Deep Reinforcement Learning for the Real-Time Inventory Rack Storage Assignment and Replenishment Problem

Sander Teck[a,*], Tú San Phạm[b], Louis-Martin Rousseau[b], and Pieter Vansteenwegen[a]

[a]*KU Leuven Institute for Mobility – CIB, KU Leuven, Celestijnenlaan 300, 3001 Leuven, Belgium*
[b]*Department of Mathematics and Industrial Engineering, Polytechnique Montreal, Canada*

Abstract

The e-commerce industry is quickly transforming towards more automation and technological advancements. With the growing intricacy of warehouse operations, there is a need for control systems that can efficiently handle this complexity. This study considers a Robotic Mobile Fulfillment System (RMFS), a semi-automated warehousing system. This system employs autonomous mobile robots (AMRs) to retrieve inventory racks from the storage area; this way, human activity is eliminated within the storage area itself. The fleet of robots both store and retrieve the inventory racks to either workstations, where human pickers are stationed that pick items from the racks, or replenishment stations, where depleted inventory racks can be restocked with items. An attractive characteristic of the RMFS is that it dynamically changes the positioning of the inventory racks based on the frequency of inventory rack requests and the state of their stock levels. The optimization objective considered in this study for the dynamic positioning problem of the racks within the storage area is to minimize the average cycle time of the mobile robots to perform retrieval and replenishment activities. We propose a deep reinforcement learning approach to train a decision-making agent to learn a policy for the storage assignment and replenishment of inventory racks. The learned policy is compared to the commonly used decision rules in the academic literature on this problem. The experimental results show the potential benefits of training an agent to learn a storage and replenishment policy. Cycle time improvements up to 5.4% can be achieved over the best-performing decision rules. This research contributes to advancing the understanding of intelligent storage assignment and replenishment strategies for the real-time decision-making process within an RMFS.

Keywords: Robotic Mobile Fulfillment System, E-commerce, Storage policy, Real-Time Decision-making, Reinforcement Learning

1. Introduction

Warehouses are vital for sustainable e-commerce development, as they are a critical component of modern supply chain operations. They serve as a central hub for companies to store, ship, and distribute goods. Efficient warehouses offer several benefits, including tracking and managing a company's inventory. Moreover, they can significantly reduce transportation costs and improve operational flexibility. The ongoing trend toward automation in warehousing environments is accelerating, and

while this automation enhances efficiency, it also introduces more operational complexity within such environments (Azadeh et al., 2019; Boysen et al., 2019; Boysen et al., 2022). Consequently, there is a growing need to address and effectively manage this increasing complexity. This underscores the importance of developing more efficient and cost-effective scheduling policies. In essence, warehouse scheduling concerns assigning limited resources to optimize the storage and distribution of goods. The efficient management of goods within the storage areas of a warehouse results in a range of advantages, including reduced lead times, improved space utilization, and reduced travel times. Moreover, automated systems in warehousing offer inherent advantages, such as scalability and flexibility. They can adapt to fluctuations in order volumes by reconfiguring themselves as needed. For instance, dynamically changing the storage layout, increasing or reducing the number of autonomous mobile robots (AMRs) used in the storage area, and more. In today's context, this scalability and flexibility are essential in meeting stricter customer expectations (e.g., same- or next-day delivery).

The Robotic Mobile Fulfillment System (RMFS) scheduling problem considered in this work refers to the challenge of coordinating the movement and tasks of both robots and human pickers in an automated warehouse setting to fulfill customer orders efficiently. A well-designed scheduling system can improve efficiency, reduce costs, and enhance customer satisfaction. The overall objective is to minimize the average cycle time of the AMRs, which also leads to the minimization of the operational expenses (i.e., wages) and the capital expenses (i.e., equipment costs, maintenance costs, etc.). To maintain its efficiency in dynamic environments, the system has to perform well in the following aspects:

1. **Flexibility and adaptability:** The system must adjust to fluctuating demands and unexpected disruptions in real time.
2. **Scalability:** The system should be able to handle growing order volumes without compromising on efficiency.
3. **Real-time monitoring and control:** Effective monitoring and control of the system are essential to ensure its performance and maintain high quality.

The prevailing studies predominantly employ simple decision rules for assigning storage locations to return inventory racks and managing their replenishment. However, these approaches prioritize returning the rack as quickly as possible, often by minimizing distance traveled, while neglecting other considerations such as queuing times and distinctions between fast- and slow-moving stock. This focus on immediate gains reveals a short-sightedness that makes decision rules too rigid for dynamic and complex warehouse environments. The study by Rimélé et al. (2021) looked into learning a storage policy. Nevertheless, their investigation was confined to small problem instances, encompassing only 36 storage locations and one workstation. Moreover, they did not include a scattered storage policy. Consequently, the intricacies associated with scalability were not addressed, limiting the

generalizability of their findings. Additionally, they did not consider the replenishment decisions, which can significantly impact the overall cycle time. Therefore, we propose a novel approach employing a deep reinforcement learning (DRL) approach to address the storage location assignment and replenishment problem. Our contributions are as follows:

1. **Integrated problem formulation:** Unlike prior studies that address replenishment and storage location assignment in isolation, we integrate both into a joint decision problem. We formulate a novel mixed-integer linear programming model to formally define the problem, which also serves as a baseline for evaluating alternative solution approaches. Additionally, we introduce a DRL model for large-scale instances to tackle these as a joint decision problem.
2. **Real-time decision making under uncertainty:** The DRL model effectively handles uncertainty within the decision-making process, enabling the agent to dynamically adapt to unknown, incoming order requests in real time. Unlike traditional methods that focus on short-term optimization, the proposed approach accounts for how current decisions impact future outcomes, fostering a more long-term perspective.
3. **Custom reward function:** Balancing replenishment frequency with minimized cycle times presents a novel challenge. The reward function is specifically designed to manage this trade-off, guiding the agent to learn a policy that minimizes both the number of replenishments and average cycle times. This is non-trivial, as replenishment introduces distinct cycle time considerations, such as queue and processing times.
4. **Efficient handling of high dimensionality in state and action space:** The RMFS environment involves high-dimensional state and action spaces, given the large number of inventory racks, AMRs, and storage locations. To scale the model effectively, we applied feature engineering, clustered storage zones, and streamlined actions choices (e.g. action masking) to reduce complexity without sacrificing decision quality.
5. **Use of explainable AI:** By decoding feature representations for SHAP value analysis, we enhance the interpretability of DRL policies within the RMFS context, providing valuable insights for the community.

The remainder of the paper is structured as follows. In the following section, the related research concerning the operational scheduling problems in an RMFS is reviewed, emphasizing the inventory rack storage assignment and replenishment problems. Section 3 provides a brief problem description of the operational problem of interest and Section 4 presents the mathematical model for the joint optimization problem. Next, Section 5 presents the proposed deep reinforcement framework. Thereafter, Section 6 elaborates on the performance of the proposed framework compared to the typically used storage policies. Finally, in Section 7, the conclusions and the directions for future research are discussed.

2. Related literature

As stated in Enright & Wurman (2011), the RMFS scheduling problem on the operational level consists of various interesting assignment problems: inventory rack selection, inventory rack storage assignment, pick order assignment, replenishment assignment, and robot assignment problems. Table 1 presents an overview of the existing literature.

Table 1. *Literature overview of relevant assignments problems in RMFS.*

Publications	Pick order assignment	Inventory rack selection	Replenishment assignment	Task assignment	Inventory rack storage assignment
Boysen et al. (2017)	✓				
Zou et al. (2017)				✓	
Weidinger et al. (2018b)					✓
Krenzler et al. (2018)					✓
Merschformann et al. (2019)	✓	✓	✓		✓
Yuan et al. (2019)					✓
Roy et al. (2019)			✓	✓	
Gharehgozli & Zaerpour (2020)				✓	
Lamballais et al. (2020)			✓		
Valle & Beasley (2021)	✓				
Xie et al. (2021)	✓	✓			
Rimélé et al. (2021)					✓
Yang et al. (2021)	✓	✓			
Teck et al. (2022)	✓	✓		✓	
Zhuang et al. (2022a)	✓	✓			
Zhuang et al. (2022b)				✓	✓
Zhang et al. (2022)	✓			✓	
Lamballais et al. (2022)	✓		✓	✓	
Lu et al. (2023)				✓	
Jiao et al. (2023)	✓	✓			
Cheng et al. (2024)	✓				
Li et al. (2024)				✓	
Zhou et al. (2024)				✓	
This study			✓		✓

From the literature overview in Table 1, we focus on the central topic of this paper, the inventory rack storage assignment and replenishment problem. Weidinger et al. (2018b) present a mixed integer programming (MIP) model for interval scheduling, which operates under the assumption of predetermined rack visits and uses a surrogate objective of minimizing the total loaded travel distance for robots, albeit without explicitly accounting for robot availability. They introduce an Adaptive Large Matheuristic Search for this model and compare it with various simple rule-based storage policies, concluding their method's superiority in achieving the surrogate objective. Krenzler et al. (2018) developed a deterministic model for inventory rack assignment, comparing it with various storage policies similar to those in Weidinger et al. (2018b), as well as with genetic algorithms and binary

integer programming approaches. Their testing, limited to single-workstation RMFS scenarios and small-to-medium-sized instances, shows that their model performs well under simplified conditions. Nevertheless, they acknowledge that their model does not extend efficiently to complex, multi-workstation environments. Additionally, their approach is based on a long-term planning horizon, limiting their flexibility to adapt to dynamic changes within the environment.

Merschformann et al. (2019) adopt a different methodology, conducting a simulation-based analysis to study the pick and replenishment processes in RMFS. They explore several rule-based assignment policies for the pick order assignment, inventory rack selection, replenishment assignment, and inventory rack storage assignment problems. The study examines random, fixed, nearest, station-based, and class-based storage location rules for storage location assignment. Their findings suggest that the storage assignment decision rules had a fairly low impact on the overall system performance. However, they note that this may be due to the limited scale of their test warehouse layout. They expect that the impact of storage location policies on travel distance could increase significantly in larger warehouse layouts, signaling that more sophisticated strategies could yield greater benefits. A concept that our work investigates.

Zhuang et al. (2022b) investigate inventory rack storage and task assignment through a matheuristic decomposition method using a rolling horizon framework combined with simulated annealing. Their study benchmarks the proposed framework against common policies like the nearest and shortest leg storage, affirming that the shortest leg policy is among the most effective in minimizing the idle time for human pickers and reducing system makespan. Yuan et al. (2019) focus on velocity-based storage policies and use a fluid model, validated through simulation, to analyze their effectiveness. Their findings suggest that class-based storage policies, particularly those with two or three classes, significantly reduce robot travel distances compared to random policies, with reductions up to 8% and 10%, respectively. Ding et al. (2024) also explore velocity-based assignment methods to minimize total rack travel distances. Beyond comparing their approach to the commonly used decision rules, they conclude that SKU-to-rack assignments, typically a tactical decision, significantly impact overall system performance, as correlations between SKUs can influence retrieval efficiency.

In most studies focusing on the RMFS problem, the inventory stock is assumed to be inexhaustible. Hence, replenishment of the inventory is unnecessary. However, this is an important concern in managing the storage area efficiently. The studies that do consider replenishment typically only apply simple decision rules. For instance, Lamballais et al. (2020) propose a Semi-Open Queueing Network to study the optimization of three key decision variables: (1) the number of inventory racks per stock-keeping unit (SKU), (2) the ratio of picking stations and replenishment stations, (3) the replenishment level per inventory rack. They conclude that spreading the SKUs (scattered storage policy) over multiple

racks can substantially improve the throughput performance. Moreover, they found that replenishing the racks before they were empty positively impacted the throughput performance. They impose an inventory replenishment level, a fraction of the maximum number of products per SKU on a rack (e.g. 30-50% stock depletion). This level indicates when an inventory rack should visit a replenishment station. The study of Lamballais et al. (2022) considers a queueing network integrated within a Markov decision process (MDP) for the RMFS pick and replenishment processes. Here, the replenishment of an inventory rack occurs after every retrieval cycle with a certain probability.

Reinforcement learning has increasingly gained attention as an effective method for addressing complex optimization problems, such as production scheduling under processing time uncertainty (Shi et al., 2020), job shop scheduling (Gabel et al., (2012); Luo (2020); Tassel et al., (2020)), and order batch scheduling in warehouses (Cals et al., (2021), among others. Recently, its application within RMFS optimization has seen notable growth. For instance, Cheng et al. (2024) utilize deep reinforcement learning to minimize processing costs in batch order scheduling, where mobile robots are dynamically assigned to workstations and orders. To improve efficiency, they substitute traditional actions with heuristic-based rules in the action space, subsequently validating this approach through simulations. Li et al. (2024) also explore reinforcement-based multi-robot task assignment and routing methods by developing a mixed-integer linear programming (MILP) model with valid inequalities and a reinforcement learning-based hyper-heuristic framework. Their framework operates on two levels, leveraging high-level and low-level heuristics. They demonstrate that including valid inequalities provides stronger bounds and enhances the model's efficiency. Similarly, Zhou et al. (2024) use an attention-based network model to optimize multi-robot scheduling with pod repositioning, incorporating problem-specific knowledge through a masking strategy that enhances the learning process for more efficient allocation and retrieval policies. The study of Rimélé et al. (2021) develops a Deep Q-learning agent with Reinforcement Learning (RL) to learn a dynamic storage policy and model their system as a Partially Observable Markov Decision Process (POMDP). They compare their method to the traditionally used storage policies in the industry on small-scale instances with only one workstation. Their results show that the proposed method, in most cases, does slightly better or is comparable to the shortest-leg policy. However, the limitation of this study is that it works with unrealistically small instances. Furthermore, many simplifying assumptions make it less relevant for practical applications (e.g. inexhaustible stock, no scattered storage policy, etc.). Because there is only one workstation, the gains over a greedy heuristic like the shortest leg policy are limited. This observation is also supported by the study of Merschformann et al. (2019).

For the above reasons, this study will focus on large-scale instances with multiple workstations. Furthermore, we also consider the state of the inventory rack and whether it is close to needing replenishment by a replenishment station. Thus, the objective of our learning agent is to select an

available storage location where the mobile robots can store the inventory rack and determine if these racks require replenishment of the stock they contain. Doing this efficiently should prevent stock-outs and result in better picking efficiency. Our learning agent is trained with deep reinforcement learning techniques, which will be discussed in more detail in the following sections.

3. The dynamic inventory rack storage assignment and replenishment problem

This study focuses on a semi-automated warehousing system called a Robotic Mobile Fulfillment System (RMFS), commonly called a rack-moving mobile robot-based warehouse. Kiva Systems introduced the RMFS in 2006, later becoming Amazon Robotics, when Amazon acquired the company in 2012 (Enright and Wurman, 2011). This system involves a rack-moving mobile robot to retrieve and store the requested inventory racks in the storage area, to workstations and replenishment stations (see Figure 1). Once unloaded, these robots can pass under stored shelves, significantly reducing travel time and traffic density in the aisles. Hence, human activity within the storage area is eliminated. Presenting inventory racks at the workstations allows for efficient picking since it eliminates unproductive human travel time. The human pickers at the workstations pick the correct items from the presented inventory racks and place them in the corresponding order totes to fulfill the retrieval requests. At the replenishment stations, items can be restocked to replenish the depleted racks during operation.

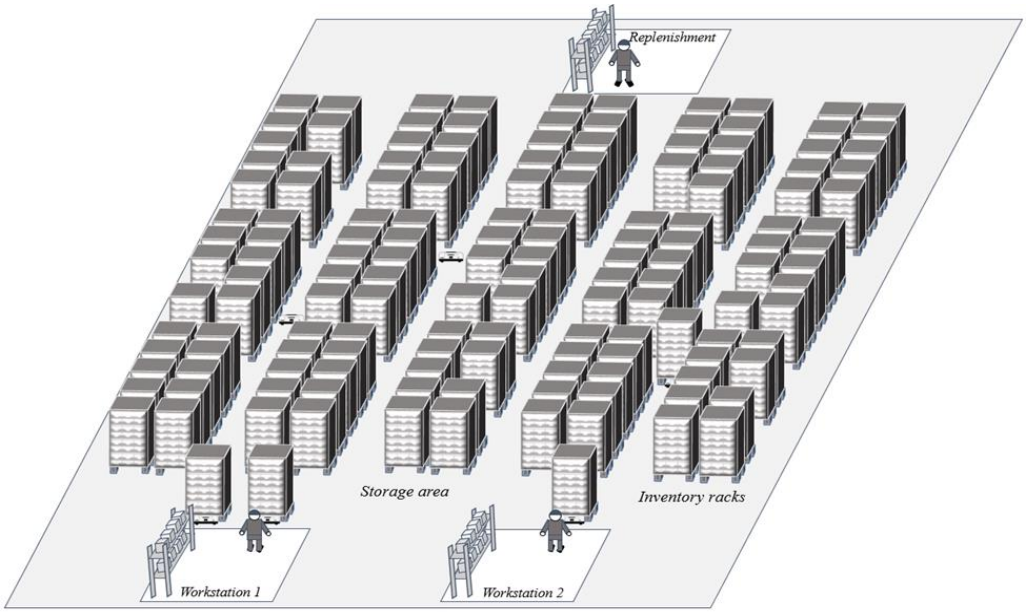

Figure 1: *A schematic overview of a RMFS layout.*

This paper focuses on the problem of learning a policy for the dynamic relocation of inventory racks within the storage area and the replenishment of these inventory racks. Each time a robot visits a workstation, after fulfilling its pick requests, a decision agent will make a real-time decision on whether to replenish or store the inventory rack at a free storage location within the storage area. The overall

objective is to minimize the robots' average cycle time, discussed in the following paragraph, as this impacts the makespan of the system and the total travel durations of the mobile robots.

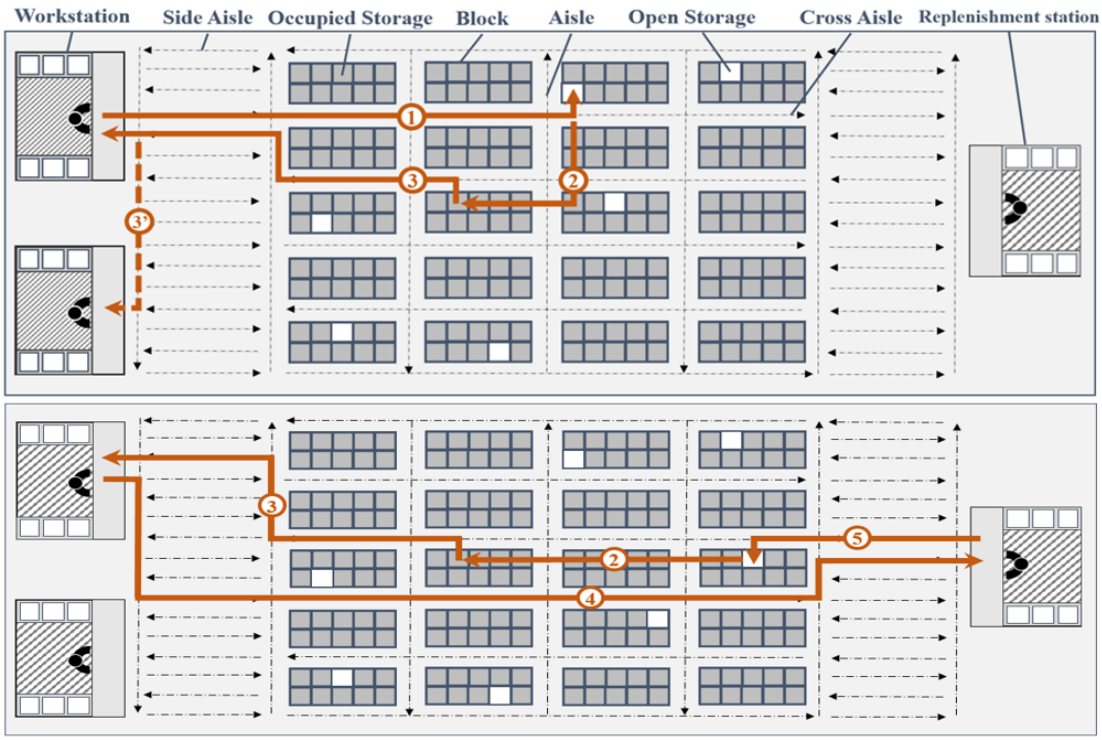

Figure 2: *(a) A retrieval cycle of a mobile robot. (b) A replenishment cycle of a mobile robot.*

Given a set of stock-keeping units (SKUs) $K = \{1, \dots, k, \dots, |K|\}$ and a set of customer orders $O = \{1, \dots, o, \dots, |O|\}$ that need to be fulfilled, each customer order o can consist of one or more order lines, with each line requiring a unique SKU k. Moreover, each order line can require different quantities to be fulfilled. Order-picking fulfills these customer orders with rack retrieval from the storage area. Figure 2 (a) illustrates the rack retrieval cycle in an RMFS; it consists of several steps. Mobile robots retrieve inventory racks from the storage area to fulfill a customer order and transport them to a workstation. Whenever a mobile robot has completed a retrieval task, typically going from a workstation back to an open storage location (step 1), it remains there at the dwell location until a new task is assigned. Whenever this robot is assigned a new retrieval task, it moves from its current dwell location to the storage location of the requested inventory rack (step 2). Thereafter, it can lift the storage rack and transport the requested rack to the correct workstation (step 3). We define this as the normal retrieval cycle. However, in some cases, the requested inventory rack is required by multiple workstations. In such a case, the robot can directly bring the inventory rack from one workstation to the other (step 3') before storing it away again in an unoccupied storage location in the storage area (step 1). Figure 2 (b) illustrates an inventory rack replenishment cycle in an RMFS. To replenish an inventory rack, after it has been retrieved from the storage area and has served a workstation (step 3), the mobile robots can bring the depleted inventory racks straight from the workstation to the replenishment station (step 4).

Here, human workers replenish the depleted stock from the inventory rack. Thereafter, the mobile robot stores the inventory rack in the storage area in a suitable unoccupied storage space (step 5). Following this, the mobile robot dwells at this location until it is assigned a new task (rack retrieval or rack replenishment).

Other than traditional warehouses, where inventory racks are fixed and stationary, the inventory rack assignment problem can dynamically change the storage area of the RMFS. In doing so, inventory racks can be strategically stored in the warehouse based on the frequency of its retrieval requests and the inventory levels of the rack itself. For instance, if the items on the rack are close to being depleted, storing them preemptively closer to the replenishment station might be interesting. The overall objective here is to minimize the average cycle time of the mobile robots.

The main assumptions are the following:

I. Robot breakdown does not occur, and battery management is not included.
II. Robots travel at a constant velocity, and processing times are deterministic.
III. Congestion and potential conflicts of mobile robots in the aisles are ignored, as unidirectional lanes are implemented to significantly reduce the likelihood of collisions and congestion. Unidirectional lanes are commonly used by major warehouse operators such as JD, Amazon, Swisslog, and Scallog (Duan et al., 2021, Lamballais et al., 2017, Xie et al., 2021, Zou et al., 2017).
IV. The mobile robots will dwell at the location where they returned their last rack until a new task is assigned.
V. Robots follow the shortest path, and unloaded robots can travel underneath the inventory racks in the storage area.
VI. The mobile robots are pooled over all the workstations and replenishment station(s).
VII. A scattered storage policy distributes the items over the inventory racks (Weidinger & Boysen, 2018a).
VIII. The maximum number of units per SKU in an inventory rack is predetermined and fixed.

4. Mathematical Model

To evaluate the performance of the heuristic and provide a baseline for comparison, we introduce a mixed-integer linear formulation for the deterministic rack storage location assignment and replenishment problem. This model assumes full information, where the order assignment to workstations, the inventory rack selection for order fulfillment, and multi-robot task assignment are predetermined. The proposed model makes decisions on where to store the inventory racks after they have fulfilled the orders at the workstations and when they should be replenished. It is assumed for this model that an inventory rack can only be requested once in the given time period. Moreover, initially each robot is queueing at a workstation with an inventory rack and predetermined fulfillment sequence. Thereafter, the sequence in which inventory racks are fulfilled is dependent on their arrival time at the workstations.

The nomenclature for the variables and parameters is as follows:

	Sets
W	Set of workstations active in the warehouse environment. $W = \{0, \dots, w, \dots, \lvert W\rvert - 1\}$
A	Set of autonomous mobile robots active in the storage area. $A = \{0, \dots, a, \dots, \lvert A\rvert - 1\}$
R	Set of replenishment stations active in the warehouse environment. $R = \{0, \dots, r, \dots, \lvert R\rvert - 1\}$
L^o	Set of open storage locations in the storage area at time event 0. $L^o = \{0, \dots, l, \dots, \lvert L^o\rvert - 1\}$
P	Set of requested inventory racks to fulfill orders at the workstations. $P = \{0, \dots, p, \dots, \lvert P\rvert - 1\}$
P^a	Set of requested inventory racks to be retrieved by robot $a \in A$.
K^p	Set of SKUs stored in inventory rack $p \in P$.
L	Set of potential storage locations in the storage area. $L = P \cup L^o$
T	Set of decision event slots. $T = \{1, \dots, \lvert P\rvert\}$
T^0	Set of the initialization slot and the subsequent decision event slots. $T^0 = \{0, \dots, \lvert T\rvert\}$
	Parameters
SL_{lp}	Initial storage availability of storage location $l \in L$ given a pod $p \in P$
PL_p	The number of picks that have to be performed to complete the order lines by pod $p \in P$
PT	Processing time of a human operator at the workstations to pick an order line
AT	Time to either unload or load an AMR with an inventory rack in the storage area
RT	Replenishment time of a human operator at the replenishment stations to restock an SKU
MS	Max stock level
LT	Threshold stock level below which replenishment is needed
C_{wr}	Travel time from workstation $w \in W$ to replenishment station $r \in R$
$C_{wlpw'}$	Travel time from workstation $w \in W$ to storage location $l \in L$, to storage location of pod $p \in P$, to workstation $w' \in W$
$C_{wrlpw'}$	Travel time from workstation $w \in W$ to replenishment station $r \in R$, to storage location $l \in L$, to storage location of pod $p \in P$, to workstation $w' \in W$
	Variables
x_{pwl}^t	Binary variable that equals 1 if inventory rack $p \in P$ is assigned to storage location $l \in L$ after completing work at workstation w∈W at time $t \in T^0$
r_{pr}^t	Binary variable that equals 1 if inventory rack $p \in P$ is assigned to a replenishment station $r \in R$ at decision event $t \in T^0$
	Auxiliary variables
y_{pl}^t	Binary variable that equals 1 if inventory rack $p \in P$ is occupying storage location $l \in L$ at decision event $t \in T^0$
w_{pw}^t	Cycle time of completing a retrieval task for inventory rack $p \in P$ at decision event $t \in T^0$ on workstation $w \in W$
z_{pw}^t	Time at which the picking of goods from inventory rack $p \in P$ is completed and the inventory rack is ready to leave the workstation $w \in W$ at decision event $t \in T^0$
u_{pwl}^t	Binary variable that aids in vacating inventory rack $p \in P$ on storage location $l \in L$ at event $t \in T^0$ as it is requested by workstation $w \in W$
v_{pk}^t	Binary variable that aids in restocking SKU $k \in K^p$ on inventory rack $p \in P$ at event $t \in T^0$
b_r^t	Tracks the time of the last completed replenishment at station $r \in R$ at event $t \in T$
s_{pk}^t	Stock level of inventory rack $p \in P$ for a given SKU $k \in K^p$ at event $t \in T$
q_{pwl}^t	Queueing time at the workstation $w \in W$ for inventory rack $p \in P$ at event $t \in T$

q_{pr}^{t} Queueing time at replenishment station $r \in R$ for inventory rack $p \in P$ at event $t \in T$

After defining the necessary sets, parameters, and variables, the mathematical formulation that incorporates these elements to derive the solution framework is presented below.

$$Min \quad \sum_{t \epsilon T} \sum_{p \epsilon P} \sum_{w \epsilon W} w_{pw}^{t} / (|T|\text{-}|A|) \tag{1}$$

Subject to

$$y_{pl}^{0} = SL_{lp} \qquad \forall l \in L, p \in P \tag{2}$$

$$\sum_{p \epsilon P} y_{pl}^{t} \leq 1 \qquad \forall t \in T, l \in L \tag{3}$$

$$\sum_{p \epsilon P} \sum_{w \epsilon W} \sum_{l \epsilon L} x_{pwl}^{t} = 1 \qquad \forall t \in T \tag{4}$$

$$\sum_{w \epsilon W} \sum_{l \epsilon L} \sum_{t \epsilon T} x_{pwl}^{t} = 1 \qquad \forall a \in A, p \in P^{a} \tag{5}$$

The objective (1) of the study is to minimize the cycle time of AMRs as they execute retrieval and storage tasks efficiently. The first set of constraints initializes the system and defines its general operations. Constraint (2) sets the initial state of storage location states, indicating whether each is occupied or vacant. Constraint (3) ensures that no more than one inventory rack can occupy a storage location at any given time. Constraint (4) enforces the requirement that each decision event must include a task assignment. Additionally, constraint (5) guarantees that all inventory rack requests are completed as part of the AMR operations.

$$\sum_{w \epsilon W} x_{pwl}^{t} \leq 1 - \sum_{p' \epsilon P} y_{p'l}^{t-1} \qquad \forall t \in T, p \in P, l \in L \tag{6}$$

$$y_{pl}^{t-1} + \sum_{w \epsilon W} x_{pwl}^{t} - u_{pl}^{t} = y_{pl}^{t} \qquad \forall t \in T, p \in P, l \in L \tag{7}$$

$$y_{pl}^{t} \geq \sum_{w \epsilon W} x_{pwl}^{t} \qquad \forall t \in T, p \in P, l \in L \tag{8}$$

$$\sum_{w \epsilon W} \sum_{l \epsilon L} x_{p-1wl}^{t} \geq \sum_{l \epsilon L} u_{pwl}^{t} \qquad \forall t \in T, a \in A, p \in \{1,\ldots,|P^{a}|\} \tag{9}$$

$$\sum_{p \epsilon P} \sum_{l \epsilon L} u_{pwl}^{t} \leq 1 \qquad \forall t \in T \tag{10}$$

$$\sum_{p \epsilon P} \sum_{l \epsilon L} u_{pwl}^{t} \leq 1 - \sum_{w \epsilon W} \sum_{l \epsilon L} x_{pwl}^{t} \qquad \forall a \in A, p \in P^{a} \tag{11}$$

$$\sum_{w \epsilon W} x_{p-1wl}^{t} + y_{pl}^{t-1} \leq 1 \qquad \forall t \in T, a \in A, p \in \{1,\ldots,|P^{a}|\}, l \in L \tag{12}$$

$$\sum_{l \epsilon L} y_{pl}^{t} \leq 1 - \sum_{w \epsilon W} \sum_{l \epsilon L} x_{p-1wl}^{t} \qquad \forall t \in T, a \in A, p \in \{1,\ldots,|P^{a}|\} \tag{13}$$

The next group of constraints governs the dynamic updates of storage locations throughout decision events. Constraints (6) prohibits the assignment of an inventory rack to a storage location unless that location is vacant. Constraints (7), (8), and (9) collectively update the storage location states. When an inventory rack is assigned to a location, that location transitions to being occupied, while the location for the subsequent rack in the AMRs sequence is marked as vacant. Constraint (10) limits the system to vacating only one location during a decision event, and constraint (11) prevents any location from being vacated after all tasks in the AMRs sequence have been completed. Constraint (12) ensures that an

inventory rack cannot be assigned to the storage location of the subsequent rack in the AMR's task sequence. Constraint (13) confirms that the storage location for the subsequent rack is indeed vacant after the preceding rack has been assigned a location.

$$z_{pw}^{t} \geq z_{p-1w\prime}^{t'} + PT + w_{pw}^{t} - M(2 - \sum_{l\epsilon L} x_{pwl}^{t} - \sum_{l'\epsilon L} x_{p-1w'l'}^{t'})$$
$$\forall t,t' \epsilon T,\ a \epsilon A,\ p \epsilon \{1,\ldots,|P^a|\}, w,w' \epsilon W \quad (14)$$

$$z_{pw}^{t} \geq z_{p\prime w}^{t'} + PT - M(2 - \sum_{l\epsilon L} x_{pwl}^{t} - \sum_{l'\epsilon L} x_{p-1w'l'}^{t'}) \quad \forall t \epsilon T,\ t' \epsilon \{1,\ldots,t\},\ p,p' \epsilon P | p \neq p', w \epsilon W \quad (15)$$

$$w_{pw'}^{t} \geq C_{wlpw'} + 2AT + q_{pw'l}^{t} - M * (2 - \sum_{l'\epsilon L} x_{pw'l'}^{t} - x_{p-1wl}^{t'}) - M \sum_{r\epsilon R} r_{p-1r}^{t'}$$
$$\forall t \epsilon T,\ t' \epsilon \{1,\ldots,t\},\ a \epsilon A,\ p \epsilon \{1,\ldots,|P^a|\},\ l \epsilon L,\ w,w' \epsilon W \quad (16)$$

$$w_{pw'}^{t} \geq C_{wrlpw'} + 2AT + RT + q_{p-1r}^{t'} + q_{pw'l}^{t} - M * (3 - \sum_{l'\epsilon L} x_{pw'l'}^{t} - x_{p-1wl}^{t'} + r_{p-1r}^{t'})$$
$$\forall t \epsilon T,\ t' \epsilon \{0,\ldots,t\},\ a \epsilon A,\ p \epsilon \{1,\ldots,|P^a|\},\ l \epsilon L,\ r \epsilon R,\ w,w' \epsilon W \quad (17)$$

$$q_{pw'l}^{t} \geq z_{pw'}^{t} - \left(z_{p-1w'}^{t'} + C_{wlpw'} + 2AT + PT\right) - M * \left(2 - \sum_{l'\epsilon L} x_{pw'l'}^{t} - x_{p-1wl}^{t'}\right) - M \sum_{r\epsilon R} r_{p-1r}^{t'}$$
$$\forall t \epsilon T,\ t' \epsilon \{0,\ldots,t\},\ a \epsilon A,\ p \epsilon \{1,\ldots,|P^a|\},\ l \epsilon L,\ w,w' \epsilon W \quad (18)$$

$$q_{pw'l}^{t} \geq z_{pw'}^{t} - (z_{p-1w'}^{t'} + C_{wrlpw'} + 2AT + PT + RT + q_{p-1r}^{t'}) - M * (3 - \sum_{l'\epsilon L} x_{pw'l'}^{t} - x_{p-1wl}^{t'} + r_{p-1r}^{t'})$$
$$\forall t \epsilon T,\ t' \epsilon \{0,\ldots,t\},\ a \epsilon A,\ p \epsilon \{1,\ldots,|P^a|\},\ l \epsilon L,\ w,w' \epsilon W \quad (19)$$

$$q_{pr}^{t} = b_{r}^{t} - (z_{pw}^{t} + C_{wr}) \quad \forall t \epsilon T,\ t' \epsilon \{0,\ldots,t\},\ p,p' \epsilon P,\ r \epsilon R, w,w' \epsilon W \quad (20)$$

$$b_{r}^{t} \geq z_{pw}^{t'} + C_{wr} + RT + q_{pr}^{t\prime} - M(1 - r_{pr}^{t'}) \quad \forall t \epsilon T,\ t' \epsilon \{0,\ldots,t\},\ p \epsilon P,\ r \epsilon R, w \epsilon W \quad (21)$$

Another set of constraints tracks the timing of task completions and determines cycle times. Constraint (14) ensures that the completion times of racks handled in sequence by an AMR follow the correct order. Constraint (15) enforces that inventory racks are completed in the sequence that they are processed at a given workstation. Constraints (16) and (17) determine the cycle time for an AMR based on whether it is performing a standard storage or replenishment cycle, respectively. Constraints (18), (19) deal with computing the queuing time at the workstations depending on the cycle that occurred beforehand, and constraint (20) computes the queueing times at the replenishment stations. Constraint (21) tracks the time of the last completed replenishment.

$$s_{pk}^{0} = MS \quad \forall p \epsilon P,\ k \epsilon K^{p} \quad (22)$$

$$s_{pk}^{t} = s_{pk}^{t-1} - \sum_{w\epsilon W} \sum_{l\epsilon L} x_{pwl}^{t}\ \mathrm{PL_p} + v_{pk}^{t} \quad \forall t \epsilon T,\ p \epsilon P,\ k \epsilon K^{p} \quad (23)$$

$$v_{pk}^{t} \geq MS - (s_{pk}^{t-1} - \sum_{w\epsilon W} \sum_{l\epsilon L} x_{pwl}^{t}\ \mathrm{PL_p}) - M(1 - \sum_{r\epsilon R} r_{pr}^{t}) \quad \forall t \epsilon T,\ p \epsilon P,\ k \epsilon K^{p} \quad (24)$$

$$v_{pk}^{t} \leq MS \sum_{r\epsilon R} r_{pr}^{t} \quad \forall t \epsilon T,\ p \epsilon P,\ k \epsilon K^{p} \quad (25)$$

$$s_{pk}^{t} \geq LT\left(\sum_{w\epsilon W} \sum_{l\epsilon L} x_{pwl}^{t} - \sum_{r\epsilon R} r_{pr}^{t}\right) \quad \forall t \epsilon T,\ p \epsilon P,\ k \epsilon K^{p} \quad (26)$$

$$\sum_{r\epsilon R} r_{pr}^{t} \leq \sum_{w\epsilon W} \sum_{l\epsilon L} x_{pwl}^{t} \quad \forall t \epsilon T,\ p \epsilon P,\ k \epsilon K^{p} \quad (27)$$

Stock management is another critical aspect of the system, governed by the following constraints. Constraint (22) initializes the stock levels of the inventory racks. Constraint (23), (24), and (25) dynamically update stock levels by accounting for stock consumption during workstation visits and

replenishment following visits to replenishment stations. Constraint (26) enforces a stock level threshold for each rack, triggering replenishment if the stock falls below this level. Constraint (27) ensures that replenishment can only occur during a decision event when that rack has been assigned a storage location.

$$x_{pwl}^{t}, y_{pl}^{t}, u_{pwl}^{t}, r_{pr}^{t} \in \{0,1\} \qquad \forall t \in T, p \in P, r \in R, l \in L, w \in W \qquad (28)$$

$$s_{pk}^{t}, v_{pk}^{t} \in \{1,2,\ldots,MS\} \qquad \forall t \in T, p \in P, k \in K^{p} \qquad (29)$$

$$z_{pw}^{t}, w_{pw}^{t}, q_{pwl}^{t}, q_{pr}^{t}, b_{r}^{t} \geq 0 \qquad \forall t \in T, p \in P, l \in L, w \in W \qquad (30)$$

Finally, the remaining constraints define the domains of decision variables. These constraints ensure that variables are appropriately restricted to binary values, integers specified bounds, or continuous non-negative values, as required by the model.

5. Methodology

This section discusses the proposed method for solving the dynamic storage assignment and replenishment problem in an RMFS. Section 5.1 details our design of a Reinforcement Learning (RL) model for the problem, including creating the action space, state space, and reward function to train the model effectively. In Section 5.2, we briefly discuss the commonly used decision rules from the academic literature in dealing with this problem.

5.1. Deep reinforcement learning agent

In the literature on decision rules concerning optimizing material handling resources, relatively simple rules are used to schedule the available resources. In cases where the decision rules are more advanced, they typically only work for a specific problem. Thus, when new problems are defined or additional constraints are added, these rules must be adapted. This process is time-consuming and requires significant in-depth expert knowledge concerning the problem. However, instead of spending much time developing these rules, letting an agent learn the best decision rules/policies is possible. To do so, the agent has to interact and learn from its environment, where storing them preemptively closer to the replenishment station might be interesting (Cals et al., 2021).

DRL combines RL techniques and Deep Learning (DL). The technique is a general framework for solving complex problems involving sequential decision-making. RL focuses on learning through trial and error. The learning agent interacts with the environment by taking actions after observing the environment. This process continues, where the environment moves to a new state (*s'*) after an action has been taken, and the learning agent makes a new action based on this new perceived state. The overarching goal of RL is to learn through a series of actions and formulate a policy (π) (Tassel et al., 2020). On its own, RL is limited to applications where the states are low-dimensional. To overcome this limitation, DL has been introduced (Luo, 2020). In short, DL involves training neural networks to

perceive high-dimensional states. The neural networks allow us to approximate either a value or policy function and can learn to map states to values. Essentially, there are three crucial elements for effective interactions between the agent and the environment (see Figure 4): the state (*s*) of the environment, an action (*a*), and a reward (*r*) (Nguyen et al., 2020).

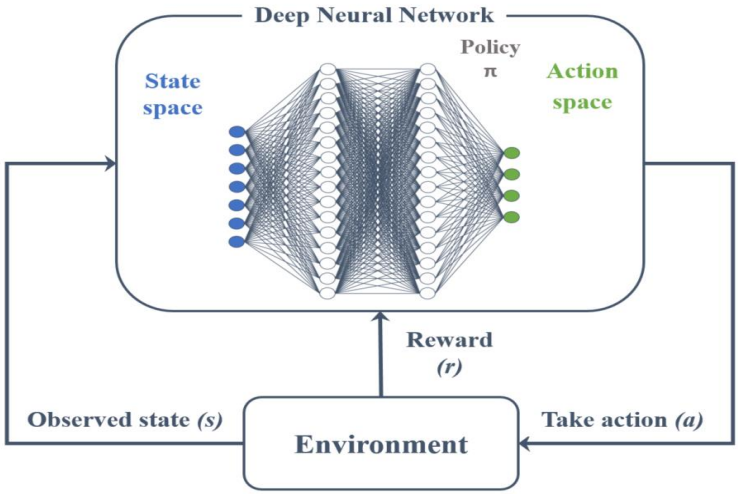

Figure 4: *Interaction scheme of a deep reinforcement learning agent and the environment.*

In the DRL field, several algorithms are typically categorized into either value-based or policy-based approaches **(**Arulkumaran et al., 2017**)**. The first approach tries to approximate the state-value function. The latter tries to learn the policy directly. Both approaches have their advantages and disadvantages. One of the main drawbacks of both these approaches is that they normally use a table memory for learning, which limits their use case for complex problems due to a lack of memory. Unlike traditional table-based approaches, which require storing all possible state-action pairs, modern policy and value-based DRL methods employ function approximators such as deep neural networks (DNNs). This significantly reduces memory requirements and enables the application of these methods for more complex problems with large state and action spaces. Another approach, the proximal policy optimization method (see Algorithm 1), has been proposed to address this issue. In Appendix A, a more detailed explanation of the policy update expressions is provided. The hyperparameters chosen for this algorithm are detailed in Table 9 (see Appendix A). This approach combines value-based, which estimates the value of actions in a given state, and policy-based approaches, which seek strategies for action selection based on the current state (Schulman et al., 2017). It separates the memory structure for an agent into both an *actor* and a *critic* and focuses on learning the policy and the state-values function (Tassel et al., 2020; Tang et al., 2021). The actor selects an action based on the current policy, and the critic evaluates this selected action. The critic informs the actor how good this action was and helps the actor adjust and improve its policy by a temporal difference (TD) error, allowing it to predict a future tendency of a selected action. The TD method reduces the error between the predicted and actual state(-action) values. The TD method updates during episodes, which helps the convergence of the learning

process (Nguyen et al., 2020). Moreover, the experience replay buffer also aids the convergence behavior of the learning process by learning from previous experiences. It is a library containing past experiences captured by an agent interacting with its environment. PPO is particularly well-suited for this problem due to its combination of the following characteristics:

- **Stability:** PPO achieves a good balance between exploration and exploitation by constraining policy updates within a clipped range, preventing overly aggressive updates that can destabilize training.
- **Sample efficiency:** By allowing multiple epochs of mini-batch updates for a given set of experiences, PPO makes efficient use of collected data, which is critical in environments where data collection can be expensive or time-consuming.
- **Versatility:** PPO has demonstrated robust performance across a wide range of environments, making it a good candidate for addressing complex, high-dimensional problems like the one discussed in this paper.

Algorithm 1 Proximal Policy Optimization

Initialize policy parameter θ, value network parameter ϕ, episodes N, clip range ϵ
Env ← initializeEnv()
for i ∈ {1, ⋯, |N|} **do**
  for t ∈ {1, ⋯, |T|} **do** *(where T are the timesteps)*
    Run policy π_θ and collect (s_t, a_t, r_t)
    Estimate advantages $\hat{A}_t = G_t - V_\phi(s_t)$ *(where G_t is calculated with Eq. 1)*
  end for
  $\pi_{old} \leftarrow \pi_\theta$
  Update actor policy with the gradient
  $\nabla L^{CLIP}(\theta) = E[\nabla(min(q(\theta)\hat{A}_t, clip(q(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t))]$, where $q(\theta) = \pi_\theta/\pi_{old}$
  Update critic value function with the gradient $\nabla L_\theta(\phi)$
end for

Since the introduction of deep learning in reinforcement learning techniques, its application in scheduling problems became possible due to its ability to perceive and interpret the complex state of the scheduling environments. However, it is important to correctly define the action space, state representation, and reward function for the learning agent(s) to learn a policy (π) properly.

5.1.1. Action space

At the beginning of the training process, the learning agent will tend to choose actions at random instead of basing itself on the value function. However, in later stages of the learning process, the probability that a chosen action is based on the value function increases. The ε denotes the probability of an action chosen randomly; this ε-value decreases over time (Shi et al., 2020). Furthermore, there are scenarios where specific actions are unavailable or impossible to execute. For instance, it should be impossible for the agent to relocate an inventory pod to a storage zone already occupied by other inventory racks. However, the action space should remain constant. Thus, these storage zones cannot be left out of the

action space. To cope with this, we mask the invalid actions (Huang and Ontañón, 2020). Action masks restrict the set of actions the agent can select and only sample actions from the valid set of actions. Moreover, it aids in improving the efficiency of the learning process of the learning agent as it reduces the exploration space. The resulting action space is a vector with a relocation action for each storage zone in the storage area and an action instructing a replenishment and a relocation move for each storage zone. Hence, the size of the action space depends on the amount of storage zones in the storage area. Once the model selects a storage zone, the specific storage location within that zone is determined using the shortest leg strategy, as detailed in Section 5.2.

Action Space = [Zone 1, Zone 2, …, Zone n, Replenishment + Zone 1, Replenishment + Zone 2, …, Replenishment + Zone n]

5.1.2. State space

The state vector represents the current state of the warehouse environment. This state vector allows the learning agent to select an action. Basically, it acts as a window for the agent to interpret the environment. Hence, when an action is taken, the state of the environment changes at a certain time step, and the state vector has to change accordingly. The scheduling agent is rewarded immediately after the action according to the policy (possibly random) chosen. Furthermore, all the features in the state space are normalized to help the learning process (Andrychowicz et al., 2020). The state features are listed in Table 2.

Table 2: *State space features.*

State	Description
Rank turnover rate	The relative rank of the inventory rack's average turnover (one feature).
Zone next retrieval	The zone of the next retrieval task of the mobile robot (one-hot encoding).
Occupation levels	The occupation level in a storage area refers to the extent to which the available storage locations within a certain zone are occupied (a feature for each zone).
Traffic density	The number of robots traveling towards each zone to retrieve an inventory pod or to the replenishment station (a feature for each zone and replenishment station).
Station Id	The identifier of the workstation where the rack is currently at (one-hot encoding).
Storage cycle time	The estimated storage cycle time for each zone (a feature for each zone).
Replenish cycle time	The estimated replenishment cycle time to each zone (a feature for each zone).
Estimated queue	The estimated queue time at the replenishment station (one feature).
Stock level	The current level of stock present on the inventory rack. (a feature for each unique SKU in the inventory rack).
SKU similarity	A similarity index score is determined for each storage zone, indicating how many SKUs in the inventory rack to be stored are already present within the storage zone.

5.1.3. Reward function

Through a reward, the scheduling agents learn whether a certain action in a certain state was a preferential decision. Hence, it allows the agents to update their policy accordingly. The reward system does not only look at the immediate impact of an action on the environment. It can also incorporate the impact of an action now on its future state by including a discount factor (γ) between $0 \leq \gamma \leq 1$. When γ is close to 0, it values a short-term result more than a long-term one and vice versa. The following function (31) is used to calculate the return value at timestep t:

$$G_t = R_t + \gamma\, R_{t+1} + \gamma^2 R_{t+2} + \ldots + \gamma^{T-t-1}\, R_T \qquad (31)$$

The intelligent learning agents aim to find an optimal policy (π^*) that maximizes the expected sum of the long-term rewards. Based on the state, either direct storage location assignment or replenishment and storage location assignment (as shown in Figure 3), the cycle time can be determined as follows:

1. **Storage action:** The cycle time estimation starts with the inventory rack's location at a workstation. We compute the travel time from the workstation to the selected storage location within a certain storage zone. Thereafter, we then calculate the travel time from the storage location to the new retrieval task's location and finally from there back to the workstation where the new rack is requested. This estimation includes the pick-and-place actions associated with handling the inventory racks at the storage locations. These calculations are performed for each storage zone that is not fully occupied, resulting in an estimated storage cycle length for each zone.
2. **Replenishment action:** For replenishment actions, we first calculate the travel time from the workstation to the assigned replenishment station. We then include an estimate of any queuing time at the replenishment station, followed by the time required for the actual replenishment process. After restocking, we calculate the time required to transport the inventory rack back to the storage area, following the same steps used in the storage action. However, in this case, the transport is from the replenishment station to the designated storage location. These replenishment cycle estimations are similarly conducted for all potential storage zones.

The reward function is expressed in Equation (32) based on the state-action pair as follows:

$$r(s,a) = \begin{cases} R_t = \bar{T} - T_s\,(\bar{T}/\,\bar{T}_s) & if\ action\ =\ storage \\ R_t = \bar{T} - T_r\,(\bar{T}\,/\,\bar{T}_r\,) + \alpha\,(\tau + T_{queue})\,S & if\ action\ =\ replenishment \end{cases}$$

After each decision the agent makes, the reward is computed based on the two primary actions described above. The reward structure is designed to balance both cycle lengths while minimizing replenishment frequency. The cycle length $\bar{T}$ represents the overall average of all cycles performed by the AMRs and is updated at the end of each complete simulation run if a better average cycle length is found. Rather than calculating it dynamically during the simulation, it is computed at the end to prevent oscillations

in the learning process. At the beginning of the model's training phase, the cycle length is initialized based on the best-performing decision rule (velocity-based storage location assignment). From there, the cycle length is updated with any improvements found in subsequent run, ensuring stability in the learning process while gradually optimizing the average cycle length over time. In the storage action, the reward encourages the agent to minimize the current storage length (T_s). This is achieved by calculating the difference between the $\bar{T}$ and the normalized T_s. The current cycle length is normalized by multiplying it by the ratio between $\bar{T}$ and the average storage cycle time ($\overline{T_s}$). This ensures that the reward reflects how much the current storage decision deviates from the overall average cycle time, with the agent striving to minimize storage cycle times, and, as a result, the average reward should converge closer to zero.

In the replenishment action, the reward is analogously designed to minimize the current replenishment cycle length (T_r). It is calculated by subtracting the normalized replenishment cycle length from $\bar{T}$, aligning the goal of minimizing replenishment times with overall system performance. Since replenishment cycles are significantly longer than storage cycles, an additional penalty factor is introduced to maintain balance. This penalty is calculated as the weight factor (α) multiplied by the sum of the replenishment time required for each SKU at the station (τ) and the queueing time (T_{queue}) the inventory rack will experience. The weight factor α is computed as the ratio between the $\overline{T_r}$ and the $\overline{T_s}$, adjusted by the ratio of average number of replenishment actions and storage actions. Furthermore, a stock level factor (S) is also introduced, calculated as the difference between the current stock level of the inventory rack and the replenishment threshold, to ensure replenishments occur only when necessary. These averages are updated after each complete simulation run.

5.2. Benchmark decision rules

We compare the proposed method against the commonly studied storage policies (see Figure 3) for autonomous storage and retrieval systems (AS/RS):

- *Fixed storage policy*: This policy simply dictates that an inventory rack has a fixed location within the storage area and should always return to this location after fulfilling its purpose.
- *Random storage policy*: This policy randomly assigns an available storage location to the inventory racks that must be returned to the storage area.
- *Nearest storage policy*: This policy looks into finding the nearest available storage location from the current location of the inventory rack that has to be returned to the storage area (Weidinger et al., 2018b).
- *Shortest leg storage policy*: This policy, in addition to looking for the nearest available storage location from the current location of the inventory rack to be returned to the storage area, considers the location of the AMR's next retrieval task (Van Den Berg et al., 2000; Rimélé et al., 2021; Zhuang et al., 2022b). Thus, it minimizes the total travel time from the location when

the storage cycle starts to the location of the AMR's next retrieval task. In essence, this policy attempts to greedily optimize the immediate cycle times of each AMR.

- *Velocity-based policy:* The policy, commonly also referred to as turnover-based, aims to minimize the cycle time of AMRs by positioning frequently requested inventory racks, those containing fast-moving items, closer to workstations. The policy is based on the approaches of Yuan et al. (2019) and Ding et al. (2024). Racks currently at the workstations are ranked by velocity, which refers to item requests' frequency. Higher-demand racks are ranked higher, and storage locations are evaluated starting from these racks. Using the shortest leg principle, locations are chosen. Once a location is found, this location is temporarily reserved and excluded from consideration for subsequent racks. This process repeats iteratively until a storage location is found for the rack under consideration. Finally, only this location is actually assigned.

Figure 3: *Visualization of the commonly used decision rules for storage location assignment.*

The order assignment and sequencing, the inventory rack selection, and the task assignment to the AMRs is considered given. We refer to our approach in Teck et al. (2023) for these assignment and sequencing problems. The replenishment strategy considered in this study, used as a benchmark for the learning agent, is based on the replenishment approach described in Lamballais et al. (2020), where the inventory level of rack j is monitored and given an inventory class. They define an inventory class as the number of units remaining on the inventory rack. They establish the replenishment level, denoted by ξ, a fraction of the maximum number of units U on the inventory rack. Thus, the replenishment point for a given rack would be ξU. Following this, they make two inventory classes: the inventory class where $j > \xi U$, in this class racks, are allowed to be returned to the storage area, and the inventory class where $j \leq \xi U$, in this class, racks have to be moved to the replenishment station.

6. Numerical experiments

The proposed solution approach has been implemented using Python 3.8, and the numerical experiments were executed on the Narval high-performance cluster of Compute Canada. The cluster hardware

specifications are 2 AMD Rome 7532 CPUs 2.4 GHz 256M cache L3. The learning time of each model was limited to 10000 episodes of each 10000 time-step or four days of running time.

In Section 6.1, a brief description of the characteristics of the generated instances is provided. Section 6.2 presents the results of the numerical experiments. Section 6.3 briefly studies the impact of the zoning strategy on the system performance. Finally, Section 6.4 provides some discussion of the model's detail, thus enhancing its explainability.

6.1. Problem instances

To assess the performance of the proposed model, we generate a set of problem instances that vary in terms of the number of storage locations within the storage area and their layout. Moreover, as the storage area expands, the number of AMRs and workstations also increases to achieve a server utilization rate of 60 to 80 percent. Each simulation run corresponds to 8 to 10 hours, a typical human operator workday. The customer orders within these problem instances consist of an average of 1.6 order lines, with most orders having only 1 or 2 order lines. In these systems, workstation capacity is defined as the number of customer orders that can be handled simultaneously. Per industry standards, the workstation capacity ranges from 4 to 20 orders. The general layout of the warehouse for all instances in this study is based on configurations used in previous research focusing on the impact of warehouse layouts on system performance (Lamballais et al., 2020). Table 3 presents additional information on the characteristics and considerations in generating the problem instances used for evaluation in this study.

Table 3: Characteristics of problem instances.

Characteristic	Unit	Value
Number of vertical shelves in a block	-	2
Number of horizontal shelves in a block	-	5
Number of storage locations	-	180 - 900
Width of side aisle	m	5
Unit length	m	1
Number of AMRs	-	10 - 18
AMR velocity	m/s	0.6
Time for pod lifting and storing	s	3
Number of workstations (*w)*	-	2 - 3
Buffer at workstations	-	5
Station order capacity	-	10
Picking time per SKU	s	10
Replenishment time per SKU (τ)	s	15
Replenishment threshold	%	30

Theoretically, the AMRs operating in an RMFS can have speeds up to 1.78 meters per second. However, this excludes the time the AMRs take to perform turns and the time lost due to congestion within the storage area. Thus, we assume a constant velocity of 0.6 meters per second. The picking time at the workstations is deterministic for each SKU retrieved from the inventory rack. Similarly, the replenishment time for restocking SKUs on the inventory racks is also deterministic. The study

considers a diverse range of storage area sizes, with storage capacity between 180 and 900 storage locations (*i*). Furthermore, depending on the storage area layouts, different zones (z) can be defined. Figure 5 shows an example of a storage area with multiple different layouts for the same storage capacity.

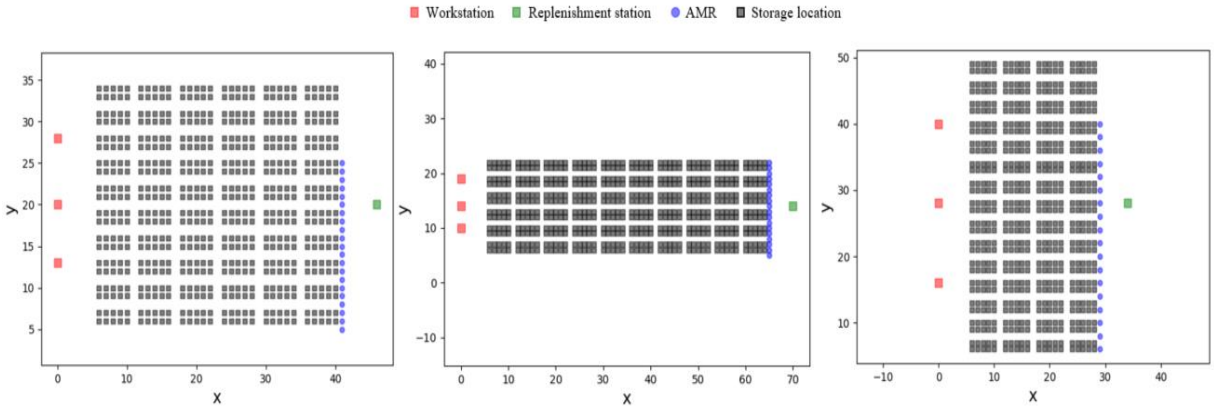

Figure 5: *Different storage area layouts for instances 10 (left), 11 (middle), and 12 (right). Instance layouts can be found through the link: https://www.mech.kuleuven.be/en/cib/rmfs.*

6.2. Numerical results

This subsection presents the results of the experimental study, comparing the proposed solution approach to commonly used decision rules. Average cycle time serves as a robust metric, encompassing both travel and processing efficiencies, and is particularly suitable for evaluating the system's dynamic demands related to joint replenishment and storage location assignments. Traditionally, storage location assignment studies prioritize minimizing total travel distance as the primary metric of system performance. However, this approach overlooks the influence of queuing and processing times at stations, which significantly impact overall system throughput.

Table 4: Comparative results of the heuristic methods and the full-information model on instances with 100 storage locations, one workstation and replenishment station, and 3 AMRs with 3-hour computation cutoff (Optimality Gap calculated as: $\frac{|\text{Objective Bound} - \text{Objective Value}|}{|\text{Objective Value}|} \times 100\%$).

Rack visits	*Full-information model*		*Shortest Leg*		*Velocity-based*		*DRL*	
	Avg. cycle time [s]	*Optimality Gap [%]*	*Avg. cycle time [s]*	*Δ [%]*	*Avg. cycle time [s]*	*Δ [%]*	*Avg. cycle time [s]*	*Δ [%]*
12	***66.0***	*63.6*	*66.7*	*1.1*	*68.3*	*3.5*	*69.7*	*5.6*
15	***77.3***	*61.2*	*82.0*	*6.1*	*85.0*	*10.0*	*84.2*	*8.9*
18	***71.6***	*58.1*	*73.3*	*2.4*	*74.0*	*3.4*	*71.8*	*0.3*

Section 4 presents the mathematical model for the deterministic joint storage location allocation and replenishment, serving as a formal foundation for analyzing the optimization problem. The results, summarized in Table 4, provide a benchmark for comparison with alternative methods, helping to contextualize their performance relative to an optimal yet computationally expensive approach. Notably, the number of rack visits is directly tied to the number of decisions made by the model. The reported optimality gap for the full-information model does not reflect the performance of the DRL

approach. The optimality gap refers to the gap obtained by Gurobi when solving the offline full-information model. Given the time limit, Gurobi is able to close only a portion of the gap, even for small-sized instances. This demonstrates the intractability of the model. However, the feasible solution it generates can be used to evaluate the performance of the heuristic. The difference (Δ) between the solution of the full-information model and the DRL solution indicate that the heuristic performs well. However, it is important to note that the scheduling horizon considered is short, whereas the alternative methods are designed to focus on longer-term optimization. Overall, these findings underscore the mathematical model's limitations in serving as a practical decision-support tool for real-world applications. Additionally, the model assumes perfect knowledge of the environment, which is unrealistic given the dynamic nature of warehouse operations.

Table 5 summarizes the average cycle times of the AMRs across various warehouse layouts. As anticipated and aligned with existing academic literature, the velocity-based approach is the top-performing decision rule approach. The shortest leg rule follows closely, exhibiting competitive performance, particularly in smaller instances. Furthermore, the random, fixed, and nearest location rules exhibit significant underperformance compared to all methods, with performance differences going up to approximately 25% compared to DRL. Notably, the DRL decision-maker consistently outperforms the other rule-based approaches, demonstrating improvement gains of up to 5.4% compared to the velocity-based method. On average, this reduction in cycle times result in a 4.1% decrease in the system's makespan and a 6.3% reduction in the total distance traveled by the mobile robots. Moreover, performance gains consistently increase alongside the size of problem instances when comparing simpler decision rules (e.g., random, fixed, nearest) to more intelligent rules (e.g., shortest leg, velocity-based, and DRL). This indicates that, in larger problem instances, more sophisticated decision-making processes offer significant advantages. This underscores the effectiveness of intelligent storage location assignment and replenishment strategies, which are better equipped to comprehend the intricacies of the problem at hand, leading to notable improvements in performance.

Table 5: Comparative study of the different decision rules and the proposed decision-maker (*i* represents the number of storage locations, *w* number of workstations, *z* number of storage zones, and Imp. the improvement compared to the best decision rule). The last column provides the performance gain of our DRL over the most effective decision rule on the same instance.

Instance	*i*	*w*	*AMR*	*z*	Avg. cycle time [s]						Imp. [%]
					Random	*Fixed*	*Nearest*	*Shortest Leg*	*Velocity-based*	*DRL*	
1	180	2	10	9	107.7	108.6	100.9	96.9	95.5	**91.1**	4.9
2				9	124.5	124.3	115.0	113.6	111.2	**106.0**	4.9
3				9	106.7	105.9	99.6	94.8	93.6	**88.9**	5.3
4	300			10	118.7	118.7	111.1	105.1	103.6	**99.0**	4.6
5				10	121.5	124.3	113.0	111.5	106.4	**101.9**	4.3
6				9	114.9	115.7	103.5	100.6	97.9	**94.6**	3.5
7	480	3	18	12	133.1	134.1	121.8	117.4	113.6	**107.9**	5.2

8		12	139.0	142.6	127.7	123.4	119.6	**113.5**	5.4
9		12	130.5	128.9	118.7	110.7	109.1	**104.4**	4.5
10	600	9	137.8	138.3	127.9	121.3	116.3	**111.6**	4.2
11		10	153.9	157.6	143.0	136.1	130.9	**124.5**	5.1
12		10	138.0	135.2	124.1	117.5	113.8	**110.2**	3.2
13	900	9	159.3	161.1	145.3	135.6	130.9	**126.2**	3.8
14		9	161.3	164.5	147.9	139.1	133.0	**127.8**	4.1
15		9	155.9	150.8	140.0	129.3	125.1	**120.8**	3.6

A more comprehensive comparison between the best-performing decision rule, velocity-based, and the proposed solution approach reveals improvements in both the storage and replenishment cycles. Table 6 presents various metrics, including average storage cycle time ($\overline{T_s}$), average replenish cycle time ($\overline{T_r}$), average wait time (T_{wait}) at the replenish stations resulting from queueing, and the average number of replenishments (*Repl.*) performed during a simulation run. Intuitively, minimizing the wait times at the replenishment stations is to be expected. However, this typically implies an increase in replenishments compared to the simple replenishment rule, which waits until the threshold value is met. Conversely, the learning agent can proactively assign an inventory rack for replenishment to mitigate longer queues. Replenishment cycles, including replenishment times, are inherently longer than storage cycles, negatively impacting the average cycle time. Consequently, a delicate balance exists between queue time and the number of replenishments.

Table 6: Comparative study of the best performing decision rule and the proposed decision-maker.

Instance	*i*	Velocity-based				DRL			
		$\overline{T_s}$*[s]*	$\overline{T_r}$ *[s]*	T_{wait} *[s]*	*Repl.*	$\overline{T_s}$*[s]*	$\overline{T_r}$ *[s]*	T_{wait} *[s]*	*Repl.*
1	180	79.6	253.5	34.2	252	**76.8**	**231.3**	**25.3**	255
2		93.2	296.6	**20.0**	208	**84.8**	**287.7**	21.2	256
3		79.0	228.5	29.7	230	**76.2**	**216.8**	**26.6**	214
4	300	92.6	279.3	**12.0**	151	**88.9**	**267.1**	18.9	149
5		94.6	294.5	17.2	154	**89.8**	**273.9**	**14.0**	174
6		89.1	251.5	19.9	140	**86.5**	**234.0**	**14.3**	141
7	480	104.0	325.1	34.0	152	**100.2**	**296.3**	**25.4**	143
8		108.0	351.3	27.3	184	**102.6**	**323.6**	**23.9**	198
9		102.5	283.0	**26.4**	145	**97.8**	**261.8**	27.7	168
10	600	109.2	313.0	**21.5**	141	**104.3**	**299.6**	28.3	153
11		119.6	378.7	**15.3**	178	**112.7**	**364.0**	24.0	194
12		109.4	278.1	**10.3**	104	**105.7**	**267.9**	24.8	117
13	900	123.8	381.5	21.3	125	**120.2**	**355.6**	**19.1**	117
14		126.6	386.4	9.7	109	**122.7**	**359.5**	**9.2**	95
15		121.1	322.3	7.5	89	**118.0**	**287.7**	**5.5**	76

The numerical results in Table 6 indicate that the DRL agent consistently achieves lower average storage and replenishment cycles across all scenarios. While waiting times are often lower compared to the velocity-based approach, this is not always the case, possibly due to their limited impact on overall average cycle time in certain scenarios. Lower wait times are more evident in instances with higher replenishments, where their cumulative effect is greater. Figure 6 illustrates the characteristic training curve of the decision-maker. In the initial stages of the learning process, the agent commences by

randomly allocating inventory racks to storage locations and replenishment activities. Gradually, the agent refines its policy through a series of trial-and-error actions.

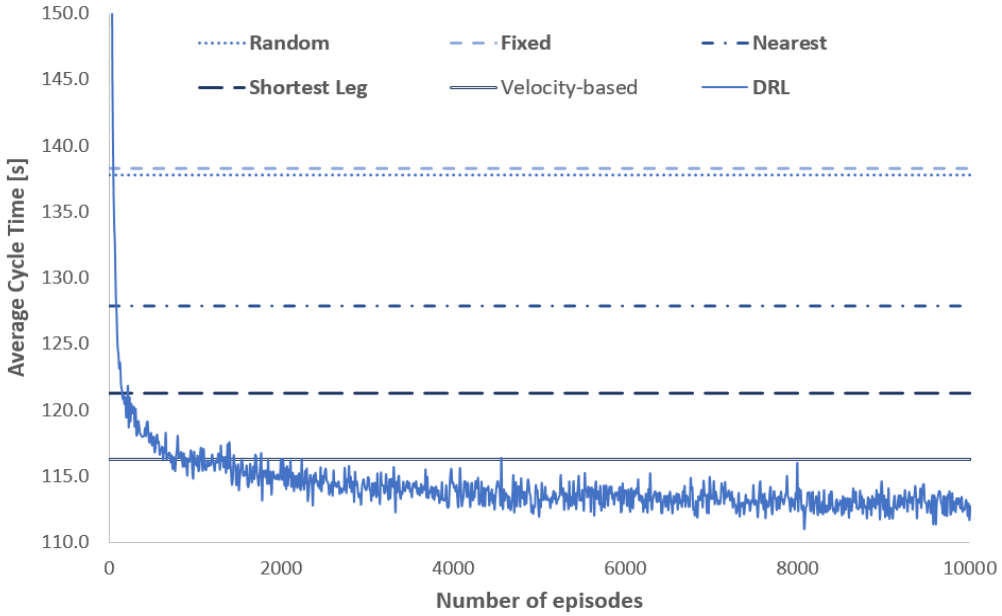

Figure 6: *Learning process of the decision-maker on a training instance with 600 storage locations.*

The models exhibit superior performance when evaluated on problem instances they were trained on, surpassing conventional decision rules. However, their utility extends beyond these specific instances. These models can effectively adapt if alternative instances maintain similarity to the originals in terms of crucial parameters such as the state space, ensuring consistency in factors like the number of storage zones, workstations, and the number of SKUs on inventory racks being provided. While certain aspects must remain unchanged to ensure compatibility, there is flexibility in how, for instance, SKUs are distributed among the inventory racks and the positioning of the stations. This adaptability enables the models to tackle diverse scenarios. Hence, the test instances are generated with the same parameters as presented in Table 3. However, the SKU distribution among the racks and order list are different. Table 7 presents a comparative analysis of the model's performance on alternative problem instances against that of the velocity-based decision rule. We showcase its generalizability, reinforcing its applicability beyond the confines of its training data.

Table 7: Validation experiments on alternative problem instances.

Instance	$\overline{T}$ [s]		Imp. [%]	*Instance*	$\overline{T}$ [s]		Imp. [%]
	Velocity-based	*DRL*			*Velocity-based*	*DRL*	
1	94.4	**90.6**	4.2	9	108.5	**104.5**	3.8
2	112.7	**107.5**	4.9	10	116.3	**112.2**	3.6
3	92.7	**89.0**	4.1	11	128.9	**124.2**	3.8
4	103.3	**98.9**	4.5	12	114.3	**110.7**	3.3
5	106.6	**102.2**	4.3	13	128.8	**125.7**	2.4
6	98.1	**95.2**	3.1	14	133.9	**128.4**	4.2
7	112.1	**109.1**	2.7	15	126.3	**121.9**	3.7
8	120.4	**115.7**	4.1				

6.3. Impact of storage zoning strategy

In the context of a specific warehouse layout, selecting different strategies significantly impacts the optimization potential available to the decision-maker. However, this decision involves a delicate trade-off between optimization potential and the duration of the decision-maker's learning process.

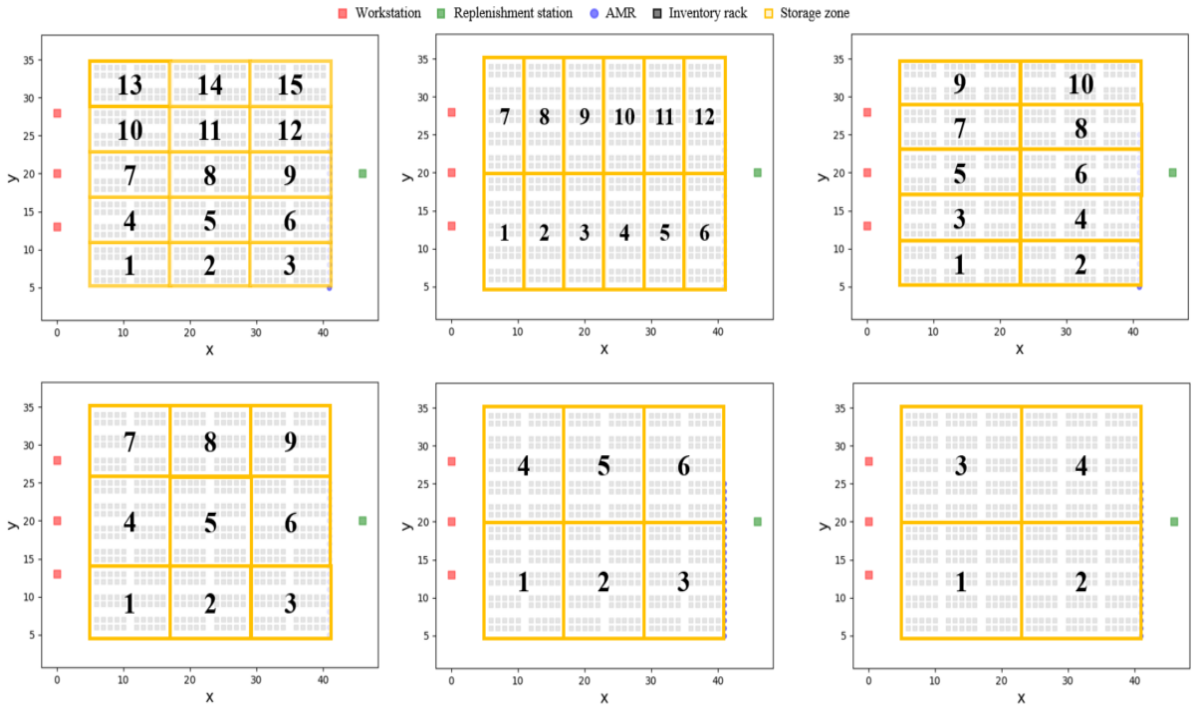

Figure 7: *Various storage zoning strategies for instance 10.*

In Figure 7, several different zoning strategies are shown for a storage area with 600 storage locations. Increasing the number of storage zones directly expands the decision-maker's action space, potentially slowing the learning process. Moreover, the size of the storage areas plays a critical role. Designing overly large zones may compromise the optimization potential, leading to suboptimal storage location assignment for slow-moving inventory in locations more suitable for faster-moving inventory. On the other hand, creating excessively small zones may lead to a loss of critical information for the decision-maker concerning specific environmental states, such as the SKU similarity. Striking the right balance is essential, as too many smaller zones can also increase the complexity of the learning relationship between adjacent zones, posing challenges for the learning agent. The subsequent table (Table 8) showcases the impact of various zoning strategies on the average cycle time within the storage area. In this context, the best design consisted of 9 storage zones.

Table 8: Impact of different storage zoning strategies on the system performance for instance 10. The gap is relative to the best average cycle time out of the series of experiments for the given instance.

z	Avg. cycle time [s]	Gap [%]
2	115.9	-3.8
4	115.3	-3.2
6	113.3	-1.5
9	**111.6**	**0.0**
10	114.6	-2.6
12	113.4	-1.6
15	115.2	-3.1

6.4. Explainable AI

One notable drawback of deep reinforcement learning is its tendency to be perceived as a "black box", where the underlying decision-making processes are difficult to interpret. This lack of transparency makes it challenging to understand why the model selects specific actions in given states, highlighting the importance of addressing the explainability issue. To shed light on the model's inner workings, we use SHapley Additive exPlanations (SHAP), which helps interpret how individual state features contribute to the decision-making process. SHAP is based on game theory, where the overall reward of a game is fairly distributed among its players. In our context, the "players" are the state features, and the goal is to understand how each feature influences the selection of a particular action.

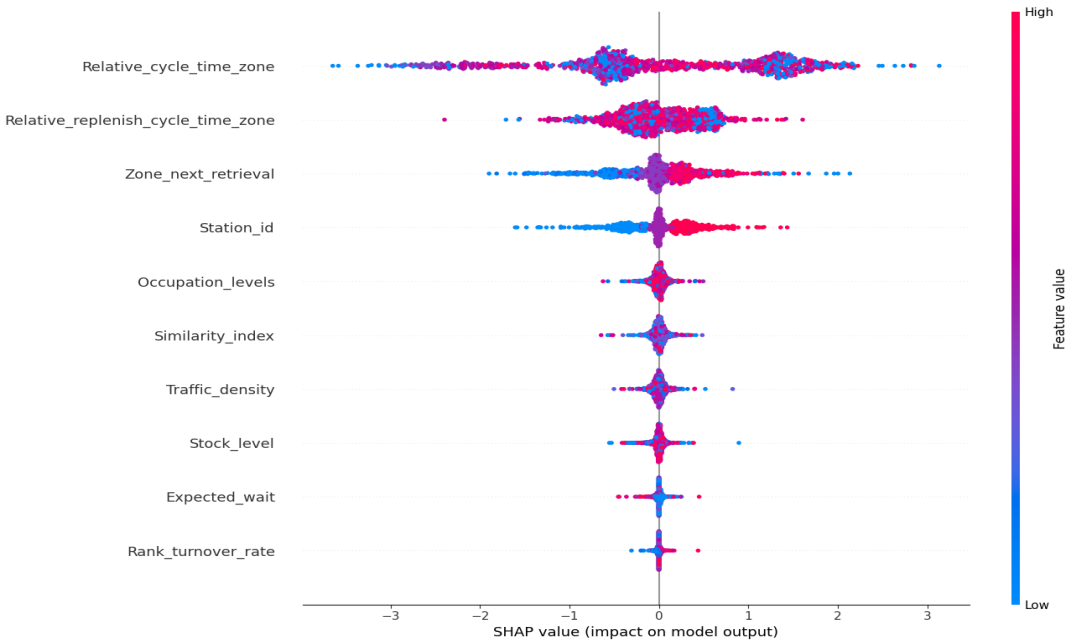

Figure 8: *A beeswarm plot illustrating the SHAP values of a set of 1000 observations.*

Figure 8 demonstrates how various state features affect SHAP values, providing insight into their impact on the model's outputs. To apply this method effectively, it is necessary to decode features that have been encoded as arrays into their real-world equivalents. For example, the station identifier is represented through one-hot encoding, where an array consists of zeros, and a single one indicates the

current location of the inventory rack at a workstation. By decoding this array (e.g., [0, 0, 1] corresponds to station identifier 3), we can better interpret the SHAP values and, in turn, the model's decision-making process.

As anticipated, the most impact stems from the cycle time estimations and information regarding the zone of the next retrieval task. While the plot does not directly convey whether higher or lower values of cycle times result in higher or lower SHAP values, the interpretation becomes more straightforward for the feature representing the zone of the next retrieval. Specifically, lower values of this zone are likely to correspond to lower SHAP values, indicating the model's preference for zones closer to the indicated retrieval zone. Similar reasoning applies to the station identifiers, as depicted in Figure 7; certain workstations are closer to specific zones, influencing the model's probability of choosing storage zones nearby. Furthermore, the stock level's impact aligns with expectations, where lower stock levels on inventory racks correlate with higher SHAP values. This makes sense, considering that replenishment actions with lower stock levels yield higher output numbers, as the replenishment actions are in the second half of the action space (Section 5.2.1).

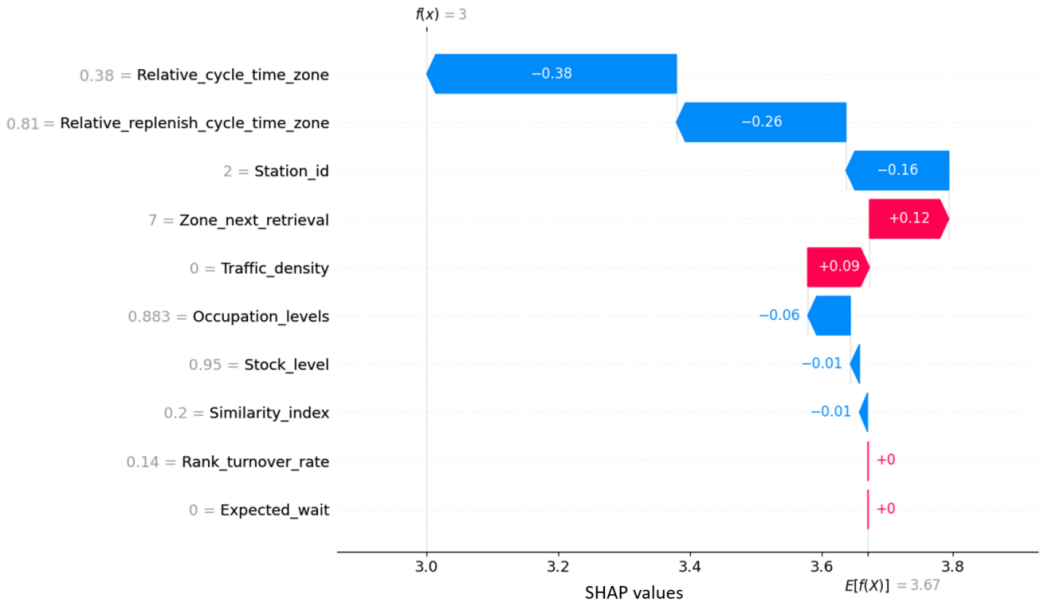

Figure 9: *A waterfall plot illustrating the impact of each feature state on the decision of an action during the warehouse operations.*

The waterfall plot in Figure 9 provides an insightful breakdown of the RL agent's decision-making process for a particular state in the environment. The x-axis shows the cumulative SHAP values, indicating the incremental impact of each feature on the model's output. At the same time, the y-axis lists the relevant features, with their actual values shown in grey text. This plot starts from the model's base prediction, E[f(x)], which represents the average output across the training set (3.67 in this case), and moves step-by-step through each feature's contribution. To interpret the waterfall plot, one reads from the bottom up. As each feature is evaluated, it adds to or subtracts from the base prediction. Red

bars signify positive contributions that push the prediction upward, while blue bars indicate negative effects that push it downward. These contributions accumulate sequentially, illustrating how each feature shifts the prediction to reach its final value.

In this example, only a few features significantly influence the decision. Specifically, features like the expected wait time and the turnover rank of the inventory rack exert minimal impact. In contrast, the next retrieval task zone, station identifier, and the estimated cycle times notably shape the output. For instance, the next retrieval task zone drives the output upwards, suggesting a preference toward storage zone 7. However, combining the station identifier and cycle time features ultimately determines the final output by anchoring it at 3, corresponding to storage zone 4. Thus, this choice has been influenced by the advantageous proximity of workstation 2 to zone 4.

7. Conclusions and Future Research

This study considers the storage location assignment and the inventory rack replenishment problem in an RMFS, wherein AMRs have the ability to retrieve and store movable inventory racks in the storage area. The primary objective of this study is to minimize the average cycle times of the mobile robots while minimizing the queuing times at the replenishment station. We examine the system performance under various warehouse settings, focusing on validating the effectiveness of the proposed deep reinforcement learning approach. We also conduct comparative assessments against commonly used decision rules in the existing academic literature.

From the experimental results, we conclude that the commonly used decision rules yield suboptimal cycle times for mobile robots in an RMFS. The proposed learning agent demonstrates the capability to enhance warehouse storage and replenishment operations significantly, the simulation results from the experimental study also support this claim. The proposed agent consistently outperforms traditional decision rules in all instances, yielding significant gains with cycle times reduced by instances. In all instances, the proposed agent consistently outperforms traditional decision rules, yielding significant gains with cycle times reduced by up to 5.4%. Furthermore, the design of storage zones is an important consideration to improve both the learning process and the quality of decisions made by the decision-maker. The size of these zones matters, as overly small zones may lead to information loss within the state environment, hindering the learning process. Alternatively, excessively large zones risk sacrificing some optimization potential. Striking the right balance in storage zone design is imperative. By utilizing explainable AI methodologies, such as SHAP, we can identify the features that significantly impact the decision-making process. This analysis not only enhances our understanding of the model behavior but also enables the future development of more sophisticated heuristic methods based on the insights gained.

While the proposed method offers significant potential, it is important to acknowledge several key limitations and outline directions for future research. First, the current learning framework does not explicitly assign storage location for inventory racks. This omission was intended to reduce the complexity of the action space, which can otherwise hinder the learning process. However, this simplification may limit the potential improvements in cycle time lengths for AMRs. Exploring whether explicit storage location assignment could enhance performance is an interesting avenue for future investigation, even if it would introduce more complexity. Secondly, alternative optimization approaches, such as predict-and-optimize methodologies, could be explored. However, balancing computational efficiency with solution quality remains a challenge, as the proposed mathematical model highlights the problem's computational complexity. To make these approaches viable, future research must focus on enhancing scalability. Finally, the model assumes minimal interactions between AMRs, which simplifies congestion management. While this assumption is valid for systems with unidirectional lanes, it may not hold in large-scale environments or environments with bidirectional lanes where congestion or conflicts between AMRs are more frequent. Addressing this limitation would require a more sophisticated, multi-agent learning approach. Additionally, introducing customer order trends (e.g., seasonality), which are common in certain e-commerce applications, could be valuable. Analyzing the model's ability to recognize and adapt to such patterns may lead to better optimization of inventory rack assignments and improve overall system efficiency.

8. Acknowledgments

The Research Foundation – Flanders (FWO) funded this research, granting the corresponding author a strategic basic research mandate (1S97322).

9. Appendices

Appendix A. Proximal Policy Optimization

PPO optimizes the policy using a surrogate objective. It employs both an actor (policy) and a critic (value function) in its updates. The core idea behind PPO is to ensure that updates do not deviate too drastically from the previous policy, thereby stabilizing the learning process.

1. Actor Policy Update (Gradient of Surrogate Loss)

The policy π_θ aims to maximize the expected cumulative reward. The actor's policy update is based on the following surrogate objective function:

$$\nabla L^{CLIP}(\theta) = E[\nabla(min(q(\theta)\hat{A}_t, clip(q(\theta),1-\epsilon,1+\epsilon)\hat{A}_t))]$$

Where:

- $q(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$ is the probability ratio between the new policy π_θ and the old policy π_{old} for a given state-action pair (s_t, a_t).

- $\hat{A}_t$ is the advantage estimate at time *t*, which measures how much better the action a_t taken at state s_t is compared to the average action.
- clip($q(\theta)$,1- $\epsilon, 1+\epsilon$) is the clip function, which limits the value of the probability ratio $q(\theta)$, ensuring that the update does not exceed a threshold ϵ in either direction.

2. Critic Value Function Update (Gradient of Value Loss)

The critic estimates the value function $V(s_t)$, which is used to calculate the advantage $\hat{A}_t = \hat{G}_t - V(s_t)$. The critic is updated by minimizing the following loss function:

$$\nabla L_\theta(\phi) = E[(V_\theta(s_t) - \hat{G}_t)^2]$$

Where:

- $V_\theta(s_t)$ is the estimated value of state s_t by the critic.
- $\hat{G}_t$ is the return at time *t*, which is computed from the rewards received and a discount factor γ.

The hyperparameters for the Proximal Policy Optimization (PPO) algorithm were selected based on established best practices and fine-tuned through empirical experimentation to suit the proposed model's dynamics. The learning rate of 0.0003 was chosen to ensure stable convergence, preventing large weight update that could cause instability and balance exploration and exploitation. The discount factor was set to 0.99 to appropriately weigh long-term and short-term rewards. Lowering this value led to an overemphasis on immediate rewards, which negatively impacted decision-making for task such as storage location assignment. Similarly, the clip range was reduced from 0.2 to 0.1, which improved stability by limiting the magnitude of policy updates and preventing overcorrection.

Table 9: Hyperparameters used for the PPO model.

Hyperparameter	Value
Hidden layers	2
Optimizer	Adam
Discount factor	0.99
Clip range	0.1
Value network coefficient	0.5
Learning rate	0.0003
Batch size	64
Episodes	10000
Time steps	10000

10. References

K. Arulkumaran, M.P. Deisenroth, M. Brundage, & A.A. Bharath (2017). Deep Reinforcement Learning: A Brief Survey. IEEE Signal Process. 34(6): 26-38. https://doi.org/10.1109/MSP.2017.2743240

M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, & O. Bachem (2020). What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *Arxiv.org/abs/2006.05990*. https://doi.org/10.48550/arXiv.2006.05990

K. Azadeh, R. De Koster, & D., Roy (2019). Robotized and Automated Warehouse Systems: Review and Recent Developments. *Transportation Science*, 53(4): 917-945. https://doi.org/10.1287/trsc.2018.0873

N. Boysen, D. Briskorn, & S. Emde (2017). Parts-to-picker Based Order Processing in a Rack Moving Mobile Robots Environment, *European Journal of Operational Research*, 262: 550-562. https://doi.org/10.1016/j.ejor.2017.03.053

N. Boysen, R. De Koster, & F. Weidinger (2019). Warehousing in the E-commerce Era: A survey. *European Journal of Operational Research*, 277(2): 396-411. https://doi.org/10.1016/j.ejor.2018.08.023

N. Boysen, S. Schwerdfeger, & K. Stephan (2022). A review of synchronization problems in parts-to-picker warehouses. *European Journal of Operational Research*, 307(3): 1374-1390. https://doi.org/10.1016/j.ejor.2022.09.035

B. Cals, Y. Zhang, R. Dijkman, & C. Van Dorst (2021). Solving the Online Batching Problem Using Deep Reinforcement Learning. *Computers and Industrial Engineering*, 156. https://doi.org/10.1016/j.cie.2021.107221

B. Cheng, T. Xie, L. Wang, Q. Tan, & X. Cao (2024). Deep Reinforcement Learning Driven Cost Minimization for Batch Order Scheduling in Robotic Mobile Fulfillment Systems. *Expert Systems with Applications*, 255: 124589. https://doi.org/10.1016/j.eswa.2024.124589

T., Ding, Y., Zhang, Z., Wang, and X., Hu. (2024). Velocity-based rack storage location assignment for the unidirectional robotic mobile fulfillment system. Transportation Research Part E: Logistics and Transportation Review 186: 103533. https://doi.org/10.1016/j.tre.2024.103533.

J. Enright, P.R. Wurman (2011). Optimization and Coordinated Autonomy in Mobile Fulfillment Systems. *In Proceedings of the AAAI workshop on automated action planning for autonomous mobile robots*, 33-38.

A. Gharehgozli, & N. Zaerpour (2020). Robot Scheduling for Pod Retrieval in a Robotic Mobile Fulfillment System. *Transportation Research: Part E*, 142. https://doi.org/10.1016/j.tre.2020.102087

T. Gabel, & M. Riedmiller (2012). Distributed Policy Search Reinforcement Learning for Job Shop Scheduling Tasks. *International Journal of Production Research*, 50 (1): 41-61. https://doi.org/10.1080/00207543.2011.571443

S. Huang, & S. Ontañón (2020). A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. *Arxiv.org/abs/2006.14171*. https://doi.org/10.48550/arXiv.2006.14171

G. Jiao, H. Li, & M. Huang (2023). Online Joint Optimization of Pick Order Assignment and Pick Pod Selection in Robotic Mobile Fulfillment Systems. *Computers and Industrial Engineering*, 175: 108856. https://doi.org/10.1016/j.cie.2022.108856

T. Lamballais, D. Roy, & R. De Koster (2020). Inventory Allocation in Robotic Mobile Fulfillment Systems. *IISE Transactions*, 52(1): 1-17. https://doi.org/10.1080/24725854.2018.1560517

T. Lamballais, D. Merschformann, D. Roy, R. De Koster, K. Azadeh, & L. Suhl (2022). Dynamic Policies for Resource Reallocation in a Robotic Mobile Fulfillment System with Time-varying Demand. *European Journal of Operational Research*, 300(3): 937-952. https://doi.org/10.1016/j.ejor.2021.09.001

K. Li, T. Liu, R. Kumar, & X. Han (2024). A Reinforcement Learning-based Hyper-heuristic for AGV Task Assignment and Route Planning in Parts-to-Picker Warehouses. *Transportation Research Part E*, 185: 103518. https://doi.org/10.1016/j.tre.2024.103518

J. Lu, C. Ren, Y. Shao, J. Zhu, & X. Lu (2023). An Automated Guided Vehicle Conflict-Free Scheduling Approach Considering Assignment Rules in a Robotic Mobile Fulfillment System. *Computers and Industrial Engineering*, 176: 108932. https://doi.org/10.1016/j.cie.2022.108932

S. Luo (2020). Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning. *Applied Soft Computing Journal*, 91: 106208. https://doi.org/10.1016/j.asoc.2020.106208

R. Krenzler, L. Xie, & Hi, Li (2018). Deterministic Pod Repositioning Problem in Robotic Mobile Fulfillment Systems. ArXiv ID: arXiv:1810.05514v1.

M. Merschformann, T. Lamballais, M. De Koster, & L. Suhl (2019). Decision Rules for Robotic Mobile Fulfillment Systems. *Operations Research Perspectives*, 6: 100-128. https://doi.org/10.1016/j.orp.2019.100128

T. Nguyen, N. Nguyen, & S. Nahavandi (2020). Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions, and Applications. *IEEE Transactions on Cybernetics*, 50 (9): 3826-3839. https://doi.org/10.1109/TCYB.2020.2977374

A. Rimélé, P. Grangier, M. Gamache, M. Gendreau, L.M. Rousseau (2021). E-commerce Warehousing: Learning a Storage Policy. ArXiv ID: arXiv: 2101.08828

D. Roy, S. Nigam, R. De Koster, I. Adan, & J. Resing (2019). Robot-storage Zone Assignment Strategies in Mobile Fulfillment Systems. *Transportation Research Part E*, 122: 119-142. https://doi.org/10.1016/j.tre.2018.11.005

J. Schulman, F. Wolski, D. Prafulla, A. Radford, & O. Klimov (2017). Proximal Policy Optimization Algorithms. arXiv preprint, arXiv:1707.06347.

D. Shi, W. Fan, Y. Xiao, T. Lin, & C. Xing (2020). Intelligent Scheduling of Discrete Automated Production Line via Deep Reinforcement Learning. *International Journal of Production Research*, 58 (11): 3362-3380. https://doi.org/10.1080/00207543.2020.1717008

H. Tang, A. Wang, F. Xue, J. Yang, & Y. Cao (2021). A Novel Hierarchical Soft Actor-Critic Algorithm for Multi-Logistics Robots Task Allocation. *IEEE Access*, 9: 42568-42582. https://doi.org/10.1109/ACCESS.2021.3062457

P. Tassel, M. Gebser, & K. Schekotihin (2020). Reinforcement Learning Environment for Job Shop Scheduling Problems. *International Journal of Computer Information Systems and Industrial Management Applications*, 12: 231-238. https://doi.org/10.48550/arXiv.2104.03760

S. Teck, & R. Dewil (2022). A Bi-level Memetic Algorithm for the Integrated Order and Vehicle Scheduling in a RMFS. *Applied Soft Computing*, 121: 108770. https://doi.org/10.1016/j.asoc.2022.108770

S. Teck, P. Vansteenwegen, & R. Dewil (2023). An Efficient Multi-Agent Approach to Order Picking and Robot Scheduling in a Robotic Mobile Fulfillment System. *Simulation Modelling Practice and Theory*, 127: 102789. https://doi.org/10.1016/j.simpat.2023.102789

C. A. Valle, & J. E. Beasley (2020). Order Allocation, Rack Allocation and Rack Sequencing for Pickers in a Mobile Rack Environment. *Computers and Operations Research*, 125. https://doi.org/10.1016/j.cor.2020.105090

J. Van den Berg, & A. Gademann (2000). Simulation Study of an Automated Storage/Retrieval System. *International Journal of Production Research*, 38(6): 1339-1356. https://doi.org/10.1080/002075400188889

F. Weidinger, & N. Boysen (2018a). Scattered Storage: How to Distribute Stock Keeping Units All Around a Mixed-Shelves Warehouse. *Transportation Science*, 52(6): 1412-1427. https://doi.org/10.1287/trsc.2017.0779

F. Weidinger, & N. Boysen (2018b). Storage Assignment with Rack-Moving Mobile Robots in KIVA Warehouses. *Transportation Science*, 52(6): 1479-1495. https://doi.org/10.1287/trsc.2018.0826

L. Xie, N. Thieme, R. Krenzler, & H. Li (2021). Introducing Split Orders and Optimizing Operational Policies in Robotic Mobile Fulfillment Systems. *European Journal of Operational Research*, 288(1): 80-97. https://doi.org/10.1016/j.ejor.2020.05.032

X. Yang, G. Hua, L. Hu, T. Cheng, & A. Huang (2021). Joint Optimization of Order Sequencing and Rack Scheduling in the Robotic Mobile Fulfilment System. *Computers and Operations Research*, 135: 105467. https://doi.org/10.1016/j.cor.2021.105467

R. Yuan, S. Graves, & T. Cezik (2019). Velocity-Based Storage Assignment in Semi-Automated Storage Systems. *Production and Operations Management*, 28(2): 354-373. https://doi.org/10.1111/poms.12925

S. Zhang, D. Zhuge, Z. Tan, & L. Zhen (2022). Order Picking Optimization in a Robotic Mobile Fulfillment System. *Expert Systems with Applications*, 209: 118338. https://doi.org/10.1016/j.eswa.2022.118338

X. Zhou, X. Shi, W. Chu, J. Jiang, L. Zhang, & F. Deng (2024). Learning to Solve Multi-AGV Scheduling Problem with Pod Repositioning Optimization in RMFS. *International Conference on Industrial Technology*. http://dx.doi.org/10.1109/ICIT58233.2024.10541015

Y. Zhuang, Y. Zhou, Y. Yuan, X. Hu, & E. Hassini (2022a). Order Picking Optimization with Rack-moving Mobile Robots and Multiple Workstations. *European Journal of Operational Research*, 300(2): 527-544. https://doi.org/10.1016/j.ejor.2021.08.003

Y. Zhuang, Y. Zhou, E. Hassini, Y. Yuan, & X. Hu (2022b). Rack Retrieval and Repositioning Optimization Problem in Robotic Mobile Fulfillment Systems. *Transportation Research Part E*, 167: 102920. https://doi.org/10.1016/j.tre.2022.102920

B. Zou, Y. Gong, X. Xu, & Z. Yuan (2017). Assignment Rules in Robotic Mobile Fulfillment Systems for Online Retailers. *International Journal of Production Research*, 55(20): 6175-6192. https://doi.org/10.1080/00207543.2017.1331050