Received 19 March 2024, accepted 1 April 2024, date of publication 8 April 2024, date of current version 16 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3385791

RESEARCH ARTICLE

Item Storage Assignment Problem in Robotic Mobile Fulfillment Systems With Nonempty Pods

RUBO LI[1], **XUDONG DENG**[1,2], **AND YUNFENG MA**[1,2]
[1]School of Management, Wuhan University of Science and Technology, Wuhan 430065, China
[2]Research Center for Service Science and Engineering, Wuhan University of Science and Technology, Wuhan 430065, China

Corresponding author: Xudong Deng (dengxudong@wust.edu.cn)

This work was supported in part by the Department of Education of Hubei Province under Grant 22D022, in part by Wuhan Science and Technology Bureau under Grant 2022010801010301 and Grant 2022010801020317, and in part by China Society of Logistics and China Federation of Logistics and Purchasing under Grant 2022CSLKT3-130 and Grant 2022CSLKT3-132.

ABSTRACT This study examines the item storage assignment problem in robotic mobile fulfillment systems (RMFSs), that determines the assignment of items to pods. Unlike previous studies on this problem that generally considered an empty warehouse, the current study addresses the problem in RMFSs with nonempty pods. The problem is formulated as a mixed-integer program with the goal of maximizing the correlation degree among items. On the basis of the characteristics of the problem, an adaptive large neighborhood search (ALNS) heuristic is designed for the solution. Numerical results show that the gap between the ALNS heuristic and the Gurobi solver is less than 0.32% in small-scale instances. In medium- and large-scale instances, the ALNS heuristic improves the objective value by at least 10% compared with the methods in literature, and it outperforms commonly used storage assignment policies for traditional warehouses (e.g., random, dedicated, and class-based storage) by more than 30%. The ALNS heuristic can also be applied to the special case where the initial state of the warehouse is empty. The computational study of the ALNS heuristic in all instances shows that it can effectively solve this problem by providing high-quality solutions and has good scalability.

INDEX TERMS Robotic mobile fulfillment system, storage assignment, adaptive large neighborhood search, warehouse.

I. INTRODUCTION

E-commerce warehouses today are facing small orders, large assortment, tight delivery schedules, and varying workloads [1]. These challenges place considerable pressure on the order fulfillment process, which is considered one of the most labor-intensive and costly operations in warehouses [2]. Many giant online retailers, such as Amazon, JD.com, and Alibaba, have employed robotic mobile fulfillment systems (RMFSs), also known as KIVA systems, to improve order picking efficiency [3], [4].

Compared with traditional picker-to-parts order picking systems, RMFSs do not require pickers to travel because they adopt a parts-to-picker order picking pattern; they offer excellent system scalability, flexible deployment, and high order throughput to meet diverse customer demands [5]. Such systems use a fleet of robots to transport inventory pods (also called movable racks or shelves) between the pod storage area and picking stations. The picker at the picking station, assisted by a pick-to-light system, retrieves items in the pods in accordance with the picking list [6], [7]. After the picking operation is completed, the robot returns the pods back to the storage area. Fig. 1 depicts the layout of a typical RMFS and its operation processes.

In addition to order picking, replenishment transactions that assign storage items to inventory pods are managed by RMFSs. Such a decision is not only the basis for subsequent decision problems, such as order batching, task assignment,

The associate editor coordinating the review of this manuscript and approving it for publication was Diego Oliva.

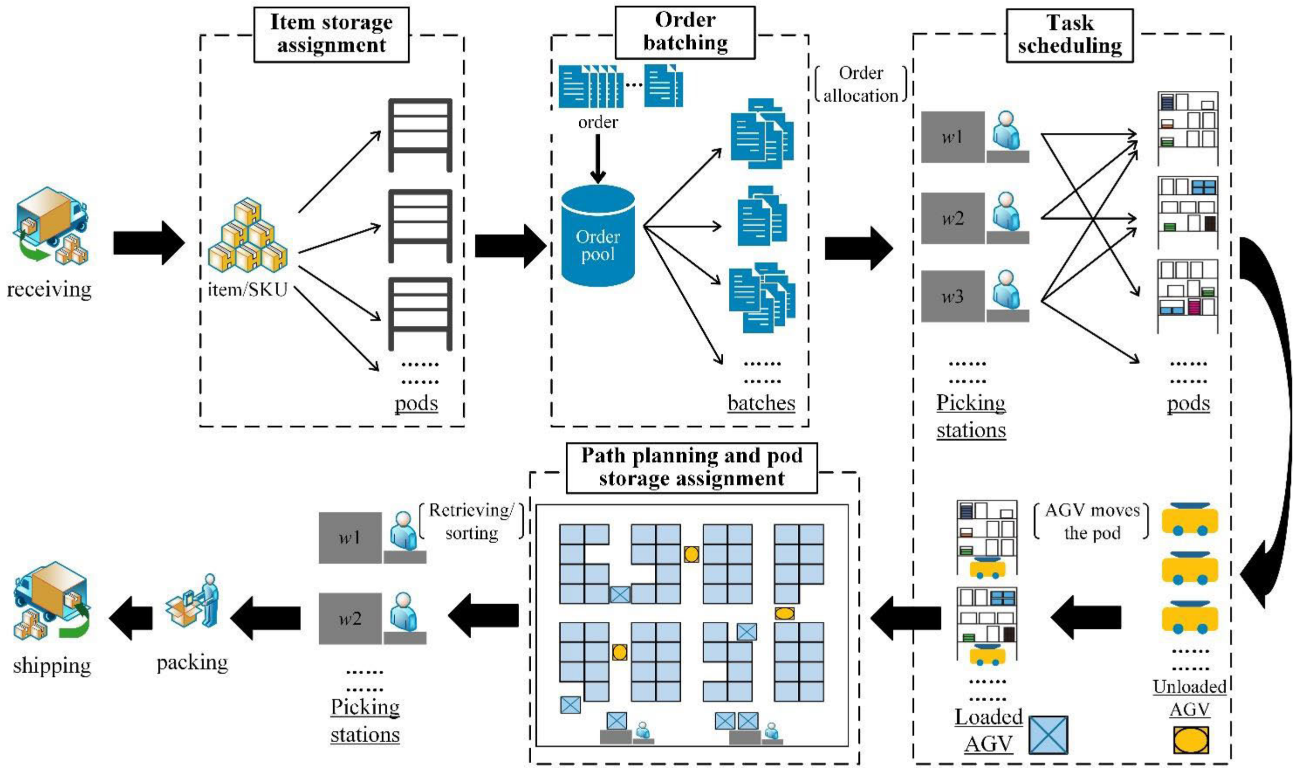


FIGURE 1. Operation processes of the RMFS warehouse.

and path planning, but also directly affects material handling costs, inventory control, and warehouse space utilization [8], [9], [10]. Given the wide variety of items and large number of pods in RMFSs, how to assign these items to pods has become one of the urgent challenges for e-commerce warehouses.

Items that frequently appear in the same order have strong correlations, and customer order analysis can be used to determine statistical correlation relationships [11]. This information should be readily available from historical order data in current information systems. In RMFS, a robot carries the entire inventory pod to a picking station whenever items are needed to fulfill a customer order. The robots can transport reduced number of pods to satisfy orders by storing correlated items in the same pods [12]. Given that RMFS generally uses identical pods and items are not dedicated to pods [5], [13], clustering correlated items in the same pods may improve order picking efficiency.

Although some previous studies have investigated item storage assignment in RMFS, they implicitly assumed that the initial state of the warehouse (all pods) is empty, or the number of empty pods is sufficient to store all replenishment items. They only needed to consider the correlation relationship between replenishment items, and the assignment restrictions on storage locations were ignored. However, in the daily operations of RMFS, the pods in the system are often partially occupied or have limited empty storage locations for replenishment items. On the basis of this fact, we consider a nonempty warehouse in this study. Moreover, given that occupied storage locations cannot be reassigned (these storage locations are unavailable), we need to consider not only the correlation relationship between replenishment items, but also the correlation relationship between replenishment items and the items already stored in the same nonempty pod. This study answers the following research questions:

1) How can items be assigned to nonempty pods in RMFS to improve order picking efficiency?
2) What improvements can be achieved by using the adaptive large neighborhood search (ALNS) heuristic designed in this study instead of the methods in literature?
3) How does optimization-based storage assignment perform compared with the storage assignment policies commonly used in traditional warehouses?

In the replenishment operation of RMFS, only the items that do not meet the required inventory levels (replenishment items) need to be replenished. Each replenishment item can be assigned to multiple pods, and the required inventory level of each item varies depending on the turnover levels. Therefore, the turnover level, scattered storage, and correlation relationship of items are simultaneously considered. First, we build a mixed-integer programming (MIP) model to optimize the assignment of replenishment items to nonempty pods, and use the maximization of the correlation degree among items as the objective function. Second, we design an ALNS heuristic for the solution. The ALNS heuristic can handle the problem under any replenishment rate level, and is applicable to the special case of an empty warehouse. Third, we test the performance of the ALNS heuristic by numerical experiments. The results show that in small-scale instances, the gap between the ALNS heuristic and the Gurobi solver is less than 0.32%. In real-world medium- and large-scale instances, the ALNS heuristic improves the objective value by at least 10% compared with methods in literature. In particular, it yields objective value increments of more than 30% compared with commonly used storage assignment policies for traditional warehouses (e.g., random, dedicated, and class-based storage). Last, we consider the properties of the items and the heterogeneous storage locations in the pods. This study contributes in the following aspects:

1) The replenishment item storage assignment (dubbed RISA) problem is studied in the nonempty RMFS.
2) A problem-specific ALNS heuristic is proposed.
3) The solution quality and efficiency of the ALNS are numerically verified.

The rest of the paper is structured as into sections. Section II provides a detailed literature review and discusses related studies on storage assignment of the RMFS. Section III describes the RISA problem in RMFS and formulates it as a MIP model. A detailed solution method for the problem is presented in Section IV. Section V presents the computational experiments that examine the performance of the proposed method. In Section VI, we further consider the properties of the items and the heterogeneous storage locations in the pods. The conclusions are given in Section VII.

II. LITERATURE REVIEW

The storage assignment decision in the traditional picker-to-parts order picking system only needs to assign items to the storage locations on the rack. Given that the rack is static and fixed, the position of the items in the storage area is determined when the items are assigned to the storage locations. For individual items, commonly used storage assignment policies include random storage, closest open location storage, dedicated storage, full turnover storage and class-based storage [2]. For different items, the relationship between items must be considered, and the items that are closely related (often appear in the same order) should be stored in adjacent locations or in the same region of the storage area, e.g., family-grouping, cluster-based assignment, and data mining-based assignment [14], [15], [16], [17], [18]. However, the storage assignment decision in RMFS is different from that in the traditional picker-to-parts order picking system. It can be divided into two basic decision problems [1], [19], [20], namely, item storage assignment and pod storage assignment, as shown in Fig. 2.

For pod storage assignment, the robot needs to return the pod to a storage position after the picking operation is finished at a picking station. Weidinger et al. [13] formalized the pod storage assignment problem as a special interval scheduling problem and designed a suitable matheuristic based on ALNS. They compared the approach with five well-established storage assignment policies (Fig. 2) and found that the shortest path storage can produce near-optimal results. Zhuang et al. [21] integrated pod retrieval and storage assignment decisions simultaneously and proposed a matheuristic decomposition method. With regard to storage policies, combined with the class-based storage policy, Yuan et al. [22] examined the performance of a velocity-based storage policy. Different from Yuan et al. [22], Li et al. [23] proposed a new turnover-rate-based decentralized storage policy that considers robot blocking and energy consumption.

With regard to item storage assignment, Kim et al. [24] and Li et al. [23] studied the item storage assignment problem with the goal of maximizing the correlation degree among items in each group and considered the situation where the correlation degree is altered. Mirzaei et al. [25] proposed an integrated cluster allocation policy that considers the correlation and turnover of items and decomposed the storage assignment problem into two phases, namely, item clustering and item assignment. Yang [26] investigated the interactive effects of the storage assignment and order batching policies in RMFS. They proposed the use of item similarity to assign item storage locations. The abovementioned studies assumed that each item can be assigned to only one storage location or one group, which is equivalent to the dedicated storage rule. However, some studies [27], [28], [29], [30] have shown that adopting scattered storage, that is, each kind of items can be distributed to several different pods, can improve order picking efficiency and is suitable for RMFS warehouses. Using information from historical customer orders, Xiang et al. [31] studied the small-scale item storage assignment problem by considering item scattered storage and solve it with the CPLEX solver. Mirzaei et al. [32] considered item correlation, turnover frequency, and inventory dispersion concurrently and proposed a correlated dispersion storage assignment policy. Ma et al. [33] further considered item classification, namely, best-selling and general-selling items. They proposed a new storage policy that covers item classification, item correlation, and scattered storage. Zhang et al. [34] considered human factors. They investigated the item storage assignment problem by considering order picking efficiency and the pickers' energy expenditure.

For zone clustering and storage assignment classification in RMFS, Keung et al. [35] proposed a data-driven approach by performing a TO-BE analysis of robotic process automation and cloud-based cyber-physical system framework. Keung et al. [36] developed a model with different storage location assignment rules and strategies under particular parties to minimize operation costs, by providing a new three-tier Industrial Internet of Things architecture that includes suppliers, RMFS, and the disposal center. Jiang et al. [37] integrated the picking and replenishment decisions and proposed a synchronization mechanism to balance the replenishment efforts and picking efficiency in a forward area applied RMFS. They considered the situation where the number of free pods (empty pods) is adequate.

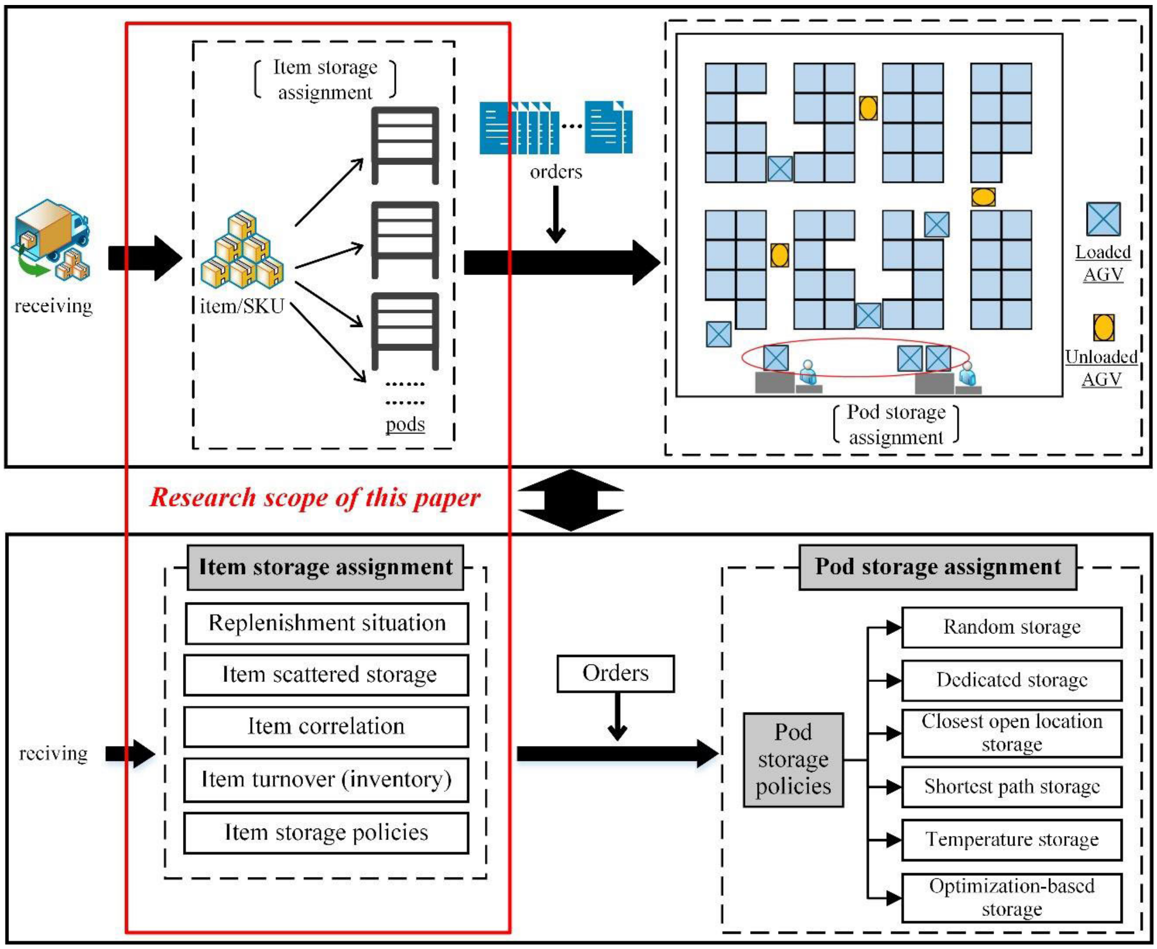


FIGURE 2. Storage assignment in the RMFS.

The difference between their work and ours is that our work considers the assignment of replenishment items to nonempty pods. Most existing scientific studies indirectly assumed that the initial state of RMFS warehouses is empty and that the storage locations in all pods can be assigned (i.e., all these storage locations are available). Guo et al. [38] studied the storage assignment problem for newly arrived items in picker-to-parts forward areas with limited open locations, whereas we studied the parts-to-picker order picking system.

The optimization objective of the item storage assignment problem in RMFS is different from that in traditional picker-to-parts order picking systems. Specifically, most of the existing studies on storage assignment in traditional picker-to-parts order picking systems aimed to minimize the total travel distance of human pickers or minimize the overall order picking time [23]. However, minimizing the number of pod visits can be the main optimization objective in RMFS [5], [37], [39]. Jiang et al. [37] justified the objective from two aspects: the number of pod visits directly reflects the movement of robots, and a reduced number of pod visits can decrease the size of the required robot fleet.

ALNS is a metaheuristic algorithm based on the idea of large neighborhood search [40], [41]. It has received considerable attention in recent years and has been successfully applied to many problem areas [42], such as vehicle routing problems [43], [44], scheduling problems [45], and pod storage assignment in RMFS warehouses [13]. The ALNS algorithm needs to design multiple destroy methods and repair methods, so applying ALNS to a new problem domain is difficult [13]. Nevertheless, by referring to the characteristics of the RISA problem, we propose a problem-specific ALNS algorithm for the solution.

In real-life e-commerce warehouses, replenishment and order picking operations are performed sequentially in a periodic manner [37], [38]. In particular, there are only some empty storage locations available in the pods after an order picking period, and the other storage locations are already occupied. The warehouse needs to replenish the items that do not meet the required inventory levels. To the best of our knowledge, existing studies on the item storage assignment problem in RMFS generally assumed an empty warehouse, which is contrary to the daily status of RMFS. To resolve this gap, this study focuses on the item storage assignment with

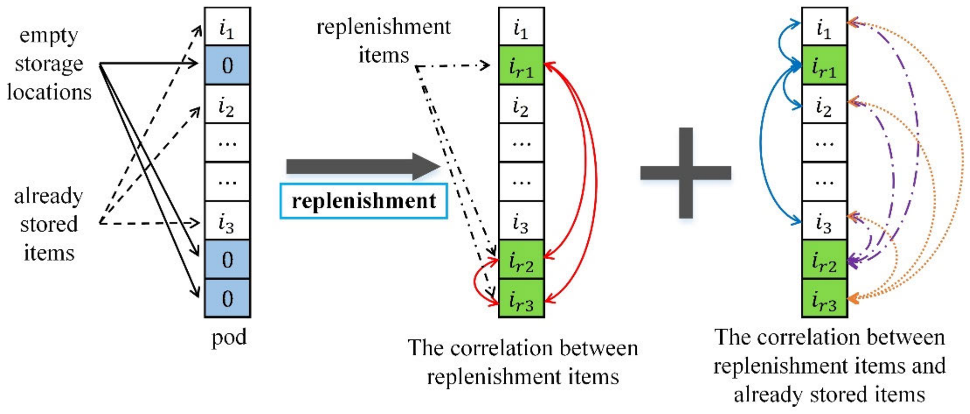

empty storage locations
already stored items
pod
replenishment
replenishment items
The correlation between replenishment items
The correlation between replenishment items and already stored items

FIGURE 3. **The correlation among items during replenishment.**

consideration of a nonempty warehouse in the replenishment operation of RMFS. It considers not only the correlation relationship between replenishment items and the items already stored in the same nonempty pod, but also the assignment restrictions of storage locations. For the specified research problem (i.e., RISA), an MIP model is built, and an efficient ALNS algorithm is proposed based on the characteristics of the problem. The model and algorithm are applicable to empty warehouses.

III. ITEM STORAGE ASSIGNMENT PROBLEM OF RMFS

A. PROBLEM DESCRIPTION

We consider N_I items stored in RMFS, and the given required inventory level of each item is D_i. These items are stored in N_M pods, and each of them has N_C storage locations (also denoted as layers or slots). D_i can be obtained by analyzing historical order data such as turnover levels, sales rates, or demand frequency. When the replenishment rate reaches a certain level, the system needs to perform the replenishment operation. When D_i is not met, the system needs to replenish item i.

In such a system, we need to assign replenishment items to the pods with empty storage locations. Each replenishment item can be assigned to several different pods simultaneously, and the required inventory level of each replenishment item has to be reached. Furthermore, the items that frequently appear in the same consumer orders can be assigned to the same pod, which can improve order picking efficiency [46]. The number of pod visits can be effectively reduced if the items ordered by many customers at the same time are stored in the same pod [24]. The relationship between these items can be measured by the correlation degree. RISA determines the storage of replenishment items in the pods to maximize the sum of the correlation degree among items.

Note that the occupied storage locations are unavailable. Thus, as shown in Fig. 3, when we choose to replenish the items, we need to consider not only the correlation between replenishment items, but also the correlation between replenishment items and the items already stored in the same nonempty pod.

In addition, the RISA problem defined above follows several assumptions:

1) Without loss of generality, we consider standard pods in RMFS.
2) We use item to represent SKU and employ the total number of storage locations required for each item i to indicate the average inventory level D_i it requires. The detailed quantity of items is simplified by assuming that only one unit of item i can be stored in each storage location in the pod. For instance, if four small items i are always stored together in one storage location, then we can treat these items as one unit of item i in the RISA problem.
3) The total remaining storage capacity is sufficient to store all replenishment items.
4) The item storage assignment decision is generally made based on historical order data within a period (such as a quarter) to analyze the correlation relationship among items. It can effectively adjust the assignment of items to adapt to seasonal changes in customer purchasing preferences. Therefore, we assume that the correlation degree of any pair of items can be obtained from historical order data.

B. JUSTIFICATION OF THE OPTIMIZATION OBJECTIVE

Existing studies of RMFS generally used the number of pod visits and the travel distance (time) of robots as the objective function [5], [13], [47], [48]. However, we take the maximization of the correlation degree among items as the objective on the basis of the following considerations.

First, because RMFS uses mobile pods, the items do not need to be assigned to a specific pod. Moreover, replenishment operations are usually performed during off-peak hours, so they are not time critical [28]. Compared with the picking operation, the replenishment operation is item-oriented rather than order-oriented. Therefore, the storage positions of pods and the travel distance of the robots that move the pods are not considered in the RISA problem.

Second, correlated storage assignment looks for correlation between items in a warehouse on the basis of their demand structure [12]. Given that item storage assignment is conducted before the order picking operation, the correlation between items needs to be analyzed based on historical order data. The higher the correlation degree is, the greater the probability that the items in the same pod will appear in the same order. In this way, the number of times that robots move the pods during the order picking operation can be effectively reduced (the travel distance of robots moving pods is also reduced), leading to an efficient and cost-saving order picking operation. This optimization objective of maximizing the sum of the correlation degree was adopted by Xiang et al. [31], Kim et al. [24], Li et al. [23], and Ma et al. [33].

The correlation degree of two items is defined as how frequently the two items are ordered together. Correlation degree is another important attribute of items that can be obtained from demand forecasts and order history [25]. Similar to Ma et al. [33], we calculated the correlation degree between items i and j in RMFS as follows:

$$r_{ij} = \begin{cases} \dfrac{O_{ij}}{O_i + O_j - O_{ij}}, & \textit{if } i \neq j; \\ 0, & \textit{if } i = j. \end{cases} \tag{1}$$

where O_i and O_j represent the number of orders that contain items i and j, respectively, and O_{ij} is the number of orders that contain both items i and j. The correlation degree between items i and j ranges from 0 to 1. However, storing only one type of item in each pod will increase the number of pod visits at the picking stations. Therefore, we set the correlation degree between each item and itself to 0, that is, $r_{ii} = 0$. This item correlation measurement method indirectly classifies the items. Given that $O_{ij} \leq \min\{O_i, O_j\}$, the correlation degree between items i and j is large only when O_i, O_j, and O_{ij} are close. For instance, best-selling item i is frequently ordered (O_i is large), whereas the general-selling item j is less frequently ordered (O_j is small); thus, the correlation degree between them is small.

C. MATHEMATICAL MODELING

On the basis of the problem description and notations summarized in Table 1, the MIP model of the RISA problem can be formulated as follows.

$$\text{Max} \left(\sum_{m\in M} \sum_{i\in I_r} \sum_{j>i} r_{ij} x_{im} x_{jm} + \sum_{m\in M} \sum_{i\in I_r} \sum_{j\in I} r_{ij} x_{im} a_{jm} \right) \tag{2}$$

$$\text{s.t.} \quad \sum_{i\in I_r} y_{im} \leq q_m, \quad \forall m \in M \tag{3}$$

$$\sum_{m\in M} y_{im} = d_i, \quad \forall i \in I_r \tag{4}$$

$$\sum_{m\in M} x_{im} \geq 1, \quad \forall i \in I_r \tag{5}$$

TABLE 1. List of notations.

Notation	Definition
Sets and parameters	
I	Set of items ($I = \{1, \ldots, N_I\}$)
I_r	Set of replenishment items ($I_r \subseteq I$)
M	Set of pods ($M = \{1, \ldots, N_M\}$)
C	Set of storage locations on each pod ($C = \{1, \ldots, N_C\}$)
i, j	Index of items ($i, j \in I$)
m	Index of pods ($m \in M$)
D_i	The total number of storage locations required for item i (required inventory level)
d_i	The number of empty storage locations required for replenishment item i
q_m	The number of empty storage locations in pod m
r_{ij}	Correlation degree between items i and j
a_{im}	1, If item i has been stored in the pod m before replenishment; 0, otherwise
b_{im}	The number of storage locations that item i has already occupied in the pod m
Decision Variables	
x_{im}	Binary variables: 1, If replenishment item i is assigned to pod m; 0, otherwise
y_{im}	Integer variables, the number of empty storage locations in pod m occupied by replenishment item i

$$x_{im} \leq y_{im}, \quad \forall i \in I_r;\, m \in M \tag{6}$$

$$y_{im} \leq x_{im} q_m, \quad \forall i \in I_r;\, m \in M \tag{7}$$

$$x_{im} \in \{0, 1\}, \quad \forall i \in I_r;\, m \in M \tag{8}$$

$$y_{im} \geq 0, \quad \forall i \in I_r;\, m \in M \tag{9}$$

The objective function (2) maximizes the sum of correlation degree among items in each pod. the first part indicates the correlation degree between replenishment items, and the second part indicates the correlation degree between the replenishment items and the items already stored in the pod. Constraint (3) ensures that the number of empty storage locations assigned to all replenishment items in each pod is smaller than or equal to the number of empty storage locations in the pod. Constraint (4) means that the number of empty storage locations assigned to each replenishment item in all pods is equal to the number of empty storage locations required for that item, that is, each replenishment item must be replenished in accordance with the required inventory level. Constraint (5) indicates that each replenishment item is assigned to at least one pod. Constraints (6) and (7) mean that if replenishment item i is assigned to pod m, it needs to occupy at least one empty storage location, but the number of empty storage locations occupied by it does not exceed the number of empty storage locations in the pod. Finally, Constraint (8) defines the binary variables, and Constraint (9) defines the integer variable.

If some pods do not have any empty storage location before replenishment, then we do not need to calculate the correlation degree among the items in these pods, because the correlation degree among the items in these pods does not change before and after replenishment. Therefore, during the replenishment operation, we only need to calculate the correlation degree among the items in the pods that have empty storage locations.

We denote the set of pods with empty storage locations as M_e. The items already stored in the pods still need to be replenished when the required number of storage locations (inventory level D_i) is not reached. Thus, in accordance with D_i, N_C, and b_{im}, d_i, I_r, q_m, and M_e can be obtained as follows:

$$d_i = D_i - \sum_{m \in M} b_{im}, \quad \forall i \in I \tag{10}$$

$$I_r = \{i \in I | d_i > 0\} \tag{11}$$

$$q_m = N_C - \sum_{i \in I} b_{im}, \quad \forall m \in M \tag{12}$$

$$M_e = \{m \in M | q_m > 0\} \tag{13}$$

In addition, we can linearize and rewrite the objective function (2). Let $z_{ijm} = x_{im} \cdot x_{jm} \in \{0, 1\}$ denotes whether or not replenishment items i and j are stored in pod m. Given that a_{jm} is known in advance, we use j_m to represent item j stored in pod m. Therefore, we can rewrite the objective function as (14) and add constraints (15), (16), and (17):

$$\text{Max} \left(\sum_{m \in M_e} \sum_{i \in I_r} \sum_{j > i} r_{ij} z_{ijm} + \sum_{m \in M_e} \sum_{i \in I_r} \sum_{j \in I} r_{ij_m} x_{im} \right) \tag{14}$$

$$z_{ijm} \le x_{im}, \quad \forall i, j \in I_r; m \in M_e \tag{15}$$

$$z_{ijm} \le x_{jm}, \quad \forall i, j \in I_r; m \in M_e \tag{16}$$

$$z_{ijm} \ge x_{im} + x_{jm} - 1, \quad \forall i, j \in I_r; m \in M_e \tag{17}$$

The storage assignment problem has been identified as NP-hard [49], [50]. As an extension of the problem, the RISA problem is also an NP-hard problem. On the basis of the characteristics of the RISA problem, we propose a ALNS heuristic in the following section.

IV. SOLUTION METHOD

This section presents the proposed ALNS for solving the RISA problem. We next introduce its overall framework, solution encoding, initial solution generation, neighborhood search, adaptive mechanism, simulated annealing acceptance criterion, and complexity analysis.

A. ALNS

The RISA problem aims to assign correlated items to appropriate pods with the goal of maximizing the sum of correlation degree among the items in the pods. Hence, a solution in the ALNS heuristic is represented by the state of item storage assignment. We use a greedy approach to generate a feasible initial solution and employ the proposed objective function to evaluate the quality of the solution. Then, the initial solution is improved by alternately destroying and repairing the solution. The use of multiple large neighborhoods within the same search allows us to find better candidate solutions in each iteration and thus traverse a more promising search path. The selection of neighborhood is controlled dynamically by using the recorded performance of the neighborhoods. The flow chart of the ALNS heuristic is given in Fig. 4.

The pseudocode of the ALNS heuristic is shown in Algorithm 1. A feasible initial solution, S_{init}, is generated by a greedy approach (Algorithm 2). Line 1 initializes the parameters of the adaptive mechanism and simulated annealing (SA) acceptance criterion, namely, the weights of destroy and repair methods w_d and w_r and temperature T. Line 2 initializes current solution S_{curr} and global best solution S_{best}. The ALNS is divided into several scoring intervals, each of which consists of several iterations (lines 4 and 6). At the beginning of each scoring interval, the scores for destroy and repair methods, θ_d and θ_r, are initialized to zero (line 5). Lines 7-10 are the most critical part of the heuristic. On lines 7 and 8, several replenishment items need to be removed from S_{curr} to generate destroyed solution S_d, and the destroy method is determined by the roulette wheel. The removed replenishment items are then inserted into S_d again to generate a new repaired solution S_r. The repair method is also determined by the roulette wheel (lines 9 and 10). Lines 11-19 determine whether the repaired solution should be accepted or not based on the solution quality and updates the best solution. Particularly, to avoid falling into a local optimum, we use a simulated annealing acceptance criterion (line 17). The score of the destroy/repair method selected in each iteration is increased by σ_1, σ_2, or σ_3 in different cases. At the end of each scoring interval, we update w_d and w_r by using the recorded scores (Section IV-E, line 22). The heuristic stops after performing a specified number of iterations.

B. SOLUTION ENCODING

Fig. 5 shows the solution encoding of the RISA problem and ALNS heuristic. A solution is encoded as a matrix with N_C rows and N_M columns. Each column corresponds to an inventory pod, and each row corresponds to a storage location of the pod. The total number of storage locations is $N = N_C * N_M$. The status of each storage location is represented by zero or i. Zero means the storage location is empty, and i refers to item i stored in the storage location. Therefore, before replenishment, we can derive information on the items stored in each pod and the state of empty storage locations in each pod (q_m) from matrix S_{rep} (the state of item storage assignment before replenishment).

In the replenishment situation, some pods may have multiple empty storage locations that need to be replenished. Given that we encode the storage location statuses of all pods, the neighborhood search (for multiple replenishment items) can cover multiple pods instead of only two. Thus, the search range is large per iteration. After the replenishment operation is completed, storage

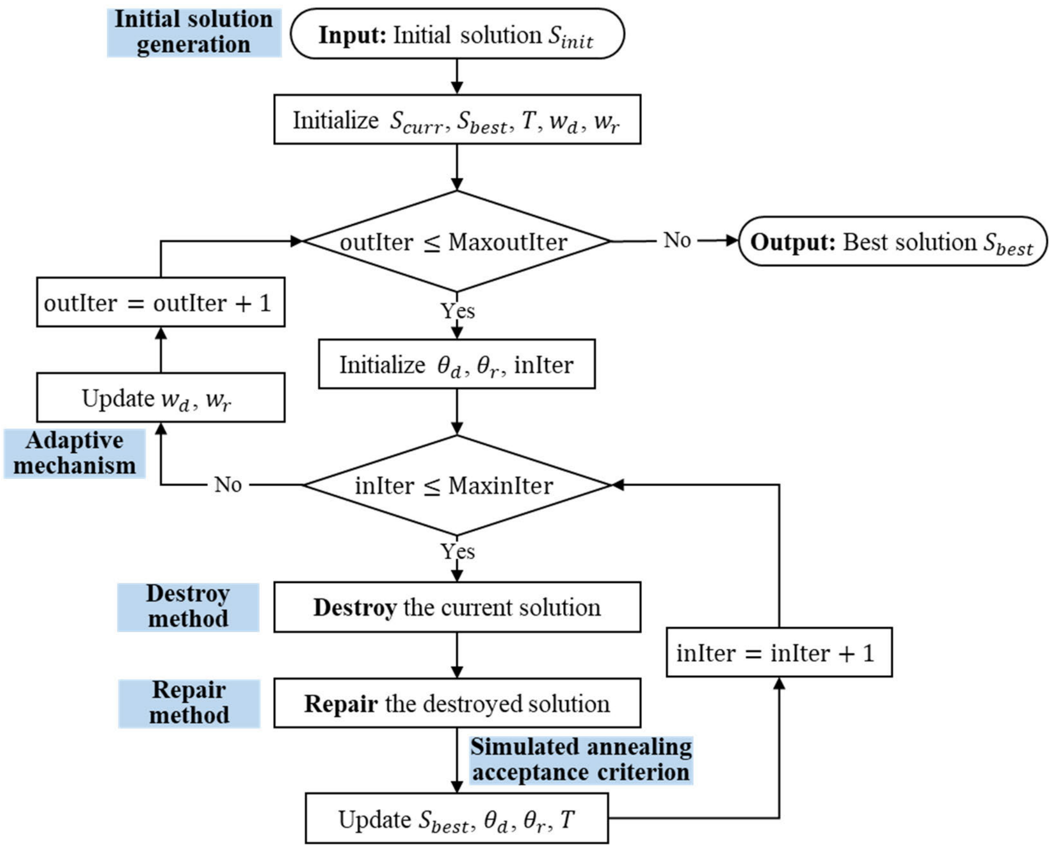
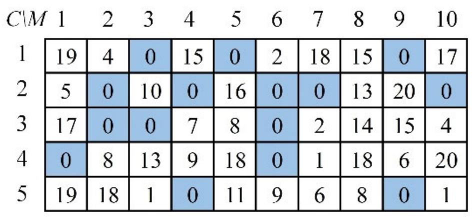

Initial solution generation
Input: Initial solution S_{init}
Initialize S_{curr}, S_{best}, T, w_d, w_r
outIter ≤ MaxoutIter
No
Output: Best solution S_{best}
outIter = outIter + 1
Yes
Initialize θ_d, θ_r, inIter
Update w_d, w_r
Adaptive mechanism
No
inIter ≤ MaxinIter
Yes
Destroy method
Destroy the current solution
inIter = inIter + 1
Repair method
Repair the destroyed solution
Simulated annealing acceptance criterion
Update S_{best}, θ_d, θ_r, T

FIGURE 4. Flow chart of the ALNS heuristic.



C\M	1	2	3	4	5	6	7	8	9	10
1	19	4	0	15	0	2	18	15	0	17
2	5	0	10	0	16	0	0	13	20	0
3	17	0	0	7	8	0	2	14	15	4
4	0	8	13	9	18	0	1	18	6	20
5	19	18	1	0	11	9	6	8	0	1

FIGURE 5. Solution encoding of the RISA problem.

location assignment of the replenishment items can be known.

C. INITIAL SOLUTION GENERATION

A greedy heuristic approach is used to generate the initial solution. Algorithm 2 presents the pseudocode of the process of feasible initial solution generation, which consists of two parts. The item correlation degree matrix, $R = (r_{ij})_{N_I \times N_I}$, is constructed based on the historical order data. For each replenishment item $i \in I_r$, if $d_i > 0$, then we find item j with the maximum correlation degree r_{ij} with item i (lines 3 and 4). Next, on line 5, we check if item j has been stored in the pod. If yes, we return to Step 1 (lines 5-11); otherwise, we proceed to Step 2 (lines 12-21).

Step 1: We check if empty storage locations l_e are present in the pods where item j is stored. If they are present, we compare the number of empty storage locations q_m of the pods with d_i. If $q_m \geq d_i$, we randomly place d_i item i in l_e. Otherwise, $d_i - q_m$ item i is randomly place in the other pods that have l_e (represented by m_e). If no l_e is available in the pods where item j is stored, then d_i item i is randomly placed in the other pods m_e.

Step 2: If item j is not present in all the pods, then item j is a replenishment item. This step checks if the pods have $q_m \geq 2$. If yes, we compare the number of these pods n_m with d_i and d_j. If $n_m \geq d_i \geq d_j$ or $d_i > n_m > d_j$ ($n_m \geq d_j \geq d_i$ or $d_j > n_m > d_i$), then we place d_j (d_i) item i and j in pairs in these pods, and put $d_i - d_j$ ($d_j - d_i$) item i (j) in the other pods m_e. If $\min\{d_i, d_j\} > n_m$, then we place n_m item i and j in pairs in these pods and place $d_i - n_m$ item i and $d_j - n_m$ item j in the other pods m_e. If no pod has $q_m \geq 2$, item i and j will be placed randomly in other pods m_e.

The algorithm ends with the condition that the number of empty storage locations required for all replenishment items is satisfied, that is, $d_i = 0$ for all $i \in I_r$.

D. NEIGHBORHOOD SEARCH

The performance and robustness of the ALNS heuristic depend on the design and choice of destroy and repair

Algorithm 1 Adaptive Large Neighborhood Search (ALNS)

```
Input: A feasible initial solution $S_{init}$
Output: Global best solution $S_{best}$
1: $T \longleftarrow T_{start}$; $w_d, w_r \longleftarrow 1$
2: $S_{curr} \longleftarrow S_{init}$; $S_{best} \longleftarrow S_{init}$
3: outIter $\longleftarrow$ 1
4: while outIter $\leq$ MaxoutIter do
                     ▷ MaxoutIter is the number of scoring intervals
5:    $\theta_d, \theta_r \longleftarrow 0$; inIter $\longleftarrow$ 1
                     ▷ Initialize the score of destroy and repair methods
6:    while inIter $\leq$ MaxinIter do
                     ▷ MaxinIter is the number of iterations
7:       destroy $\longleftarrow$ Roulettewheel(destroy methods)
8:       $S_d \longleftarrow$ destroy($S_{curr}$)
9:       repair $\longleftarrow$ Roulettewheel(repair_methods)
10:      $S_r \longleftarrow$ repair($S_d$)
11:      if $f(S_r) > f(S_{best})$ then
12:         $S_{best} \longleftarrow S_r$
13:         $\theta_d, \theta_r \longleftarrow \theta_d, \theta_r + \sigma_1$
14:      elseif $f(S_r) > f(S_{curr})$ then
15:         $S_{curr} \longleftarrow S_r$
16:         $\theta_d, \theta_r \longleftarrow \theta_d, \theta_r + \sigma_2$
17:      elseif $S_r$ is accepted by $e^{(f(S_r)-f(S_{curr}))/T}$ then
18:         $S_{curr} \longleftarrow S_r$
19:         $\theta_d, \theta_r \longleftarrow \theta_d, \theta_r + \sigma_3$
20:      Update $T$
21:      inIter = inIter + 1
22:   Update $w_d, w_r$
23:   out Iter = outIter + 1
24: return $S_{best}$
```

Algorithm 2 Initial Solution Generation

```
Input: $S_{rep}, D_i, R$
Output: $S_{init}$
1: $I, M, d_i, I_r \longleftarrow S_{rep}, D_i$
2: $S_{init} \longleftarrow S_{rep}$
3: for $i \in I_r$ and $d_i > 0$ do
4:    find $j$ with the maximum $r_{ij}$ with $i$
5:    if $j$ on pod $m$ then
6:       if $m$ have $l_e$ and $(q_m \geq d_i)$ then
7:          place $d_i$ item $i$ in pod $m$
8:       elseif $m$ have $l_e$ and $(q_m < d_i)$ then
9:          place $q_m$ item $i$ in pod $m$ and $(d_i - q_m)$ item $i$ in the other pods $m_e$
10:      else $q_m = 0$
11:         place $d_i$ item $i$ in other pods $m_e$
12:   else $j \in I_r$
13:      if the pods $m$ have $q_m \geq 2$ then
14:         if $(n_m \geq d_i \geq d_j)$ or $(d_i > n_m > d_j)$ do
15:            pair $d_j$ item $i$ and $j$ place in $n_m m$, place $(d_i - d_j)$ item $i$ in the other pods $m_e$
16:         elseif $(n_m > d_j > d_i)$ or $(d_j > n_m > d_i)$ do
17:            pair $d_i$ item $i$ and $j$ place in $n_m m$, place $(d_j - d_i)$ item $j$ on other pods $m_e$
18:         else
19:            pair $n_m$ item $i$ and $j$ in $m$, place $(d_i - n_m)i$ and $(d_j - n_m)j$ in the other pods $m_e$
20:   else
21:      place $d_i$ item $i$ and $d_j$ item $j$ in other pods $m_e$
22:   Update $q_m, d_i$
23: return $S_{init}$
```

methods. We introduce several destroy and repair methods to improve the incumbent solution.

1) DESTROY METHOD

In this section, we introduce three destroy methods. All of them use current solution S_{curr} as the input. A destroyed solution S_d and a set of removed replenishment items I_d are the output of the destroy methods.

a: RANDOM DESTROY

The random destroy method selects L replenishment items randomly and removes them from the current solution. The procedure of random destroy is shown in Algorithm 3. Lines 2 and 3 set the minimum (L_{min}) and maximum (L_{max}) replenishment items that can be removed. N_m is the total number of pods with replenishment items. We randomly select L replenishment items to be removed. ξ is a uniform random number between zero and one (line 4). Next, we randomly remove L replenishment items from S_d (lines 5 and 6).

b: POD DESTROY

The pod destroy method removes only one replenishment item from each of the randomly selected L pods where

Algorithm 3 Random Destroy

```
Input: $S_{curr}$
Output: $S_d, I_d$
1: $S_d \longleftarrow S_{curr}$
2: $L_{min} \longleftarrow 2$
3: $L_{max} \longleftarrow \max(\lceil \sqrt{N_M} \rceil, 10)$
4: $L \longleftarrow L_{min} + \lceil (L_{max} - L_{min}) * \xi \rceil$
5: $S_d \longleftarrow$ randomly remove $L$ replenishment items from $S_d$
6: $I_d \longleftarrow$ the $L$ replenishment items that have been removed
```

replenishment items are stored. The procedure of the pod destroy is shown in Algorithm 4. We count the number of pods (N_m) with replenishment items and randomly select L of them (lines 2-5). For each pod, we randomly remove one of the replenishment items i from it and update S_d and I_d (lines 6-9).

c: WORST DESTROY

The worst destroy method selects the replenishment items that appear to be assigned to the wrong pods. Given replenishment item i assigned to a pod, we define the correlation degree reduction of the replenishment item as $\nabla_i = f(S) - f_{-i}(S)$ where $f(S)$ is the correlation degree of the solution and $f_{-i}(S)$ is the correlation degree of the

Algorithm 4 Pod Destroy

```
Input: $S_{curr}$
Output: $S_d, I_d$
1: $S_d \longleftarrow S_{curr}$
2: $N_m \longleftarrow$ count the number of pods where replenishment items are stored
3: $L_{min} \longleftarrow 2$
4: $L_{max} \longleftarrow \max(\left\lceil \sqrt{N_M} \right\rceil, 10)$
5: $L \longleftarrow L_{min} + \lceil (L_{max} - L_{min}) * \xi \rceil$
6: for $m \longleftarrow 1$ to $L$ do
7:     randomly remove a replenishment item $i$ from pod $m$
8:     $S_d \longleftarrow$ remove replenishment item $i$ from $S_d$
9:     $I_d \longleftarrow$ add replenishment item $i$ to $I_d$
```

Algorithm 5 Worst Destroy

```
Input: $S_{curr}$
Output: $S_d, I_d$
1: $S_d \longleftarrow S_{curr}$
2: The number of replenishment items $N_r$ is divided into $L$ and $(N_r - L)$
3: for $i \longleftarrow 1$ to $L$ do
4:     calculate the correlation degree reduction $\nabla_i$ of the replenishment item $i$
5: Sorting replenishment items in ascending order of $\nabla_i$, and removing the first $\lceil L/2 \rceil$ replenishment items
6: Randomly remove $L - \lceil L/2 \rceil$ replenishment items among the other $(N_r - L)$ replenishment items
7: Update $S_d, I_d$
```

solution without replenishment item i. Removing replenishment items with a small correlation degree reduction and placing them in other pods to obtain an improved solution value are feasible and reasonable. We introduce the worst destroy method that removes replenishment items with low ∇_i.

The worst destroy method is shown in Algorithm 5. On line 2, we divide the total number of replenishment items (N_r) into two parts L and $N_r - L$. For these L replenishment items, we successively calculate the correlation degree reduction of each replenishment item and remove the $\lceil L/2 \rceil$ replenishment items with the smallest correlation degree reduction (lines 3-5). For the other $N_r - L$ replenishment items, we randomly remove $L - \lceil L/2 \rceil$ replenishment items (line 6).

2) REPAIR METHOD

This section describes two kinds of repair methods. They adopt a destroyed solution S_d and a set of removed replenishment items I_d as their input and a new repaired solution S_r as their output. Given that L replenishment items have been removed, we now have L empty storage locations, denoted by the set Ω_e.

Algorithm 6 Max Correlation Repair

```
Input: $S_d, I_d$
Output: $S_r$
1: $S_r \longleftarrow S_d$
2: while $I_d \neq \emptyset$ do
3:     $\Omega_e \longleftarrow$ the set of empty locations
4:     for $i \in I_d$ do
5:         $\Delta_{i,l_e} \longleftarrow i$ reinsert in each empty location of $\Omega_e$
6:         $\alpha_i \longleftarrow \max\{\Delta_{i,l_e}\}$
7:     $i \longleftarrow \max\{\alpha_i\}$
8:     $S_r \longleftarrow$ insert $i$ at its maximum correlation location
9:     $I_d \longleftarrow$ remove $i$ from $I_d$
10: return $S_r$
```

a: MAX CORRELATION REPAIR

The max correlation repair method is a greedy construction heuristic. The pseudocode of the max correlation repair method is shown in Algorithm 6. It performs at most L iterations as it inserts one replenishment item into each iteration (lines 3-6). Let Δ_{i,l_e} denote the increment in the objective value caused by assigning replenishment item i to empty storage location l_e. We define $\alpha_i = \max_{l_e \in \Omega_e}\{\Delta_{i,l_e}\}$ as the correlation degree increment of assigning replenishment item i to its best storage location (increases the objective value the most). This location is denoted by the *maximum correlation location*. Then, we choose the replenishment item i that maximizes α_i for $i \in I_d$ and insert it into its maximum correlation location (lines 7 and 8). This process is repeated until all replenishment items have been assigned or no more replenishment items can be assigned. In each iteration, we only choose one pod (the one we assign to). Thus, we do not need to recalculate the correlation degree increment for all other pods. This feature can be used to increase the speed of the repair method during the actual implementation of the algorithm.

b: REGRET REPAIR

Regret heuristics have been applied to the vehicle routing problems with time windows [51] and pickup and delivery problems with time windows [41]. It can also be used for other combinatorial optimization problems such as the generalized assignment problem [41]. The regret heuristic can be used for RISA because of the problem's characteristics. The pseudocode of the regret repair method is shown in Algorithm 7.

The regret repair method improves the max correlation repair method by adopting forward-look information during the selection of the replenishment items to insert. Unlike in the max correlation repair method, in the regret repair method, *regret value* δ_i is defined as

$$\delta_i = \max_{l_e \in \Omega_e} \left\{\Delta_{i,l_e}\right\} - \operatorname{submax}_{l_e \in \Omega_e} \left\{\Delta_{i,l_e}\right\} \quad (18)$$

Algorithm 7 Regret Repair

```
Input: $S_d, I_d$
Output: $S_r$
1: $S_r \longleftarrow S_d$
2: while $I_d \neq \emptyset$ do
3:   $\Omega_e, n_e \longleftarrow$ the set of empty locations and the number of the empty locations
4:   if $n_e \geq 2$ do
5:     for $i \in I_d$ do
6:       $\Delta_{i,l_e} \longleftarrow i$ reinsert in each empty location of $\Omega_e$
7:       $\alpha_i \longleftarrow \max\{\Delta_{i,l_e}\}$
8:       $\delta_i \longleftarrow \max\{\Delta_{i,l_e}\} - \text{submax}\{\Delta_{i,l_e}\}$
9:     $i \longleftarrow \max\{\delta_i\}$
10:    $S_r \longleftarrow$ insert $i$ at its maximum correlation location
11:    $I_d \longleftarrow$ remove $i$ from $I_d$
12:  else
13:      $S_r \longleftarrow$ insert $i$ into the last empty storage location
14:      $I_d \longleftarrow$ remove $i$ from $I_d$
15: return $S_r$
```

when the number of empty storage locations, n_e, in set Ω_e is greater than 2 (lines 4-8). The regret value is the difference in the correlation degree increment between assigning replenishment item i to its best storage location and its second-best storage location. In each iteration, the regret repair method inserts replenishment item i that maximizes δ_i. Then, replenishment item i is inserted into its maximum correlation location (lines 9-11). If only one empty storage location is left, it inserts the last replenishment item directly (lines 12-14).

E. ADAPTIVE MECHANISM

In Section IV-D, we defined three destroy methods (random, pod, and worst destroy) and two kinds of repair methods (max correlation and regret repair). During the iterative search process of the heuristic, each destroy and repair method is assigned a weight that determines its probability of being selected. The destroy and repair methods that perform well in the search process (they improve the solution many times) have a large weight.

In accordance with Ropke and Pisinger [41], we assign weights to the different methods and use a roulette wheel selection principle. If we have k methods with weights w_i, $i \in \{1, 2, \ldots, k\}$, then we select method i with probability $w_i \big/ (\sum_{i=1}^{k} w_i)$. The destroy method is selected independently of the repair method (and vice versa). The weights of the destroy and repair methods can be automatically adjusted using statistics from the previous iterations. When an iteration of the ALNS heuristic is completed, the score for the destroy and repair method used in the iteration is increased in different cases, as shown in Table 2.

After we have finished a scoring interval, we recalculate the weight for all methods to be used in the next scoring interval as follows:

TABLE 2. Score adjustment parameters.

Parameter	Description
σ_1	If the repaired solution is a new global best solution
σ_2	If the repaired solution is better than the current solution
σ_3	If the repaired solution is accepted by the simulated annealing acceptance criterion

$$w_{i,j+1} = \begin{cases} w_{ij}\left(1-\lambda\right) + \lambda \dfrac{\theta_i}{\mu_i}, & \text{if } \mu_i \neq 0; \\ w_{ij}\left(1-\lambda\right), & \text{if } \mu_i = 0, \end{cases} \tag{19}$$

where w_{ij} is the weight of method i used in scoring interval j, θ_i is the total score of method i obtained during the last scoring interval, and μ_i is the number of times method i was used during the last scoring interval. $\lambda \in [0, 1]$ is the reaction factor that controls the reaction speed of weight adjustment.

F. SIMULATED ANNEALING ACCEPTANCE CRITERION

We only accept solutions that are superior to the current one. This acceptance criterion entails the risk of falling into a local optimum. To overcome this issue, we use a simulated annealing acceptance criterion to increase search diversity. It accepts solutions that are worse than the current one with a certain probability. Given current solution S_{curr}, we accept a new deteriorating repaired solution S_r with the probability $e^{(f(S_r)-f(S_{curr}))/T}$ (the RISA problem is a maximization problem), where $T > 0$ denotes the current temperature. The temperature starts from T_{start} and decreases at a cooling rate of $\varsigma \in (0, 1)$ until the freezing temperature, T_{end}, is met.

G. COMPLEXITY ANALYSIS OF THE PROPOSED ALGORITHM

According to the pseudocode of the ALNS heuristic, the time complexity of the initial solution generation is $O(N_r)$ (Algorithm 2), where N_r is the total number of types of the replenishment item. The time complexities of the three destroy methods, namely, Algorithms 3-5, are $O(1)$, $O(L)$, and $O(L \times N_C)$, respectively, where L is the number of replenishment items that need to be removed in each iteration, and N_C is the number of storage locations in each pod. The time complexity of each repair method is $O(L \times L \times N_C)$. Therefore, the time complexity of the ALNS heuristic is

$$O\left(N_r + iter \times (1 + L + L \times N_C + 2 \times L^2 \times N_C)\right),$$

where $iter$ is the total number of iterations (MaxoutIter $\times$ MaxinIter). Given that L equals $\sqrt{N_M}$ at the maximum, the time complexity of the ALNS heuristic can be simply expressed as $O\left(iter \times N\right)$, where N is the total number of storage locations of all pods.

TABLE 3. Parameter for instance generation.

Symbol	Description	Small	Medium	Large
$\left\lceil \widecheck{N_I}, \widehat{N_I} \right\rceil$	Min./Max. type of items	[20, 100]	[500, 3000]	[5000, 10000]
$\left\lceil \widecheck{N_M}, \widehat{N_M} \right\rceil$	Min./Max. number of pods	[10, 100]	[500, 2000]	[4000, 5000]
$\left\lceil \widecheck{N_C}, \widehat{N_C} \right\rceil$	Min./Max. capacity of each pod	5	[5,9]	[5, 9]
η	Replenishment rate	0.25	0.25	0.25

V. COMPUTATIONAL EXPERIMENTS

This section verifies the performance of ALNS through numerical experiments. We generate the instances because the RISA problem has no available test data. The details of the generated instances are presented in Section V-A. The performance of the ALNS heuristic is presented in Section V-B. In Section V-C, we compare the storage assignment derived by optimization (i.e., the RISA problem solved by ALNS) with well-known storage assignment policies applied in traditional warehouses. We perform a sensitivity analysis in Section V-D.

All computations are executed on a 64-bit PC with an Intel Core-i7-8565U CPU (1.8 × 1.99 GHz) and 8 GB of RAM. The procedures are implemented using MATLAB R2015a, and the MIP model is solved using the standard solver Gurobi (version 9.5.1).

A. INSTANCE GENERATION AND PARAMETERS

The performance of the proposed ALNS heuristic is evaluated in small-, medium- and large-scale test instances. The small-scale instances can still be solved to produce optimal or reasonable results by the standard solver Gurobi, and the medium- and large-scale instances are solved in realistic scenarios where optimal solutions cannot be obtained. Table 3 summarizes the parameters inputted to our generator.

The settings of the instance generation and algorithm experiments are elaborated as follows:

1) With reference to literature [31], [33], [48], we set the parameters of the warehouse (N_I, N_M, and N_C) to generate different test instances, as shown in Table 3. Then, we randomly generate the required inventory level of each item D_i and item correlation degree matrix R. The state of item storage assignment before replenishment, S_{rep}, is randomly generated based on the warehouse parameters.
2) The replenishment operation is based on the replenishment rate η, which is the ratio of the total number of empty storage locations N_e to the total number of storage locations in all pods N. $\eta = 0$ means that the pod has no empty storage location, and no replenishment is required. $\eta = 1$ means all pods are empty. If the replenishment operation is performed when η is low, the warehouse needs to be replenished frequently. Otherwise, an out-of-stock situation could arise in the picking process [28]. Moreover, the replenishment operation is usually launched by following the (s, S) policy [52]. The replenishment operation is performed when the replenishment rate $\eta = 25\%$ or $N_e = 1000$, that is, N_e accounts for 25% of N or $N_e = 1000$ before replenishment.
3) We tune the parameters of the ALNS heuristic by referring to literature [40], [41]. We start with a good range and then tune one parameter once. The process is repeated until all parameters are tuned. As shown in Table 4, the best parameters settings are selected for all computational experiments.
4) Ten independent runs are performed for each instance to make the numerical results reliable, and we use the average result. In Gurobi, the CPU time required to solve the small-scale instances is limited to 1800 seconds. The ALNS heuristic and other comparison methods have the same parameters. The gap between the ALNS heuristic and the other methods is calculated as

$$\text{Gap} = \frac{f\ (\text{ALNS}) - f(X)}{f(X)} \times 100\% \tag{20}$$

where f (ALNS) represents the objective value of the ALNS heuristic and f (X) is the objective value of the comparison method. For comparison purposes, we have marked the best results in bold.

B. ALGORITHM PERFORMANCE

In this section, we assess the efficiency and performance of the ALNS heuristic. We also investigate the effects of different destroy and repair methods.

1) COMPARISON WITH THE STANDARD SOLVER GUROBI

We set the small-scale instances to about the maximum instance size that Gurobi can solve with reasonable resources. The numerical results are shown in Table 5, where negative values indicate that the results obtained by Gurobi are better than those obtained by ALNS. The gap in the objective values of ALNS and Gurobi varies from -0.32% to 3.37%. The running time of Gurobi increases sharply as the problem size increases, so in the slightly large cases, Gurobi cannot obtain the optimal solution within a short time. ALNS can derive the optimal solution in some small-scale instances. Although Gurobi can obtain a good solution in some small-scale cases, the gap between ALNS and Gurobi is smaller than 0.32%. When the type of items N_I exceeds 50 and the number of pods

TABLE 4. Parameters for algorithm.

Parameter	Description	Value
MaxoutIter	The number of scoring intervals (segment)	120
MaxinIter	The number of iterations of each scoring interval	100
T_{start}	Initial temperature	1
T_{end}	Freezing temperature	0.001
ς	Cooling rate	0.998
λ	Reaction factor	0.1
σ	Scores	$\sigma_1 = 40, \sigma_2 = 20, \sigma_3 = 10$
L	Scale of destruction	$\left[2, \max\left(\left\lceil\sqrt{N_M}\right\rceil, 10\right)\right]$

N_M exceeds 40, the results of ALNS are better than those of Gurobi.

We record the time during the performance comparison with Gurobi to show that ALNS solves fast and can provide high-quality solutions. Order picking operations in e-commerce retailer warehouses are performed under considerable time pressure because of the tight delivery schedules promised to the customers, but replenishment operations (e.g., once a day or half a day) can be postponed to off-peak periods [28]. The ALNS heuristic can still provide ideal solutions within 40 minutes even in the large-scale instances. Therefore, we do not record the running time of the ALNS heuristic during the comparison with other heuristics and storage assignment policies in the medium- and large-scale instances.

2) COMPARISON WITH EXISTING HEURISTICS

Other general heuristic algorithms, such as greedy construction heuristic (GCH) [25], variable neighborhood search (VNS) [37], and simulated annealing (SA) or hybrid optimization algorithm [33], can also be used to solve the item storage assignment problem. However, these methods are developed for warehouses with an empty initial state or other situations. They are not directly applicable to the RISA problem because some pods may have no empty storage locations, whereas others may have multiple empty storage locations. Therefore, we make some adjustments in these algorithms to make them adapt to the RISA problem. In addition, we test the performance of ALNS in an empty warehouse (see Appendix).

GCH is similar to the greedy heuristic we used to find the feasible initial solution, so we apply this framework to GCH. For the VNS algorithm, we need to design different neighborhood structures. Similar to literature [33], we exchange single or many different replenishment items between two pods or among multiple pods. However, we need to identify the matching pods first. Pods without replenishment items or those with a mismatched number of replenishment items cannot be processed. We have to find these pods not only to store replenishment items, but also to match the number of replenishment items. Then, we apply the SA acceptance criterion to VNS (i.e., VNS-SA). We also design an adaptive VNS (AVNS) by using an adaptive mechanism (each neighborhood structure, rather than individual methods, is assigned a weight).

The numerical results of the medium- and large-scale instances are shown in Table 6 and Table 7, respectively. In all the medium-scale instances, ALNS outperforms the four comparison algorithms in terms of solution quality. The gap between GCH and ALNS ranges from 29.0% to 37.0%, and the average value is 32.3%. The gap between VNS, VNS-SA, AVNS and ALNS ranges from 11.8% to 15.9%, and the average values are 14.3%, 14.7% and 14.4%, respectively. In all the large-scale instances, ALNS also performs better than the four comparison algorithms in terms of solution quality. In particular, ALNS outperforms GCH with a gap between 31.4% to 47.7%, where the average value is 38.6%. The gap between VNS, VNS-SA, AVNS and ALNS ranges from 14.7% to 22.1%, and the average values are 17.3%, 17.1%, and 17.2%, respectively.

The RISA problem aims to maximize the sum of correlation degree among items in pods by assigning correlated items to appropriate pods. A high correlation degree can reduce the number of pod visits during order picking operations [33]. As a result, the travel distance of robots and the order retrieval time can be reduced accordingly [23], [25], [32]. The high performance of the ALNS heuristic in the RISA problem has a considerable positive effect on the travel distance of robots and order retrieval time. It will result in more efficient and cost-saving order picking operations. This can help e-commerce retailers meet growing customer demands and maintain high-quality logistic service.

In conclusion, the ALNS heuristic shows a better performance than GCH, VNS, VNS-SA, and AVNS in all instances. The reasons for this result are as follows:

1) For GCH, although the replenishment items have the maximum correlation degree with themselves, the correlation degree with the other items in the same pod is not considered. Meanwhile, we aim to maximize the sum of correlation degree among all items in each pod.
2) Solution encoding and neighborhood search. VNS approaches (VNS-SA, AVNS) encode the solution for each individual pod. Even when multiple pairs of pods are present, the neighborhood structure of VNS

TABLE 5. Performance comparison between ALNS and Gurobi in small-scale instances.

N_I-N_M-N_C	Gurobi		ALNS		Gap (%)
	Obj.	Time (s)	Obj.	Time (s)	
20-10-5	**25.4649**	0.96	**25.4649**	13.73	0
20-12-5	**29.9847**	1.83	**29.9847**	13.8	0
20-14-5	**38.1205**	11.34	**38.1205**	18.89	0
20-16-5	**43.5207**	165.24	**43.5207**	20.13	0
20-18-5	**46.939**	1800	46.7888	19.52	-0.32
20-20-5	**53.19**	1800	**53.19**	21.61	0
30-20-5	**56.303**	1800	**56.303**	21.9	0
30-30-5	**87.378**	1800	87.3072	31.75	-0.08
30-40-5	**103.4046**	1800	103.1977	34.17	-0.20
50-40-5	113.4	1800	**113.5586**	32.64	0.14
50-50-5	148.746	1800	**149.0937**	41.3	0.23
50-60-5	176.127	1800	**176.649**	42.98	0.30
80-60-5	183.205	1800	**186.1926**	45.42	1.63
80-70-5	208.05	1800	**211.8454**	45.04	1.82
80-80-5	235.823	1800	**240.6522**	55.91	2.05
100-80-5	253.185	1800	**253.8293**	55.84	0.25
100-90-5	286.3264	1800	**288.4559**	60.25	0.74
100-100-5	296.5367	1800	**306.5379**	62.11	3.37

TABLE 6. Performance comparison between ALNS and other algorithms in medium-scale instances.

N_I-N_M-N_C	GCH	VNS	VNS-SA	AVNS	ALNS	Gap (%)			
						GCH	VNS	VNS-SA	AVNS
500-500-5	1229.7	1461.1	1442.5	1453.2	**1661.5**	35.1	13.7	15.2	14.3
600-500-5	1225.3	1457.4	1449.0	1470.8	**1647.6**	34.5	13.1	13.7	12.0
700-500-5	1218.4	1463.9	1455.8	1468.2	**1645.2**	35.0	12.4	13.0	12.1
800-500-5	1203.5	1443.8	1453.6	1441.9	**1636.6**	35.9	11.8	12.6	13.5
900-500-5	1203.2	1445.6	1448.3	1456.3	**1647.9**	37.0	14.0	13.8	13.2
1000-500-5	1210.6	1461.4	1456.6	1451.5	**1658.2**	37.0	13.5	13.8	14.2
1000-1000-7	3050.1	3459.0	3460.0	3465.9	**4005.2**	31.3	15.8	15.8	15.6
1200-1000-7	3043.3	3465.4	3457.1	3472.5	**3983.5**	30.9	14.9	15.2	14.7
1400-1000-7	3016.1	3464.3	3446.0	3452.1	**3978.8**	31.9	14.9	15.5	15.3
1600-1000-7	3071.8	3478.5	3469.4	3472.4	**4001.8**	30.3	15.0	15.3	15.2
1800-1000-7	3004.0	3448.9	3446.0	3452.1	**3993.4**	32.9	15.8	15.9	15.7
2000-1000-7	3050.5	3499.1	3488.9	3496.8	**4028.8**	32.1	15.1	15.5	15.2
2000-2000-9	4160.4	4708.8	4680.1	4711.3	**5390.3**	29.6	14.5	15.2	14.4
2200-2000-9	4172.9	4691.5	4679.8	4688.2	**5395.4**	29.3	15.0	15.3	15.1
2400-2000-9	4173.3	4700.9	4710.2	4705.5	**5383.1**	29.0	14.5	14.3	14.4
2600-2000-9	4167.5	4714.2	4717.8	4720.9	**5398.6**	29.5	14.5	14.4	14.3
2800-2000-9	4167.9	4716.3	4697.5	4708.1	**5410.0**	29.8	14.7	15.2	14.9
3000-2000-9	4162.4	4694.5	4682.7	4690.8	**5392.1**	29.5	14.9	15.1	15.0
Average	-	-	-	-	-	32.3	14.3	14.7	14.4

(VNS-SA, AVNS) is limited to the exchange of replenishment items between two pods. Therefore, the solution space of each iteration by VNS (VNS-SA, AVNS) is small, and the solution easily falls into a local optimum. In the ALNS heuristic, each neighborhood can be considered a unique combination of a destroy and repair method. The design of destroy and repair methods can effectively diversify and intensify the search. The large neighborhood also allows the ALNS heuristic to navigate the solution space eas-

TABLE 7. Performance comparison between ALNS and other algorithms in large-scale instances.

N_I-N_M-N_C	GCH	VNS	VNS-SA	AVNS	ALNS	Gap (%)			
						GCH	VNS	VNS-SA	AVNS
5000-4000-5	2103.9	2518.8	2525.1	2523.7	**3004.2**	42.8	19.3	19.0	19.0
6000-4000-5	2061.5	2505.5	2516.3	2502.2	**3001.5**	45.6	19.8	20.1	20.0
7000-4000-5	2048.1	2494.7	2498.7	2452.3	**2993.8**	46.1	20.0	19.8	22.1
8000-5000-5	2043.2	2511.9	2522.3	2530.2	**2984.3**	46.1	18.8	18.3	17.9
9000-5000-5	2062.1	2508.3	2527.2	2511.4	**2995.2**	45.2	19.4	18.5	19.3
10000-5000-5	2029.8	2499.1	2511.0	2520.3	**2998.9**	47.7	20.0	19.4	19.0
5000-4000-7	3157.6	3660.8	3646.4	3662.5	**4258.7**	34.9	16.3	16.8	16.3
6000-4000-7	3126.3	3653.1	3662.3	3651.7	**4249.1**	35.9	16.3	16.0	16.4
7000-4000-7	3117.9	3649.5	3657.0	3647.0	**4243.2**	36.1	16.3	16.0	16.3
8000-5000-7	3073.6	3645.9	3640.3	3642.7	**4277.0**	39.2	17.3	17.5	17.4
9000-5000-7	3099.9	3643.2	3647.7	3650.2	**4283.7**	38.2	17.6	17.4	17.3
10000-5000-7	3090.7	3630.3	3635.8	3632.3	**4266.4**	38.0	17.5	17.3	17.5
5000-4000-9	4154.5	4743.4	4754.7	4760.8	**5459.9**	31.4	15.1	14.8	14.7
6000-4000-9	4142.8	4768.1	4772.1	4766.1	**5478.3**	32.2	14.9	14.8	14.9
7000-4000-9	4137.2	4765.6	4762.7	4772.9	**5510.0**	33.2	15.6	15.7	15.4
8000-5000-9	4116.3	4753.0	4749.8	4758.4	**5503.2**	33.7	15.8	15.9	15.7
9000-5000-9	4123.2	4755.1	4768.5	4760.3	**5531.5**	34.2	16.3	16.0	16.2
10000-5000-9	4075.3	4742.2	4755.9	4752.0	**5473.9**	34.3	15.4	15.1	15.2
Average	-	-	-	-	-	38.6	17.3	17.1	17.2

ily. The problem-specific solution encoding and the destroy/repair methods can cover multiple pods instead of only two. The search range is large per iteration and the overall search speed of the solution space is fast. Thus, the ALNS heuristic can find high-quality solutions with high convergence speed.

The time complexity of the ALNS heuristic is related to the total number of storage locations in all pods, namely, the size of the RMFS warehouse. In the small-scale instances, the ALNS heuristic can obtain a good solution or even an optimal one within a short time. In the medium- and large-scale instances, the solutions obtained by the ALNS heuristic are better than those produced by the comparison algorithms. Moreover, although the replenishment operation is not time-critical and can be postponed to off-peak hours [28], the ALNS heuristic can still produce good solutions within an acceptable time frame. The performance of the ALNS heuristic in all instances shows that it has good scalability and ensures a proper trade-off between solution quality and time consumption.

3) EFFECT OF THE DESTROY AND REPAIR METHODS

To examine the effect of each destroy/repair method and their combination with each other, we record how frequently they are selected when solving the medium-scale instances with ALNS. The three destroy methods (random destroy-RD, pod destroy-PD, worst destroy-WD) and the two repair methods (max correlation repair-MR, regret repair-RR) can be merged into six combinations, i.e., RD-MR, RD-RR, PD-MR, PD-RR, WD-MR, WD-RR. In the adaptive mechanism of ALNS, the probability of each combination being selected is dynamically adjusted in accordance with to the combination's performance, and the combinations that lead to a large improvement in the solution are selected more frequently. In Table 8, we show the frequency percentages (invoking times) of the destroy/repair methods and six combinations. The invoking frequencies of the three destroy methods (RD, PD and WD) and two repair methods (MR and RR) do not vary considerably. WD-RR (23.9%) is the most promising among the combinations. WD (40.7%) with RR (58.1%) frequently generates better solutions, so its probability of being selected is high.

C. COMPARISON OF TRADITIONAL STORAGE ASSIGNMENT POLICIES

This section compares the RISA problem solved by ALNS with well-known storage assignment policies for traditional warehouses, including random storage, dedicated storage, closest open location storage, full-turnover storage and class-based storage [2].

Random storage randomly assigns items to fitting empty storage locations with an equal probability. This policy (also called RND) can be applied directly to RMFS by randomly assigning replenishment items to pods with empty storage locations. **Closest open location storage** assigns items to the empty storage locations closest to the depot. This policy is not directly transferable to the RISA problem. In order to apply the basic idea of closest open location storage, we recommend

TABLE 8. Frequency of selected destroy/repair method combinations.

Destroy method	RD	RD	PD	PD	WD	WD
Repair method	MR	RR	MR	RR	MR	RR
Rel. usage combinations (%)	12.3	19.4	12.8	14.8	16.8	23.9
Rel. usage destroy method (%)	31.7 (RD)		27.6 (PD)		40.7 (WD)	
Rel. usage repair method (%)	41.9 (MR)			58.1 (RR)		

a policy called largest correlation storage (LCS). It is similar to the greedy rule we used to find an initial solution, so we apply this framework to LCS. **Dedicated storage** (DS) stores each item in the fixed storage locations. As mentioned before, storing each type of item in dedicated pods will increase the number of pods visits in RMFS. We also test DS where one type of item is stored in only one pod. **Full-turnover storage** assigns items to storage locations based on their turnover rate. Items with high turnover rates are assigned close to the depot. **Class-based storage** (CBS) divides items into several classes, where each class is randomly stored in a dedicated storage area. To transfer the basic idea of full-turnover storage and CBS, we divide the items into two classes [33], namely, best-selling items ($D_i > 3$) and general-selling items ($D_i \leq 3$). The items in each class are assigned to different pods, and items of the same class are stored randomly.

We apply the medium- and large-scale instances, and present the results of the comparison in Table 9 and Table 10. The average performance gap between RISA and RND, LCS, DS, and CBS is 44.5%, 32.2%, 198.1%, and 42.0% in the medium-scale instances, respectively. Even the best among them (i.e., LCS) leads to 32.2% lower correlation degree than the results of the optimized storage assignment. In the large-scale instances, the average performance gap between RISA and RND, LCS, DS, and CBS is 45.6%, 38.6%, 161.6% and 44.3%, respectively. Among the four policies, DS has the worst results in the medium- and large-scale instances. Storing each type of item in dedicated pods increases the number of pod visits during order picking operations. Therefore, the four storage assignment policies are unsuitable for the RISA problem in RMFS warehouses.

D. SENSITIVITY ANALYSIS

This section analyzes the effects of various parameters, including the number of pods, number of storage locations in each pod, and average scatter level of items, on the performance of proposed ALNS and RISA.

1) EFFECT OF THE NUMBER OF PODS

Experiments are performed on instances with $N_I = 1000$, $N_C = 5$ and different N_M (number of pods) to demonstrate the effect of N_M on the performance of RISA. The results are given in Fig. 6. With the increase in N_M, the heuristics and storage assignment policies present approximately linear increasing trends. In the RISA problem, the variants of the VNS algorithm (VNS, VNS-SA, and AVNS) do not differ considerably in performance. The performance of the proposed ALNS and RISA is better than that of the other heuristics and storage assignment policies.

2) EFFECT OF THE NUMBER OF STORAGE LOCATIONS IN EACH POD

The change in the objective values regarding N_C (the number of storage locations in each pod) under a fixed type of items ($N_I = 1000$) and number of pods ($N_M = 500$) is depicted in Fig. 7. A large pod capacity means a pod can store many different items at the same time. The objective values of all the heuristics and storage assignment policies increase as the number of storage locations in each pod increases. The results also indicate that our RISA and the proposed ALNS have excellent performance.

3) EFFECT OF THE SCATTER LEVEL OF ITEMS

This section examines the effect of items' scatter level on the RISA with a fixed number of pods ($N_M = 500$) and number of storage locations in each pod ($N_C = 5$). We define the scatter level of items, s, as the average number of storage locations to be assigned for each item:

$$s = \frac{N_C \times N_M}{N_I}$$

The total type of items N_I must not exceed the total number of storage locations ($N_C \times N_M$) in the warehouse, that is, $N_I \leq N_C \times N_M$. $s = 1$ indicates that the total type of items is equal to the total number of storage locations, namely, each item corresponds to one storage location on the average. When the total number of storage locations is determined, the larger s is, the fewer the types of items are, and the more the storage locations that each item occupies (i.e., the more scattered the items stored in the pods are).

The type of items and expected results of RISA obtained by varying the items' scatter level are shown in Table 11 and Fig. 8. The scatter level of the items does not have a considerable effect on RISA, suggesting that high inventory dispersion is not always good. This result may be good for warehouse managers because scattered storage of items remarkably increases the replenishment workload. Although the scattered storage strategy can effectively reduce the distance to which robots move pods, with the increase in the scatter level, the type of items stored in the warehouse in the same storage area decreases correspondingly, or many pods are needed to store the same number of item types.

TABLE 9. Performance comparison between RISA and storage assignment policies in medium-scale instances.

N_I-N_M-N_C	RND	LCS	DS	CBS	RISA	Gap (%)			
						RND	LCS	DS	CBS
500-500-5	1096.8	1229.7	453.0	1120.7	**1661.5**	51.5	34.1	266.8	48.2
600-500-5	1087.7	1225.3	507.7	1130.3	**1647.6**	51.4	34.0	224.5	45.7
700-500-5	1097.1	1218.4	578.1	1125.7	**1645.2**	50.0	34.4	184.6	46.1
800-500-5	1082.7	1203.5	621.8	1124.9	**1636.6**	51.2	34.8	163.2	45.4
900-500-5	1085.0	1203.2	713.5	1128.1	**1647.9**	51.9	35.8	131.0	46.0
1000-500-5	1092.5	1210.6	779.3	1132.4	**1658.2**	51.8	38.0	112.8	46.4
1000-1000-7	2809.5	3050.1	1058.7	2821.7	**4005.2**	42.6	31.7	281.1	41.9
1200-1000-7	2774.7	3043.3	1195.6	2823.4	**3983.5**	43.6	30.5	233.2	41.1
1400-1000-7	2782.4	3016.1	1396.7	2802.1	**3978.8**	43.0	31.8	184.9	42.0
1600-1000-7	2803.0	3071.8	1527.6	2813.2	**4001.8**	42.8	30.7	162.0	42.2
1800-1000-7	2768.1	3004.0	1633.9	2826.7	**3993.4**	44.3	32.9	144.4	41.3
2000-1000-7	2812.9	3050.5	1751.7	2815.3	**4028.8**	43.2	32.4	130.0	43.1
2000-2000-9	3874.4	4160.4	1401.6	3917.5	**5390.3**	39.1	29.5	284.6	37.6
2200-2000-9	3875.2	4172.9	1524.9	3912.3	**5395.4**	39.2	29.3	253.8	37.9
2400-2000-9	3878.6	4173.3	1619.6	3913.7	**5383.1**	38.8	29.3	232.4	37.5
2600-2000-9	3893.4	4167.5	1725.7	3906.4	**5398.6**	38.7	29.8	212.8	38.2
2800-2000-9	3903.3	4167.9	1839.1	3933.9	**5410.0**	38.6	29.9	194.2	37.5
3000-2000-9	3873.5	4162.4	2001.4	3919.4	**5392.1**	39.2	29.9	169.4	37.6
Average	-	-	-	-	-	44.5	32.2	198.1	42.0

TABLE 10. Performance comparison between RISA and storage assignment policies in large-scale instances.

N_I-N_M-N_C	RND	LCS	DS	CBS	RISA	Gap (%)			
						RND	LCS	DS	CBS
5000-4000-5	1950.5	2103.9	915.7	1969.6	**3004.2**	54.0	42.8	228.1	52.5
6000-4000-5	1949.7	2061.5	1047.5	1973.1	**3001.5**	53.9	45.6	186.5	52.1
7000-4000-5	1942.9	2048.1	1192.3	1964.6	**2993.8**	54.1	46.1	151.1	52.3
8000-5000-5	1959.5	2043.2	1139.1	1987.1	**2984.3**	52.3	46.1	162.0	50.2
9000-5000-5	1965.5	2062.1	1262.0	1973.8	**2995.2**	52.4	45.2	137.3	51.7
10000-5000-5	1956.3	2029.8	1356.2	1980.4	**2998.9**	53.2	47.7	121.1	51.4
5000-4000-7	2947.9	3157.6	1308.2	2971.1	**4258.7**	44.5	34.9	225.5	43.3
6000-4000-7	2942.4	3126.3	1547.6	2953.6	**4249.1**	44.4	35.9	174.6	43.8
7000-4000-7	2949.9	3117.9	1694.7	2984.5	**4243.2**	43.8	36.1	150.4	42.2
8000-5000-7	2958.5	3073.6	1657.3	2986.6	**4277.0**	44.6	39.2	158.1	43.2
9000-5000-7	2963.6	3099.9	1806.4	2981.4	**4283.7**	44.5	38.2	137.1	43.6
10000-5000-7	2961.2	3090.7	1930.7	2987.1	**4266.4**	44.1	38.0	120.9	42.8
5000-4000-9	3945.2	4154.5	1723.1	3965.7	**5459.9**	38.4	31.4	216.8	37.7
6000-4000-9	3933.6	4142.8	2060.0	3981.5	**5478.3**	39.3	32.2	165.9	37.5
7000-4000-9	3933.0	4137.2	2219.4	3962.2	**5510.0**	40.1	33.2	148.3	39.1
8000-5000-9	3960.7	4116.3	2095.3	3986.6	**5503.2**	38.9	33.7	162.6	38.0
9000-5000-9	3944.8	4123.2	2290.5	3994.4	**5531.5**	40.2	34.2	141.4	38.4
10000-5000-9	3945.1	4075.3	2467.8	4001.8	**5473.9**	38.8	34.3	121.8	36.8
Average	-	-	-	-	-	45.6	38.6	161.6	44.3

VI. EXTENSION

In this section, we further consider the different properties of items and the heterogeneous storage locations in the pods. In the real-world scenario, the storage locations in the pods in RMFS may have different sizes. The properties of different items (such as quantity, weight, size, fragility, etc.) also vary [53]. The compatibility between replenishment items and storage locations needs to be considered when making decisions to replenish items. For example, heavy or large items have to be stored at the bottom of the pods, and light

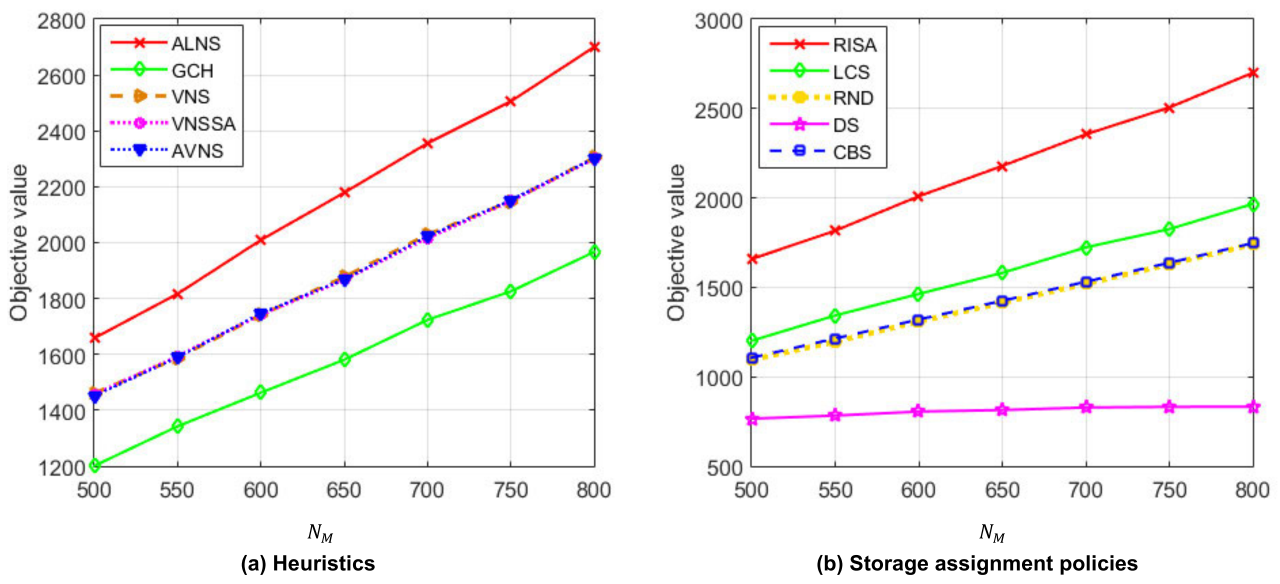


FIGURE 6. The effect of number of pods: (a) Heuristics; (b) Storage assignment policies.

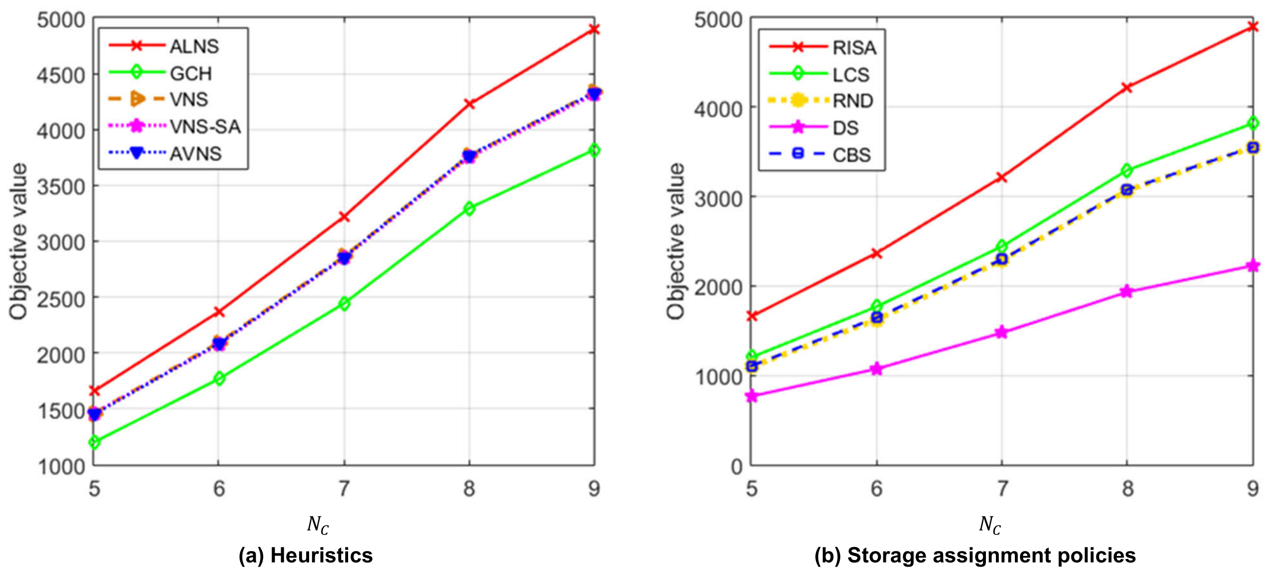


FIGURE 7. The effect of number of storage locations each pod: (a) Heuristics; (b) Storage assignment policies.

TABLE 11. The effect of the scattered level of items.

s	1	2	3	4	5	6	7	8	9	10
N_I	2500	1250	833	625	500	417	357	312	278	250
RISA	1682.8	1637.4	1655.6	1669.4	1668.1	1674.8	1650.6	1642.7	1641.9	1627.4

and small items can be placed in the middle or upper storage locations of the pods. Furthermore, the quantity pairing of the items should be considered in the calculation of the correlation degree between the items in the pod.

By using the notations in Table 1 and Table 12, we can rewrite the objective function as (21) and add constraints (22) to (26) as follows:

$$\text{Max} \sum_{m\in M_e}\left(\sum_{c\in C_{me}}\sum_{c'>c}\sum_{i\in I_r}\sum_{j\in I_r}\frac{r_{ij}v_{imc}v_{jmc'}}{\max\{y_{im},y_{jm}\}} + \sum_{c\in C_{me}}\sum_{c'\in C_{ma}}\sum_{i\in I_r}\sum_{j\in I}\frac{r_{ij}v_{imc}h_{jmc'}}{\max\{y_{im},b_{jm}\}}\right) \tag{21}$$

$$v_{imc} \le x_{im}, \quad \forall i \in I_r;\, m \in M_e;\, c \in C_{me} \tag{22}$$

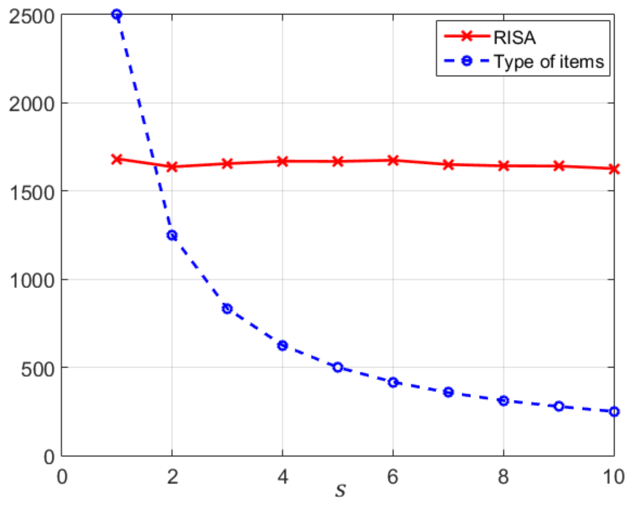


FIGURE 8. The effect of the scattered level of items.

TABLE 12. Additional notations.

Notation	Definition
Sets and parameters	
C_{me}	Set of empty storage locations in pod m ($C_{me} \subseteq C$)
C_{ma}	Set of storage locations already occupied by items in pod m ($C_{ma} \subseteq C$ and $C_{ma} \cup C_{me} = C$)
c, c'	Index of storage locations in a pod ($c, c' \in C$)
u_{imc}	1, If replenishment item i can be stored in the empty storage location c in pod m; 0, otherwise
h_{imc}	1, If item i has been stored in the storage location c in pod m before replenishment; 0, otherwise
Decision variables	
v_{imc}	Binary variables: 1, If replenishment item i is assigned to the empty storage location c in pod m; 0, otherwise

$$v_{imc} \leq u_{imc}, \quad \forall i \in I_r; m \in M_e; c \in C_{me} \tag{23}$$

$$\sum_{c \in C_{me}} v_{imc} = y_{im}, \quad \forall i \in I_r; m \in M_e \tag{24}$$

$$\sum_{i \in I_r} v_{imc} \leq 1, \quad \forall m \in M_e; c \in C_{me} \tag{25}$$

$$v_{imc} \in \{0, 1\}, \quad \forall i \in I_r; m \in M_e; c \in C_{me} \tag{26}$$

The objective function (21) further considers the quantity pairing of the items in each pod. Constraint (22) indicates that if replenishment item i is assigned to an empty storage location c in pod m, it must be assigned to pod m. Constraint (23) ensures that compatibility between replenishment item i and empty storage location c. The number of empty storage locations occupied by replenishment item i in pod m is determined by Constraint (24). Constraint (25) ensures that only one unit of replenishment items can be assigned to each empty storage location. Constraint (26) defines the binary variables.

When solving RISA problem in business practice (i.e., in consideration of the abovementioned factors), we need to add a judgment condition for allocating replenishment items. That is, we screen out the empty storage locations that are compatible with the replenishment items and choose from them. Notably, considering these conditions does not change the essence of the RISA problem and does not affect the performance of the proposed algorithm. Thus, our proposed algorithm is a good choice for use in business practice.

RMFS is a kind of parts-to-picker system. Due to different warehousing environment and system selection, different parts-to-picker systems implement various technical components, have different layout design, and organize the picking process slightly different. However, they all have in common that the storage bins or moveable racks are transported to the picking workstations by automated devices such as autonomous mobile robots, shuttles, and conveyors. Then, the human picker in the workstation picks items from bins or racks into the corresponding customer bins [39]. Thus, the findings of this study can also be applied to some other parts-to-picker warehouses, such as some automated storage and retrieval systems (ASRS).

VII. CONCLUSION

This study investigates the RISA problem in RMFS warehouses. In particular, some empty storage locations are available in the pods after an order picking period, and the other storage locations are already occupied. The items that do not meet the required inventory levels need to be replenished. When we decide to replenish these items, we need to consider not only the correlation between replenishment items, but also the correlation between replenishment items and the items already stored in the same pod. Therefore, we focus on the replenishment situation with simultaneous consideration of item turnover level, scattered storage, and correlation relationship in RMFS. We describe the problem in detail and formulate it as a MIP model.

In consideration of the characteristics of the RISA problem, we propose a suited ALNS heuristic to solve this problem. The heuristic can handle the problem at any replenishment rate level and can be applied to the special case where the initial state of the warehouse is empty. The computational study shows that the ALNS heuristic can effectively solve this problem and provide high-quality solutions compared with the methods in literature. Furthermore, we compare the proposed optimization-based storage assignment with commonly used storage assignment policies for traditional warehouses. Numerical results show that our optimization-based storage assignment has remarkable advantages over other policies.

The following part presents the limitations and scopes to consider in further research:

1) Reliable historical order data can be used to obtain the robust relationship between different items. This study assumes that the correlation degree of any pair

TABLE 13. **Performance comparison between ALNS and other algorithms in medium-scale instances (empty warehouse).**

N_I-N_M-N_C	GCH	VNS	VNS-SA	AVNS	ALNS	Gap (%)			
						GCH	VNS	VNS-SA	AVNS
200-100-5	489.5	691.2	690.8	689.3	**758.6**	55.0	9.8	9.8	10.0
200-110-5	538.3	755.5	759.1	752.8	**834.1**	55.0	10.4	9.9	10.8
200-120-5	551.2	824.6	819.6	829.1	**905.7**	64.3	9.8	10.5	9.2
200-130-5	606.7	889.3	880.5	881.7	**979.1**	61.4	10.1	11.2	11.0
200-140-5	673.1	953.3	953.9	951.7	**1047.2**	55.6	9.8	9.8	10.0
200-150-5	696.4	1022.8	1016.2	1023.6	**1136.5**	63.2	11.1	11.8	11.0
300-100-7	960.8	1363.9	1352.4	1360.0	**1492.3**	55.3	9.4	10.3	9.7
300-110-7	1087.6	1490.2	1483.0	1495.1	**1643.4**	51.1	10.3	10.8	9.9
300-120-7	1184.3	1624.4	1615.1	1622.7	**1784.3**	50.7	9.8	10.5	10.0
300-130-7	1258.9	1749.8	1742.6	1747.5	**1946.2**	54.6	11.2	11.7	11.4
300-140-7	1316.2	1870.6	1875.3	1871.2	**2083.3**	58.3	11.4	11.1	11.3
300-150-7	1465.6	2009.5	2003.4	2005.3	**2235.1**	52.5	11.2	11.6	11.5
500-100-9	1638.5	2255.1	2241.6	2252.8	**2452.1**	49.7	8.7	9.4	8.8
500-110-9	1840.2	2465.3	2472.7	2474.1	**2723.8**	48.0	10.5	10.2	10.1
500-120-9	1923.3	2678.4	2659.6	2665.9	**2949.7**	53.4	10.1	10.9	10.6
500-130-9	2097.1	2887.9	2862.3	2875.7	**3210.4**	53.1	11.2	12.2	11.6
500-140-9	2295.1	3105.3	3091.8	3102.9	**3443.3**	50.0	10.9	11.4	11.0
500-150-9	2435.6	3326.1	3306.4	3316.9	**3698.2**	51.8	11.2	11.8	11.5
Average	-	-	-	-	-	54.6	10.4	10.8	10.5

TABLE 14. **Performance comparison between ALNS and other algorithms in large-scale instances (empty warehouse).**

N_I-N_M-N_C	GCH	VNS	VNS-SA	AVNS	ALNS	Gap (%)			
						GCH	VNS	VNS-SA	AVNS
800-200-5	1136.2	1359.8	1360.0	1361.8	**1552.7**	36.7	14.2	14.2	14.0
800-210-5	1106.8	1430.3	1435.1	1424.3	**1610.8**	45.4	12.6	12.2	13.1
800-220-5	1237.0	1498.2	1492.6	1494.8	**1697.4**	37.2	13.3	13.7	13.6
800-230-5	1211.8	1554.4	1555.5	1551.7	**1771.2**	46.1	13.9	13.9	14.1
800-240-5	1316.5	1620.4	1628.7	1625.6	**1834.5**	39.3	13.2	12.6	12.9
800-250-5	1364.4	1693.2	1685.2	1689.1	**1932.1**	41.6	14.1	14.7	14.4
1000-250-7	2580.9	3301.9	3305.3	3312.8	**3774.3**	46.2	14.3	14.2	13.9
1000-260-7	2657.9	3425.8	3426.2	3442.2	**3919.4**	47.5	14.4	14.4	13.9
1000-270-7	2721.0	3557.7	3563.2	3576.5	**4073.2**	49.7	14.5	14.3	13.9
1000-280-7	2804.7	3671.1	3669.9	3682.8	**4209.8**	50.1	14.7	14.7	14.3
1000-290-7	2880.9	3795.0	3811.2	3816.4	**4366.4**	51.6	15.1	14.6	14.4
1000-300-7	3025.5	3925.7	3929.3	3937.0	**4527.6**	49.6	15.3	15.2	15.0
1200-300-9	4845.2	6486.0	6491.1	6493.9	**7365.9**	52.0	13.6	13.5	13.4
1200-310-9	5058.7	6700.7	6695.9	6690.7	**7710.4**	52.4	15.1	15.2	15.2
1200-320-9	5093.4	6857.2	6868.2	6877.8	**7957.1**	56.2	16.0	15.9	15.7
1200-330-9	5354.6	7087.1	7109.8	7114.9	**8140.5**	52.0	14.9	14.5	14.4
1200-340-9	5461.8	7298.3	7302.9	7308.1	**8339.1**	52.7	14.3	14.2	14.1
1200-350-9	5614.7	7507.7	7510.2	7507.3	**8697.5**	54.9	15.8	15.8	15.9
Average	-	-	-	-	-	47.8	14.5	14.3	14.2

of items can be derived from historical order data. However, in some e-commerce warehouses, customer demand fluctuates considerably in different sales periods. Future orders may differ from historical orders, so the correlation between items may also change. Therefore, the item storage assignment problem that considers dynamic demand will be the direction of our future research.

2) Some parameters of the warehouse setup in the computational experiments are referenced to actual warehouse

and relevant literature. However, this study does not test out the proposed method in a real-life scenario or on actual enterprise data. Future research could examine how the proposed method performs in actual warehouse environments.

3) Future research could also consider the cost of replenishment operations, including the cost of item scattered storage and correlated storage. Doing so will address important research questions on how often and when replenishment operations should be performed. In addition, studying the dynamic item storage assignment problem in some warehouses where replenishment and picking occur simultaneously remains difficult.
4) Although the trend toward automated processes is continuing and robots are becoming increasingly intelligent, the human advantage cannot be completely replaced. In the foreseeable future, many technical problems in robot picking are still expected to occur; not all of them can be resolved, and human pickers will remain an integral part of order picking [39], [54]. Furthermore, human pickers are the bottleneck resource in parts-to-picker systems. The working states of human pickers considerably influence picking performance [55], [56], [57]. Therefore, considering human factors in the item storage assignment problem is an interesting aspect to explore.

APPENDIX

In this appendix, we test the performance of ALNS heuristic for solving the special case where the warehouse is empty ($\eta = 1$). Mathematically, a larger replenishment rate η means a larger number of empty storage locations, and thus making the problem more combinatorial and difficult to solve. We adjust the scale of the test instances for the empty warehouse to obtain a fair trade-off between solution quality and search time.

The numerical results of medium- and large-scale instances of the empty warehouse are shown in Table 13 and 14, respectively. ALNS performs better than all four comparison algorithms on all medium- and large-scale instances. Specifically, ALNS outperforms GCH with an average gap of 54.6% in the medium-scale instances and 47.8% in the large-scale instances. The average gap between VNS, VNS-SA, AVNS and ALNS is more than 10% and 14% in the medium-scale instances and large-scale instances, respectively.

REFERENCES

[1] N. Boysen, R. de Koster, and F. Weidinger, "Warehousing in the e-commerce era: A survey," *Eur. J. Oper. Res.*, vol. 277, no. 2, pp. 396–411, Sep. 2019, doi: 10.1016/j.ejor.2018.08.023.

[2] R. de Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *Eur. J. Oper. Res.*, vol. 182, no. 2, pp. 481–501, Oct. 2007, doi: 10.1016/j.ejor.2006.07.009.

[3] A. Bolu and Ö. Korçak, "Adaptive task planning for multi-robot smart warehouse," *IEEE Access*, vol. 9, pp. 27346–27358, 2021, doi: 10.1109/ACCESS.2021.3058190.

[4] W. Wang, Y. Wu, J. Zheng, and C. Chi, "A comprehensive framework for the design of modular robotic mobile fulfillment systems," *IEEE Access*, vol. 8, pp. 13259–13269, 2020, doi: 10.1109/ACCESS.2020.2966403.

[5] N. Boysen, D. Briskorn, and S. Emde, "Parts-to-picker based order processing in a rack-moving mobile robots environment," *Eur. J. Oper. Res.*, vol. 262, no. 2, pp. 550–562, Oct. 2017, doi: 10.1016/j.ejor.2017.03.053.

[6] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–20, 2008.

[7] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *IEEE Spectr.*, vol. 45, no. 7, pp. 26–34, Jul. 2008.

[8] J. J. R. Reyes, E. L. Solano-Charris, and J. R. Montoya-Torres, "The storage location assignment problem: A literature review," *Int. J. Ind. Eng. Computations*, vol. 10, no. 2, pp. 199–224, 2019, doi: 10.5267/j.ijiec.2018.8.001.

[9] J. Gu, M. Goetschalckx, and L. F. McGinnis, "Research on warehouse design and performance evaluation: A comprehensive review," *Eur. J. Oper. Res.*, vol. 203, no. 3, pp. 539–549, Jun. 2010, doi: 10.1016/j.ejor.2009.07.031.

[10] J. Gu, M. Goetschalckx, and L. F. McGinnis, "Research on warehouse operation: A comprehensive review," *Eur. J. Oper. Res.*, vol. 177, no. 1, pp. 1–21, Feb. 2007, doi: 10.1016/j.ejor.2006.02.025.

[11] I. G. Lee, S. H. Chung, and S. W. Yoon, "Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations," *Comput. Ind. Eng.*, vol. 139, Jan. 2020, Art. no. 106129, doi: 10.1016/j.cie.2019.106129.

[12] M. S. Islam and M. K. Uddin, "Correlated storage assignment approach in warehouses: A systematic literature review," *J. Ind. Eng. Manage.*, vol. 16, no. 2, p. 294, Jun. 2023, doi: 10.3926/jiem.4850.

[13] F. Weidinger, N. Boysen, and D. Briskorn, "Storage assignment with rack-moving mobile robots in KIVA warehouses," *Transp. Sci.*, vol. 52, no. 6, pp. 1479–1495, Dec. 2018, doi: 10.1287/trsc.2018.0826.

[14] M. Wang, R.-Q. Zhang, and K. Fan, "Improving order-picking operation through efficient storage location assignment: A new approach," *Comput. Ind. Eng.*, vol. 139, Jan. 2020, Art. no. 106186, doi: 10.1016/j.cie.2019.106186.

[15] R.-Q. Zhang, M. Wang, and X. Pan, "New model of the storage location assignment problem considering demand correlation pattern," *Comput. Ind. Eng.*, vol. 129, pp. 210–219, Mar. 2019, doi: 10.1016/j.cie.2019.01.027.

[16] K.-W. Pang and H.-L. Chan, "Data mining-based algorithm for storage location assignment in a randomised warehouse," *Int. J. Prod. Res.*, vol. 55, no. 14, pp. 4035–4052, Jul. 2017, doi: 10.1080/00207543.2016.1244615.

[17] J. Li, M. Moghaddam, and S. Y. Nof, "Dynamic storage assignment with product affinity and ABC classification—A case study," *Int. J. Adv. Manuf. Technol.*, vol. 84, nos. 9–12, pp. 2179–2194, Jun. 2016, doi: 10.1007/s00170-015-7806-7.

[18] Y.-F. Chuang, H.-T. Lee, and Y.-C. Lai, "Item-associated cluster assignment model on storage allocation problems," *Comput. Ind. Eng.*, vol. 63, no. 4, pp. 1171–1177, Dec. 2012, doi: 10.1016/j.cie.2012.06.021.

[19] M. Merschformann, T. Lamballais, M. B. M. de Koster, and L. Suhl, "Decision rules for robotic mobile fulfillment systems," *Oper. Res. Perspect.*, vol. 6, Dec. 2019, Art. no. 100128, doi: 10.1016/j.orp.2019.100128.

[20] K. Azadeh, R. De Koster, and D. Roy, "Robotized and automated warehouse systems: Review and recent developments," *Transp. Sci.*, vol. 53, no. 4, pp. 917–945, Jul. 2019, doi: 10.1287/trsc.2018.0873.

[21] Y. Zhuang, Y. Zhou, E. Hassini, Y. Yuan, and X. Hu, "Rack retrieval and repositioning optimization problem in robotic mobile fulfillment systems," *Transp. Res. E, Logistics Transp. Rev.*, vol. 167, Nov. 2022, Art. no. 102920, doi: 10.1016/j.tre.2022.102920.

[22] R. Yuan, S. C. Graves, and T. Cezik, "Velocity-based storage assignment in semi-automated storage systems," *Prod. Oper. Manage.*, vol. 28, no. 2, pp. 354–373, Feb. 2019, doi: 10.1111/poms.12925.

[23] X. Li, G. Hua, A. Huang, J.-B. Sheu, T. C. E. Cheng, and F. Huang, "Storage assignment policy with awareness of energy consumption in the Kiva mobile fulfilment system," *Transp. Res. E, Logistics Transp. Rev.*, vol. 144, Dec. 2020, Art. no. 102158, doi: 10.1016/j.tre.2020.102158.

[24] H.-J. Kim, C. Pais, and Z. M. Shen, "Item assignment problem in a robotic mobile fulfillment system," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 1854–1867, Oct. 2020, doi: 10.1109/TASE.2020.2979897.

[25] M. Mirzaei, N. Zaerpour, and R. de Koster, "The impact of integrated cluster-based storage allocation on parts-to-picker warehouse performance," *Transp. Res. E, Logistics Transp. Rev.*, vol. 146, Feb. 2021, Art. no. 102207, doi: 10.1016/j.tre.2020.102207.

[26] N. Yang, "Evaluation of the joint impact of the storage assignment and order batching in mobile-pod warehouse systems," *Math. Problems Eng.*, vol. 2022, pp. 1–13, Apr. 2022, doi: 10.1155/2022/9148001.

[27] T. L. Tessensohn, D. Roy, and R. B. M. De Koster, ''Inventory allocation in robotic mobile fulfillment systems,'' *IISE Trans.*, vol. 52, no. 1, pp. 1–17, Jan. 2020, doi: 10.1080/24725854.2018.1560517.

[28] F. Weidinger and N. Boysen, ''Scattered storage: How to distribute stock keeping units all around a mixed-shelves warehouse,'' *Transp. Sci.*, vol. 52, no. 6, pp. 1412–1427, Dec. 2018, doi: 10.1287/trsc.2017.0779.

[29] R. Yuan, T. Cezik, and S. C. Graves, ''Stowage decisions in multi-zone storage systems,'' *Int. J. Prod. Res.*, vol. 56, nos. 1–2, pp. 333–343, Jan. 2018, doi: 10.1080/00207543.2017.1398428.

[30] T. Cezik, S. C. Graves, and A. C. Liu, ''Velocity-based stowage policy for a semiautomated fulfillment system,'' *Prod. Oper. Manage.*, 2022, doi: 10.1111/poms.13745.

[31] X. Xiang, C. Liu, and L. Miao, ''Storage assignment and order batching problem in kiva mobile fulfilment system,'' *Eng. Optim.*, vol. 50, no. 11, pp. 1941–1962, Nov. 2018, doi: 10.1080/0305215x.2017.1419346.

[32] M. Mirzaei, N. Zaerpour, and R. B. M. de Koster, ''How to benefit from order data: Correlated dispersed storage assignment in robotic warehouses,'' *Int. J. Prod. Res.*, vol. 60, no. 2, pp. 549–568, Jan. 2022, doi: 10.1080/00207543.2021.1971787.

[33] Z. Ma, G. Wu, B. Ji, L. Wang, Q. Luo, and X. Chen, ''A novel scattered storage policy considering commodity classification and correlation in robotic mobile fulfillment systems,'' *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 2, pp. 1020–1033, Apr. 2023, doi: 10.1109/TASE.2022.3178934.

[34] J. Zhang, N. Zhang, L. Tian, Z. Zhou, and P. Wang, ''Robots' picking efficiency and pickers' energy expenditure: The item storage assignment policy in robotic mobile fulfillment system,'' *Comput. Ind. Eng.*, vol. 176, Feb. 2023, Art. no. 108918, doi: 10.1016/j.cie.2022.108918.

[35] K. L. Keung, C. K. M. Lee, and P. Ji, ''Data-driven order correlation pattern and storage location assignment in robotic mobile fulfillment and process automation system,'' *Adv. Eng. Informat.*, vol. 50, Oct. 2021, Art. no. 101369, doi: 10.1016/j.aei.2021.101369.

[36] K. L. Keung, C. K. M. Lee, and P. Ji, ''Industrial Internet of Things-driven storage location assignment and order picking in a resource synchronization and sharing-based robotic mobile fulfillment system,'' *Adv. Eng. Informat.*, vol. 52, Apr. 2022, Art. no. 101540, doi: 10.1016/j.aei.2022.101540.

[37] M. Jiang, K. H. Leung, Z. Lyu, and G. Q. Huang, ''Picking-replenishment synchronization for robotic forward-reserve warehouses,'' *Transp. Res. E, Logistics Transp. Rev.*, vol. 144, Dec. 2020, Art. no. 102138, doi: 10.1016/j.tre.2020.102138.

[38] X. Guo, R. Chen, S. Du, and Y. Yu, ''Storage assignment for newly arrived items in forward picking areas with limited open locations,'' *Transp. Res. E, Logistics Transp. Rev.*, vol. 151, Jul. 2021, Art. no. 102359, doi: 10.1016/j.tre.2021.102359.

[39] N. Boysen, S. Schwerdfeger, and K. Stephan, ''A review of synchronization problems in parts-to-picker warehouses,'' *Eur. J. Oper. Res.*, vol. 307, no. 3, pp. 1374–1390, Jun. 2023, doi: 10.1016/j.ejor.2022.09.035.

[40] D. Pisinger and S. Ropke, ''A general heuristic for vehicle routing problems,'' *Comput. Oper. Res.*, vol. 34, no. 8, pp. 2403–2435, Aug. 2007, doi: 10.1016/j.cor.2005.09.012.

[41] S. Ropke and D. Pisinger, ''An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows,'' *Transp. Sci.*, vol. 40, no. 4, pp. 455–472, Nov. 2006.

[42] S. T. Windras Mara, R. Norcahyo, P. Jodiawan, L. Lusiantoro, and A. P. Rifai, ''A survey of adaptive large neighborhood search algorithms and applications,'' *Comput. Oper. Res.*, vol. 146, Oct. 2022, Art. no. 105903, doi: 10.1016/j.cor.2022.105903.

[43] Z. Luo, H. Qin, D. Zhang, and A. Lim, ''Adaptive large neighborhood search heuristics for the vehicle routing problem with stochastic demands and weight-related cost,'' *Transp. Res. E, Logistics Transp. Rev.*, vol. 85, pp. 69–89, Jan. 2016, doi: 10.1016/j.tre.2015.11.004.

[44] V. C. Hemmelmayr, J.-F. Cordeau, and T. G. Crainic, ''An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics,'' *Comput. Oper. Res.*, vol. 39, no. 12, pp. 3215–3228, Dec. 2012, doi: 10.1016/j.cor.2012.04.007.

[45] P. Bodnar, R. de Koster, and K. Azadeh, ''Scheduling trucks in a cross-dock with mixed service mode dock doors,'' *Transp. Sci.*, vol. 51, no. 1, pp. 112–131, Feb. 2017, doi: 10.1287/trsc.2015.0612.

[46] E. A. Frazele and G. P. Sharp, ''Correlated assignment strategy can improve any order-picking operation,'' *Ind. Engineer*, vol. 21, pp. 33–37, Aug. 1989.

[47] A. Gharehgozli and N. Zaerpour, ''Robot scheduling for pod retrieval in a robotic mobile fulfillment system,'' *Transp. Res. E, Logistics Transp. Rev.*, vol. 142, Oct. 2020, Art. no. 102087, doi: 10.1016/j.tre.2020.102087.

[48] H. Qin, J. Xiao, D. Ge, L. Xin, J. Gao, S. He, H. Hu, and J. G. Carlsson, ''JD.Com: Operations research algorithms drive intelligent warehouse robots to work,'' *INFORMS J. Appl. Analytics*, vol. 52, no. 1, pp. 42–55, Jan. 2022, doi: 10.1287/inte.2021.1100.

[49] S. C. Graves, W. H. Hausman, and L. B. Schwarz, ''Storage-retrieval interleaving in automatic warehousing systems,'' *Manage. Sci.*, vol. 23, no. 9, pp. 935–945, May 1977.

[50] W. H. Hausman, L. B. Schwarz, and S. C. Graves, ''Optimal storage assignment in automatic warehousing systems,'' *Manage. Sci.*, vol. 22, no. 6, pp. 629–638, Feb. 1976.

[51] J.-Y. Potvin and J.-M. Rousseau, ''A parallel route building algorithm for the vehicle routing and scheduling problem with time windows,'' *Eur. J. Oper. Res.*, vol. 66, no. 3, pp. 331–340, May 1993, doi: 10.1016/0377-2217(93)90221-8.

[52] H. de Vries, R. Carrasco-Gallego, T. Farenhorst-Yuan, and R. Dekker, ''Prioritizing replenishments of the piece picking area,'' *Eur. J. Oper. Res.*, vol. 236, no. 1, pp. 126–134, Jul. 2014, doi: 10.1016/j.ejor.2013.12.045.

[53] T. van Gils, K. Ramaekers, A. Caris, and R. B. M. de Koster, ''Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review,'' *Eur. J. Oper. Res.*, vol. 267, no. 1, pp. 1–15, May 2018, doi: 10.1016/j.ejor.2017.09.002.

[54] R. de Koster, ''Warehousing 2030,'' in *Global Logistics and Supply Chain Strategies for the 2020s*, R. Merkert and K. Hoberg, Eds. Cham, Switzerland: Springer, 2030, pp. 243–260.

[55] E. H. Grosse, C. H. Glock, and W. P. Neumann, ''Human factors in order picking: A content analysis of the literature,'' *Int. J. Prod. Res.*, vol. 55, no. 5, pp. 1260–1276, Mar. 2017, doi: 10.1080/00207543.2016.1186296.

[56] E. H. Grosse, C. H. Glock, M. Y. Jaber, and W. P. Neumann, ''Incorporating human factors in order picking planning models: Framework and research opportunities,'' *Int. J. Prod. Res.*, vol. 53, no. 3, pp. 695–717, Feb. 2015, doi: 10.1080/00207543.2014.919424.

[57] Z. Wang, J. Sheu, C. Teo, and G. Xue, ''Robot scheduling for mobile-rack warehouses: Human–robot coordinated order picking systems,'' *Prod. Oper. Manage.*, vol. 31, no. 1, pp. 98–116, Jan. 2022, doi: 10.1111/poms.13406.

RUBO LI was born in Huangshi, Hubei, China, in 1994. He received the M.S. degree in management science and engineering from Wuhan University of Science and Technology, Wuhan, China, in 2021, where he is currently pursuing the Ph.D. degree with the School of Management. His research interests include warehouse management, optimization, and heuristics.

XUDONG DENG was born in Yunmeng, Hubei, China, in 1964. He received the B.S. degree in applied mathematics from Huazhong University of Science and Technology, Wuhan, China, in 1985, and the M.S. degree in management science and engineering from Wuhan University of Science and Technology, Wuhan, China, in 1991. He is currently a Professor with the School of Management, Wuhan University of Science and Technology. His research interests include supply chain management, logistics, and optimization.

YUNFENG MA was born in Jilin, China, in 1972. He received the Ph.D. degree in management science and engineering from Huazhong University of Science and Technology, Wuhan, China, in 2005. He is currently a Professor with the School of Management, Wuhan University of Science and Technology. His research interests include optimization, operations management, and warehouse management.