Reinforcement learning for pod retrieval scheduling and robot dispatching in the multi-deep compact robotic mobile fulfillment system

Jie Shao[a,b,1], Mingzhe Li[a,b,1], Mingyao Qi[a,b], Peng Yang[a,b]*

[a]Division of Logistics and Transportation, Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China.

[b]Institution of data and information, Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China.

*Corresponding author

ABSTRACT

To facilitate pod storage and retrieval, conventional robotic mobile fulfilment systems (RMFSs) have always used a single-deep pod configuration. Such use results into aisles that require much space and limits the use of storage space. As the warehousing footprint narrows especially in cities, high-density robotic warehousing systems have attracted much attention both from academia and industry. Inheriting the merits of single-deep RMFSs, multi-deep RMFSs could improve storage space utilization by using fewer aisles. However, the retrieval scheduling and robot dispatching in multi-deep RMFSs are challenging and have not yet been investigated. This study formulates this problem as an integer-programming model and develops the reinforcement learning and simulation-based multi-strategy heuristic approaches to jointly schedule the pod retrieval and dispatch the robot. Based on the extensive numerical experiments, the proposed reinforcement learning could yield as good solution as commercial solver for small-scale instances. As the instance size increases, the reinforcement learning approach performs better than the simulation-based multi-strategy heuristic and classical rule-based algorithm commonly used in practice in terms of solution quality and efficiency. Our study offers valuable insights for improving the operational efficiency of multi-deep RMFSs in the context of high space utilization.

Keywords: Robotic warehouses; multi-deep; RMFS; reinforcement learning; integer programming

1. Introduction

Robotic mobile fulfilment systems (RMFSs) are easily configured and efficient parts-to-pick systems

[1] These authors contributed equally to this work.

that have been widely applied in warehouses. In RMFSs, robots carry moveable pods (pods) to picking stations where workers pick the required items. RMFSs reduce unproductive picker movements (Wurman et al., 2008). To facilitate pod storage and retrieval, single-deep RMFS layouts are common. The pods are located near the aisles, and each pod can be accessed directly. Despite a high picking efficiency, the space utilization of single-deep RMFSs is not sufficiently efficient, making them less competitive in metropolitan areas where land is limited and expensive. Therefore, increasing space utilization is of significance to improve RMFS competitiveness.

Adopting a multi-deep warehouse layout is an economic approach to enhance space utilization by reducing aisle occupancy. Some well-known RMFS providers, such as Quicktron and Geekplus Technology Co., Ltd. have piloted the multi-deep layout in practice, as shown in Figure 1. Recently, scholars have also paid attention to studies on multi-deep RMFSs. Yang et al. (2022) evaluated the performance of multi-deep compact RMFSs using a semi-open queueing network model and provided insights into system design. Li et al. (2021) developed a simulation-based approach to investigate the task fulfillment decision in multi-deep compact RMFSs.

In multi-deep RMFSs, retrieving the targeted pods mostly requires a series of associated movements of adjacent pods. Pod reshuffling is inevitable. In a sense, multi-deep RMFSs increase the space utilization at the cost of losing some efficiency. To mitigate the efficiency effect, high-quality pod retrieval scheduling and robot dispatching are crucial for multi-deep RMFSs. However, the literature has rarely investigated pod retrieval optimization in multi-deep RMFSs. Therefore, we formulate the pod retrieval scheduling and robot dispatching problem as an integer-programming model. This task is challenging because it requires numerous obstructing pods in multi-deep RMFS to be arranged. To solve the problem, we develop reinforcement learning and simulation-based multi-strategy heuristic approaches to jointly schedule the pod retrieval and dispatch the robot. Reinforcement learning has been proven effective in solving many complex problems in recent years whereas its application remains rare in pod retrieval scheduling and robot dispatching. We are also interested in and expect to bridge this gap by exploring reinforcement learning's potential to optimize pod retrieval operations in multi-deep compact RMFSs.

The main contributions of this paper are as follows: 1) We first investigate the joint pod retrieval scheduling and robot dispatching problems in multi-deep compact RMFSs to improve operational efficiency in the context of high-density compact storage. 2) We formulate this problem and develop a

simulation-based multi-strategy heuristic to solve it, which is also used as baseline to investigate the performance of reinforcement learning algorithm. 3) We apply artificial intelligence technology in this field and develop a reinforcement learning algorithm to optimize pod retrieval in multi-deep compact RMFSs. The resulting solution outperforms the simulation-based multi-strategy heuristic both in quality and efficiency.

The remainder of this paper is organized as follows. Section 2 reviews and summarizes the related research. Section 3 describes and formulates the problem. In Section 4, we develop the simulation-based multi-strategy heuristic. In Section 5, we design a reinforcement learning algorithm to further optimize the pod retrieval scheduling and robot dispatching. In Section 6, we conduct numerical experiments and discuss the experimental results. Section 7 offers concluding remarks.

Figure 1. Multi-deep compact RMFS in practice (Source: Geekplus Technology Co., Ltd)

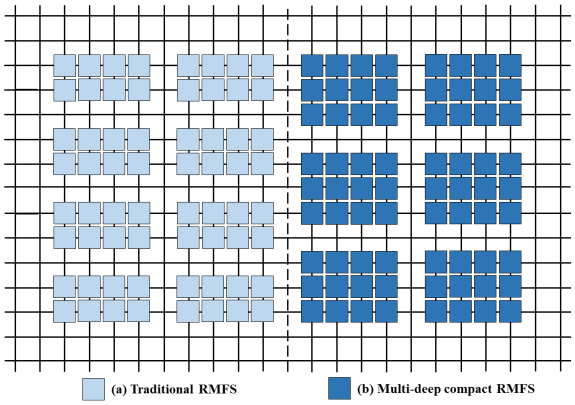


Figure 2. Pod arrangement in RMFSs

2. Literature Review

2.1 Robotic mobile fulfilment systems

The RMFS research has mainly focused on three major categories: system analysis, design optimization, and operational planning and control. For a comprehensive overview of RMFSs and other intelligent storage systems, readers are referred to Azadeh et al. (2019), Boysen et al. (2019), Boysen et al. (2021), da Costa Barros and Nascimento (2021), Zhen and Li (2022). System analyses and design optimizations have utilized analytical methods to generate closed-form expressions, such as the expected distance traveled by a machine, for performance metrics to evaluate layout designs or operational strategies. Previous studies by Lamballais et al. (2017), Yuan and Gong (2017), Zou et al. (2018), Roy et al. (2019), Gong et al. (2021) have evaluated RMFS performance while considering various factors such as parti-

tioned and unpartitioned cases, dedicated and shared robots, battery management, picking and replenishment processes, and customer classes. In terms of operational planning and control, studies have mainly focused on robot dispatching, route planning, task assignment, and scheduling. As this study falls under the category of operational planning and control, we do not discuss the former two categories in detail but focus instead on the latter.

The study of operational planning and control in RMFS encompasses order scheduling decisions such as order assignment (assigning orders to picking stations), order sequencing, and pod selection. After scheduling orders as tasks, the next step is to sequence and schedule the pods corresponding to the tasks and finally to plan paths for the robots and to consider traffic control to avoid or resolve conflicts and deadlocks. Pod retrieval scheduling (including pod sequencing and pod scheduling) is an important decision problem that aims to achieve the appropriate deployment of task resources under specific constraints and achieve optimal system performance (Cao et al., 2018). Several studies, such as Weidinger et al. (2018) and Ji et al. (2020), have examined the storage space allocation problem from multiple perspectives, but in these works, the pod retrieval order is fixed, or a simple strategy is used to sort them. In contrast, Boysen et al. (2017) and Zhuang et al. (2022) have proposed various strategies for pod sequencing and scheduling. Since storage space allocation, pod sequencing, and pod scheduling problems are related to order picking efficiency, some scholars, such as Gharehgozli and Zaerpour (2020), have studied both problems. Another important research direction is robot path planning and traffic control. Merschformann et al. (2019) proposed five path planning algorithms for real-time robot dispatching to avoid RMFS conflicts and deadlocks. Müller et al. (2020) proposed scheduling models with various automated guided vehicle (AGV) depots, implementing the typical strategies for addressing deadlocks such as prevention, avoidance, detection, and resolution.

Most of the existing RMFS studies have focused on traditional systems, and relatively little research has been done on multi-deep compact RMFSs. Li et al. (2021) utilized simulation to examine robot dispatching for multi-deep compact RMFSs, while Yang et al. (2022) evaluated system performance using a network-queuing approach. However, to the best of our knowledge, no study has investigated the retrieval for multi-deep compact RMFSs using an optimization approach.

2.2 Puzzle-based storage systems

Since multi-deep compact RMFSs include puzzle-based storage (PBS) systems, these must be reviewed

and analyzed. PBS system is an automated compact storage system with high-density shelving arranged on a square grid, featuring one or more empty locations referred to as "buffers" that can be used for temporary parking. The studies on PBS and retrieval systems have centered on three primary categories: system analysis, design optimization, and operational planning and control. This paper focuses solely on PBS operational planning and control.

The expected retrieval time of items in a single fixed-position escort was first evaluated by Gue and Kim (2007), Rohit et al. (2010), Yalcin et al. (2019), MA et al. (2022) have extended this work to multiple, random-position escorts. However, all of these articles have focused solely on single-load movements, an approach that oversimplifies the problem and leads to inefficient retrieval times. Therefore, it is necessary to consider simultaneous movements and congestion. To address this problem, Yu et al. (2022) proposed constructive optimization algorithms that can solve the single-item retrieval problem for simultaneous and block movements. Alfieri et al. (2012) considered a limited number of AGVs and developed a heuristic to optimize their scheduling and pod movements. Bukchin and Raviv (2022) developed an exact dynamic planning algorithm that can solve the problem of retrieving individual items allowing simultaneous or block movements. However, those studies have focused only on the retrieval of individual items, which may not fully reflect the complexities of real-world warehouse operations. Therefore, several researchers—including Gue et al. (2013), Mirzaei et al. (2017), Zou and Qi (2021), He et al. (2023)—have proposed methods to address the multi-item retrieval problem. Notably, He et al. (2023) developed a compact, mixed integer-programming model and a deep, reinforcement learning approach that effectively solves medium-to-large scale instances of the multi-escort, multi-pick station, and multi-item retrieval problem for PBS. These studies have been particularly influential in inspiring our research, as we recognize that retrieval in multi-deep compact RMFSs is essentially a multi-robot, multi-picking station, and multi-task retrieval problem for multiple, integrated PBS.

2.3 Reinforcement learning and its applications

Our research is closely linked to reinforcement learning, which allows automated agents to interact with the environment, such as playing Atari games (Mnih et al., 2015) or mastering the game of Go (Silver et al., 2017). In optimization problems, methods can be categorized as exact methods, which guarantee optimal solutions, or heuristics, which sacrifice optimality for computational efficiency. Heuristics are

often rule based and can be seen as decision-making strategies. We propose parameterizing certain heuristic strategies using neural networks (NNs) trained to develop improved algorithms.

The use of NNs in solving combinatorial optimization problems can be traced back to Hopfield and Tank (1985), who used Hopfield's network to solve a small-scale traveling salesman problem (TSP). Since then, NNs have been applied to various, related problems Smith (1999), with most solutions learned from scratch in an online manner. More recently, NNs have also been used offline to solve related problems. Vinyals et al. (2015) and Bello et al. (2016) proposed and improved a pointer network for solving the TSP, work that was further extended to the vehicle routing problem by Nazari et al. (2018). Recently, scholars have integrated deep learning models into reinforcement learning, leading to the development of a deep Q-Learning network (DQN). Mnih et al. (2013) and Mnih et al. (2015) pioneered the successful application of DQNs to control strategy learning in the Atari 2600 game, demonstrating the potential of deep-learning models for reinforcement learning. Subsequent studies have introduced various enhancements to DQNs, such as the dual DQN proposed by Van Hasselt et al. (2016), which separates action selection from value assessment to prevent overestimation. Another variant, the dueling DQN proposed by Wang et al. (2016), employs two independent estimators for the state value and competitive advantage functions, leading to improved policy evaluation, especially when the Q values are close. Schaul et al. (2015) were inspired by human thought processes and emphasized the crucial role of prioritized experience replay in DQN training, demonstrating how DQNs with prioritized experience replay can improve sample utilization efficiency. For more in-depth insights into reinforcement learning, Sutton and Barto (2018) is a recommended source.

Table 1. Research perspectives related to multi-deep compact RMFSs

Article	System	Storage mode	Methodology	Research content
Lamballais et al. (2017) Yuan and Gong (2017) Zou et al. (2018) Roy et al. (2019) Gong et al. (2021)	RMFS	Noncompact	Analytic	Layout design
Weidinger et al. (2018) Ji et al. (2020) Boysen et al. (2017) Zhuang et al. (2022) Gharehgozli and Zaerpour (2020)			Optimization	Pod scheduling
Merschformann et al. (2019) Müller et al. (2020)			Simulation	Robot dispatching
Gue and Kim (2007)	PBS	Compact	Heuristic	Single-item retrieval with single escorts and single IO points
Rohit et al. (2010)			IP	Single-item retrieval with multiple escorts and single IO points
Yalcin et al. (2019)			Heuristic	
MA et al. (2022)			Heuristic	
Yu et al. (2022)			Heuristic	Multi-item retrieval with multiple escorts and single IO points
Alfieri et al. (2012)			Heuristic	

Bukchin and Raviv (2022)			DP	
Gue et al. (2013)			Heuristic	
Mirzaei et al. (2017)			Heuristic	Multi-item retrieval with multiple escorts and multiple IO points
Zou and Qi (2021)			Heuristic	
He et al. (2023)			IP & RL	
Li et al. (2021)	RMFS	Compact	Simulation	Robot dispatching
Yang et al. (2022)			Analytic	Layout design
This paper	RMFS	Compact	IP & RL	Pod scheduling & Robot dispatching

Notes: IP = integer programming, DP = dynamic programming, and RL = reinforcement learning

We have conducted a thorough literature review on the related work—including RMFS, PBS and reinforcement learning—as presented above and summarized in Table 1. Our analysis reveals that rarely has the previous research specifically addressed the problem of pod retrieval and robot dispatch in multi-deep compact RMFSs. Moreover, to the best of our knowledge, no prior work has attempted to tackle this problem using a reinforcement learning approach to learn about and overcome potential challenges. This study aims to bridge this gap and advance the current research.

3. Problem description and formulation

3.1 Multi-deep compact RMFSs

Figure 3 depicts the fundamental layout of multi-deep compact RMFSs, which contain $W * L$ blocks (3 wide x 4 long). The picking stations can be arranged symmetrically or asymmetrically around the warehouse (in this paper, we assume a uniform and even distribution of picking stations on the left-hand side). The pods are grouped into blocks that are separated by vertical and horizontal aisles. Each block measures $m * n$ (3*4), meaning that it has m rows and n columns.

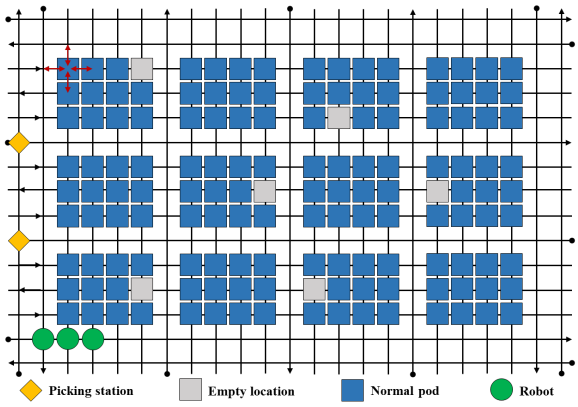


Figure 3. Layout of multi-deep compact RMFSs

In a traditional single-deep RMFS, each pod is connected directly to an aisle, and each block has a depth of one (block depth refers to the maximum number of locations between any pod in a block and the nearest aisle). This layout allows for individual access to each pod. However, in multi-deep compact

RMFSs, other pods must be moved out of the way before retrieving a target pod. To ensure the smooth operation of multi-deep compact RMFSs, a few empty locations must be reserved. Doing so guarantees that the storage system functions properly, as depicted in Figure 3.

3.2 Pod retrieval process

This paper aims to address the pod retrieval scheduling and robot dispatching problem of multi-deep compact RMFS. Our goal is to optimize pod and robot scheduling to complete all tasks in the shortest time, given a series of tasks that each involve a target pod and potentially multiple pods obstructing its movement. From a task perspective, the pod retrieval process consists of 5 steps, as shown in Figure 4. 1) The robot travels to the location of the obstructing pod. 2) The robot carries the obstructing pod to an empty location. 3) The robot travels to the location of the target pod. 4) The robot carries the target pod to the picking station. 5) The robot carries the finished pod to its new storage location. Notably, these processes can be executed by distinct robots, engaging in a collaborative effort to collectively achieve pod retrieval.

The following assumptions are used through this paper. 1) All aisles are unidirectional to prevent collisions and deadlocks. 2) Reserved empty positions are distributed randomly throughout the storage area. 3) Items on the pods can always meet the demand of incoming orders. 4) The targeted pod cannot visit different picking stations continuously. 5) The point of service completion strategy is used for the robot's staging points. 6) Not considered are the time taken by the robot to lift and unload the pod or the service time of the picking station. As we primarily focus on pod retrieval scheduling, neglecting these times and treating them as fixed values are reasonable. 7) The robot moves at a constant speed, without considering its acceleration and deceleration motion.

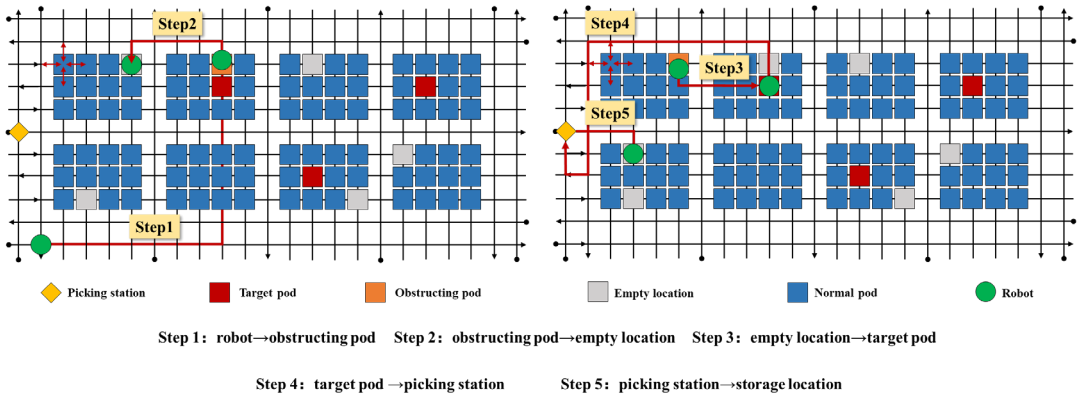


Figure 4. Retrieval process in multi-deep compact RMFSs

For convenience, the abbreviations and symbols involved in the study are listed in Appendix A,

Tables A1 and A2.

3.3 Integer programming formulation

This section presents an integer programming model that covers various problem settings, such as multiple tasks, picking stations, robots, and empty locations. The tasks and empty locations are randomly placed in the warehouse, while the picking stations are located on the left side, and the robots are located below the warehouse. The number of tasks, picking stations, robots, and empty locations can be set to any feasible value. To design the decision variables accurately, we first briefly describe the system state transitions.

We consider a multi-deep compact RMFS with $W * L$ blocks. The state of the system at each timestamp is determined by the occupancy of each grid, which can be an empty position, a target pod, a picking station, an aisle, or a robot. To accurately describe the occupancy status and changes of the grid, we introduce three sets of binary variables.

1) e_i^t: whether position i is an empty position at time t; if so, then $e_i^t = 1$, otherwise $e_i^t = 0$.

2) g_i^t: whether position i is a target pod at time t; if so, then $g_i^t = 1$, otherwise $g_i^t = 0$.

3) h_i^t: whether position i has a robot at time t; if so, then $h_i^t = 1$, otherwise $h_i^t = 0$.

Furthermore, we introduce the following four sets of binary decision variables to represent the movement of the robot and pod, as well as the completion of the task.

1) $m_{r,i,j}^t$: whether robot r moves from position i to position j at time t; if so, then $m_{r,i,j}^t = 1$, otherwise $m_{r,i,j}^t = 0$.

2) $x_{r,i,j}^t$: whether robot r moves the pod from position i to position j at time t; if so, then $x_{r,i,j}^t = 1$, otherwise $x_{r,i,j}^t = 0$.

3) $b_{r,i,j}^t$: whether robot r moves the target pod from position i to position j at time t; if so, then $b_{r,i,j}^t = 1$, otherwise $b_{r,i,j}^t = 0$.

4) f^t: whether all tasks have been completed at time t; if so, then $f^t = 0$, otherwise $f^t = 1$.

As illustrated in Figure 5, the left side of the figure shows the states of each grid in the system at time $t-1$. At time t, the robot moves from grid 49 to 50, the normal pod moves from grid 29 to 21, and the target pod moves from grid 20 to 12, resulting in the corresponding state transitions shown on the right side of the figure. Therefore, the robot scheduling process can be interpreted as a sequence of state changes in the system, starting from the initial state and ending when all tasks are completed.

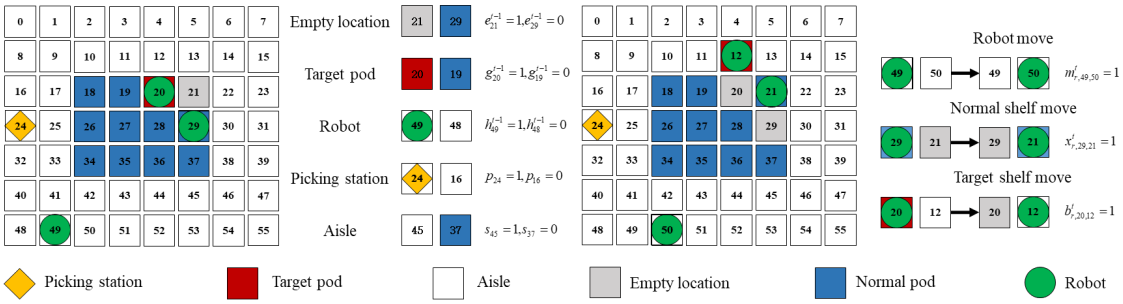


Figure 5. System state changes over time

Building upon the aforementioned considerations, we aim to model the problem using integer programming. To facilitate the modeling process, we introduce an input parameter T that represents the longest system operation time. This parameter implies that all tasks must be completed within a predetermined time limit, T, which can be set to a sufficiently large value. In summary, we can formulate the retrieval problem for multi-deep compact RMFSs as follows.

$$min\, Z = \sum_{t=1}^{T} f^t \tag{1}$$

$$s.t. \quad \sum_{i=1}^{N}\sum_{j=1}^{N} m_{r,i,j}^{t} \le 1, \quad \forall r \in R \quad \forall t = 0,\dots,T \tag{2}$$

$$\sum_{r=1}^{R}\sum_{j=1}^{N} m_{r,i,j}^{t} \le 1, \quad \forall i \in N \quad \forall t = 0,\dots,T \tag{3}$$

$$\sum_{r=1}^{R}\sum_{i=1}^{N} m_{r,i,j}^{t} \le 1, \quad \forall j \in N \quad \forall t = 0,\dots,T \tag{4}$$

$$m_{r,i,j}^{t} \le A_{i,j}, \quad \forall i \in N \quad \forall j \in N \quad \forall r = R \quad \forall t = 0,\dots,T \tag{5}$$

$$m_{r,i,j}^{t} \le h_i^t, \quad \forall i,j \in N \quad \forall r \in R \quad \forall t = 0,\dots,T \tag{6}$$

$$m_{r,i,j}^{t} \le 1 - h_j^t, \quad \forall i,j \in N \quad \forall r \in R \quad \forall t = 0,\dots,T \tag{7}$$

$$x_{r,i,j}^{t} \le m_{r,i,j}^{t}, \quad \forall i,j \in N \quad \forall r \in R \quad \forall t = 0,\dots,T \tag{8}$$

$$x_{r,i,j}^{t} \le e_i^t, \quad \forall i,j \in N \quad \forall r \in R \quad \forall t = 0,\dots,T \tag{9}$$

$$x_{r,i,j}^{t} \le 1 - e_j^t, \quad \forall i,j \in N \quad \forall r \in R \quad \forall t = 0,\dots,T \tag{10}$$

$$\sum_{i=1}^{N} x_{r,j,i}^{t} \ge \sum_{i=1}^{N} x_{r,i,j}^{t-1}, \quad \forall j \in \{j \in N | s_j = 1\} \quad \forall r \in R \quad \forall t = 1,\dots,T \tag{11}$$

$$h_i^t + \sum_{r=1}^{R}\sum_{j=1}^{N} m_{r,i,j}^{t-1} - \sum_{r=1}^{R}\sum_{j=1}^{N} m_{r,j,i}^{t-1} = h_i^{t-1}, \quad \forall i \in N \quad \forall t = 1,\dots,T \tag{12}$$

$$e_i^t + \sum_{r=1}^{R}\sum_{j=1}^{N} x_{r,i,j}^{t-1} - \sum_{r=1}^{R}\sum_{j=1}^{N} x_{r,j,i}^{t-1} = e_i^{t-1}, \quad \forall i \in N \quad \forall t = 1,\dots,T \tag{13}$$

$$g_i^t + \sum_{r=1}^{R}\sum_{j=1}^{N} b_{r,i,j}^{t-1} - \sum_{r=1}^{R}\sum_{j=1}^{N} b_{r,j,i}^{t-1} = g_i^{t-1}, \quad \forall i \in \{i \in N | p_i = 0\} \quad \forall t = 1,\dots,T \tag{14}$$

$$f^t \ge g_i^t, \quad \forall i \in N \quad \forall t = 0,\dots,T \tag{15}$$

$$f^T = 0 \tag{16}$$

$$x_{r,i,j}^{t} \le f^t, \quad \forall i,j \in N \quad \forall r \in R \quad \forall t = 0,\dots,T \tag{17}$$

$$f^t \in \{0,1\}, \quad \forall t = 0,\dots,T \tag{18}$$

$$e_i^t, g_i^t, h_i^t \in \{0,1\}, \quad \forall i \in N \quad \forall t = 0, \dots, T \tag{19}$$

$$m_{r,i,j}^t, x_{r,i,j}^t, b_{r,i,j}^t \in \{0,1\}, \quad \forall i \in N \quad \forall j \in N \quad \forall r = R \quad \forall t = 0, \dots, T \tag{20}$$

Objective Function (1) aims to minimize the total time needed to complete all tasks. Constraint (2) ensures that each robot can move at most once each timestamp. Constraint (3) limits to one the number of robots that can depart from each location at each timestamp. Similarly, Constraint (4) restricts to one the number of robots that can arrive at each location at each timestamp. Constraint (5) enforces the reachability condition for each robot location at each timestamp. Constraints (6) and (7) dictate that a move is allowed only if a robot exists at that location, and an arrival is allowed only if no robot exists at that location. Constraint (8) stipulates that a pod can only be moved when a robot move is made. Constraint (9) mandates that no pod carried by a robot at any timestamp can be placed at an empty location. Constraint (10) specifies that the pod handled by each robot at each timestamp can be placed only in an empty position. Constraint (11) ensures that the robot cannot stay on the aisle. Constraints (12), (13), and (14) update the robot position state, pod state, and target pod state, respectively. Finally, Constraints (15), (16), and (17) guarantee that all tasks are completed.

4. Simulation-based multi-strategy heuristic framework

We first propose a simulation-based multi-strategy heuristic framework to solve the above IP model and yield the baseline solution to assess the performance of the proposed reinforcement learning algorithm in section 5. This framework includes a processing layer, a scheduling layer, and an execution layer, which are comprised of six agents, as illustrated in Figure 6. The pseudo-code of the algorithm can be found in Appendix B. When the system receives a series of orders, the order processing agent first groups the orders into batches before assigning them to the appropriate picking station agents and creating tasks. The picking station agent then transmits the task information to the information management agent, which dispatches the robot and pod agents to complete their respective tasks. The system's environment is modeled as an agent, which collaborates with the picking station, robot, pod, and information management agents to regulate traffic and prevent congestion, collisions, and deadlocks among the robots.

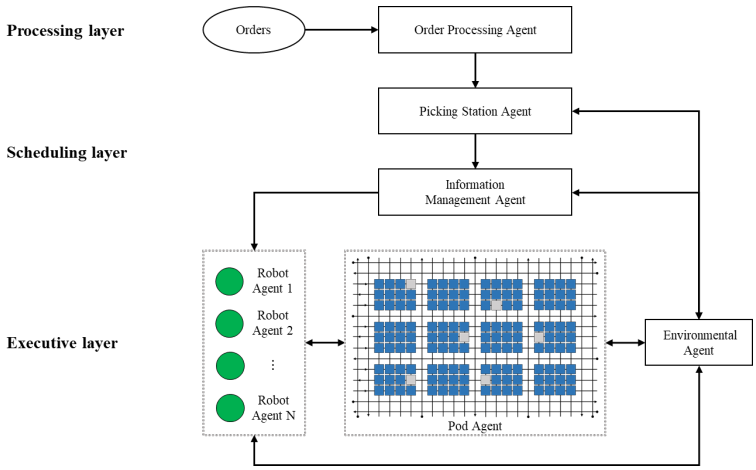


Figure 6. Collaborative multi-agent framework of multi-deep compact RMFSs

4.1 Processing layer

The processing layer is primarily responsible for the batch formation of external orders, task creation, and distribution to the relevant picking stations. The picking stations then determine the order in which tasks should be completed by pod sequencing. In contrast to the traditional RMFS approach in which the robots can move directly to complete tasks, in multi-deep compact RMFSs, robots may encounter obstructions that impede their movement toward the target pod. To overcome this obstacle, we introduce two definitions that enable the identification of the obstructing pod to facilitate efficient task completion.

Definition 1—Optimal Direction: the direction in which the target pod can be moved from the storage location to the nearest aisle as soon as possible.

Definition 2—Obstructing pod: the pod that blocks the route of the target pod in the Optimal Direction.

Considering that the essence of moving a target pod or an obstructed pod is to move a pod to an area, we decompose a single task into several subtasks, including moving an obstructed pod, moving a target pod, and moving a pod back to the storage area after picking. First, we define the task set to be handled by the system at a certain stage as $O = \{o_1, o_2, \cdots, o_O\}$. Assuming that the number of obstructed pods above, below, left, and right of the target pod required for a certain task o_o is $d_o^1, d_o^2, d_o^3, d_o^4$, respectively, then the total number of obstructed pods to be moved can be calculated by Equation 18.

$$d_o^* = \min\{d_o^1, d_o^2, d_o^3, d_o^4\}, \forall o \in O \tag{18}$$

Next, we decompose the main task into multiple subtasks, where each subtask corresponds to a single robot dispatch. As a result, the subtask set for the primary task o_o, known as $subor_o$, can be

expressed using Equation 2, where $N_o = d_o^* + 2$, $s_o^1 \sim s_o^{N_o-2}$ represents the obstructing pod, $s_o^{N_o-1}$ represents the target pod, and $s_o^{N_o}$ represents the returning pods.

$$subor_o = \{s_o^1, s_o^2, \cdots, s_o^{N_o}\}, \forall o \in O \tag{19}$$

As illustrated in Figure 7, the optimal movement direction is upward, and the two pods that impede the target pod's movement are considered its obstructing pods. Consequently, we decompose the main task into four subtasks, which are then assigned to available robots in sequence for efficient task completion.

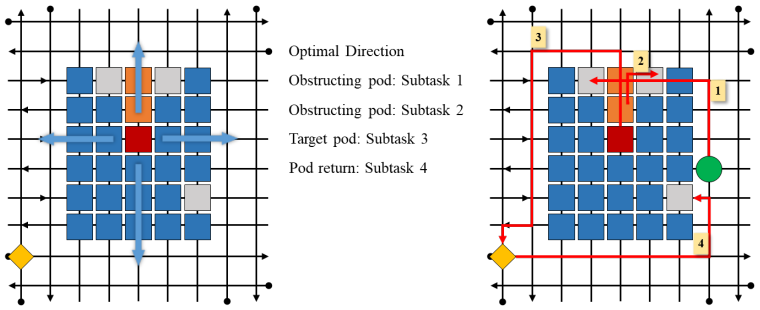


Figure 7. Dismantling the main task to form subtasks

4.2 Scheduling layer

The scheduling layer is primarily responsible for pod scheduling and robot task assignment. It involves selecting a destination for each subtask and assigning it to an available robot. Given order o_o, the subtask set is $subor_o = \{s_o^1, s_o^2, \cdots, s_o^{N_o}\}$. The available robots are represented by set $R = \{r_1, r_2, \cdots, r_{N_r}\}$, while the picking stations and empty locations are denoted by sets $P = \{p_1, p_2, \cdots, p_{N_p}\}$ and $E = \{e_1, e_2, \cdots, e_{N_e}\}$, respectively, and are dynamically updated as the system evolves. The robot scheduling for each subtask $s_o^{n_o}$ involves two decisions.

The first decision is to allocate an available robot to the subtask and schedule the robot's movement from its current location to the subtask's location. We adopt the strategy of binding the subtask to the robot that is closest to it. We use the Manhattan distance as our distance measure, as shown in Equation 20. We first calculate the distance between each idle robot and the subtask, as expressed in Equation 21. We then select the idle robot with the shortest distance to complete the subtask, as shown in Equation 22.

$$d_{i,j} = |x_i - x_j| + |y_i - y_j| \tag{20}$$

$$d_{r_{n_r}, s_o^{n_o}} = \left|x_{r_{n_r}} - x_{s_o^{n_o}}\right| + \left|y_{r_{n_r}} - y_{s_o^{n_o}}\right|, \forall\ r_{n_r} \in R \tag{21}$$

$$d_1 = \min\left\{d_{r_1,s_o^{n_o}}, d_{r_2,s_o^{n_o}} \cdots, d_{r_{N_r},s_o^{n_o}}\right\} \tag{22}$$

Decision 2 involves assigning a destination for each subtask, depending on whether it involves an obstructing, target, or returning pod. For obstructing pods, an empty location must be assigned as the destination. To determine the nearest empty location, we use the Manhattan distance, as shown in Equations 23 and 24. In the case of target pods, a picking station is assigned as the destination, using Equations 25 and 26. For returning pods, the process is similar to that of obstructing pods, and an empty location is assigned as the destination using Equations 27.

$$d_{s_o^{n_o},e_{n_e}} = \left|x_{s_o^{n_o}} - x_{e_{n_e}}\right| + \left|y_{s_o^{n_o}} - y_{e_{n_e}}\right|, \forall\ e_{n_e} \in E \tag{23}$$

$$d_2^1 = \min\left\{d_{s_o^{n_o},e_1}, d_{s_o^{n_o},e_2}, \cdots, d_{s_o^{n_o},e_{N_e}}\right\} \tag{24}$$

$$d_{s_o^{n_o},p_{n_p}} = \left|x_{s_o^{n_o}} - x_{p_{n_p}}\right| + \left|y_{s_o^{n_o}} - y_{p_{n_p}}\right|, \forall\ p_{n_p} \in P \tag{25}$$

$$d_2^2 = \min\left\{d_{s_o^{n_o},p_1}, d_{s_o^{n_o},p_2}, \cdots, d_{s_o^{n_o},p_{N_p}}\right\} \tag{26}$$

$$d_2^3 = d_2^1 \tag{27}$$

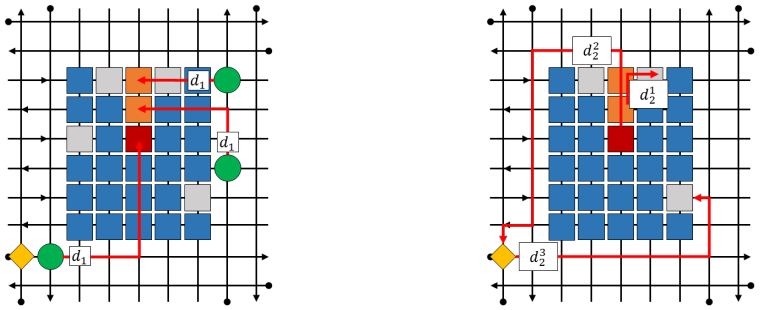


Figure 8. Assigning subtasks and dispatching idle robots

4.3 Executive layer

The execution layer plays two vital roles, namely, path planning and traffic control. To plan the optimal path for a robot, we employ the Dijkstra algorithm, as illustrated in Figure 8. When the robot is unloaded, it can move under the pod, whereas when it has a load, it is permitted to pass only through the aisle or an empty position. To avoid collisions and deadlocks among multiple robots, we introduce a conflict-free mechanism, as depicted in Figure 9, that involves three stages: individual path planning, overall conflict detection, and resolution. In the first stage, the robot agent uses the Dijkstra algorithm to plan its route from its current location to its destination. In the second stage, the robot agent identifies potential conflicts between its intended route and those of other robots using a collision detection algorithm.

If no conflicts arise, the planned route is executed; otherwise, conflicts must be resolved in the third stage.

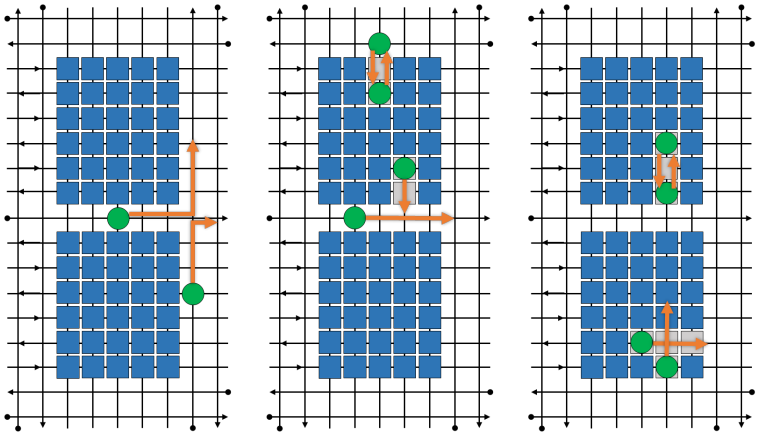


Figure 9. Collision and deadlock situations between multiple robots

5. Reinforcement learning method

We propose the reinforcement learning method to strengthen the ability of efficiently obtaining high-quality pod retrieval scheduling solution by leveraging the existing patterns and experiences associated with pod retrieval scheduling in multi-deep compact RMFSs. First, we design a suitable reinforcement learning environment for the multi-deep compact RMFSs. Next, attention models are constructed for pod retrieval scheduling. Lastly, we implement REINFORCE with the Rollout Baseline algorithm to train the model.

5.1 Environment

The training of a reinforcement learning model relies on the environment in which the subject interacts and learns from its experience. The agent receives specific feedback, or rewards, based on the current state of the environment and the actions taken. Positive rewards enhance the probability of action, while negative rewards weaken it. Over time, the agent learns to choose the best action based on the cumulative rewards received, ultimately optimizing its performance.

The iterative process for multi-deep compact RMFSs reinforcement learning can be described as follows. First, the system environment is initialized by randomly generating a certain number of target tasks and empty locations, and setting several robots and picking stations at designated passages. Then, the tasks are assigned based on the task information and the availability of empty robots. The environment is updated accordingly, and corresponding rewards are received. The cycle continues until all tasks

are completed, at which point the iteration terminates and the environment is reset. The system states, actions, and rewards are defined as follows.

State: To enable the agents in the reinforcement learning model to learn from their experience and interact with the environment, they must be provided with information about the environment. In this paper, we use a variety of variables—such as the grid's horizontal and vertical coordinates, empty locations, target pods, robots, picking stations, and aisles—to represent the entire system's state and to provide feedback on the state after each action.

Actions: Since our goal is to learn the pod retrieval scheduling process, the actions in our reinforcement learning model have two primary decisions. One is task to assign to an idle robot, and the other is which location to use as the destination for that task.

Reward: To encourage the agent to find the best possible solution, the environment provides a positive reward only when all tasks are completed. If any tasks remain incomplete, the reward is zero, thus incentivizing the agent to discover the termination as quickly as possible.

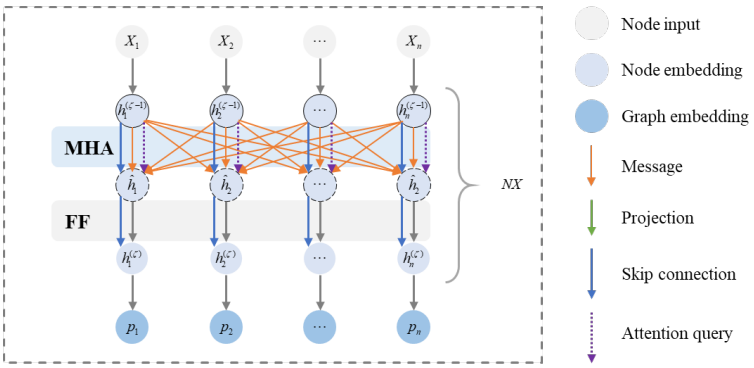


Figure 10. Attention model

5.2 Attention models

To account for the pod retrieval scheduling characteristics of multi-deep compact RMFSs, we employed the attention model, as illustrated in Figure 10. The model remains the same for various problems, but we must define the inputs, outputs, and model structures appropriately. Since pod retrieval scheduling requires two decisions, we used the attention model to learn the decision processes for both pod sequencing and pod scheduling. Doing so allowed us to build a two-layer attention model, in which the upper layer is responsible for deciding the pod sequencing, and the result serves as the input to the lower layer attention model, which then determines the pod scheduling. This approach improves the model's expressiveness and accuracy in decision-making.

Input: To represent either the pod sequencing or pod scheduling problem as a graph, we define instance s with n nodes, in which each node corresponds to a grid in the reinforcement learning environment. Specifically, node i is described by feature X_i, which consists of the coordinates of the node and various basic node state information, as shown in Equation 28. This representation enables us to model both pod sequencing and scheduling as graph problems, which can be efficiently solved using graph-based algorithms.

$$X_i = \{x_i, y_i, e_i, g_i, h_i, p_i, s_i\} \tag{28}$$

Output: We define $\pi^1, \pi^2,$ and π as solutions for pod sequencing, pod scheduling and pod retrieval scheduling, respectively. The solution for pod sequencing is denoted as $\pi^1 = \{\pi_1^1, \pi_2^1, \cdots \pi_t^1\}$, and $\pi_t^1 \neq \pi_{t'}^1, \forall t \neq t'$. The solution for pod scheduling is denoted as $\pi^2 = \{\pi_1^2, \pi_2^2, \cdots \pi_t^1\}$, and the solution for pod retrieval scheduling is denoted as $\pi = \{\pi_1, \pi_2, \cdots \pi_t\}$. We use a network model of attention model to select corresponding solutions π^1, π^2, π given instance s, and we define random policies $p(\pi^1|s), p(\pi^2|s), p(\pi|s)$ for pod sequencing, pod scheduling and pod retrieval scheduling, respectively. These policies can be factorized and parameterized by $\theta_1, \theta_1, \theta$ as shown in Equations. 29, 30, and 31.

$$p_{\theta_1}(\pi^1|s) = \prod_{t=1}^{n} p_{\theta_1}(\pi_t^1|s, \pi_{1:t-1}^1) \tag{29}$$

$$p_{\theta_2}(\pi^2|s) = \prod_{t=1}^{n} p_{\theta_2}(\pi_t^2|s, \pi_{1:t-1}^2) \tag{30}$$

$$p_\theta(\pi|s) = p_{\theta_1,\theta_2}(\pi^1, \pi^2|s) = \prod_{t=1}^{n} p_{\theta_1}(\pi_t^1|s, \pi_{1:t-1}^1, \pi_{1:t-1}^2)\, p_{\theta_2}(\pi_t^2|s, \pi_{1:t}^1, \pi_{1:t-1}^2) \tag{31}$$

For multi-deep compact RMFSs, we begin by determining the task and empty position nodes, which are then used as input for the attention model. The model calculates and generates solutions π^1, π^2, π under the corresponding random policy, determining which task is assigned to which empty location. This decision updates the system state, which serves as input for the next decision-making process. This operational sequence is repeated until all tasks are completed and the cutoff point is reached. See Appendix C.1 for the detailed network structures for encoding, decoding, etc.

We can illustrate the decision-making process of the attention model with an example problem that involves three tasks to be completed and three available empty locations, as shown in Figure 11. The model uses the corresponding features as its input and outputs the node with the highest probability, which becomes the input for the next step. As demonstrated in Figure 11, the task order is 3>2>1, with

Task 3 assigned to Empty Location 1 (3→1), and Tasks 2 and 1 assigned to Empty Locations 2 and 3, respectively (2→2, 1→3).

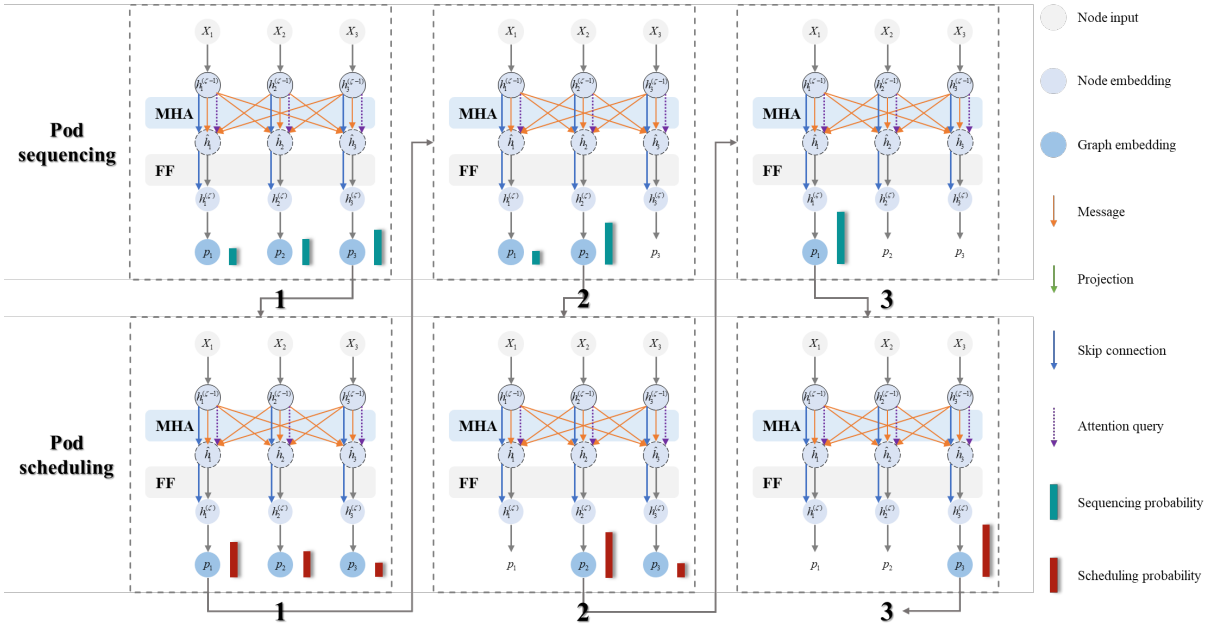


Figure 11. Attention model for the pod sequencing and pod scheduling problems

5.3 Reinforce with greedy rollout baseline

Our attention model generates a probability for instance s, which can be sampled to obtain a solution. To train the model, we define loss function $\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(L(\pi)]$, where $L(\pi)$ is the cost of completing all tasks. We optimize $\mathcal{L}$ using gradient descent and the REINFORCE (Williams, 1992) gradient estimator with baseline $b(s)$, as shown in Equation 32. Doing so ensures that we obtain an optimal solution that minimizes the total time required to complete all tasks.

$$\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}\left[\left(L(\pi) - b(s)\right)\nabla log p_\theta(\pi|s)\right] \tag{32}$$

We utilize Adam (Kingma & Ba, 2014) reinforce with a greedy rollout baseline for the training algorithm for the attention model. Further details can be found in Appendix C.2.

6. Numerical experiments

We perform extensive numerical experiments to evaluate the effectiveness of the proposed heuristic and reinforcement learning methods. Initially, we compare the solutions obtained from the heuristic and reinforcement learning methods with that from commerce solver on small-scale instances. Furthermore, we introduce several rule-based and greedy algorithms commonly used in traditional single-deep RMFS, including those for solving rack sequence, rack allocation, and rack scheduling. Later, we benchmark

our proposed simulation-based multi-strategy heuristic and test the efficiency of reinforcement learning methods on medium and large-scale instances.

The experiments are conducted on a Windows operating system equipped with a 2.90 GHz CPU and 16 GB RAM. The integer programming model is solved using Gurobi 10.0.1. The attention model is configured with 2 layers and 2 heads, with embedding and hidden dimension of 128. During training, we utilize the Adam optimizer with a learning rate of 0.001, a discount rate of 0.95, a batch size of 100, and a batch number of 300 to optimize the model's performance.

6.1 Experimental setting

We generate a set of standard cases for our experiments, which include small-scale instances as well as medium- and large-scale instances. Table 2 provides details of the small-scale instances, while Table 3 provides details of the medium- and large-scale instances. Each standard instance comprises five elements: the number of blocks ($W * L$) and the size of each block ($m * n$), the number of robots (R), the number of target tasks (O), the ratio of empty locations (REL), and the number of picking stations (P).

We category the instances into two groups for different purposes. First, the small-scale instances I-R1 to I-R22 (Table 2) are used to compare the optimal solutions obtained by the heuristic and reinforcement learning methods with the IP model. Second, the medium- and large-scale instances II-R1 to II-R30 (Table 3) are employed to evaluate the performance of the heuristic and reinforcement learning methods in solving medium- and large-scale problems.

Table 2. Small-scale instances I-R1~I-R22

ID	$W*L\times m\times n$	R	O	REL	P	**ID**	$W*L\times m\times n$	R	O	REL	P
I-R1	1*1*3*4	1	1	0.2	2	I-R12	1*1*3*6	2	1	0.2	2
I-R2	1*1*3*6	1	1	0.2	2	I-R13	1*2*3*4	2	1	0.2	2
I-R3	1*2*3*4	1	1	0.2	2	I-R14	1*2*3*6	2	1	0.2	2
I-R4	1*2*3*6	1	1	0.2	2	I-R15	2*2*3*4	2	1	0.2	3
I-R5	2*2*3*4	1	1	0.2	3	I-R16	2*2*3*6	2	1	0.2	3
I-R6	2*2*3*6	1	1	0.2	3	I-R17	2*3*3*4	2	1	0.2	3
I-R7	2*3*3*4	1	1	0.2	3	I-R18	2*3*3*6	2	1	0.2	3
I-R8	2*3*3*6	1	1	0.2	3	I-R19	1*1*3*4	2	2	0.2	2
I-R9	3*3*3*4	1	1	0.2	4	I-R20	1*1*3*6	2	2	0.2	2
I-R10	3*3*3*6	2	1	0.2	4	I-R21	1*2*3*4	2	2	0.2	2
I-R11	1*1*3*4	2	1	0.2	2	I-R22	1*2*3*6	2	2	0.2	2

Table 3. Medium- and large-scale instances II-R1~II-R30

ID	$W*L\times m\times n$	R	O	REL	P	ID	$W*L\times m\times n$	R	O	REL	P
II-R1	2*6*3*3	6	20	0.2	3	II-R16	4*12*5*5	15	70	0.2	5
II-R2	2*8*3*3	9	30	0.2	3	II-R17	4*14*5*5	18	80	0.2	5
II-R3	2*10*3*3	12	40	0.2	3	II-R18	4*16*5*5	21	90	0.2	5
II-R4	2*12*3*3	15	50	0.2	3	II-R19	5*6*6*6	6	50	0.2	6
II-R5	2*14*3*3	18	60	0.2	3	II-R20	5*8*6*6	9	60	0.2	6
II-R6	2*16*3*3	21	70	0.2	3	II-R21	5*10*6*6	12	70	0.2	6
II-R7	3*6*4*4	6	30	0.2	4	II-R22	5*12*6*6	15	80	0.2	6
II-R8	3*8*4*4	9	40	0.2	4	II-R23	5*14*6*6	18	90	0.2	6
II-R9	3*10*4*4	12	50	0.2	4	II-R24	5*16*6*6	21	100	0.2	6

II-R10	3*12*4*4	15	60	0.2	4	II-R25	6*6*7*7	6	60	0.2	7
II-R11	3*14*4*4	18	70	0.2	4	II-R26	6*8*7*7	9	70	0.2	7
II-R12	3*16*4*4	21	80	0.2	4	II-R27	6*10*7*7	12	80	0.2	7
II-R13	4*6*5*5	6	40	0.2	5	II-R28	6*12*7*7	15	90	0.2	7
II-R14	4*8*5*5	9	50	0.2	5	II-R29	6*14*7*7	18	100	0.2	7
II-R15	4*10*5*5	12	60	0.2	5	II-R30	6*16*7*7	21	110	0.2	7

6.2 Small-scale instances

To evaluate the performance of the heuristic and reinforcement learning methods, we compare their results with the IP solution and summary them in Table 4. For each instance, we conduct 100 repeated experiments and take the average as the final result. The results show that the difference between the heuristic and reinforcement learning methods and the Gurobi optimal solution is 6.05% and 4.17%, respectively, and the average times required (ATRs) are 18.31 and 17.99, respectively. These findings suggest that the heuristic and reinforcement learning methods produce solutions that are close to the optimal solution in the small-scale instances. However, we notice that the reinforcement learning method does not always outperform the heuristic method and is even weaker than the heuristic method in a few cases. We suspect that the reason is the small number of tasks because the reinforcement learning method primarily sequences and schedules pods, which is not advantageous when the number of tasks is limited. To test this hypothesis, we conduct a second set of experiments on small-scale instances. During the experiment, we observe that the reinforcement learning method was weaker than the heuristic method when the number of tasks was below a certain value. Nevertheless, when the number of tasks increase, the reinforcement learning method exhibits consistent advantages, as depicted in Figure 12. These observations led us to question whether the reinforcement learning method's advantages increase along with the size of the instances. Therefore, we conduct further numerical experiments on medium- and large-scale instances.

In addition, we refer to several papers that study traditional single-deep RMFS and introduce some classical rule-based algorithms for rack sequence (Wang et al., 2022), rack allocation (Weidinger et al., 2018), and rack dispatching (Zhuang et al., 2022), respectively. Furthermore, we combine the three rule-based algorithms into R-ALL. We also test its performance on small-scale instances, with results presented in Table 4. From the results, it can be observed that traditional single-deep RMFS algorithms fail to effectively address similar problems in multi-deep RMFS.

Table 4. IP, Heuristic, and RL in the small-scale instances

ID	Gurobi		Heuristic			RL			R-ALL		
	Obj_g	$Time_g$	Obj_h	$Time_h$	Δ_h^{obj}	Obj_{rl}	$Time_{rl}$	Δ_{rl}^{obj}	Obj_{r-all}	$Time_{r-all}$	Δ_{r-all}^{obj}
I-R1	17.92	8.07	18.42	0.00	2.81%	18.48	0.00	3.14%	20.49	0.00	14.36%

I-R2	19.45	7.07	20.48	0.00	5.27%	20.35	0.00	4.61%	23.56	0.00	21.14%
I-R3	24.88	7.51	25.79	0.00	3.68%	25.12	0.00	0.97%	26.67	0.00	7.20%
I-R4	21.35	4.88	21.86	0.00	2.39%	21.65	0.00	1.41%	28.59	0.00	33.89%
I-R5	22.09	11.49	22.78	0.00	3.13%	22.28	0.00	0.85%	28.79	0.00	30.33%
I-R6	17.43	13.24	17.66	0.00	1.32%	17.63	0.00	1.15%	30.11	0.00	72.77%
I-R7	17.17	6.83	17.24	0.00	0.40%	17.54	0.00	2.15%	30.01	0.00	74.77%
I-R8	13.31	12.05	13.84	0.00	3.98%	13.86	0.00	4.17%	23.58	0.00	77.16%
I-R9	11.38	25.64	13.09	0.00	15.00%	12.56	0.00	10.33%	18.63	0.00	63.75%
I-R10	23.55	12.77	24.28	0.00	3.12%	24.38	0.00	3.53%	30.75	0.00	30.57%
I-R11	15.34	100.94	16.18	0.00	5.48%	16.05	0.00	4.61%	20.27	0.00	32.15%
I-R12	15.97	114.27	16.40	0.00	2.71%	16.69	0.00	4.49%	23.56	0.00	47.51%
I-R13	20.22	157.54	20.53	0.00	1.55%	20.77	0.00	2.71%	26.66	0.00	31.86%
I-R14	19.58	159.72	19.96	0.00	1.94%	19.66	0.00	0.40%	28.59	0.00	46.03%
I-R15	14.51	175.23	14.75	0.00	1.65%	14.78	0.00	1.86%	22.38	0.00	54.24%
I-R16	12.03	104.84	13.10	0.00	8.87%	13.10	0.00	8.89%	20.44	0.00	69.88%
I-R17	12.88	91.02	14.46	0.00	12.30%	13.52	0.00	4.97%	21.21	0.00	64.70%
I-R18	12.86	109.14	13.86	0.00	7.74%	13.85	0.00	7.72%	22.77	0.00	77.09%
I-R19	17.22	98.71	19.96	0.00	15.92%	17.58	0.00	2.09%	28.49	0.00	65.45%
I-R20	17.02	131.19	18.32	0.00	7.62%	18.33	0.00	7.71%	26.93	0.00	58.25%
I-R21	17.07	155.52	19.24	0.00	12.69%	18.54	0.00	8.62%	28.79	0.00	68.68%
I-R22	18.13	161.45	20.57	0.00	13.45%	19.11	0.00	5.41%	27.21	0.00	50.08%
Average	17.33	75.87	18.31	0.00	**6.05%**	17.99	0.00	**4.17%**	25.39	0.00	**49.63%**

$Note. \Delta_h^{obj} = (Obj_h - Obj_g)/Obj_g;\ \Delta_{rl}^{obj} = (Obj_{rl} - Obj_g)/Obj_g;\ \Delta_{r-all}^{obj} = (Obj_{r-all} - Obj_g)/Obj_g.$

6.3 Medium- and large-scale instances

To assess the efficacy of the heuristics and reinforcement learning methods in solving medium and large-scale instances, we perform numerical experiments on cases II-R1~II-R30. We conduct 100 repetitions for each set of cases to mitigate the impact of randomness on the results. The outcome of the experiments is presented in Table 5, since the time of R-ALL can be almost negligible, we did not calculate its gap with the heuristic.

Table 5. Heuristic, RL and R-ALL in the medium- and large-scale instances

ID	Heuristic		RL		R-ALL		Gap		
	Obj_h	$Time_h$	Obj_{rl}	$Time_{rl}$	Obj_{r-all}	$Time_{r-all}$	Δ_{rl}^{obj}	Δ_{rl}^{time}	Δ_{r-all}^{obj}
II-R1	156.54	0.01	147.75	0.00	192.79	0.00	-5.62%	-17.55%	23.15%
II-R2	200.49	0.01	190.05	0.01	240.66	0.00	-5.21%	-14.59%	20.04%
II-R3	238.17	0.03	225.10	0.03	287.00	0.00	-5.49%	-13.35%	20.50%
II-R4	281.46	0.05	265.95	0.05	332.56	0.00	-5.51%	-13.69%	18.16%
II-R5	323.83	0.09	302.75	0.07	385.09	0.00	-6.51%	-15.10%	18.92%
II-R6	362.29	0.13	337.80	0.11	430.24	0.00	-6.76%	-17.15%	18.76%
II-R7	299.06	0.03	269.45	0.02	386.57	0.00	-9.90%	-18.00%	29.26%
II-R8	339.03	0.07	308.65	0.05	428.60	0.00	-8.96%	-17.13%	26.42%
II-R9	373.40	0.13	343.55	0.10	479.55	0.00	-7.99%	-20.15%	28.43%
II-R10	411.29	0.22	379.65	0.18	528.76	0.00	-7.69%	-19.66%	28.56%
II-R11	465.63	0.35	430.30	0.30	592.42	0.00	-7.59%	-14.26%	27.23%
II-R12	511.49	0.54	471.45	0.43	645.87	0.00	-7.83%	-19.87%	26.27%
II-R13	497.69	0.10	432.65	0.07	682.22	0.00	-13.07%	-28.55%	37.08%
II-R14	510.14	0.24	451.20	0.18	695.61	0.00	-11.55%	-24.89%	36.36%
II-R15	549.11	0.45	489.15	0.34	747.48	0.00	-10.92%	-24.59%	36.12%
II-R16	584.26	0.77	519.45	0.56	800.16	0.00	-11.09%	-26.85%	36.95%
II-R17	639.43	1.19	571.30	0.89	861.18	0.00	-10.65%	-25.35%	34.68%
II-R18	685.71	1.70	623.70	1.35	927.79	0.00	-9.04%	-20.56%	35.30%
II-R19	784.71	0.35	634.05	0.21	1098.89	0.00	-19.20%	-38.54%	40.04%
II-R20	764.14	0.76	644.00	0.50	1068.12	0.00	-15.72%	-33.98%	39.78%
II-R21	768.83	1.37	656.30	0.93	1092.47	0.00	-14.64%	-32.11%	42.10%
II-R22	825.94	2.29	711.85	1.60	1155.19	0.00	-13.81%	-30.42%	39.86%
II-R23	865.71	3.50	748.34	2.44	1199.80	0.00	-13.56%	-30.42%	38.59%
II-R24	896.00	4.81	796.23	3.45	1287.95	0.00	-11.14%	-28.34%	43.74%

II-R25	1179.43	1.06	902.70	0.52	1705.75	0.00	-23.46%	-50.74%	44.63%
II-R26	1047.71	2.10	855.15	1.21	1529.15	0.00	-18.38%	-42.54%	45.95%
II-R27	1044.57	3.69	869.89	2.28	1555.35	0.00	-16.72%	-38.36%	48.90%
II-R28	1094.43	5.95	914.91	3.88	1617.75	0.00	-16.40%	-34.88%	47.82%
II-R29	1154.14	9.17	975.54	5.89	1694.15	0.00	-15.47%	-35.79%	46.79%
II-R30	1187.71	12.90	1000.26	8.70	1731.30	0.00	-15.78%	-32.57%	45.77%
Average	634.74	1.80	548.97	1.21	879.35	0.00	**-11.52%**	**-26.00%**	**34.21%**

$Note. \Delta_{rl}^{obj} = (Obj_{rl} - Obj_h)/Obj_h;\ \Delta_h^{time} = (Time_{rl} - Time_h)/Time_h;\ \Delta_{r-all}^{obj} = (Obj_{r-all} - Obj_h)/Obj_h.$

The reinforcement learning method is superior to the heuristic method for solving medium- and large-scale instances, with average solution times of 634.74 and 548.97, respectively. The improvements in solution quality and time averages 11.52% and 26.00%, respectively. Meanwhile, compared to other traditional single-deep RMFS methods, our proposed heuristic also outperform them by 34.21%.

These findings demonstrate that reinforcement learning methods are more suitable for tackling medium- and large-scale problems and can significantly outperform the proposed heuristic methods, as shown in Figure 13. We observe a clear trend as the problem size increased, with both the reinforcement learning and the heuristic methods demonstrating improved solution qualities and reduced solution time gaps, as depicted in Figure 14. The reinforcement learning method outperforms the heuristic approach by approximately 20% in terms of solution quality and 40% in terms of solution time, further highlighting the superiority of reinforcement learning methods in solving large-scale instances.

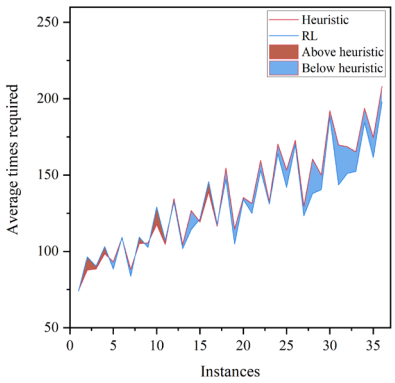


Figure 12. Comparison of RL, heuristic, and IP in small-scale instances

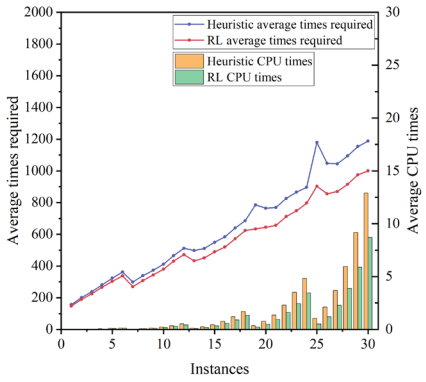


Figure 13. Comparison of heuristic and RL in medium- and large-scale instances

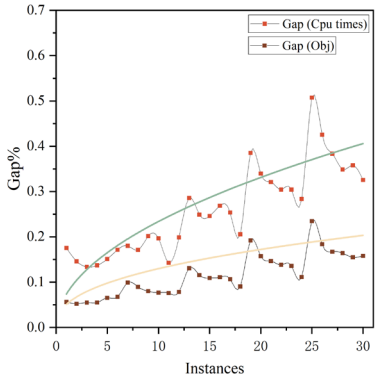


Figure 14. Gaps between heuristic and RL in Obj and CPU times

6.4 Sensitivity analyses on the reinforcement learning approach

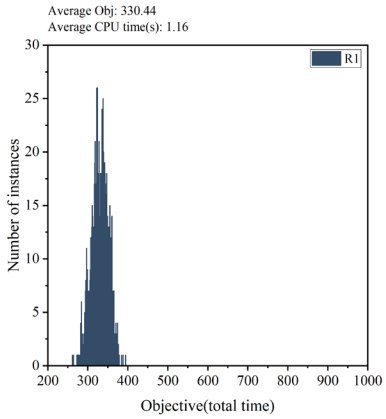
Managers sometimes need to estimate task completion times for a specific warehouse layout with a known number of target pods, robots, and picking stations. They also need to understand how changes in warehouse factors—such as the number of tasks and robots, empty position rate, and block layout $m * n$—impact system efficiency. To address these needs, we perform sensitivity analyses of the reinforcement learning approach using a series of five instances. Table 6 presents the basic information about these instances.

Table 6. Sensitivity analyses instances

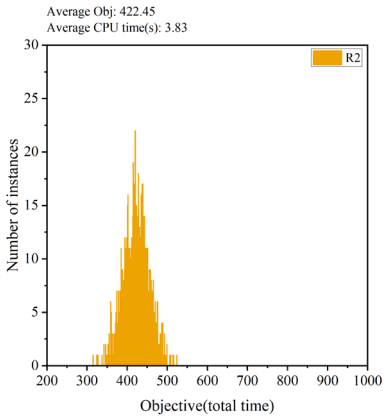
ID	$W * L \times m \times n$	R	O	REL	P
III -R1	2*8*3*3	5	30	0.2	3
III -R2	3*8*4*4	5	30	0.2	3
III -R3	4*8*5*5	5	30	0.2	3
III -R4	5*8*6*6	5	30	0.2	3
III -R5	6*8*7*7	5	30	0.2	3

6.4.1 RL results in different instance series

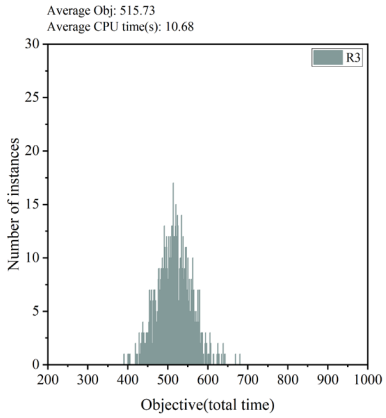
We initially perform 1000 repetitions of the experiment on the 5 selected instances to study the time distribution required for completing all tasks, as depicted in Figure 15. Generally, the time required for completing all tasks is moderate, with few instances of either too short or too long times for a given map. While the CPU time for the reinforcement learning method relies on the total number of tasks to be completed, which indicates the number of state transitions and attention model calls, it is still relatively short.



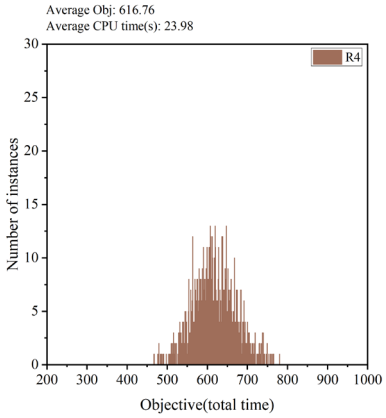
(a) III-R1 instance set



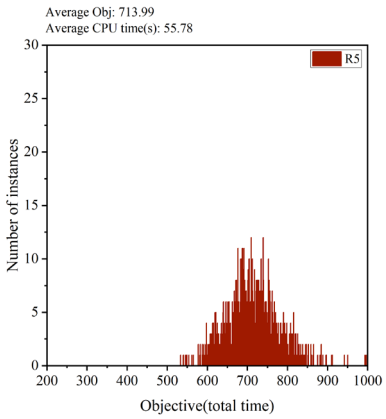
(b) III-R2 instance set



(c) III-R3 instance set



(d) III-R4 instance set



(d) III-R5 instance set

Figure 15. Results of RL with different grid sizes

6.4.2 Varying the targeted pod numbers

We conduct 100 repetitions of the experiment on the 5 instances to investigate the relationship between the number of targeted pods and the ATRs to complete all tasks. The results are summarized in Appendix D1, Table D1, and as shown in Figure 16, the relationship was nearly linear between the number of

target tasks and the ATRs for different warehouse layouts. Moreover, larger warehouses have faster ATRs increases.

6.4.3 Varying the number of robots

We conducted 100 repetitions of the experiment on the 5 instances to examine the relationship between the ATRs to complete all tasks and the number of robots, and the results are presented in Appendix D.2, Table D2, as shown in Figure 17. As the number of robots in the system increases, the time required to complete all tasks decreases exponentially and eventually converges to a constant value. Two possibilities explain this behavior: system efficiency has reached its maximum capacity, and system congestion becomes more severe as the number of robots increases, resulting in decreased efficiency.

6.4.4 Varying the ratio of empty locations

We perform 100 replicate trials on the 5 instances to investigate the relationship between the ATRs to complete all tasks and the empty position rate. The results are presented in Appendix D.3, Table D3 and Figure 18. As the empty position rate increases, the ATR to complete all tasks decreases gradually at different warehouse sizes and eventually plateaued. Larger arithmetic examples are more sensitive to changes in the empty position rate. This result is because systems with more empty positions have fewer obstructing pods to be moved, improving efficiency. However, an excessively high empty position rate does not further enhance system efficiency, and the empty position rate has an optimal value.

6.4.5 Varying the block length and width

We conduct 100 replicate trials on the 5 instances to observe the relationship between the ATRs to complete all tasks and the block length and width. The results are shown in Appendix D.4, Table D4 and illustrated in Figure 19. As the block length or width increases, the ATRs to complete all tasks increases dramatically. This result is because the block depth increases, and the robot must spend more time moving the obstructing pod, reducing system efficiency.

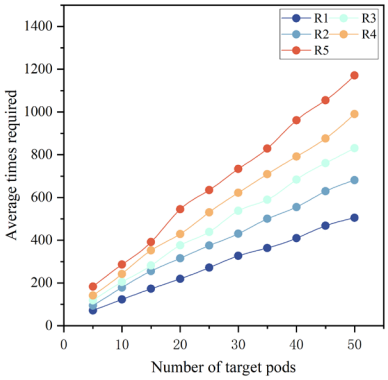


Figure 16. ATRs as the target pod numbers change

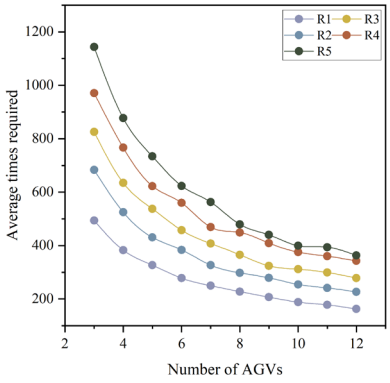


Figure 17. ATRs as the AGV numbers change

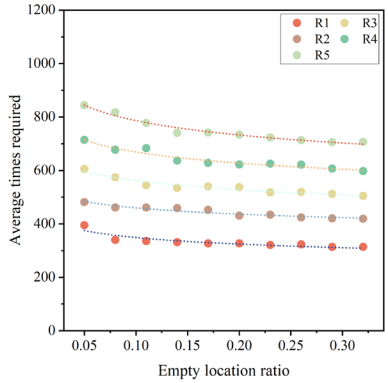


Figure 18. ATRs as the empty location ratios change

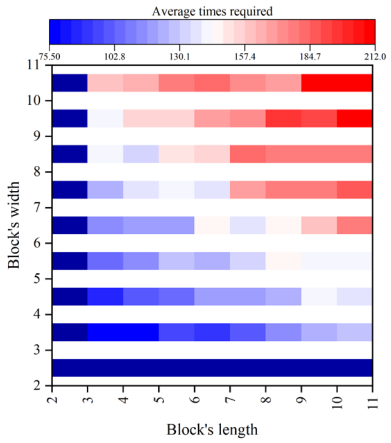


Figure 19. ATRs as the block's length and width changes

7. Conclusion and future research

In this study, we tackle the challenging problem of pod retrieval scheduling and robot dispatching in multi-deep compact RMFSs to increase operational efficiency in the context of high space utilization. To achieve this goal, we formulate this problem as an integer programming model and develop a simulation-based multi-strategy heuristic approach (used as baseline) and reinforcement learning methods to optimize pod retrieval scheduling and robot dispatching in multi-deep RMFSs.

Our extensive numerical experiments yield several noteworthy findings, outlined below. The proposed reinforcement learning methods exhibit superior performance to sovle the challenging pod retrieval scheduling and robot dispatching in multi-deep compact RMFSs, especially in the term of computational effeciency when solving medium- and large-scale instances. When the multi-deep compact RMFSs has a threshold number of robot configurations, adding robots does not significantly improve

efficiency, especially for small warehouses. The empty location ratio has a significant impact on efficiency. Large warehouses require a larger empty location ratio, but the efficiency gains become negligible when it exceeds 0.1. We suggest setting this value within the range of 0.05 to 0.1. We recommend increasing the length–width ratio of the block layout to balance space utilization and operational efficiency.

Our research can offer valuable insights into how multi-deep compact RMFSs can enhance operational efficiency while maintaining high space utilization and holds great potential for real-world applications in automated warehouses, providing high-quality, real-time, and implementable solutions for warehouse managers.

Although our approach has addressed pod retrieval scheduling in multi-deep compact RMFSs, it has a few limitations. The integer programming model is highly complex, making it challenging to find the best solution for medium and large instances. Additionally, different warehouse configurations require different system environments, so the reinforcement learning models need to be repeatedly trained, which is time consuming. Future research could focus on developing efficient algorithms to up model training and provide tighter bounds. Another potential avenue is to explore migratory-learning strategies to improve the generalization ability of reinforcement learning models.

References

Alfieri, A., Cantamessa, M., Monchiero, A., & Montagna, F. (2012). Heuristics for puzzle-based storage systems driven by a limited set of automated guided vehicles. *Journal of Intelligent Manufacturing*, *23*(5), 1695.

Azadeh, K., De Koster, R., & Roy, D. (2019). Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, *53*(4), 917-945.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Boysen, N., Briskorn, D., & Emde, S. (2017). Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research*, *262*(2), 550-562.

Boysen, N., de Koster, R., & Füßler, D. (2021). The forgotten sons: Warehousing systems for brick-and-mortar retail chains. *European Journal of Operational Research*, *288*(2), 361-381.

Boysen, N., De Koster, R., & Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, *277*(2), 396-411.

Bukchin, Y., & Raviv, T. (2022). A comprehensive toolbox for load retrieval in puzzle-based storage systems with simultaneous movements. *Transportation Research Part B: Methodological*, *166*, 348-373.

Cao, Z., Lin, C., Zhou, M., & Huang, R. (2018). Scheduling semiconductor testing facility by using cuckoo search algorithm with reinforcement learning and surrogate modeling. *IEEE Transactions on Automation Science and Engineering*, *16*(2), 825-837.

da Costa Barros, Í. R., & Nascimento, T. P. (2021). Robotic mobile fulfillment systems: A survey on recent developments and research opportunities. *Robotics and Autonomous Systems*, *137*, 103729.

Gharehgozli, A., & Zaerpour, N. (2020). Robot scheduling for pod retrieval in a robotic mobile fulfillment system. *Transportation Research Part E: Logistics and Transportation Review*, *142*, 102087.

Gong, Y., Jin, M., & Yuan, Z. (2021). Robotic mobile fulfilment systems considering customer classes. *International Journal of Production Research*, *59*(16), 5032-5049.

Gue, K. R., Furmans, K., Seibold, Z., & Uludağ, O. (2013). GridStore: a puzzle-based storage system with decentralized control. *IEEE Transactions on Automation Science and Engineering*, *11*(2), 429-438.

Gue, K. R., & Kim, B. S. (2007). Puzzle‐based storage systems. *Naval Research Logistics (NRL)*, *54*(5), 556-567.

He, J., Liu, X., Duan, Q., Chan, W. K. V., & Qi, M. (2023). Reinforcement learning for multi-item retrieval in the puzzle-based storage system. *European Journal of Operational Research*, *305*(2), 820-837.

Hopfield, J. J., & Tank, D. W. (1985). "Neural" computation of decisions in optimization problems. *Biological cybernetics*, *52*(3), 141-152.

Ji, T., Zhang, K., & Dong, Y. (2020). Model-based Optimization of Pod Point Matching Decision in Robotic Mobile Fulfillment System. 2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA),

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lamballais, T., Roy, D., & De Koster, M. (2017). Estimating performance in a robotic mobile fulfillment system. *European Journal of Operational Research*, *256*(3), 976-990.

Li, X., Yang, X., Zhang, C., & Qi, M. (2021). A simulation study on the robotic mobile fulfillment system in high-density storage warehouses. *Simulation Modelling Practice and Theory*, *112*, 102366.

MA, Y., CHEN, H., & YU, Y. (2022). An efficient heuristic for minimizing the number of moves for the retrieval of a single item in a puzzle-based storage system with multiple escorts. *European Journal of Operational Research*, *301*(1), 51-66.

Merschformann, M., Lamballais, T., De Koster, M., & Suhl, L. (2019). Decision rules for robotic mobile fulfillment systems. *Operations Research Perspectives*, *6*, 100128.

Mirzaei, M., De Koster, R. B., & Zaerpour, N. (2017). Modelling load retrievals in puzzle-based storage systems. *International Journal of Production Research*, *55*(21), 6423-6435.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Ostrovski, G. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533.

Müller, M., Ulrich, J. H., Reggelin, T., Lang, S., & Reyes-Rubiano, L. S. (2020). Comparison of deadlock handling strategies for different warehouse layouts with an AGVS. 2020 Winter Simulation

Conference (WSC),
Nazari, M., Oroojlooy, A., Snyder, L., & Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, *31*.
Rohit, K. V., Taylor, G. D., & Gue, K. R. (2010). Retrieval time performance in puzzle-based storage systems. IIE Annual Conference. Proceedings,
Roy, D., Nigam, S., de Koster, R., Adan, I., & Resing, J. (2019). Robot-storage zone assignment strategies in mobile fulfillment systems. *Transportation Research Part E: Logistics and Transportation Review*, *122*, 119-142.
Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.
Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., . . . Bolton, A. (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354-359.
Smith, K. A. (1999). Neural networks for combinatorial optimization: a review of more than a decade of research. *Informs journal on Computing*, *11*(1), 15-34.
Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. Proceedings of the AAAI conference on artificial intelligence,
Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. *Advances in neural information processing systems*, *28*.
Wang, Z., Feng, C., Muruganandam, R., Ang, W. T., Tan, S. Y. M., & Latt, W. T. (2016). Three-dimensional cell rotation with fluidic flow-controlled cell manipulating device. *IEEE/ASME Transactions on Mechatronics*, *21*(4), 1995-2003.
Weidinger, F., Boysen, N., & Briskorn, D. (2018). Storage assignment with rack-moving mobile robots in KIVA warehouses. *Transportation Science*, *52*(6), 1479-1495.
Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, 5-32.
Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, *29*(1), 9-9.
Yalcin, A., Koberstein, A., & Schocke, K.-O. (2019). An optimal and a heuristic algorithm for the single-item retrieval problem in puzzle-based storage systems with multiple escorts. *International Journal of Production Research*, *57*(1), 143-165.
Yang, P., Jin, G., & Duan, G. (2022). Modelling and analysis for multi-deep compact robotic mobile fulfilment system. *International Journal of Production Research*, *60*(15), 4727-4742.
Yu, H., Yu, Y., & de Koster, R. (2022). Dense and fast: Achieving shortest unimpeded retrieval with a minimum number of empty cells in puzzle-based storage systems. *IISE Transactions*, *55*(2), 156-171.
Yuan, Z., & Gong, Y. Y. (2017). Bot-in-time delivery for robotic mobile fulfillment systems. *IEEE Transactions on Engineering Management*, *64*(1), 83-93.
Zhen, L., & Li, H. (2022). A literature review of smart warehouse operations management. *Frontiers of Engineering Management*, *9*(1), 31-55.
Zhuang, Y., Zhou, Y., Yuan, Y., Hu, X., & Hassini, E. (2022). Order picking optimization with rack-moving mobile robots and multiple workstations. *European Journal of Operational Research*, *300*(2), 527-544.

Zou, B., Xu, X., & De Koster, R. (2018). Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system. *European Journal of Operational Research*, *267*(2), 733-753.

Zou, Y., & Qi, M. (2021). A heuristic method for load retrievals route programming in puzzle-based storage systems. *arXiv preprint arXiv:2102.09274*.