and $y_j^0 = 0$. In addition, we propose the following auxiliary variables:

$U_j^{i,k}$: The jth SKU in the ith storage unit is replenished by the kth SKU group, then $U_j^{i,k} = 1$, otherwise $U_j^{i,k} = 0$;

S_i^k: There exists a SKU in the ith storage unit that is replenished by the kth SKU group, then $S_i^k = 1$, otherwise $S_i^k = 0$.

Construct the following mathematical planning model:

Objective function:

$$\min \sum_{i=1}^{n} \sum_{k=1}^{K} S_i^k \tag{1}$$

s.t.

$$\sum_{j=1}^{m} y_j^k = s, k = 1,2,\dots,K \tag{2}$$

$$\sum_{k=1}^{K} \left| y_j^k - y_j^{k-1} \right| = 2\,, j = 1,2,\dots,m \tag{3}$$

$$y_j^k - y_j^k y_j^{k-1} \leq 1, k = 2,3,\dots,K \tag{4}$$

$$y_j^{k-1} - y_j^k y_j^{k-1} \leq 1, k = 2,3,\dots,K \tag{5}$$

$$\sum_{k=1}^{K} U_j^{i,k} = x_j^i, i = 1,2,\dots,n;\ j = 1,2,\dots,m \tag{6}$$

$$U_j^{i,k} x_j^i \leq y_j^k, i = 1,2,\dots,n;\ j = 1,2,\dots,m; k = 1,2,\dots,K \tag{7}$$

$$S_i^k \geq U_j^{i,k}, i = 1,2,\dots,n;\ j = 1,2,\dots,m;\ k = 1,2,\dots,K \tag{8}$$

Equation (1) is the objective function, representing to minimize the number of times all storage units are out of storage area; Eq. (2) restricts that all SKU groups must contain all SKUs; Eq. (3) restricts that each SKU can only appear once on the buffer region; Eq. (4) and Eq. (5) both restrict that only one SKU in the front and back SKU groups is changed; Eq. (6) restricts that each SKU of each storage unit needs to get replenished; Eq. (7) indicates that the kth SKU group contains the ith storage unit's jth SKU; and Eq. (8) is the formula for the number of times each storage unit is out of storage.

Clearly, the mathematical model above is nonlinear. To transform it into a linear model, we introduce auxiliary variables α_j^k, β_j^k, and $\gamma_j^{i,k}$ that take the value 0 or 1. Here, $\alpha_j^k = \left|y_j^k - y_j^{k-1}\right|$, $\beta_j^k = y_j^k y_j^{k-1}$, $\gamma_j^{i,k} = U_j^{i,k} x_j^i$.

Furthermore, add the following constraints:

$$\alpha_j^k \geq y_j^k - y_j^{k-1}, j = 1,2,\dots,m; k = 2,3,\dots,K \tag{9}$$

$$\alpha_j^k \geq y_j^{k-1} - y_j^k, j = 1,2,\dots,m; k = 2,3,\dots,K \tag{10}$$

$$\beta_j^k \leq y_j^k, j = 1,2,\dots,m; k = 2,3,\dots,K \tag{11}$$

$$\beta_j^k \leq y_j^{k-1}, j = 1,2,\dots,m; k = 2,3,\dots,K \tag{12}$$

$$\beta_j^k \geq y_j^k + y_j^{k-1} - 1, j = 1,2,\dots,m; k = 2,3,\dots,K \tag{13}$$

$$\gamma_j^{i,k} \leq U_j^{i,k}, i = 1,2,\dots,n;\ j = 1,2,\dots,m; k = 1,2,\dots,K \tag{14}$$

$$\gamma_j^{i,k} \leq x_j^i, i = 1,2,\dots,n;\ j = 1,2,\dots,m; k = 1,2,\dots,K \tag{15}$$

$$\gamma_j^{i,k} \geq U_j^{i,k} + x_j^i - 1, i = 1,2,\dots,n;\ j = 1,2,\dots,m; k = 1,2,\dots,K \tag{16}$$

Equations (9)–(16) represent the constraints required for model linearization, which limit the values of the relevant variables. Therefore, Eqs. (3), (4), (5), and (7) are transformed as follows:

$$\sum_{k=1}^{K} \alpha_j^k = 2\ , j = 1,2,\dots,m \tag{17}$$

$$y_j^k - \beta_j^k \leq 1, j = 1,2,\dots,m; k = 2,3,\dots,K \tag{18}$$

$$y_j^{k-1} - \beta_j^k \leq 1, j = 1,2,\dots,m; k = 2,3,\dots,K \tag{19}$$

$$\gamma_j^{i,k} \leq y_j^k, i = 1,2,\dots,n;\ j = 1,2,\dots,m; k = 1,2,\dots,K \tag{20}$$

The mathematical model is converted into a linear programming (LP) model, which can be directly solved using the linear programming module of the commercial solver Gurobi. The linear programming model has limited ability to solve this problem and can solve this problem only on a small scale, so we propose a greedy and simulated annealing algorithm based on SKU centrality and correlation to solve the large-scale problem. Notations for parameters and variables used in the model are listed in Table 2. Please note that the setting of B affects the value of the system's known variable x_j^i, thereby indirectly influencing the problem. However, it does not belong to the parameters or constraints of the mathematical model used to solve the problem.

Table 2. Parameters and decision variables.

Notations	Explanations
Parameters	
n	Number of storage units
m	Number of SKU types
s	Number of buffer locations
K	Number of groups of SKUs
B	Maximum number of SKU types within a storage unit
i	Storage unit index
j	SKU index
k	SKU group index
x_j^i	$x_j^i = 1$ if the ith storage unit stores the jth SKU, otherwise $x_j^i = 0$
Decision variables	
y_j^k	$y_j^k = 1$ if the kth SKU group contains the jth SKU, otherwise $y_j^k = 0$
Auxiliary variables	
$U_j^{i,k}$	$U_j^{i,k} = 1$ if the jth SKU in the ith storage unit is replenished by the kth SKU group, otherwise $U_j^{i,k} = 0$
S_i^k	$S_i^k = 1$ if there exists a SKU in the ith storage unit that is replenished by the kth SKU group, otherwise $S_i^k = 0$
α_j^k	$\alpha_j^k = \lvert y_j^k - y_j^{k-1} \rvert$
β_j^k	$\beta_j^k = y_j^k y_j^{k-1}$
$\gamma_j^{i,k}$	$\gamma_j^{i,k} = U_j^{i,k} x_j^i$

3. Greedy algorithm based on SKU centrality and correlation

3.1. Closeness centrality of SKUs

Closeness centrality was introduced by Alex Bavelas in 1950 as a measure of the average of the shortest distances from a node to each of the other points in its connectivity component. The concept helps select the closest point to each point within the connectivity component and, hence, is widely used in application scenarios such as key social node discovery and functional place siting. In this paper, the closeness centrality of SKU is used to indicate the degree of association with other SKUs, the value range is [0,1], and the larger the value is, the greater the degree of association with other SKUs.

In addition, we define the correlation and distance between SKUs. x_{th} SKU and y_{th} SKU are considered correlated with each other if there exists at least one storage unit storing x_{th} SKU and y_{th} SKU at the same time. If there are N storage units storing both x_{th} SKU and y_{th} SKU, the distance between x_{th} SKU and y_{th} SKU is considered to be $d(x, y) = 1/N$. The closeness centrality of x_{th} SKU is calculated as Eq. (21):

$$C_x = \left(\frac{A_x}{m-1}\right)^2 \frac{1}{\sum_{x=1}^{m-1}\sum_{y=x+1}^{m} d(x,y)} \tag{21}$$

where A_x is the number of SKU types related with x_{th} SKU, m is the number of SKU types, and $d(x,y)$ is the distance between x_{th} SKU and y_{th} SKU.

3.2. Correlation between SKUs

The Apriori algorithm, introduced by Agrawal and Srikant in 1994, is a widely recognized method for mining frequent itemsets and generating association rules from transaction data. The Apriori algorithm directly yields the required itemsets and their sorted support values, enabling us to identify multiple associated SKUs and form the first SKU group. In contrast, clustering algorithms like K-Means cannot efficiently handle diverse scenarios. While they can group similar SKUs into clusters, they fail to provide comparisons of multi-SKU correlation within clusters or between clusters. Therefore, they are unsuitable for the problem addressed in this paper. Similarly, the Jaccard index used to measure SKU correlation cannot directly yield the desired results for this paper due to the same limitations. If applied, auxiliary rules would need to be added. Thus, we apply the Apriori algorithm to calculate the correlation between SKUs. The steps of the Apriori algorithm are outlined as follows:

Step 1: Generate Candidate Itemsets of Size 1. Generate the candidate itemsets of size 1, which are all SKUs contained within the storage unit. The support of each candidate itemset is calculated by scanning all storage units and their included SKUs. Itemsets that do not meet the minimum support threshold are pruned and discarded.

Step 2: Iterative Generation of Candidate Itemsets. In each subsequent iteration, candidate itemsets of size k are generated by joining frequent itemsets of size $k-1$ from the previous iteration. The support of each candidate itemset is calculated by scanning all storage units and their included SKUs. Candidate itemsets that do not meet the minimum support threshold are pruned.

Step 3: Termination Condition. The algorithm terminates when no more candidate itemsets can be generated, meaning that no frequent itemsets exist for the current iteration.

3.3. Greedy algorithm based on SKU centrality and correlation

Based on the calculated SKU centrality and correlation between SKUs, the Greedy algorithm based on SKU centrality and correlation (GASCC) is proposed in this section. The algorithm uses the concept of weight of SKU groups, which represents the ability of replenishment of a particular SKU group, and the larger its value, the more replenishment tasks can be accomplished. The weight W_z of SKU group z is calculated as Eq. (22):

$$W_z = \sum_{i=1}^{n} R(z,i) \tag{22}$$

where n denotes the storage unit number and $R(z,i)$ denotes the number of SKU types shared by storage unit i and SKU group z. The flowchart of GASCC is shown in Figure 6.

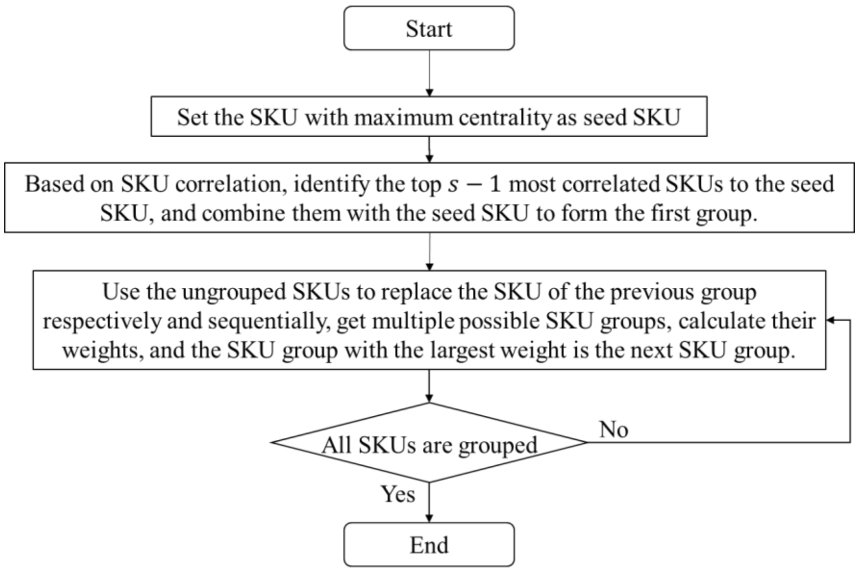

Start
Set the SKU with maximum centrality as seed SKU
Based on SKU correlation, identify the top $s-1$ most correlated SKUs to the seed SKU, and combine them with the seed SKU to form the first group.
Use the ungrouped SKUs to replace the SKU of the previous group respectively and sequentially, get multiple possible SKU groups, calculate their weights, and the SKU group with the largest weight is the next SKU group.
All SKUs are grouped
No
Yes
End

Figure 6. The flowchart of GASCC.

The pseudocode of GASCC is shown in Algorithm 1.

```
Algorithm 1 GASCC
Input: All SKUs;
Output: Ordered group set
1: Set ungrouped SKUs set as all SKUs;
2: Using the Apriori algorithm, obtain the itemsets for SKUs and their support values;
3: Set the SKU with maximum centrality as seed SKU.
4: Sort all itemsets containing seed SKU in descending order by support.
5: Insert the first $s$ SKUs containing the seed SKU from the top itemsets into ordered group set;
6: Delete all SKUs of the last group of ordered group set from ungrouped SKUs set;
7: while ungrouped SKUs set is not empty
8:     for $i$ from 1 to $s$
9:         foreach $sku$ in ungrouped SKUs set
10:            Use $sku$ to replace the $i$th SKU of the last group of ordered group set, and form a new group;
11:            Calculate the weight of the new group;
12:            Combined into $\{sku$, the new group, weight$\}$ as a row and insert it into Weight List;
13:        end
14:    end
15: Identify the row with the highest weight from Weight List as the selected row.
16: Insert the group of the selected row into ordered group set;
17: Delete the $sku$ of the selected row from ungrouped SKUs set;
18: Clear Weight List;
19: end
20: return Ordered group set;
```

4. Simulated annealing algorithm based on SKU centrality and correlation

The simulated annealing algorithm is one of the most commonly used meta-heuristics for solving combinatorial optimization problems and is widely used in sequencing problems, which is characterized by its simplicity and efficiency and avoids the model from converging into a local optimum too quickly. Thus, it has become a common heuristic algorithm framework, and we propose an SKU centrality and correlation based simulated annealing algorithm (SASCC) using the simulated annealing algorithm as the basic framework.

4.1. Generative encoding and generating a new solution

Assuming that the replenishment process generates ordered K SKU groups and the system has s buffer locations, the encoding constructed in this paper is in the form of a $2 \times (K + s - 1)$ matrix, as shown in Table 3. The first row is the move-out SKU, which corresponds to the move-in SKU in the second row. Columns represent the generation order of SKU groups, indicating that the previous SKU group moves out of the SKUs in the first row and into the SKUs in the second row, where the first s elements of the first row are by default 0, which indicates that the first SKU group has only the move-in SKUs. Since the solution of the problem cannot be directly embodied in the encoding, but rather, there is encoding generation, we refer to it as generative encoding.

Once generative coding is defined, the method for A to generate new solutions becomes relatively simple. It requires only selecting two SKUs and then replacing them with each other in generative coding. If the two SKUs are SKU_1, SKU_2, replace all the SKU_2 in the original coding with SKU_1, and replace all the SKU_1 in the original coding with SKU_2.

Table 3. Generative coding of solutions.

Generate serial No.	1	2	…	s	$s+1$	$s+2$	…	$K+s-1$
move-out SKU:	0	0	…	0	SKU_{s+1}^1	SKU_{s+2}^1	…	SKU_{s+K-1}^1
move-in SKU:	SKU_1^2	SKU_2^2	…	SKU_s^2	SKU_{s+1}^2	SKU_{s+2}^2	…	SKU_{s+K-1}^2

4.2. Decoding and initial solution

The decoding process of the generative coding proposed in this paper is to perform the operations of moving into and out of SKUs sequentially in accordance with the generation order to obtain the solution form, i.e., a group of K SKUs arranged in an ordered manner. According to Table 3, the initial SKU group consists of the first s move-in SKUs, namely SKU_1^2, SKU_2^2, …, SKU_s^2, which is the first row of Table 4. If SKU_{s+1}^1 in Table 3 represents SKU_1^2, then SKU_1^2 in the first row of Table 4 exits and is converted to SKU_{s+1}^1, form the second row of Table 4. Similarly, if SKU_{s+2}^1 in Table 3 represents SKU_2^2, then SKU_2^2 in the second row of Table 4 exits and is converted to SKU_{s+2}^1, form the third row of Table 4. The final conversion occurs in the $K + s - 1$th column of Table 3. If SKU_{s+K-1}^1 in Table 3 represents SKU_s^2, then SKU_s^2 in the first row of Table 4 is removed and converted to SKU_{s+K-1}^2, form the last row of Table 4. The entire decoding process is complete, forming an ordered list of SKU groups, as shown in Table 4, where each row represents a SKU group.

The GASCC utilizes the concept of greedy algorithms, combining SKU closeness centrality and correlation between SKUs, which can solve the problem quickly and effectively, so the result of GASCC is applied as the initial solution of the simulated annealing algorithm. The result of GASCC requires decompilation, which involves reverse-engineering the results back to their encoded form.

Table 4. Decoding results of generative coding.

SKU_1^2	SKU_2^2	…	SKU_s^2
SKU_{s+1}^2	SKU_2^2	…	SKU_s^2
SKU_{s+1}^2	SKU_{s+2}^2	…	SKU_s^2
…	…	…	…
…	…	…	SKU_s^2
…	…	…	SKU_{s+K-1}^2

4.3. Main Process of Simulated Annealing Algorithm

Step 1: Initialization. Set the initial temperature $T = T_0$, the initial solution X_0, the number of iterations L, the termination temperature T_{end}, the temperature decay coefficient α, so that the number of iterations $i = 0,\ X = X_0$;

Step 2: If temperature $T < T_{end}$, terminate the algorithm, X_{best} is the final solution; otherwise, execute Step 3.

Step 3: Judge i, if $i \leq L$, execute steps Step 4 to Step 6; otherwise, make $i = 0, T = \alpha T$, return to Step 2;

Step 4: Generate a new solution X_{new}.

Step 5: Calculate the increment $\Delta E = E(X_{new}) - E(X)$. Here, $E(X_{new})$ and $E(X)$ are the objective function values of X_{new} and X, respectively.

Step 6: If $\Delta E < 0$, accept X as the current new solution, otherwise, accept X as the current new solution with probability $exp\ (-\Delta E/T)$. If the current new solution is finally accepted, let $X = X_{new}$. Let $i = i + 1$, return to step 3.

The flowchart for SASCC is shown in Figure 7.

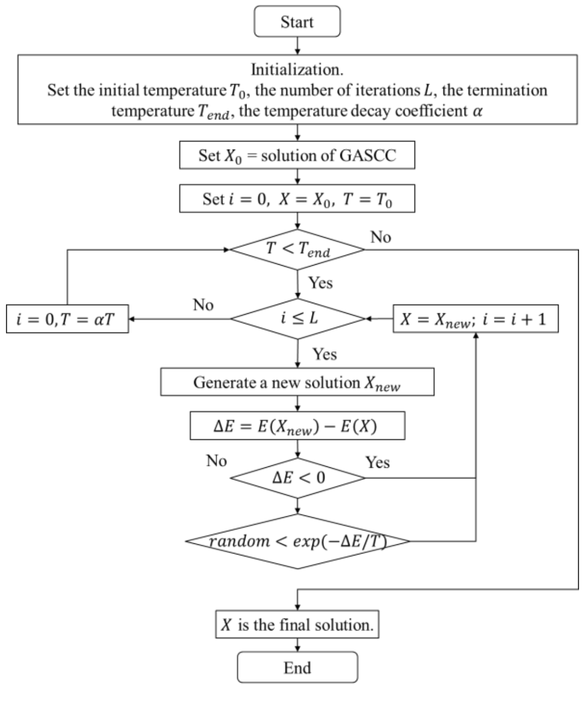

Start
Initialization.
Set the initial temperature T_0, the number of iterations L, the termination temperature T_{end}, the temperature decay coefficient α
Set X_0 = solution of GASCC
Set $i = 0,\ X = X_0,\ T = T_0$
$T < T_{end}$
No
Yes
$i = 0, T = \alpha T$
No
$i \leq L$
$X = X_{new};\ i = i + 1$
Yes
Generate a new solution X_{new}
$\Delta E = E(X_{new}) - E(X)$
No
Yes
$\Delta E < 0$
$random < exp(-\Delta E/T)$
X is the final solution.
End

Figure 7. The flowchart of SASCC.

5. Computational experiments

In this paper, comparative experiments are designed to measure the performance of linear programming models, SASCC and GASCC. The problem sizes that can be solved by the linear programming models in the acceptable time are considered as small-scale problems, and their corresponding experiments are small-scale problem experiments designed to analyze the capability of the linear programming models. Problems of other scales are regarded as large-scale problems, and the objectives of the experimental design for large-scale problems are: First, to find out the change rule of the optimization effect of SASCC and GASCC with the increase of the scale; second, to derive the influence of the system parameters on the optimization effect of the algorithms; and third, to get the validity of the SASCC and GASCC proposed in this paper by comparing with the classical heuristic algorithms. The problem in this paper is based on a real-life order picking scenario from the parts warehouse of a manufacturing company. The stored goods are mechanical parts, and since many of these parts are used in combination, it is necessary to employ a clustered storage strategy. The experimental parameters for the large-scale problem are primarily based on real-world industrial data. The experimental parameter settings for the small-scale experiments are designed to assess the solution capability of the linear programming model, so the problem scale is reduced based on real-world scenarios. All experimental tests are conducted on a personal computer with an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz and 16GB RAM. We useused Python 3.10.3 to call Gurobi Optimizer 8.0 for the linear programming, and use Matlab R2019a to run other algorithms.

To measure the quality of the solutions obtained by different algorithms, we calculate the percentage deviation (RPD) of each algorithm using Eq. (23):

$$RPD_{method} = \frac{C_{method} - C_{best}}{C_{best}} \tag{23}$$

where C_{best} is the optimal objective function value for all algorithms to solve a case, and C_{method} is the objective function value obtained when using the “method” algorithm for a case. RPD_{method} reflects the difference between the “method” algorithm and the current optimal solution, and the smaller the value of RPD_{method} is, it means that the closer the result is to the optimal solution, and the better the optimization effect.

In addition, we consider an extreme case where each SKU in each storage unit needs to be replenished individually, and the case where the number of storage unit outgoing is the sum of the number of SKU types contained in all storage units, which is the worst solution to this problem, denoted by “Ultra”. The maximum optimization power of the algorithm can be derived by comparing the difference between “Ultra” and other algorithms.

5.1. Experiments on small-scale problems

Small-scale problems are problems with a small number of storage units, SKUs, and buffer locations, which can be solved by linear programming algorithms, and the experiments are designed with the following objectives: 1. To measure the performance of the linear programming algorithm and to obtain the maximum size of the problem that can be solved by the algorithm; and 2. The effectiveness of GASCC and SASCC in solving small-scale problems. In the experiment, the number of storage units n takes the values of 10, 15, and 20, respectively, the number of SKU types m is 10, 15, and 20, respectively, the maximum number of SKU types within a storage unit B is 3, and the number of buffer locations s is 4, 6, and 8, respectively, with a total of 27 experimental scenarios, each of which is experimented with 10 times, and each of which is performed using Ultra, LP, Random, SASCC, and GASCC, respectively. The number of times all storage units are replenished in and out of the warehouse and the RPD of each method is calculated, and the upper limit of Gurobi's time for solving the LP problem is set to 1 hour. The parameters of SASCC are $T_0 = 1$, $L = 100$, $T_{\text{end}} = 0.01$, and $\alpha = 0.95$, and the initial solution is GASCC.

Table 5. Statistics of the results of each algorithm for small-scale problems.

m	n	s	RPD (%)					Average objective value	Runtime (s)		
			Ultra	LP	Random	SASCC	GASCC		LP	SASCC	GASCC
10	10	4	151.3	0	58.7	18.9	30.2	10.6	3.103	0.079	0.792
		6	184	0	56	4	20	10	0.050	0.027	0.632
		8	164	0	18	0	4	10	0.006	0.024	0.588
	15	4	128.9	0	57.7	17.5	29.6	17.6	76.103	0.041	0.996
		6	164.7	0	34.6	8	23.8	15.2	0.718	0.047	0.912
		8	168	0	17.3	0	5.3	15	0.022	0.042	0.854
	20	4	118.3	0	48.8	14.9	24	24.2	99.557	0.064	1.319
		6	167.4	0	37.6	4.9	27	20.8	1.769	0.089	1.227
		8	157	0	8	0	8	20	0.032	0.051	1.101
Average			156.0	0	37.4	7.6	19.1	15.9	20.151	0.052	0.936
15	10	4	174	0	102	54	70	10	3.194	0.040	0.794
		6	184	0	92	10	34	10	0.214	0.044	0.701
		8	184	0	70	2	16	10	0.112	0.045	0.670
	15	4	128	0	75.3	32	40.8	18.4	500.039	0.091	1.190
		6	178.7	0	68	20	44	15	12.905	0.093	1.036
		8	182.7	0	45.3	1.3	21.3	15	0.238	0.098	0.981
	20	4	115.7	0	62.5	30.8	38.7	25.4	610.036	0.165	1.559
		6	154.2	0	59.4	18.5	40.5	21.8	1020.046	0.192	1.453
		8	184	0	46	12	28	20	1.586	0.149	1.307
Average			165.0	0	68.9	20.1	37.0	16.2	238.708	0.102	1.077
20	10	4	184	0	116	66	76	10	4.133	0.054	0.879
		6	184	0	104	22	54	10	0.925	0.061	0.792
		8	190	0	82	6	24	10	0.670	0.075	0.732
	15	4	127.3	0	78.4	43.1	52.4	18	340.063	0.146	1.328
		6	180.3	0	97.6	39.5	51.2	15.2	21.692	0.134	1.231
		8	188	0	81.3	10.7	37.3	15	1.044	0.170	1.165
	20	4	103	0	56.5	37.6	43.2	26.4	3600.111	0.276	1.963
		6	158.8	0	80.6	34	53.2	21.4	3600.071	0.267	1.650
		8	183	0	71	22	39	20	11.872	0.226	1.483
Average			166.5	0	85.3	31.2	47.8	16.2	842.287	0.157	1.247

Table 5 counts the computational results of each algorithm in the small-scale experiments, including the RPDs of the five methods of Ultra, LP, Random, SASCC, and GASCC, the running times of the three algorithms proposed in this paper, namely, LP, SASCC, and GASCC, and the average objective values across multiple cases in each scenario. All these values originate from the LP. It can be found that the LP algorithm can get the exact optimal solution, and its results are optimal among all the algorithms. In some cases, LP cannot get the optimal solution within one hour, but the current solution obtained by LP is optimal among all algorithms, which shows that LP can get better quality solutions for solving small-scale problems. However, the running time of LP increases rapidly with the size of the problem. When the number of storage units n or the number of SKU types m reaches the size of 20, the probability of not being able to complete the solution within one hour increases sharply. The running time of LP depends on the case, which is volatile, and smaller-sized cases may take longer to solve. For example, some of the cases with n of 10, m of 15, and B of 6 have a larger above-average running time. Among the three algorithms proposed in this paper, GASCC has the shortest running time, and the average RPD of all cases of GASCC is 0.346, comparing with the RPD of Ultra and Random 1.625 and 0.639. It can be seen that GASCC can also improve the efficiency of replenishment, so that GASCC can get a more satisfactory solution quickly. The average RPD of SASCC is 0.196, which can continue to improve by 0.150 on the basis of GASCC, showing that the framework of the simulated annealing algorithm proposed in this paper and its improvement of encoding, decoding, and generating a new solution are effective. This also shows the effectiveness of SASCC in solving the problem. The running time of SASCC has a small increase in completing the initial solution solving of GASCC, and overall, the running time is acceptable.

5.2. Experiments on large-scale problems

To further explore the performance of SASCC and GASCC, large-scale problem experiments are designed in this paper. This part of the experiment can determine the changes of the optimization effect of SASCC and GASCC without taking into consideration the problem size; derive the influence of system parameters on the optimization effect of the algorithms; and verify the effectiveness of SASCC and GASCC proposed in this paper. In the experiment, the number of storage units n and the number of SKU types m are taken as 50, 60, 70, 80, 90, and 100, the maximum number of SKU types B within a storage unit is 3, and the number of buffer locations s is 4, 8, 12, and 16, respectively, and there are a total of 144 experimental scenarios, where each experimental scenario is experimented 10 times. Each experiment uses Ultra, Random, SASCC, and GASCC to obtain the number of times all storage units are replenished in and out of the warehouse, respectively, and we calculate the RPD for each method. The parameters of SASCC take the values of $T_0 = 0.1$, $L = 200$, $T_{\text{end}} = 0.001$, and $\alpha = 0.995$, and the initial solution is GASCC.

(1) The impact of problem size on the optimization effect of algorithms

The number of storage units and SKUs directly determines the size of the problem, and compare the optimization performance of SASCC and GASCC relative to Random and Ultra methods when the number of storage units and SKUs are varied, respectively. Figure 8(a) shows the RPD of each algorithm when the number of storage units is fixed at 50 and the number of SKU types is different, and Figure 8(b) shows the RPD of each algorithm when the number of SKU types is fixed at 50, and the number of storage units is different. The RPD values and the average optimal objective values for different problem sizes are presented in Table 6. Since SASCC consistently yields the optimal results, all average objective values presented here correspond to SASCC outcomes. Figure 8 does not show the results of the SASCC algorithm because its RPD is always 0, indicating that the SASCC algorithm can obtain the relatively optimal solution under various conditions. Therefore, SASCC is applicable for solving large-scale problems. The average RPD of GASCC in all the experiments in this part of the experiment is 0.048, which is much smaller than the average RPD of the small-scale case. On the one hand, this is because the large-scale experiments cannot get the optimal solution, and can use only the relatively optimal solution obtained by SASCC, which is lower in quality than LP; on the other hand, as the problem scale increases, the difficulty of further optimizing SASCC based on GASCC also increases. Of course, by comparing the RPD of Random and Ultra, 0.110 and 0.432, it can be seen that the optimization effect of SASCC and GASCC is obvious. So, if the algorithm is used in practice, it will improve the replenishment efficiency by at least 10%. In addition, it is observed that under the established algorithm parameters, there is a decreasing trend in the optimization effect of SASCC and GASCC as the problem size increases, which can also be found by comparing the RPD of SASCC and GASCC for small-sized problems. The reason for this may be that as the problem size increases, the range of the solution space becomes larger, and the heuristic algorithms search for a better solution with more difficulty and are more likely to fall into a local optimum.

Figure 9 demonstrates the average running time of SASCC and GASCC for solving cases of different sizes. It can be found that the running time of both SASCC and GASCC increases significantly with the increase of the problem size. However, the increase of the storage unit has a greater impact on the running time of the two algorithms, and the use of GASCC to solve the initial solution takes a larger proportion of the running time in SASCC, with an average of 42.79%. Analyzing the GASCC operation steps, it can be found that calculating the closeness centrality of SKUs and the correlation between SKUs consumes a large amount of runtime, and the increase in storage units affects the computational complexity.

Table 6. Average RPD and objective value of various algorithms for different problem sizes.

		Average objective value	GASCC (%)	Random (%)	Ultra (%)
$m = 50$:					
	60	151.0	4.8	11.0	43.2
	70	183.1	4.5	14.3	42.2
n	80	210.4	3.9	9.2	39.8
	90	230.2	5.1	12.7	42.6
	100	266.3	4.6	6.5	36.0
Average		208.2	4.6	10.7	40.8
$n = 50$:					
	60	166.2	4.7	12.8	41.4
	70	179.3	3.3	15.9	39.3
m	80	240.2	6.9	19.6	42.0
	90	264.1	4.7	17.1	34.5
	100	294.1	4.5	16.8	31.7
Average		228.8	4.8	16.4	37.8

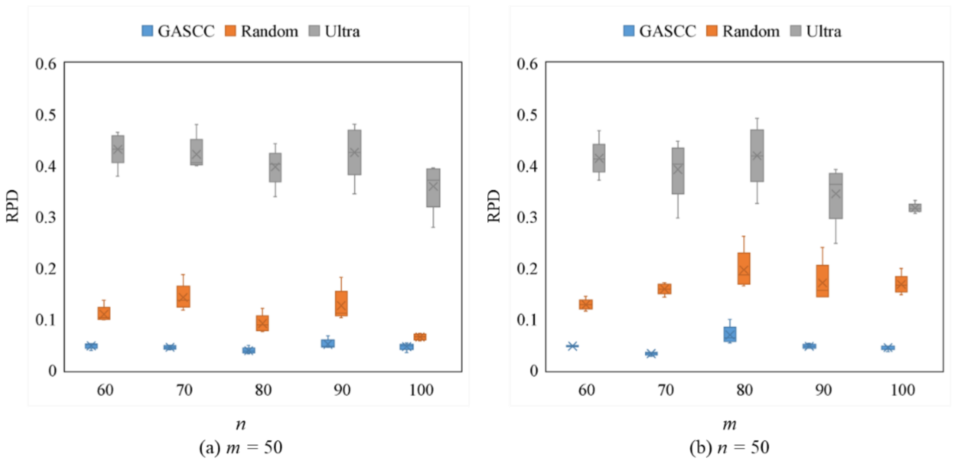


Figure 8. Optimization effect of algorithms for different problem sizes.

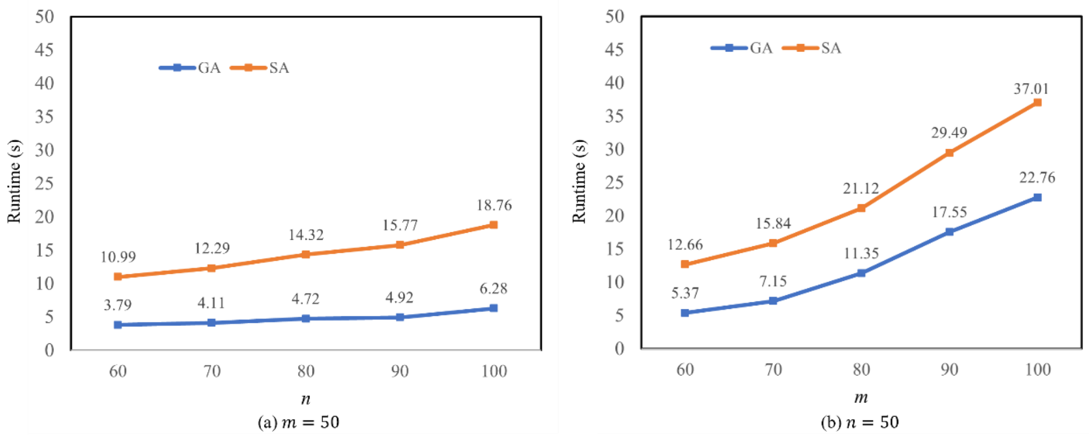


Figure 9. SASCC and GASCC runtime for different problem sizes.

(2) Parameter sensitivity analysis

In actual production operations, the number of SKU types stored in each storage unit (B) and the number of replenishment buffer locations (s) are typically parameters configurable by managers. Therefore, in this section, we conduct a sensitivity analysis on these two parameters. In the experiments, B is set to values of 2, 3, 4, 5, and 6, while s is set to 4, 8, 12, and 16. This results in $5 \times 4 = 20$ experimental scenarios, with 10 cases randomly generated for each scenario. We calculate the RPD for each case and group the results by parameter values to compute the average RPD for each parameter, as shown in Figure 10. The RPD values and the average optimal objective values for each parameter are presented in Table 7, where the average optimal objective values are also derived from SASCC. The SASCC results are not displayed in the figure because its average RPD was consistently 0, indicating that SASCC effectively solves the problem at hand and achieves high-quality solutions. The results for other algorithms in the figure represent the gap between those algorithms and SASCC. Figure 10(a) reveals that the gap between Random and SASCC diminishes as the number of SKU types stored per storage unit increases. This occurs because a larger B value increases the number of SKU itemsets, while SASCC generates groups satisfying more storage units, thereby improving optimization effectiveness. Increasing buffer locations substantially reduces the total SKU count entering and exiting storage units, lowering the objective function value of the optimal solution. This accentuates SASCC's optimization effect, widening its RPD gap with other algorithms. Overall, the gap between GASCC and SASCC remains relatively stable, with GASCC exhibiting similar parameter-dependent variations to SASCC. As shown by the average objective values in Table 7, the greater the number of SKU types per storage unit or the fewer the buffer locations, the higher the average objective, indicating a larger total number of storage unit in/out operations required.

The experiments confirm the effectiveness of both GASCC and SASCC, and warehouse managers should consider implementing both algorithms in their systems. Furthermore, when the number of SKU types stored per storage unit and the replenishment buffer size are large, the optimization effects of both algorithms become more pronounced, making it even more advisable for managers to utilize GASCC and SASCC for system optimization. Moreover, the larger these parameters are, the greater the impact of the algorithms.

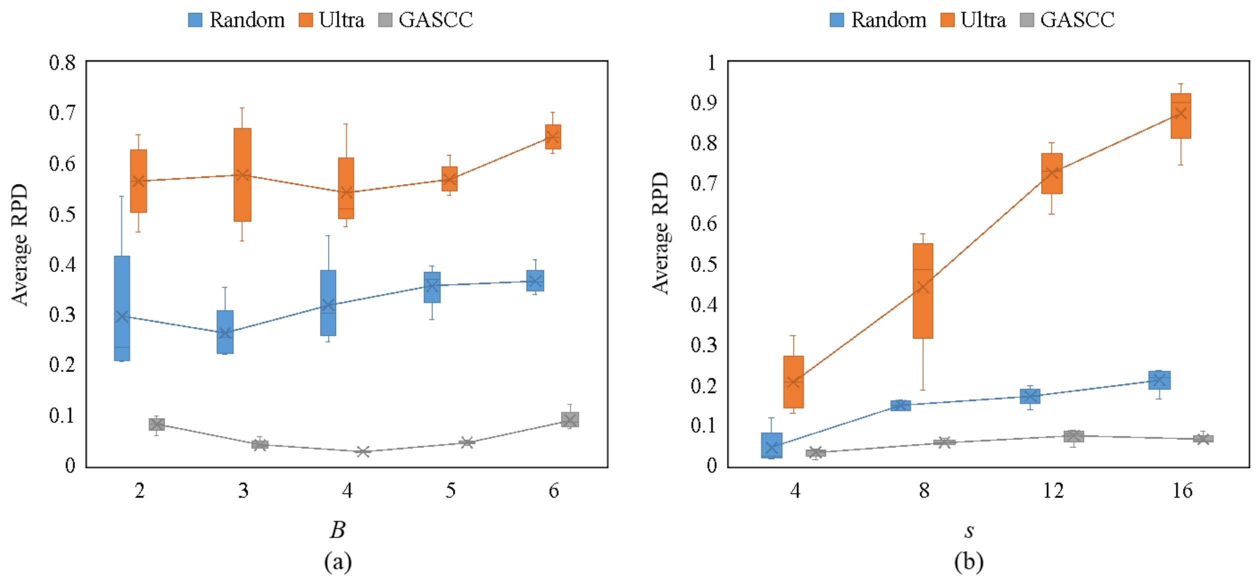


Figure 10. Parameter sensitivity analysis.

Table 7. Average RPD and objective value of algorithms for different parameters.

		Average objective value	GASCC (%)	Random (%)	Ultra (%)
B	2	190.8	8.2	29.6	56.4
	3	248.9	4.1	26.3	57.7
	4	266.4	2.7	31.9	54.2
	5	309.6	4.6	35.6	56.8
	6	322.4	9.0	36.6	65.3
Average		267.6	5.7	32.0	58.1
s	4	321.2	3.2	4.4	20.7
	8	277.4	5.7	14.9	44.3
	12	238.6	7.4	17.1	72.5
	16	183.6	6.5	21.2	87.5
Average		255.2	5.7	14.4	56.3

(3) Comparison of GASCC and SASCC with different heuristic algorithms

In this paper, experiments are also designed to prove the effectiveness of GASCC and SASCC in comparison with traditional algorithms. The traditional algorithms used as comparisons include the simulated annealing algorithm (SA), the genetic algorithm (GA), and the particle swarm optimization (PSO). In this paper, when selecting the parameter values of PSO, GA and SA, we attempt to ensure the full utilization of the algorithm's ability, such as the number of populations and the number of iterations, which are selected to be larger values in order to improve the quality of the solution. The initial solution for SA, the initial particle swarm for PSO, and the initial population for GA are generated using the Random method. The way to generate new solutions, new particles, or new chromosomes is the method in 4.1. The parameters are as follows: The number of PSO populations is 1000, the number of iterations is 500, the inertia weight is 0.96, the cognitive coefficient is 0.7, the social coefficient is 0.3, and the proportion of GASCC in the initial solution is 0.75; the initial temperature of SA is 0.1, the number of iterations is 2000, the termination temperature is 2×10^{-6}, and the temperature decay coefficient is 0.98; and the number of GA populations is 1000, the number of iterations is 1000, the percentage of GASCC in the initial solution is 0.5, the number of crossover chromosomes per generation is 2 Pg, and the number of mutations is Pg.

Table 8 compares the RPD of SASCC with the traditional algorithms SA, GA, and PSO. The optimization effect of SASCC is significantly better than the other algorithms, and the average RPD for all the cases is 0.022, especially in the case of smaller scale. The optimal solution of the problem is obtained by the SASCC algorithm, so the RPD is almost zero. Of course, when the problem size is larger, the RPD of SASCC increases, indicating that the instability of the algorithm increases when the problem is larger, but in general, it is significantly better than other algorithms. The Random method represents the actual production method, with an RPD of 0.280, reflecting the fact that there is a lot of space for optimization in actual production. Traditional PSO and GA have RPDs of 0.130 and 0.135, which are worse than other algorithms for problem solving. The average RPD of GASCC is lower than that of traditional PSO, GA, and SA, and does not differ much from that of SASCC, suggesting that the use of GASCC as the initial solution by SASCC plays a larger role. SA is superior to PSO and GA, and heuristic algorithms based on trajectory strategies are superior to population-based heuristics, which also verifies the correctness of selecting SA as the algorithm framework in this paper.

Table 8. Comparison of optimization effects of multiple algorithms.

B	s	Random (%) (Avg.±std.)	GASCC (%) (Avg.±std.)	SA (%) (Avg.±std.)	GA (%) (Avg.±std.)	PSO (%) (Avg.±std.)	SASCC (%) (Avg.±std.)
	8	16.3±16.4	2.9±1.5	2.6±2.6	6.1±2.8	7.9±4	0.4±0.2
4	12	21.7±10.7	3.1±0.7	3.9±4.1	6.3±6.7	7.4±5.9	0.1±0.1
	16	22.4±12	2±1.6	4.1±1.1	9.1±5.1	9.4±11.7	0±0
	8	24.9±26.3	5.5±0.2	8.8±2.2	17±16.9	16.7±14.7	0±0
5	12	27.1±25.4	6.1±0.6	11.2±1.5	18.4±13	13.5±8.3	2.7±1.2
	16	24.5±19.7	7.9±3.3	11±9.9	15.1±9.1	14±11.6	2.5±0.6
	8	39.2±22	10±6.2	9.6±6.3	14.7±9.4	15.8±13.5	5.1±1.8
6	12	34.6±29.1	11.6±6.6	11.4±7.2	18.4±15.4	16.5±16.8	3.8±0.8
	16	41.5±16.9	15.5±3.6	11.1±2.2	16.3±10.2	15.6±9.1	4.8±1.2

(4) The impact of SKU centrality and correlation on algorithms

To evaluate the impact of SKU centrality and correlation on the algorithm, we design the comparative experiments in this section. Before testing, we modify GASCC to create three new algorithms. To compare the role of SKU centrality, we retain the order correlation component while constructing two greedy algorithms: GA1_max and GA1_random. GA1_max changes the 2nd line in the Algorithm 1 pseudocode to "Set the most frequently occurring SKU in orders as seed SKU"; and GA1_random changes the 2nd line in the pseudocode to "Set random SKU as seed SKU". The other steps remain unchanged. To evaluate the effectiveness of SKU correlation, we retain the SKU centrality component. Thus, we have removed lines 2 and 4 from Algorithm 1 pseudocode and the 5th line to "Randomly select $s-1$ SKUs, combine them with the seed SKU to form a new group, and insert it into ordered group set". The other steps remain unchanged. This algorithm is named GA2_random. Additionally, we include the SA algorithm using a random solution as the initial solution for comparison, named SA_random. Table 9 shows the RPD values for different algorithms. Comparing GASCC with GA1_max and GA1_random in Table 9 reveals that SKU centrality computation enhances algorithm performance. The average PRD comparison between GASCC and GA2_random shows that SKU correlation also contributes positively. Overall, SKU centrality and correlation serve as auxiliary factors in the greedy algorithm, while the core design, trial replacement of each SKU and selection of the optimal one, plays a decisive role. Additionally, comparing SASCC and SA_random reveals that GASCC significantly influences SA as an initial solution. This indicates that GASCC based on SKU centrality and correlation plays a crucial role in SA algorithm design.

Table 9. The impact of SKU centrality and correlation on algorithms.

B	s	GASCC (%) (Avg.±std.)	GA1_max (%) (Avg.±std.)	GA1_random (%) (Avg.±std.)	GA2_random (%) (Avg.±std.)	SASCC (%) (Avg.±std.)	SA_random (%) (Avg.±std.)
4	8	2.9±1.5	3.9±1.5	3.8±1.9	4.1±2.3	0.4±0.2	2.6±1.3
	12	3.1±0.7	4.3±3.8	3.9±2.7	4.2±1.4	0.1±0.1	3.9±3.5
	16	2±1.6	3.1±1.6	2.8±1.3	3±1.5	0±0	4.1±1
5	8	5.5±0.2	6.5±1.5	6.5±5.5	6.8±5.9	0±0	8.8±5.4
	12	6.1±0.6	7.3±4.4	7.1±3.3	7.3±7	2.7±1.2	11.2±6.1
	16	7.9±3.3	9±8.5	8.9±3	9.2±1.5	2.5±0.6	11±10.5
6	8	10±6.2	10.9±9.5	10.9±6.9	11±10.6	5.1±1.8	9.6±8.4
	12	11.6±6.6	12.6±9.6	12.5±8.3	12.9±5.2	3.8±0.8	11.4±5.7
	16	15.5±3.6	16.3±13.7	16.3±12.2	16.6±8.7	4.8±1.2	11.1±9.4

6. Conclusions

In this paper, we solved the problem of parallel replenishment optimization for "parts-to-picker" order picking systems under clustered storage strategy. We established a linear programming model and calculated the closeness centrality of SKUs and the correlation between SKUs. Based on these, we proposed a greedy algorithm based on SKU centrality and correlation (GASCC) and a simulated annealing algorithm based on SKU centrality and correlation (SASCC). Our results revealed that the introduction of GASCC and SASCC can substantially improve the efficiency of replenishment. GASCC has an advantage in the solution time, and it can be applied to problems with solution time requirements. SASCC can obtain approximate optimal solutions, which are better than other algorithms. For problems of different sizes, SASCC can optimize solutions by 5% to 25% compared to Random, GASCC, SA, GA, and PSO. Moreover, the method is suitable for problems of various scales. In addition, the results also showed that the smaller the number of SKU types stored in a storage unit and the more the number of buffer locations are, the more obvious the optimization effect is. Thus, decision-makers should pay more attention to the introduction of scientific algorithms in such application scenarios in order to improve the operational efficiency of the system.

The parallel replenishment optimization problem of the "parts-to-picker" order picking system under the clustered storage strategy belongs to a relatively new practical problem, there are fewer related literature studies. Thus, we aimed to complete the solution to the problem and did not explore the main framework and details of the algorithm, which will be continued in future research.

In addition, we did not address the specific operations involved in buffer replenishment, which is also a complex and important issue. For example, queuing phenomena may occur in buffer locations, and transportation time may be required for replenishment operations. If mobile carriers are involved in transportation, path congestion may also occur, further complicating the issue. However, these are all issues that arise in actual operations and are worthy of in-depth study.

Author contributions

Xin Wang: Conceptualization, Methodology, Software, Writing–original draft, Formal analysis; Yaohua Wu: Supervision, Validation, Writing–review and editing.

Use of Generative-AI tools declaration

No Artificial Intelligence (AI) tools were used in the creation of this article.

Acknowledgments

The authors are deeply grateful to the handling editor and reviewers for their constructive comments and suggestions.

Conflict of interest

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. R. Q. Zhang, M. Wang, X. Pan, New model of the storage location assignment problem considering demand correlation pattern, *Comput. Ind. Eng.*, **129** (2019), 210–219. https://doi.org/10.1016/j.cie.2019.01.027
2. M. Wang, R. Q. Zhang, K. Fan, Improving order-picking operation through efficient storage location assignment: A new approach, *Comput. Ind. Eng.*, **139** (2020), 106186. https://doi.org/10.1016/j.cie.2019.106186

3. J. Li, M. Moghaddam, S. Y. Nof, Dynamic storage assignment with product affinity and ABC classification—a case study, *Int. J. Adv. Manuf. Technol.*, **84** (2016), 2179–2194. https://doi.org/10.1007/s00170-015-7806-7
4. J. Zhang, S. Onal, S. Das, The dynamic stocking location problem–Dispersing inventory in fulfillment warehouses with explosive storage, *Int. J. Prod. Econ.*, **224** (2020), 107550. https://doi.org/10.1016/j.ijpe.2019.107550
5. A. Foroughi, N. Boysen, S. Emde, M. Schneider, High-density storage with mobile racks, Picker routing and product location, *J. Oper. Res. Soc.*, **72** (2021), 535–553. https://doi.org/10.1080/01605682.2019.1700180
6. R. Yuan, J. Li, W. Wang, J. Dou, L. Pan, Storage assignment optimization in robotic mobile fulfillment systems, *Complexity*, **1** (2021), 4679739. https://doi.org/10.1155/2021/4679739
7. K. L. Keung, C. K. M. Lee, P. Ji, Data-driven order correlation pattern and storage location assignment in robotic mobile fulfillment and process automation system, *Adv. Eng. Inform.*, **50** (2021), 101369. https://doi.org/10.1016/j.aei.2021.101369
8. N. Yang, Evaluation of the joint impact of the storage assignment and order batching in mobile-pod warehouse systems, *Math. Probl. Eng.*, **1** (2022), 9148001. https://doi.org/10.1155/2022/9148001
9. M. Çelik, C. Archetti, H. Süral, Inventory routing in a warehouse, The storage replenishment routing problem, *Eur. J. Oper. Res.*, **301** (2022), 1117–1132. https://doi.org/10.1016/j.ejor.2021.11.056
10. R. Takano, T. Higashi, Tamura H, M. Sugi, J. Ota, Mixed-load transportation scheduling of multiple agents in a warehouse environment, *Adv. Rob.*, **25** (2011), 1557–1576. https://doi.org/10.1163/016918611X579538
11. H. De Vries, R. Carrasco-Gallego, T. Farenhorst-Yuan, R. Dekker, Prioritizing replenishments of the piece picking area, *Eur. J. Oper. Res.*, **236** (2014), 126–134. https://doi.org/10.1016/j.ejor.2013.12.045
12. B. I. Kim, S. S. Heragu, R. J. Graves, A. St. Onge, Realization of a short cycle time in warehouse replenishment and order picking, *Int. J. Prod. Res.*, **41** (2003), 349–364. https://doi.org/10.1080/00207540210166321
13. J. A. Haverhals, J. A. C. Resing, J. Bierbooms, Approximation of the optimal inventory replenishment policy for multi-items used in a warehouse. PhD thesis, Eindhoven institute of Technology, 2019, 64, 67, 100. https://pure.tue.nl/ws/portalfiles/portal/130176862/Thesis_Jasmijn_Haverhals.pdf
14. S. Wang, Y. Yang, M-GAN-XGBOOST model for sales prediction and precision marketing strategy making of each product in online stores, *Data Technol. Appl*, **55** (2021), 749–770. https://doi.org/10.1108/DTA-11-2020-0286
15. K. K. Chandriah, R. V. Naraganahalli, RNN/LSTM with modified Adam optimizer in deep learning approach for automobile spare parts demand forecasting, *Multimed. Tools Appl.*, **80** (2021), 26145–26159. https://doi.org/10.1007/s11042-021-10913-0
16. R. V. Joseph, A. Mohanty, S. Tyagi, S. Mishra, S. K. Satapathy, S. N. Mohanty, A hybrid deep learning framework with CNN and Bi-directional LSTM for store item demand forecasting, *Comput. Electr. Eng.*, **103** (2022), 108358. https://doi.org/10.1016/j.compeleceng.2022.108358

17. Y. Yuan, L. Zhen, J. Wu, X. Wang, Quantum behaved particle swarm optimization of inbound process in an automated warehouse, *J. Oper. Res. Soc.*, **74** (2023), 2199–2214. https://doi.org/10.1080/01605682.2022.2129488
18. J. Hui, S. Zhi, W. Liu, C. Chu, F. Zhang, An integrated implementation framework for warehouse 4.0 based on inbound and outbound operations, *Mathematics-Basel*, **13** (2025), 2276. https://doi.org/10.3390/math13142276
19. L. Zhen, Z. Tan, R. de Koster, X. He, S. Wang, H. Wang, Optimizing warehouse operations with autonomous mobile robots, *Transport. Sci.*, 2025, https://doi.org/10.1287/trsc.2024.0800
20. K. R. Mokarrari, T. Sowlati, J. English, M. Starkey, Optimization of warehouse picking to maximize the picked orders considering practical aspects, *Appl. Math. Model.*, **137** (2025), 115585. https://doi.org/10.1016/j.apm.2024.06.037
21. J. Wu, Z. Yang, L. Zhen, W. Li, Y. Ren, Joint optimization of order picking and replenishment in robotic mobile fulfillment systems, *Transport. Res. E-Log.*, **194** (2025), 103930. https://doi.org/10.1016/j.tre.2024.103930
22. C. Barnhart, A. Jacquillat, A. Schmid, Robotic warehousing operations, a learn-then-optimize approach to large-scale neighborhood search, *INFORMS J. Optim.*, **7** (2025), 171–194. https://doi.org/10.1287/ijoo.2024.0033
23. A. Zarinchang, K. Lee, I. Avazpour, J. Yang, D. Zhang, G. K. Knopf, Adaptive warehouse storage location assignment with considerations to order-picking efficiency and worker safety, *J. Ind. Prod. Eng.*, **41** (2024), 40–59. https://doi.org/10.1080/21681015.2023.2263009