Dispatching AGVs with Battery Constraints using Deep Reinforcement Learning

Computers & Industrial Engineering 187 (2024) 109678

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Dispatching AGVs with battery constraints using deep reinforcement learning

Nitish Singh, Alp Akcay *, Quang-Vinh Dang, Tugce Martagan, Ivo Adan

Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

ARTICLE INFO

Keywords:
Dispatching
Automated guided vehicles
Deep reinforcement learning
Mixed-integer linear programming

ABSTRACT

This paper considers the problem of real-time dispatching of a fleet of automated guided vehicles (AGVs) with battery constraints. AGVs must be immediately assigned to transport requests, which arrive randomly. In addition, the AGVs must be repositioned and recharged, awaiting future transport requests. Each transport request has a soft time window with late delivery incurring a tardiness cost. This research aims to minimize the total costs, consisting of tardiness costs of transport requests and travel costs of AGVs. We extend the existing literature by making a distinction between parking and charging nodes, where AGVs wait idle for incoming transporting requests and satisfy their charging needs, respectively. Also, we formulate this online decision-making problem as a Markov decision process and propose a solution approach based on deep reinforcement learning. To assess the quality of the proposed approach, we compare it with the optimal solution of a mixed-integer linear programming model that assumes full knowledge of transport requests in hindsight and hence serves as a lower-bound on the costs. We also compare our solution with a heuristic policy used in practice. We assess the performance of the proposed solutions in an industry case study using real-world data.

1. Introduction

There is an increasing shift in the industry towards data-driven manufacturing systems, in which all devices possess their own 'intelligence'. This intelligence enables a free flow of data and direct communication of devices within production and logistics. The literature often refers to a factory embedding these intelligent features as 'smart factory' (Vuksanović, Ugarak, & Korčok, 2016). One of the ambitions of smart factories is to control operations at shop floor with little human interference (De Ryck, Versteyhe, & Debrouwere, 2020). To support this ambition, mobile robots (henceforth referred to as AGVs) promise a flexible, efficient and dynamic means of transporting materials. In this setting, we define AGVs as driverless and programmable vehicles that transport materials in facilities such as manufacturing plants and warehouses.

An example of a smart factory is the Brainport Industries Campus (BIC), a new high-tech campus in Eindhoven, Netherlands. The BIC is currently testing a fully self-driving logistics system (Brainport Industries Campus, 2020), increasing the need for new planning algorithms (Singh, Dang, Akcay, Adan, & Martagan, 2022). Inspired by the smart factory initiatives at the BIC, we consider the problem of real-time control of a fleet of AGVs with battery constraints. Our optimization problem consists of three interdependent decisions made in real-time: dispatching of AGVs, assigning transport requests to AGVs, and recharging AGVs. Moreover, our problem setting involves several distinguishing features that challenge the real-time decision making:

(i) Aspects related to AGV charging. Battery management is a critical factor in AGV dispatching. Electric AGVs have a limited battery, constraining their availability. Moreover, AGV dispatching rules need to distinguish the charging (electrified) and parking (non-electrified) stations. *Charging* stations are equipped with an infrastructure to charge AGVs, where AGVs can either charge or park without charging. On the other hand, *parking stations* do not have an infrastructure for battery charging, and hence AGVs can only park.

(ii) Capacity and time restrictions. Capacity and time constraints can significantly affect the performance of AGV systems (Vis, 2006). In common practice, the capacities of the charging and parking stations are limited (because of space or financial limitations). Moreover, transport requests are typically associated with a time window, leading to high tardiness costs for late deliveries.

(iii) Spatial and temporal patterns. Spatial (e.g., the specific location of an AGV in a factory) and temporal (e.g., job arrival charecteristics during the planning horizon patterns) present an important information for AGV dispatching decisions. For example, certain locations on the shop floor can become busier than others during specific time periods in a day. However, this spatial and temporal information is dynamic and involves uncertainty, as it depends on the complex

* Corresponding author.
E-mail addresses: n.singh1@tue.nl (N. Singh), a.e.akcay@tue.nl (A. Akcay), q.v.dang@tue.nl (Q.-V. Dang), t.g.martagan@tue.nl (T. Martagan), i.adan@tue.nl (I. Adan).

https://doi.org/10.1016/j.cie.2023.109678
Received 27 June 2023; Received in revised form 26 September 2023; Accepted 9 October 2023
Available online 18 October 2023

interactions between AGV dispatching, assignment, and recharging decisions. To achieve an effective fleet management system, there is an increasing need for dynamically learning and reacting to these spatial and temporal patterns over time.

To capture the aforementioned problem features, we build an optimization model based on Markov decision processes (MDP), and develop a deep reinforcement learning (DRL) approach to solve industry-size problem instances. Two important advances in DRL are autoencoders (Bank, Koenigstein, & Giryes, 2020) and noisy nets (Fortunato et al., 2017). Autoencoders are neural nets that learn to encode input data into compressed formats, extracting and representing information from input data of high dimensions into learnt lower dimensions. Noisy nets introduce stochastic behavior directly into the parameters of the DRL policy. By introducing randomness into the parameters, the policy's exploration is not driven by the experimenter, rather, a problem-specific exploration strategy is produced.

The main contributions and novelty of this work are summarized as follows. (i) We develop an MDP model to optimize three interdependent decisions on dispatching, assigning transport requests, and recharging of AGVs. As a novel feature, the MDP model simultaneously captures the practically-relevant aspects related to AGV charging, capacity, and time restrictions, and information on spatial and temporal patterns. (ii) We propose a DRL-based framework which uses autoencoders and NoisyNets to support AGV dispatching decisions in which AGVs respond in real-time to transport requests but have to comply with capacity constraints at stations (i.e., at most one AGV can be accommodated at each station). (iii) We extend a mixed-integer linear programming (MILP) model proposed by Singh et al. (2022) to solve our problem in hindsight with the perfect information assumption, where the arrival times and all other characteristics of the transport requests are known in advance. The extended model can serve as a benchmark for the DRL policy (i.e., upper bound on the optimal reward) for small case instances. We also compare the DRL policy to two other benchmarks, i.e., the random allocation policy and a practitioner's heuristic used by our industry partner. (iv) We illustrate the use of the developed models using an industry case study from Klein Mechanische Werkplaats Eindhoven (KMWE), which is one of the high-tech companies located in the BIC. Numerical analysis shows that our DRL framework significantly outperforms the benchmark heuristics, as it exploits the spatial and temporal information on AGVs and transport requests.

Industry feedback indicates that most companies rely on the prepackaged fleet management software provided by the AGV manufacturers. The existing software is usually composed of simple dispatching rules. However, research reveals that while dispatching rules are simple and easy to implement, their performance is limited by the intuitive reasoning used to design them (Le-Anh & De Koster, 2006). Therefore, there is an increasing need for comprehensive optimization models and flexible planning algorithms to manage AGV fleets. The MDP model and DRL approach developed in this research are generalizable, and can be broadly applied in situations when a central fleet owner has to commit to capacity constraints at nodes while dynamically serving transport requests.

The remainder of this paper is organized as follows. We summarize relevant literature in Section 2. The problem is formulated in Section 3. We present the MDP model in Section 4 and present our solution approach in Section 5. Benchmark policies are presented in Section 6. In Section 7, the industry case study, computational results, and sensitivity analyses are provided. Finally, we provide conclusions and future research directions in Section 8.

2. Literature review

Dispatching can be considered as a crucial and also one of the most challenging design aspects of an AGV management and control system (De Ryck et al., 2020). Le-Anh and De Koster (2006) defined dispatching as selecting and assigning tasks to vehicles, including the routes that vehicles travel to accomplish these tasks. If all tasks are known prior to the planning period, the problem can be solved simultaneously. Literature often refers to this type of problems as static or offline dispatching problems. In practice, tasks are often unknown in advance and are revealed or modified over time, i.e. dispatching relies on real-time information. Hence, the dispatching problem needs to be solved sequentially. Literature often refers to this type of problem as a real-time, dynamic, or online dispatching problem (Vis, 2006). This research considers a dynamic AGV dispatching problem.

AGV dispatching can be considered as a special case of traditional problems such as the Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), and dial-a-ride problem (DARP). Nonetheless, AGV dispatching contains shorter travel distances, shorter planning horizons, higher traffic density, vehicle interference, and battery-charging problems (Le-Anh & De Koster, 2006). Similar to TSP, VRP, and DARP, AGV dispatching belongs to the class of NP-hard problems. A widely-known characteristic of NP-hard problems is that if the number of tasks and vehicles grows, the number of solutions will be enormous. Hence, only for cases with a relatively small number of vehicles and tasks, an exact algorithm can find an optimal solution within a polynomial (reasonable) amount of time. A widely used stream of heuristics that is well-suited to cope with increasing problem size is the stream of dispatching rules. Most dispatching rules used in literature are single-attribute rules, which dispatch vehicles based on one parameter. De Koster, Le-Anh, and Van Der Meer (2004) evaluated several well-known single-attribute dispatching rules in different settings to reduce pallet waiting times. They found that, in general, distance-based dispatching rules such as Nearest Workstation First (NWF) and Nearest Vehicle First (NVF) perform significantly better compared to time-based dispatching rules such as Modified First Come First Served (MODFCFS).

Although single-attribute dispatching rules have been widely studied in the literature, multiple studies have shown that multi-attribute dispatching rules outperform these single-attribute rules (Bilge et al., 2006; Confessore, Fabiano, & Liotta, 2013; Guan & Dai, 2009; Le-Anh & De Koster, 2004, 2005; Singh, Sarngadharan, & Pal, 2011; Zamiri Marvizadeh & Choobineh, 2014). Le-Anh and De Koster (2004) proposed a multi-attribute dispatching rule (Multi-Att) considering the empty vehicle travel time and balancing workload among workstations for systems with a large number of vehicles. They also modified this Multi-Att rule by adding a power coefficient, creating a second dispatching rule: Multi-Mod. Le-Anh and De Koster (2005) developed the combined dispatching rule Combi. This rule combines multi-attribute dispatching based on a waiting time component and an empty-travel distance component with vehicle reassignment while parking. Singh et al. (2011) studied the AGV scheduling problem for distributing uniform materials from a truck dock to machining units in an automotive machine shop. They proposed and assessed various dispatching rules (e.g., single destination, multiple destinations with priorities) for assigning jobs to AGVs. Confessore et al. (2013) proposed a vehicle-initiated dispatching rule for dynamically assigning transportation tasks to AGVs, taking into account a number of factors simultaneously, such as pick, drop, and travel times, battery recharging, capacity constraints, and congestion and error issues. Zamiri Marvizadeh and Choobineh (2014) proposed several AGV dispatching algorithms to balance the workload content among the work centers based on their input and output queue sizes. In brief, these rules often encompass several attributes, such as travel time, waiting time, distance to pick-up locations, and capacity of input and/or output buffers.

Dispatching rules are widely known to be simple and easy to implement in practice. Still, at the same time, they are also criticized for having a myopic view, i.e. not learning from patterns in historical data. A method that has proven to do a much better job learning from historical data is reinforcement learning (RL). Recently, RL, and in particular deep reinforcement learning (DRL), has also acquired attention and appreciation in the field of routing problems. Both Kamoshida and Kazama (2017) and Rhazzaf and Masrour (2021) introduced a route

planning method for an AGV order picking system using a model-free Deep Q-network (DQN). Similar to Rhazzaf and Masrour (2021) and Zhang, Sang, Li, Han, and Duan (2022), we use a multi-objective reward function comprising of vehicle travel times and request tardiness in our study. Additionally, we use novel advancements over the vanilla DQN approach of other studies for decision making.

Missing from all previous works are the challenges that arise with the electrification of vehicles. The limited but growing stream of the Electric Vehicle Routing Problem (EVRP) discusses these challenges. Due to the use of a battery instead of a combustion engine, EVs have a limited driving range compared to traditional Internal Combustion Engine Vehicles (ICEVs). Hence, (re)charging needs to be considered in routing problems' operational and tactical decision making. Many studies employ full recharging (FR) of batteries once vehicles need to recharge. Keskin and Çatay (2016) relaxed this FR restriction by introducing the Electrical Vehicle Routing Problem with Time Windows and Partial Recharge (EVRPTW-PR). The authors showed that the partial recharging strategy might save from recharging time, allowing the vehicle to catch an otherwise missed customer time window. In our study, we allow partial recharging of AGVs.

An often neglected area within the EVRP is the congestion that may arise at charging stations (CS) with a finite capacity. Many studies consider chargers to be available at every station (Pettit, Glatt, Donadee, & Petersen, 2019; Shi, Gao, Wang, Yu, & Ioannou, 2019). Our work distinguishes from this setting in two ways. The first extension entails considering stations with chargers and stations without chargers. Therefore, the charging component becomes less straightforward than in standard literature. The second extension is a result of relaxing the assumption of infinite capacity at the stations. We model each station with fixed capacity of one, i.e., at most one vehicle can be at any station at any time. Sweda, Dolinskaya, and Klabjan (2017) addressed a path-finding problem in which CSs are unavailable with a certain probability. Therefore, the vehicle dynamically decides upon its path and when to charge. Related to this problem, Froger, Mendoza, Jabali, and Laporte (2017) built a two-stage matheuristic for a situation in which stations have a limited number of chargers (one, two, or three). In the first stage, a metaheuristic builds a pool of routes (bases on iterated local search), while not yet considering the capacity of the CSs.

To conclude this literature review, we selected the studies that we consider most similar to this research. Pettit et al. (2019) employed a single-agent Trust Region Policy Optimization (TRPO) DRL algorithm to learn an agent when to serve a request and when to recharge its battery. Wei, Yan, Zhang, Xiao, and Wang (2022) used DQN algorithm with a self-attention mechanism to dispatch an idle workstation to an AGV. Hu, Jia, He, Fu, and Liu (2020) developed a DQN algorithm to select a suitable dispatching rule. Similar to the works of Holler et al. (2019) and Lin, Zhao, Xu, and Zhou (2018), Shi et al. (2019) employed a DRL algorithm with decentralized learning and centralized decision making, allowing both scalability and coordination. The objective of this work was to minimize customer waiting times, electricity costs, and operating costs. Kullman, Cousineau, Goodson, and Mendoza (2022) developed two DDQN policies aiming to maximize the fleet's revenue as a whole. Nevertheless, to the best of our knowledge, our paper is the first to consider real-time transport requests with soft time windows, partial charging at charging stations, fleet repositioning, capacity constraint at stations, and distinction in station types. We describe the considered problem in detail in the next section.

3. Problem description

We consider a central operator that controls a fleet of AGVs represented by the set V. The AGVs serve the transport requests that randomly arrive during a time horizon of length H. We let R denote the set of these transport requests. Since the transport requests arrive randomly, the set R is unknown to the decision maker at the beginning of the time horizon. The AGVs travel over a two-dimensional space that includes a set of nodes denoted by N. A node can either be a pickup-and-delivery, parking, or charging node. We let O denote the set of pickup-and-delivery nodes, P denote the set of parking nodes, and C denote the set of charging nodes. The parking and charging nodes are also referred to as the station nodes and denoted with S, i.e., $S = P \cup C$. We assume that at most one AGV can be at a station node at any time. The pickup and delivery nodes of transport request $r \in R$ are denoted with n_r^S and n_r^D, respectively, where n_r^S, $n_r^D \in O$. Additionally, each transport request has a time window $[e_r, l_r]$, where e_r is the arrival time and l_r is the latest delivery time of transport request r. The transport request $r \in R$ has a handling time h_r for loading at a pickup node and also for unloading at a delivery node.

We refer to the amount of delay in the completion of transport request r as the tardiness of request r, and denote it with τ_r. The tardiness is penalized by a tardiness cost per time unit, denoted with c_τ. The higher the number of AGV movements, the more the operational and maintenance-related costs for the AGVs. Therefore, unnecessary travel of an AGV is undesired in real life. Thus, we introduce a travel cost per time unit, denoted with c_δ, representing the cost charged per time unit for each traveling AGV.

During AGV travel, the battery level discharges proportionally to the travel time with a discharging rate d. Battery also discharges at the same rate during unloading and loading activities. If an AGV is located at a node and has a charge level less than a critical threshold $\underline{b}$ percent, it must be instructed to recharge, and the recharging duration should be long enough to allow the charge level to reach at least this critical threshold. The charging rate of an AGV is denoted with c. The critical threshold $\underline{b}$ is a prespecified parameter for a given network to assure that an AGV can reach any destination and a charging station right afterwards if needed. We emphasize that an AGV is not permitted to visit a charging station while performing a transport request (i.e., it first needs to complete the delivery). The AGVs can be routed to any station node, and we refer to this as the repositioning action. The repositioning action moves an AGV to an appropriate charging station to recharge its battery or moves the AGV to an appropriate parking station to wait idle. An AGV is allowed to stay in a charging station even after it is fully recharged.

We make certain assumptions to comply with real-world scenarios. First of all, when a new transport request arises, the operator responds immediately, either by assigning it to an AGV or by rejecting it. A transport request may be rejected by the central operator in anticipation of more jobs with less tardiness in the future, and this leads to a rejection cost denoted by c_d. The assignment of a transport request to an AGV is considered as feasible when the AGV is either idle, traveling to the delivery node of the previously assigned transport request, or charging at or repositioning to one of the station nodes. In addition, for the assignment to be feasible, the AGV's battery level must be at least $\underline{b}$ at the time of assignment. Once assigned, we do not allow modifying a transport request anymore. If there is a transport request but there is no feasible AGV, this situation is referred to as an infeasibility, and the request must be rejected. The AGVs are assumed to be equipped with hardware that assures collision-free movements on the shop floor.

The objective of the central operator is to minimize the expected total cost during the time horizon H. The central operator may prioritize travel cost or the tardiness cost by adjusting the so-called weight coefficients η_1 and η_2 that correspond to the travel and tardiness costs, respectively. When a new transport request enters the system, the decision maker needs to decide whether to accept or reject the incoming transport request, and if accepted, it is decided which AGV is assigned to this transport request. When an AGV finishes the delivery of a transport request, the decision maker is allowed to take a repositioning action for each AGV, i.e., to decide whether an AGV needs to be repositioned, and if so, to which station node.

4. Model

We formulate the problem described in Section 3 as a finite-horizon, continuous-time Markov Decision Process (MDP) model. As common in the reinforcement learning literature, we refer to the period that starts at time 0 and ends at time H as an episode.

Decision epochs. The decision epoch $k \in \{1, \dots, K\}$ is the moment a new transport request enters the system or an AGV finishes the delivery of a transport request, whichever occurs first. The value of K is not known upfront due to the randomness in the system (i.e., the number of transport requests that will arrive during an episode is not known at the beginning of the episode).

States. The system state at decision epoch k is denoted by $s_k \in S$ with S denoting the state space. In general, we represent the system state as $s = \left(s^t \oplus s^r \oplus s^V\right)$, where

$$s^t = \begin{bmatrix} s_1^t \\ s_2^t \end{bmatrix} = \begin{bmatrix} \text{Time of the day (in seconds)} \\ \text{Day of the week (one-hot encoded)} \end{bmatrix},$$

$$s^r = \begin{bmatrix} s_1^r \\ s_2^r \\ s_3^r \\ s_4^r \end{bmatrix} = \begin{bmatrix} \text{Binary variable indicating the existence of a new transport request} \\ \text{The pickup location of the new transport request} \\ \text{The delivery location of the new transport request} \\ \text{Latest delivery time of the new transport request} \end{bmatrix},$$

and $s^V = (s^1 \oplus \cdots \oplus s^{|V|})$ with

$$s^v = \begin{bmatrix} s_1^v \\ s_2^v \\ s_3^v \end{bmatrix} = \begin{bmatrix} \text{The charge level of AGV } v \\ \text{Location of AGV } v \\ \text{AGV } v\text{'s currently assigned tasks} \end{bmatrix}$$

and $\oplus$ denoting the concatenation operator.

The time component s^t contains the time of the day and the day of the week. The request component s^r captures whether the current decision epoch is triggered due to a new transport request, and if so, the information related to this transport request. To be specific, if there is a new transport request, then s_1^r is set to 1, and s_2^r and s_3^r, which are two-dimensional vectors, are set to the Cartesian coordinates of the pickup and delivery nodes of the transport request, respectively. Finally, s_4^r is set to the latest delivery time of this transport request. If the decision epoch is not triggered by a new transport request, then s_1^r is equal to 0, and the state variables s_2^r, s_3^r and s_4^r are all set to $\emptyset$. The state of AGV v is captured by s^v, where s_1^v represents the charge level of AGV v. The variable s_2^v is a two-dimensional vector representing the Cartesian coordinates of AGV v, and the variable s_3^v represents the task list of AGV v. We let s_3^v consist of three elements, i.e., $s_3^v = (s_{31}^v, s_{32}^v, s_{33}^v)$ where the state component s_{31}^v represents the current task, and s_{32}^v and s_{33}^v are planned future tasks of AGV v (note that an AGV can have at most two planned future tasks at any moment according to the problem description in Section 3). Specifically, the state component s_{3i}^v itself consists of three elements, i.e., $s_{3i}^v = (s_{3i1}^v, s_{3i2}^v, s_{3i3}^v)$, where s_{3i1}^v represents the type of task i, and s_{3i2}^v and s_{3i3}^v represent the Cartesian coordinates of the start and end locations of task i, respectively. The task type can be one of the following: idle when AGV v is located at a parking node (0), charging when AGV v is located at a charging node (1), unloaded travel when AGV v is repositioning to a station node (2), unloaded travel to a transport request's origin (3), loaded travel when AGV v is on its way to a transport request's delivery node (4). That is, $s_{3i}^v \in \{0, 1, \dots, 4\}$.

State initialization. Let s_0 denote the system state at the beginning of each episode. In this state, there is no outstanding transport request, and all of the AGVs are idle at a certain station node with some percentage amount of battery level, denoted by b_v for AGV $v \in V$. The value of b_v is assumed to be no less than $\underline{b}$ for all $v \in V$, meaning that no recharging is needed for any AGV at the beginning of an episode.

Actions. The set of actions in a decision epoch depends on whether the decision epoch is triggered by the arrival of a new transport request or the completion of a transport request. If the decision epoch is triggered by the arrival of a new transport request, we denote the action taken by $a^r \in \{0\} \cup V$. If no AGV is assigned to the incoming transport request (either because there is no feasible AGV assignment or the transport request is rejected), then a^r is equal to 0. On the other hand, a^r is set equal to v if AGV $v \in V$ is assigned to the incoming transport request. As notational convention, we let $a^r = \emptyset$ if the decision epoch is triggered by the completion of a transport request. In this type of decision epoch, repositioning decisions must be made for all the AGVs which are idle at a parking node, just completed the delivery of a load, or at a charging node with battery level no less than $\underline{b}$. We let $\mathbf{a}^V = (a^1, \dots, a^V)$ denote the actions taken in a decision epoch triggered by the completion of a transport request, where $a^v \in \{0\} \cup S$ represents the station node to which AGV v is re-positioned. Here $a^v = 0$ means no repositioning action is taken for AGV v. We let $\mathbf{a}^V = \emptyset$ if the decision epoch is not triggered by the completion of a transport request but with the arrival of a new transport request.

State transitions & Costs. At decision epoch k, the agent observes the state s_k and then takes the action $\mathbf{a}_k = (a^r, \mathbf{a}^V)$. Subsequently, the decision epoch $k' = k + 1$ is triggered when a new transport request arrives or an AGV completes its delivery, whichever occurs first (given that the end of the horizon H is not reached yet). We next describe how the state s_k is updated to the next state $s_{k'}$ in the subsequent decision epoch k'.

First, the time component s^t is updated with the current time and day of the week at decision epoch k'. The request state s^r is updated depending on the event triggering the new epoch. To be specific, if a new transport request is triggering the next epoch k', s_1^r is set to 1, and the request's origin, destination, and the latest delivery time are captured in s_2^r, s_3^r, and s_4^r, respectively. Otherwise, s_1^r is set to 0, and s_2^r, s_3^r and s_4^r are all set to $\emptyset$.

Next, the AGV states S^V are updated as follows. First of all, the charge level and location of each AGV is updated based on what has happened between decision epoch k and k', i.e., for each $v \in V$, s_1^v and s_2^v are updated by using s^t and the information captured in the (not yet updated) state variable s_3^v that correspond to the tasks of AGV v. Next, the state variable s_3^v is updated for each $v \in V$. For each AGV, there are three possible situations at any decision epoch. It can either be assigned to a transport request, be repositioned to a station node, or receive no action (i.e., neither assigned to a transport request nor repositioned to a station node):

- When the transport request r is assigned to AGV v, the state s_3^v is updated as follows: If the AGV v is currently idle, charging or conducting an unloaded travel (i.e., its current task type is 0, 1, 2 or 3), the tasks of AGV v are set such that $s_{311}^v = 3$, $s_{312}^v = s_2^v$, $s_{313}^v = s_2^r$, $s_{321}^v = 4$, $s_{322}^v = s_2^r$, and $s_{323}^v = s_3^r$. Since there is no second future task to keep in the memory in this situation, s_{33}^v is set to $\emptyset$. If AGV v is currently conducting a loaded travel (i.e., its type is 4), then, the tasks of AGV v are set such that $s_{321}^v = 3$, $s_{322}^v = s_2^v$, $s_{323}^v = s_2^r$, $s_{331}^v = 4$, $s_{332}^v = s_2^r$, and $s_{333}^v = s_3^r$.
- When AGV v is repositioned to a station node $n \in S$ with Cartesian coordinate (n_x, n_y), the assigned tasks of AGV v are set such that $s_{311}^v = 2$, $s_{312}^v = s_2^v$, $s_{313}^v = (n_x, n_y)$. Then, the type of second task is set to $s_{321}^v = 0$ if $n \in P$, and $s_{321}^v = 1$ if $n \in P$ (i.e., the task type is retained in the state variable as either parking task or charging task depending on which station node the AGV is repositioned to) and s_{322}^v and s_{323}^v are both set to $\emptyset$ as the corresponding tasks do not include travel.
- When AGV v receives no action, s_3^v remains unchanged.

Costs. At each decision epoch, all of the costs incurred during the time interval between the last and the current decision epochs are charged. To be specific, suppose that the interval starts at decision epoch $k-1$ with state s_{k-1} and ends at decision epoch k where the action a_k is taken and the new state s_k is obtained. Then, the cost $C(s_{k-1}, a_k, s_k)$ is incurred at decision epoch k, consisting of travel, tardiness, and rejection costs. The travel cost is determined by the travel time variable $\delta(s_{k-1}, s_k)$ that reflects the total travel time of the AGV fleet between decision epochs $k-1$ and k. The tardiness cost is charged when the decision epoch k is triggered by a new transport request and the transport request is assigned to an AGV but it will be delivered after its latest delivery time. The tardiness of a transport request, which we denote with $\tau(a^r, s)$, can be precisely calculated at the time of its assignment by utilizing the characteristics of the transport request (captured in the request component s^r of the system state), the current tasks of the assigned AGV a^r (captured in the AGV component s^{a^r} of the system state, where $a^r \in V$), and the current time (captured in the time component s^t of the system state). Consequently, in addition to the travel costs, the following cost is charged at each decision epoch:

$$\beta(a^r, s) = \begin{cases} c_\tau \cdot \tau(a^r, s) & a^r \in V \\ c_d & s^r_1 = 1, a^r = 0 \\ 0 & s^r_1 = 0. \end{cases} \tag{1}$$

In Eq. (1), there are three cases. The first case represents that the incoming transport request is assigned to AGV a^r and accounts for the corresponding tardiness cost. The second case represents there is an incoming request ($s^r_1 = 1$) but it is not assigned to any AGV ($a^r = 0$), leading to the rejection cost c_d. Finally, the third case represents that the decision epoch was not triggered by a new transport request, and therefore, there is no cost related to a task assignment.

Altogether, the immediate cost function charged at decision epoch k is defined as

$$C(s_{k-1}, \mathbf{a}_k, s_k) = \eta_1 \cdot c_\delta \cdot \delta(s_{k-1}, s_k) + \eta_2 \cdot \beta(a^r_k, s_k) \tag{2}$$

for $k \in \{1, \dots, K\}$ with $a^r_k \in \{0\} \cup V$ denoting the AGV assignment action at decision epoch k. The moment that the time hits H the problem ends, and the travel cost of the AGV fleet between the last decision epoch K and the time H is incurred. This additional travel cost is given by $\eta_1 \cdot c_\delta \cdot \delta(s_K, s_{K+1})$, where s_{K+1} denotes the updated state variables at time H.

Objective Function. The objective is to learn an optimal policy $\pi^\star$ that minimizes the expected total cost during time horizon H (consisting of K decision epochs), conditional on initial state s_0. A policy π consists of a sequence of decision rules $\left(X^\pi_1(s_1), \dots, X^\pi_K(s_K)\right)$ that map state s_k at decision epoch k to a feasible action. Then the optimal policy is given by

$$\pi^\star = \min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=1}^{K} C\left(s_{k-1}, X^\pi_k\left(s_k\right), s_k\right) + \eta_1 \cdot c_\delta \cdot \delta(s_K, s_{K+1})\right] \tag{3}$$

where Π is the set of all policies.

5. Solution approach

In this section, we describe our primary solution approach for the MDP model described in Section 4. Reinforcement Learning (RL) is associated with the specific case of a MDP in which the agent does not necessarily have any prior knowledge of the environment (Wiering & van Otterlo, 2012). To optimize its policy, the agent interacts with the environment through states, actions, and a feedback mechanism of scalar rewards. The agent learns an optimal policy $\pi^\star$ (see Eq. (3)) to maximize the expected total reward. Optimal policy $\pi^\star$ corresponds to an optimal action-value function, which literature often refers to as optimal *Q-function* $q_*(s, a)$ that satisfies:

$$q_*(s, a) = \mathbb{E}\left[R(s, a) + \gamma \max_{a'} q_*\left(s', a'\right) \mid s, a\right] \tag{4}$$

In Eq. (4), we adopt the common convention in DRL literature, and cast the problem as reward maximization (instead of cost minimization) with $R(s, a)$ denoting the immediate reward obtained by taking action a in state s and the maximum expected discounted reward that can be achieved from the subsequent state–action pair (s', a'). Discount factor $\gamma \in (0, 1]$ denotes the weight we apply to future rewards. We aim to make the so-called *Q-values* (i.e., the value of a given state–action pair (s, a)) as close as possible to the right-hand side of Eq. (4). Eventually, the Q-value will converge to the optimal Q-value q_*. We obtain convergence by updating the Q-value over time to reduce the loss. Let R_t denote the reward collected in decision epoch t. We define loss as follows:

$$\begin{aligned} q_*(s, a) - q(s, a) &= \text{ loss} \\ \mathbb{E}\left[R(s, a) + \gamma \max_{a'} q_*\left(s', a'\right)\right] - \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] &= \text{ loss} \end{aligned} \tag{5}$$

In the remainder of this section, we describe the approaches taken in this study for developing the proposed agent responsible for assignment of transport request to a particular AGV and carrying out the repositioning actions. Sections 5.1 and 5.2 describe the exploration and learning strategies of our proposed agent. Section 5.3 introduces additional state variables. Section 5.4 describes the repositioning rule used in this study.

5.1. Noisy exploration strategy

While learning, the agents needs a strategy to determine what action to take. A famous exploration strategy is *epsilon greedy strategy* (Sutton & Barto, 2018). This strategy has two main components: exploration and exploitation. While exploration, the agent takes a random action, and while exploitation, it takes an action based on information already known (for example, gained during exploration), i.e., the action corresponding to the highest Q-value. Exploration is controlled by a parameter, ϵ, which is initially set to a high value (for example, equal to 1). The way exploration–exploitation trade-off works in practice is that a random value between 0 and 1 is generated at each training step. If this value is larger than ϵ, we choose an exploitation action. Otherwise, we take an exploration action. In most studies, ϵ is decayed based on a fixed schedule (for example, linearly). However, these methods, separate the mechanism of generalization from that of exploration, i.e., these methods are controlled by the experimenter, rather than based on interactions with the environment being explored. Therefore we make use of NoisyNets, first presented in Fortunato et al. (2017), for a parameterized exploration of the environment. NoisyNets utilize learned perturbations of the network weights to drive exploration. The perturbations are sampled from a noise distribution, and the variance of the perturbation is a parameter that is learned using gradients from the reinforcement learning loss function alongside the other parameters of the agent. We utilize the *Factorized Gaussian noise*, which uses an independent noise per each output and another independent noise per each input, totaling $p + q$ unit Gaussian variables (for p inputs and q outputs to the layer), and thereby limiting the computational overhead in the case of single-thread agents such as ours. Since our problem is quite complex and contains a large number of input features and actions, this exploration strategy greatly enhances the performance of our agent compared to the vanilla implementation of the D3QN agent since a NoisyNet produces a problem-specific exploration strategy as opposed to fixed exploration strategy used in standard DQN, i.e., the degree of exploration is contextual and depends on the state being explored. While more gradients are required due to the increase in the number of parameters in the linear layers of the network, the computational overhead is minimal since the weights are simple affine transformations of the noise. For an exact implementation, we refer

the reader to Fortunato et al. (2017) Further, different from Fortunato et al. (2017), we utilize the NoisyNets in the value and the advantage networks in our implementation. Henceforth, we will refer to the proposed agent as the *noisy dueling double deep Q network (ND3QN)* agent. ND3QN agent is used to assign an AGV to a new transport request. It also uses a dispatching rule to make the repositioning decisions for sending an AGV to a parking or charging node. In Section 5.4, we present this dispatching rule that takes the repositioning actions. The repositioning actions will be used by the ND3QN agent while making the assignment decisions of the incoming transport requests to AGVs. A ND3QN network uses two streams of computation, one for the state value function and the other for the advantage function (i.e., a measure of how much a certain action is a good or bad decision at a certain state). The value stream approximates the state values and the advantage stream provides the relative advantage for each action (Wang et al., 2016). We provide a general explanation of Deep Q-Learning and its variants in Appendix A.

5.2. Prioritized replay and multi-step learning

In our solution approach, we implement prioritized experience replay (PER) first proposed by Schaul, Quan, Antonoglou, and Silver (2016) which is an enhancement over uniformly sampling batches (with batch size N_{batch}) from a replay memory introduced by Mnih et al. (2015). Learning speed can be increased by sampling the experiences more frequently to enable the agent to learn more at a given time. The PER samples experiences where the agent was more surprised by (big loss-value) or experiences that are not yet sampled before (unknown loss-value). As Schaul et al. (2016) describe in their proposed framework, PER can be controlled by two parameters: α and β. The first parameter prioritizes the sampling of experiences. With $\alpha = 0$, all experiences are sampled with equal probability. However, by changing the sampling distribution, PER introduces a bias. Therefore, the authors introduce β to correct this bias. They correct the bias using importance-sampling weights that fully compensate for non-uniform probabilities. The authors anneal the amount of importance-sampling correction over time, by defining a schedule on β that reaches 1 at the end of learning. We linearly anneal β from its initial value β_0 to 1 in β_{steps} training steps.

Lastly, as presented by Sutton (1988), forward-view multi-step learning considers a learning procedure in which the correctness of a prediction is revealed more than one step after making a prediction. Matching this idea with Deep Q-Learning, we store the state of n steps ahead, denoted by s^n, in the replay memory instead of storing the subsequent state. Moreover, rather than using the reward of only the subsequent state, we use the sum of discounted rewards up to state s^n (Hessel et al., 2018).

5.3. Additional state variables

In our implementation, we introduce additional state variables derived from the available state information in the MDP formulation described in Section 4 for more efficient learning of the proposed ND3QN agent. We extend s^r by introducing state variable s^r_5, representing the distance between the transport request's origin and destination. Also, we use the information in state variables in s^v to generate state variables s^v_4, s^v_5, s^v_6, and s^v_7, where s^v_4 is set to the time at which AGV v will be available for carrying a new transport request, s^v_5 is set to the charge level at time s^v_4, and s^v_6 is set to the destination of the currently assigned transport request. If AGV v is currently idle or charging, then s^v_6 is set to the current location of AGV v. Finally, s^v_7 is set to the distance of AGV v from the origin of the request if the AGV is feasible, and it is set to -1 if the AGV is infeasible for the assignment.

Since neural networks are known to be sensitive to the scale of input features, we normalize state variables s^t_1, s^r_4, s^r_5, s^v_4, s^v_5, and s^v_7 such that they take continuous values between zero and one. Further, state components containing Cartesian coordinates are represented as

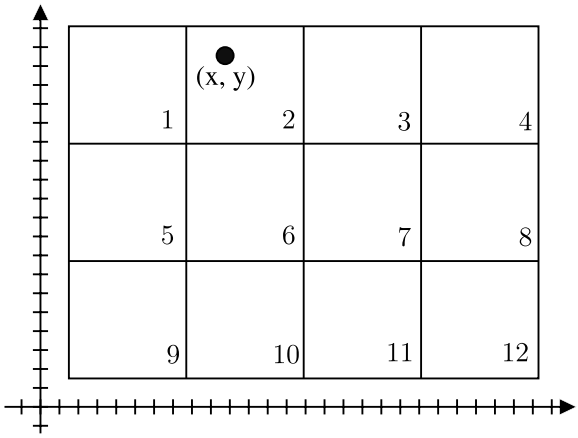

Fig. 1. Given 2D layout is sub-divided into L tiles such that Cartesian coordinates can be represented as one-hot vectors.

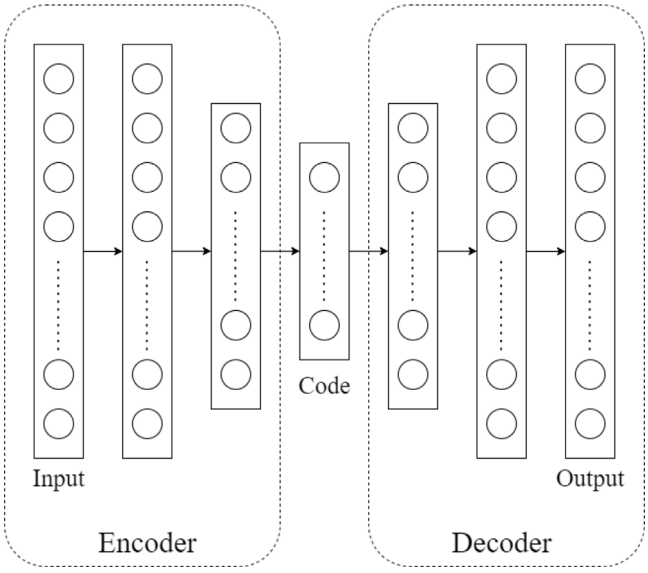

Fig. 2. Autoencoder architecture used to create code for location information.

one hot vectors by tiling the given layout (shown in Appendix C) into L tiles. Thus, each (x, y) coordinate falls inside exactly one tile. For example, the coordinate shown in Fig. 1 lies in tile 2 and is represented as $[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$. We convert this L dimensional vector representation into a U dimensional encoded representation using a trained autoencoder network, where U is typically much lesser than L (Hinton & Salakhutdinov, 2006). Details are provided in Appendix D. In short, an autoencoder is a special type of neural network that is trained to copy its input to its output. It consists of an encoder network, which in our case, takes an input and converts it into codes with a reduced dimensionality, and a decoder network, that decodes those codes back into the input (Vincent, Larochelle, Bengio, & Manzagol, 2008). This way a trained encoder network converts the input data into learned representations with much lesser dimensions, and in our case, compresses the location information from L dimensional to a U dimensional vector. Fig. 2 shows the autoencoder network used in this study.

Since it is conventional in reinforcement-learning literature to maximize rewards instead of minimizing costs, we define the reward as negation of the cost function shown in Eq. (2). Further, we normalize the reward for stable learning across problem instances of varying

scales (Mnih et al., 2015). To be specific, the corresponding reward function returns a real value r_k such that $r_k \in [0, 1]$ in each decision epoch k. We set $c_d = 0$ (i.e., the minimum reward, or equivalently, the highest possible cost) in our experiments.

5.4. Dispatching rule for repositioning decisions

A repositioning decision must be made for each AGV in the fleet when a decision epoch is triggered by the completion of a transport request. This includes deciding whether to send an AGV to a charging node for recharging or to send it to a parking node to wait for its next assignment. In either case, it needs to be decided to which specific node the AGV is sent. It is also possible to do no repositioning action for an AGV. Considering the finite capacity and heterogeneity of the station nodes, the ND3QN agent makes the AGV repositioning decisions based on a modified distance-based dispatching rule.

This rule is described as follows: First, the agent checks whether AGV v is currently charging with battery level still less than the critical threshold, already repositioning to a station node, waiting idle at a parking node, or serving a transport request. For these task types, it takes the no-repositioning action ($a^v = 0$). The reasons for taking the no-repositioning action to AGVs with those task types are (i) repositioning an AGV that is already repositioning or parked is undesirable, (ii) serving transport requests are non-preemptable tasks, and (iii) for AGVs that are already charging (and not yet reached the critical threshold), the charging must continue as much as possible.

Next, we consider AGVs that are idle (i.e., just completed the delivery of a transport request, or at a charging node with a battery level greater than the critical threshold) as remaining candidates for repositioning. The agent first checks whether AGV v has a charge level is greater than or equal to the critical threshold ($s_1^v \geq \underline{b}$). If this is true and if the AGV is not located at one of the station nodes yet, the agent assigns the AGV to the closest station node ($a^v \in S$), which is not occupied yet. On the other hand, if the AGV's charge level is less than the critical threshold ($s_1^v < \underline{b}$), the agent tries to assign the AGV to the closest available charging station ($a^v \in C$). If all charging stations are fully occupied, the agent swaps the to-be-charged AGV with the AGV at the closest charging station with a charge level greater than the critical threshold. The agent then assigns the swapped AGV to the closest available parking station.

6. Benchmark policies

In this section, we outline the policies through which we assess the performance of the ND3QN agent. In Section 6.1, we introduce the *Multi-Att* policy, a competitive benchmark inspired by the electric-vehicle dispatching literature. We present the *Random* policy in Section 6.2. This policy randomly assigns a feasible AGV to a new transport request and serves as an upper bound (on cost) to show what performance advantage can be obtained by employing more sophisticated policies. Both policies use the same repositioning actions as the ND3QN agent. Finally, we present a mixed-integer linear program (MILP) in Section 6.3 that assumes full knowledge of future transport requests and hence serves as a lower bound on the cost of the optimal policy.

6.1. Multi-att policy

Multi-Att policy is a multi-attribute dispatching rule inspired by a rule-based dispatching strategy commonly used in dial-a-ride problems. Maciejewski, Bischoff, and Nagel (2016) argue that a popular taxi dispatching rule is to assign the nearest idle vehicle to a new transport request. Although they consider this strategy a good starting point, they also argue that today's technology could improve this myopic strategy. To improve the dispatching performance, they propose to include the capability of vehicles looking into the near future and communicating their expected time until availability (ETA) with each other.

In our context, the central fleet owner can also determine the ETA for all AGVs in the fleet. We propose the Multi-Att policy, which works as follows. The agent first identifies the feasible AGVs which have an ETA less than a specified parameter $\mathcal{T}_{max}$. Subsequently, among these AGVs, the agent determines which one has the smallest distance to the pick-up location of the transport request and assigns that AGV to the transport request. If no AGVs are feasible when a new transport request arises, the agent rejects the request. To summarize, we employ a distance-based rule that uses information from the near future to balance the traveled distance by AGVs and the tardiness of transport requests.

6.2. Random policy

Similarly to Multi-Att, the Random policy rejects a request if none of the AGVs is feasible. Otherwise, the policy assigns a random AGV from the set of feasible AGVs to the new transport request.

6.3. Mixed-integer linear program

We extend the MILP model presented in Singh et al. (2022) by allowing rejection of transport requests and incurring rejection costs in the objective function. In addition, we extend their model by considering unit capacity at each station node where only a limited number of AGVs can charge at the same time. The MILP is solved with the perfect information assumption, i.e., we relax the real-time request arrival scenario and solve the instance sets in hindsight, i.e., the MILP optimizes AGV travel for charging and repositioning considering the requests to be served. The results then serve as a benchmark to calculate the performance gaps from other policies. The MILP model is presented in Appendix B.

7. Computational experiments

In this section, we present the experiments for assessing the performance of the proposed ND3QN agent and to compare it with the benchmark heuristics and the MILP. Firstly, Section 7.1 discusses the case study design, including the generation of test instances. Subsequently, Section 7.2 compares the performance of the ND3QN agent and benchmark heuristics with the performance of the MILP for small-scale scenarios. Finally, Section 7.3 provides further insights on the performance of the ND3QN agent and sensitivity analyses in a real-world case study.

7.1. Case study design

We utilize the problem instances in this research provided by our industry partner KMWE, a company specialized in precision machining.[1] The layout used for this case study is based on KMWE's production facility in the BIC. This layout is shown in Fig. 6 (see Appendix C). We assume that loads are always ready for pickup when an AGV arrives at a pickup-and-delivery node. In practice, around 900 requests arise over a time horizon H, which is equal to 24 h. Episodes start (and end) at 7:00 am, corresponding to the start of a working day. As discussed with industry practitioners from KMWE and the BIC, we set discharging rate $d = 0.0055\%/s$, charging rate $c = 0.011\%/s$, and critical threshold $\underline{b} = 20\%$. The AGVs' initial charges are drawn uniformly from $[\underline{b}, Q]$. The initial locations of vehicles are set randomly. We set the initial set of stations $S = 18$, of which nine are charging stations and nine are parking stations. Lastly, we set $c_\tau = 1$ unit/s and $c_\delta = 1$ unit/s. A summary of parameter values is shown in Table 1.

[1] Problem instances are available from https://github.com/nitman118/drl-od-h-data.

Table 1
Parameter values.

Parameter	Description	Value
H	Planning horizon	24 h
$\|C\|$	Number of charging stations	9
$\|P\|$	Number of parking stations	9
c	Charging rate	0.011%/s
d	Discharging rate	0.0055%/s
$\underline{b}$	Critical charge level	20%
c_τ	Tardiness cost	1 unit/s
c_δ	Travel time cost	1 unit/s
$\mathcal{T}_{max}$	Maximum ETA	600 s

Before evaluation, we first trained the ND3QN on a separate set of 1000 episodes, which typically takes 8 h of training time on a computer with Intel Core i7-4710MQ CPU @ 2.50 GHz CPU and 16 GB RAM. We note that 200 episodes are sufficient to obtain a stable width of around 2%–5% of the objective value in all our instances. The weight coefficients η_1 and η_2 are both set to 1. The MILP and heuristics are programmed in Python v3.7.3. Additionally, we used the Gurobi Python package for solving the MILP. In terms of computational efficiency, it is noteworthy that our neural network-based agent can make decisions almost instantaneously during deployment, owing to its pre-trained weights, making the neural network approach suitable for real-time applications.

7.2. Performance comparison of alternative solution approaches

In this section, we study the performance of the proposed ND3QN agent and the MILP. We also compare their performance with the Multi-Att policy and the Random policy. Since the MILP model cannot be solved for large problem sizes encountered in real life, in this section we focus on the problem sizes smaller than the case study presented in Section 7.1 (the large instances of the case study will be considered in Section 7.3). To be specific, we let the requests $|R|$ range from 3 to 20, $H = 0.5$ h, and $|V| = 6$. We generate ten instances for each parameter set and run each instance 10 times, making a total of 100 iterations for each parameter set. We run the MILP model on one processor core and limit the computational time to 15 s for MILP-15. We also extend the time for the MILP to 120 s (MILP-120) to check whether the MILP can reach the optimal value in more computational time. Next to the objective values, we also calculate percentage gaps of the ND3QN policy with respect to the other policies. Note that since we are comparing costs, a negative gap indicates an improvement in the objective value of the ND3QN policy against the compared approach.

Table 2 shows that the MILP-15 is not able to solve any considered instance scale to optimality, while the MILP-120 can optimally solve small instances, i.e., with three and four requests. However, they can only provide the best feasible solutions with an increase in the number of requests. Also, not all iterations provide a solution when requests exceeds 4 for the MILP-15 and 10 for the MILP-120. Further, the MILP-15 and MILP-120 cannot find any solutions beyond 10 and 17 requests, respectively.

The ND3QN, Multi-Att, and Random methods can find solutions for all instance scales presented in Table 2. Also, the ND3QN outperforms the MILP-15 for most cases for which the latter can find a solution. On the other hand, the ND3QN shows an average deterioration of around 24% compared to the MILP-120 for the instances from 3 to 15 requests. With the instances beyond 17 requests, the ND3QN starts outperforming the MILP-120. The ND3QN is also superior to the Multi-Att and Random methods for all the instances, with around 18% and 54% improvement, respectively.

Several reasons explain the relatively large gaps between the ND3QN and the MILP-120. First, since the MILP-120 is a static method, it possesses all information about transport requests before planning them. In contrast, the ND3QN can only observe the transport requests at the time they enter the system. The MILP models know the locations and time of all requests in advance and obtain an essential advantage in reaching new requests faster with less travel time. However, the ND3QN's performance compared to MILP-15 is more representative than the MILP-120 in a real-life situation since models should respond quickly to requests. While a response time of 15 s is still acceptable, 120 s is already troublesome. As a last reason for the relatively large gaps, the ND3QN is originally designed and trained for bigger scenarios. We can obtain the ND3QN's full potential in scenarios with a longer time horizon (24 h) and more requests (around 900), which are taken into account in the next section.

7.3. Sensitivity analyses

In this section, we study the performance of the ND3QN agent and provide managerial insights by performing sensitivity analyses on common instance scales in practice. We exclude the MILP from the sensitivity analyses since it could not find solutions for larger-scale instances (see Table 2). We introduce TW as the probability of a request having a tight time window. The tightness of a time-window is measured by the difference between the earliest pickup time and the latest delivery time, and a value of TW, e.g., equal to 0.5, indicates that transport requests have a 50% chance of having tight time-windows, where $TW \in [0, 1]$. As base scenario, we use $H = 24$ h, $TW = 0.5$, $|R| \sim 900$, and $|V| = 12$. We evaluate instances over 200 episodes. We analyze the effect of varying the number of requests and AGV fleet size, time-window tightness, and stations type in Sections 7.3.1, 7.3.2, and 7.3.3, respectively.

7.3.1. Impact of number of requests and AGV fleet size

In this experiment, we train the ND3QN agent for four fleet sizes: 9, 12, 15, and 18 for a scenario with $|R| = 900$ (i.e., the ND3QN was not specifically trained for other request sizes in this section). We then varied $|R|$ in $\{450, 900, 1800\}$. $|R| = 1800$ represents a busy day with approximately double the number of request arrivals in comparison to the base scenario, whereas $|R| = 450$ represents a less busy day and contains approximately half of the number of requests realized in the base scenario. The average costs obtained by the ND3QN, Multi-Att, and Random methods are summarized in Table 3. The results show that the ND3QN performs the best across all the fleet sizes and across all request sizes. The ND3QN's improvement in objective value over the Multi-Att increases from 9 to 12 AGVs, whereas this gap decreases when the fleet size increases further. Overall, the ND3QN's best performance is obtained with 12 AGVs. The ND3QN's superior performance with respect to the Random remains constant, whereas its improvement over the Multi-Att slightly increases with the request sizes. Especially for the busy day scenario, we expect a ND3QN agent specifically trained on that request size to obtain a higher advantage than the one trained on the base scenario. However, the results show that the ND3QN agent is equipped to handle fluctuations in request arrivals. In general, since the proposed methodology leverages deep neural networks that are known for their ability to generalize on unseen data, the model can accommodate various layouts, variations in pickup and delivery distributions, spatial and temporal patterns, and even scenarios like mechanical failures or breakdowns, provided they are represented in the training data.

Moreover, the cost in Eq. (7) comprise of travel and tardiness costs, and in Figs. 3 and 4 we report their gaps individually for the base scenario. We observe that ND3QN outperforms the Multi-Att with superior performance (larger gap) on the mean total tardiness costs compared to mean total travel costs, which may indicate that the ND3QN agent is able to utilize the spatial and temporal information of available fleet and arriving requests in order to achieve lower operational costs, and the savings on tardiness costs is higher than in travel costs. Also, Fig. 3 shows that the performance gap on tardiness costs becomes larger with increase in the number of requests. In addition, Fig. 4 shows that with

Table 2
Performance comparison of the policies.

$\|R\|$	MILP-15	MILP-120	Random	Multi-Att	ND3QN	Gap (%)			
						ND3QN vs. MILP-15	ND3QN vs. MILP-120	ND3QN vs. Multi-Att	ND3QN vs. Random
3	60.15	58.40[a]	130.30	85.00	58.88	−2.11	0.82	−30.73	−54.81
4	78.35[b]	65.28[a]	171.50	101.25	80.23	2.39	22.90	−20.76	−53.21
5	93.52[b]	73.06	193.66	104.00	86.15	−7.88	17.91	−17.16	−55.51
7	175.97[b]	95.18	293.51	151.00	129.75	−26.26	36.32	−14.07	−55.79
10	432.00[b]	127.82	379.29	219.75	169.13	−60.85	32.31	−23.03	−55.40
12	–	168.43[b]	467.41	263.75	209.94	–	24.64	−20.40	−55.08
15	–	249.07[b]	623.95	319.75	265.34	–	6.53	−17.01	−57.47
17	–	342.00[b]	679.00	362.50	280.48	–	−17.98	−22.62	−58.69
20	–	–	810.67	439.00	355.68	–	–	−18.98	−56.12

(–) Feasible solution not found.
[a] Optimal solution found by all iterations.
[b] Not all iterations found a solution.

Table 3
Performance comparison with varying request sizes R.

$\|R\|$	$\|V\|$	ND3QN	Multi-Att	Gap (%)		
				Random	ND3QN vs. Multi-Att	ND3QN vs. Random
450	9	7 225.26	8 332.40	13 166.85	−13.29	−45.13
900	9	16 635.25	20 105.98	28 623.57	−17.26	−41.88
1800	9	27 542.11	33 681.30	46 403.28	−18.23	−40.65
450	12	7 454.84	9 325.82	17 846.76	−20.06	−58.23
900	12	13 998.33	18 238.56	35 029.90	−23.25	−60.04
1800	12	27 968.27	35 239.53	67 823.59	−20.63	−58.76
450	15	7 210.09	8 399.80	17 536.93	−14.16	−58.89
900	15	16 009.52	18 259.22	38 554.87	−12.32	−58.48
1800	15	29 449.68	33 743.80	69 824.06	−12.73	−57.82
450	18	7 950.74	8 376.00	15 259.69	−5.08	−47.90
900	18	17 258.94	18 349.94	42 005.32	−5.95	−58.91
1800	18	31 549.79	33 780.10	70 055.86	−6.60	−54.96

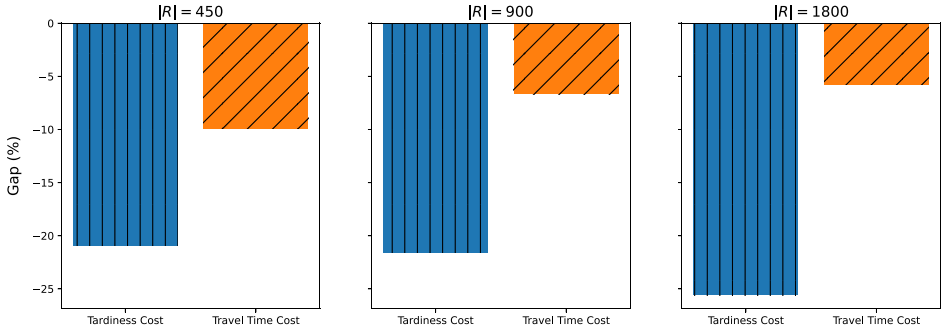

Fig. 3. Gap in performance of ND3QN with Multi-Att on average tardiness costs and average travel time costs for various request sizes $|R|$.

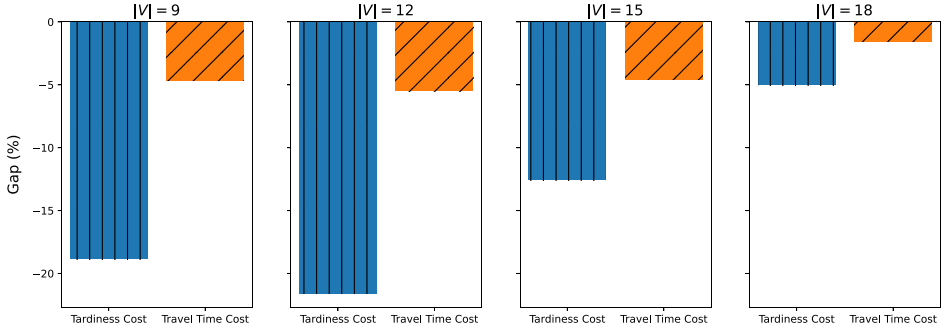

Fig. 4. Gap in performance of ND3QN with Multi-Att on average tardiness costs and average travel time costs for various fleet sizes $|V|$.

an increase in fleet size the performance gaps on travel time costs and tardiness costs diminish. The reduction in costs is higher when moving from 12 to 18 AGVs which may indicate that with an increase in fleet size the potential to save on costs may be lower since AGVs are more readily available.

7.3.2. Impact of time-window tightness

We train the ND3QN agent for various probabilities of having tight time windows, i.e., $TW = \{0.2, 0.5, 0.8\}$. The average costs obtained by the ND3QN and Multi-Att are presented in Table 4. The results show that the ND3QN's improvement over the Multi-Att remains stable

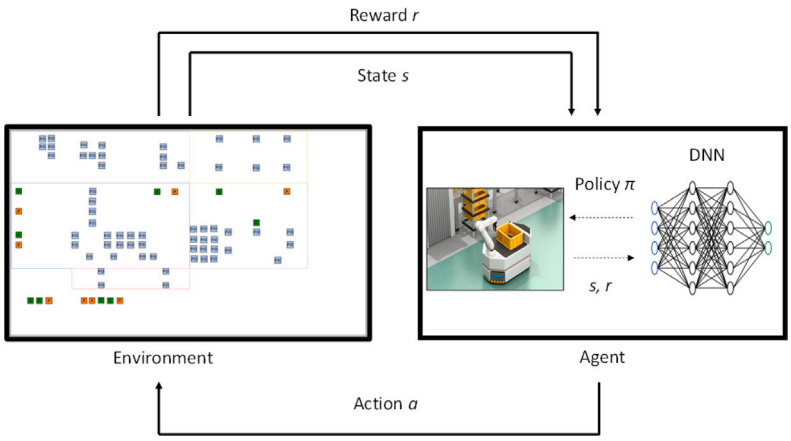

Fig. 5. The perception-action learning loop.

Table 4
Performance comparison with varying probability of time-window tightness TW.

TW	Multi-Att	ND3QN	Gap (%)
0.2	17 491.89	13 571.12	−22.41
0.5	18 238.56	13 998.33	−23.25
0.8	19 405.08	15 546.86	−19.88

Table 5
Performance comparison with varying $|C|$.

$\|C\|$	Multi-Att	ND3QN	Gap (%)
9	18 238.56	13 998.33	−23.25
18	17 966.05	13 735.89	−23.55

when increasing the probability TW. It implies that the ND3QN's performance is reliable when the tightness of the time windows change.

7.3.3. Impact of station types

In this experiment, we trained a ND3QN agent that considers only charging stations. Average costs obtained by ND3QN and Multi-Att for this scenario is reported in Table 5. The objective performance gap between ND3QN and Multi-Att increases with changing the station type to only charging stations. Hence, changing the station type to only charging stations could be beneficial but comes at considerable investment cost for each additional charging station.

8. Conclusions

This research investigates the problem of real-time vehicle-request allocation, repositioning, and recharging of a fleet of AGVs. The main novelty of this research consists of simultaneously addressing transport requests with soft due dates, loading and unloading activities at pick-up and delivery nodes, heterogeneous stations, and a capacity constraint at these stations. We also formulated a MILP model to minimize the total cost, including the tardiness and rejection costs related to the transport requests as well as the AGV travel costs. Since the MILP model could not solve instances beyond 17 requests, we proposed a ND3QN model that could find good-quality solutions for industry-size instances within a reasonable amount of computational time. Computational experiments on real-world data reveal that the proposed model outperforms the practitioners heuristic Multi-Att by up to 23%. The ND3QN showed a significant advantage over Multi-att concerning the productive travel of the AGVs, implying a more efficient use of the AGVs. Lastly, we provided sensitivity analyses with respect to several problem parameters. An interesting future research may include a multi-agent approach to solve the proposed problem with the proposed agent to also learn the optimal repositioning and recharging tasks for the fleet.

CRediT authorship contribution statement

Nitish Singh: Conceptualization, Methodology, Software, Writing – original draft. **Alp Akcay:** Supervision, Conceptualization, Writing – original draft. **Quang-Vinh Dang:** Formal analysis, Investigation. **Tugce Martagan:** Supervision, Writing – review & editing. **Ivo Adan:** Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by a grant from Provincie Noord-Brabant in the Netherlands for the project "Advanced Manufacturing Logistics" (grant number C2218902/4301942, 2017). We would also like to thank Koen Herps from KMWE for providing data and his support during this research.

Appendix A. DQN algorithm

In Q-learning, the agent learns the Q-values for all state–action pairs. However, since we consider a large-scale system, learning all unique state–action pairs becomes computationally intractable. Hence, instead of using value estimation to calculate the optimal Q-function, we use a *Deep Neural Network* to estimate this function. Combining Q-learning with the use of a Deep Neural Network is called *Deep Q-learning* or *DQN*. The way the agent interacts with the environment is captured by the perception-action learning loop as presented in Fig. 5. The DQN's objective remains the same as in regular Q-learning: minimizing the loss (see Eq. (5)). The weights of the Artificial Neural Network are updated via Stochastic Gradient Descent (SGD) and backpropagation, like regular Neural Nets. During training, we use a technique called *experience replay*, in which the agent's experiences are stored in a data set called *replay memory*. A key reason for using replay memory is to break the correlation between consecutive samples (Liu & Zou, 2019). The agent's experience at time t consists of the state of the environment s_t, the action a_t taken from that state, the reward r_{t+1} obtained by

taking action a_t from state s_t, and the consecutive state s_{t+1} and is denoted as follows.

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1}) \tag{6}$$

In practice, the size of the replay memory is often set with finite size N_{replay}, in which only the last N_{replay} experiences are stored. Subsequently, we randomly sample a batch of experiences from the replay memory to train the network. This training is prosecuted by passing the batch of experiences to the network as input. Henceforth, we will refer to this network as the *policy network* since its objective is to find the optimal policy. Next, the 'max-term' in Eq. (5) is calculated by a forward pass of s_{t+1} to the network. Now that we dispose of both the target Q-values and actual Q-values, we can calculate the loss and update the network's weights. However, simultaneously updating the network's weights, the actual Q-values and target Q-values update in the same direction, leading to the optimization 'chasing its own tail'. To overcome this issue, we introduce a *target network*, which is a clone of the policy network with 'frozen' weights. We update these weights with the policy network's weights after a N_{steps} steps, thereby reducing correlations with the target (Mnih et al., 2015).

Double DQN

The DQN agent may get stuck in a region of local optima due to overestimation of Q values. In the earlier section, we discussed about two Q Networks, one being the policy network and the other being the target Q Network which is a clone of the policy network. We also discussed that the target Q network is not updated very frequently and instead it is updated only after a certain number of steps. The above highlighted overestimation problem may become even more significant if the actions are taken on the basis of the target network whose values are not frequently updated. However, we would like to continue using the target Q Network as it offers better and more stable target values for the update. To combine the best of both worlds, the "Double DQN" algorithms propose to select the action on the basis of the policy network but to use the values of the target state–action value corresponding to the particular state–action from the target network. By doing so we can simultaneously overcome both the "overestimation" problem of Q Values while also avoiding the instability in the target values (Sewak, 2019).

Dueling Double DQN (D3QN)

A dueling double DQN builds on the double DQN architecture. While in double DQN, a particular layer could be connected to the layer before and the layer after, a dueling architecture is non-sequential. In D3QN, the model layers branches into two different streams. The first of these branches correspond to the value function which estimates the Q value of a given state. The second branch computes the value of the "advantage" of taking a particular action in the current state (Sewak, 2019; Wang et al., 2016).

Appendix B. Mixed integer linear program

We define the mathematical problem on a directed graph $G = (N, A)$, where N is the set of nodes as described in Section 3. We denote the set of arcs as $A = \{(i,j)|i,j \in N, i \neq j\}$. Each arc is associated with a travel time denoted by t_{ij}. Additionally, the set of charging requests and parking requests are captured by $B = \{1, 2, \dots, n_B\}$ and $E = \{1, 2, \dots, n_E\}$, respectively, where n_B and n_E are safe upper bounds, allowing multiple requests per charging and parking node, such that every AGV can charge and park as many times as needed. As a result, not all requests of B and E are required to be scheduled. Note that a charging request is assigned to reposition an AGV to a charging node while a parking request repositions it to a parking node. Further, we set the source and destination nodes of such a charging or parking request ($n_r^S = n_r^D, \forall r \in B \cup E$) equal to each other and set the earliest pickup and latest delivery time to zero and infinity, respectively. We denote the total set of requests considered for the MILP by $T = R \cup B \cup E$. Lastly, we formulate the mathematical model as follows.

Decision variables

$x_{rr'v}$	Binary variable, equal to 1 if AGV v performs request r immediately prior to request r', 0 otherwise
z_r	Binary variable, equal to 1, if request r does not appear in any tour (i.e., r is rejected), 0 otherwise
a_{rv}^S	Arrival time of AGV v at the source node of request r
a_{rv}^D	Arrival time of AGV v at the destination node of request r
f_{rv}^S	Finish (pick-up) time of AGV v at the source of request r
f_{rv}^D	Finish (delivery) time of AGV v at the destination of request r
d_{rv}^S	Battery discharge of AGV v after travel to and loading at the source of request r
d_{rv}^D	Battery discharge of AGV v after travel to and unloading at the destination of request r
$y_{rr'vv'}$	Binary variable, equal to 1 if request r of AGV v is done prior to request r' of AGV v' at the same charging or parking station ($n_r^S = n_{r'}^S$), 0 otherwise

The mathematical model of the described problem can be formulated as follows:

Objective (7) minimizes the weighted sum of tardiness costs of requests, travel costs of AGVs, and rejection costs of requests, where τ_r is the tardiness of request r, π_{rv} is the travel time of AGV v when performing request r, and z_r is the number of rejected transport requests. Note that η_1 and η_2 are the weight coefficients used to prioritize the travel costs and tardiness, respectively.

$$\min \quad \eta_1 \sum_{v \in V} \sum_{r \in T} c_\delta \, \pi_{rv} + \eta_2 \sum_{r \in R} c_\tau \, \tau_r + \sum_{r \in R} c_d \, z_r \tag{7}$$

We must satisfy the following constraints:

$$\sum_{r' \in T} \sum_{v \in V} x_{rr'v} \leq 1 \qquad \forall r \in T \tag{8}$$

$$x_{rrv} = 0 \qquad \forall r \in T, \forall v \in V \tag{9}$$

$$\sum_{r \in R} x_{rr''v} - \sum_{r' \in T} x_{r''r'v} = 0 \qquad \forall r'' \in T, \forall v \in V \tag{10}$$

$$z_r = 1 - \sum_{r' \in R} \sum_{v \in V} x_{rr'v} \qquad \forall r \in R \tag{11}$$

Constraints (8) impose that if a request is accepted, then at most one AGV performs the request. Self-visits are avoided by Constraints (9). Constraints (10) conserve the incoming and outgoing arcs for all requests. Constraints (11) count the number of requests that do not appear in any tour of any AGV, or in other words, count the number of rejected requests.

$$a_{rv}^S \geq e_r \qquad \forall r \in T, \forall v \in V \tag{12}$$

$$a_{rv}^S + h_r^S \leq f_{rv}^S \qquad \forall r \in T, \forall v \in V \tag{13}$$

$$a_{rv}^D + h_r^D \leq f_{rv}^D \qquad \forall r \in T, \forall v \in V \tag{14}$$

$$f_{rv}^S + t_{n_r^S n_r^D} \leq a_{rv}^D \qquad \forall r \in T, \forall v \in V, \tag{15}$$

$$f_{rv}^D + t_{n_r^S n_r^D} - L(1 - x_{rr'v}) \leq a_{r'v}^S \qquad \forall r, r' \in T, r \neq r', \forall v \in V \tag{16}$$

Constraints (12) ensure that AGVs start loading at the source node of a request after the earliest pickup time. Constraints (13) and (14) ensure that AGVs are only allowed to leave the nodes after completing the time needed for material handling. Constraints (15) and (16) determine the arrival times at the destination and source nodes, respectively. Constraints (16) also prevent sub-tours, where L is a large positive constant.

$$\tau_r \geq f_{rv}^D - l_r \qquad \forall r \in T, \forall v \in V \tag{17}$$

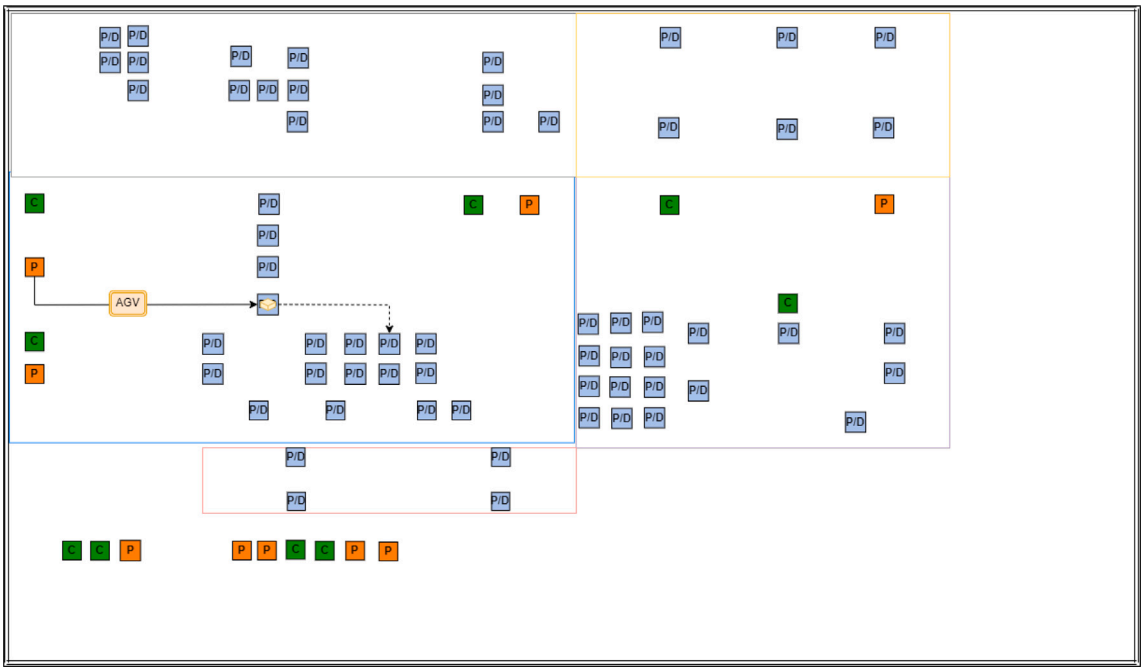

Fig. 6. Layout used for this case study. We highlight one movement of an AGV en route to the pickup-and-delivery nodes on this layout.

$$\pi_{r'v} \geq (t_{n_r^D n_{r'}^S} + t_{n_{r'}^S n_{r'}^D})\, x_{rr'v} \qquad \forall r, r' \in T, r \neq r', \forall v \in V \tag{18}$$

Constraints (17) and non-negative Constraints (28) define the tardiness of each request. Constraints (18) determine the travel time of an AGV to perform a request.

$$a_{r'v'}^S + L\,(1 - y_{rr'vv'}) \geq f_{rv}^S \qquad \forall r, r' \in B \cup E, r \neq r', n_r^S = n_{r'}^S, \; \forall v, v' \in V, v \neq v' \tag{19}$$

$$a_{rv}^S + L\, y_{rr'vv'} \geq f_{r'v'}^S \qquad \forall r, r' \in B \cup E, r \neq r', n_r^S = n_{r'}^S, \; \forall v, v' \in V, v \neq v' \tag{20}$$

Constraints (19) and (20) make sure that if two parking or charging requests have the same source (and destination), they can never be at that node during the same time.

$$d_{0v}^S = Q - b_v \qquad \forall v \in V \tag{21}$$

$$d_{rv}^D \geq d_{rv}^S + d\,(t_{n_r^S n_r^D} + h_r^D) \qquad \forall r \in R, \forall v \in V \tag{22}$$

$$d_{r'v}^S \geq d_{rv}^D + d\,(t_{n_r^D n_{r'}^S} + h_{r'}^S) - L(1 - x_{rr'v}) \qquad \forall r, r' \in R, r \neq r', \forall v \in V \tag{23}$$

$$d_{rv}^D \geq d_{rv}^S - c\,(a_{rv}^D - a_{rv}^S) \qquad \forall r \in B, \forall v \in V \tag{24}$$

$$0 \leq d_{rv}^S \leq Q - \underline{b} \qquad \forall r \in R, \forall v \in V \tag{25}$$

$$0 \leq d_{rv}^D \leq Q \qquad \forall r \in R, \forall v \in V \tag{26}$$

$$0 \leq d_{rv}^S \leq Q \qquad \forall r \in B, \forall v \in V \tag{27}$$

Constraints (21) set the initial battery discharge levels. Constraints (22) determine the battery discharge after traveling from a pickup to a delivery node and unloading there, whereas Constraints (23) determine the battery discharge after traveling from a delivery to a pickup node and loading activity. Constraints (24) define the battery discharge of an AGV after visiting a charging station. It is reduced by an amount of charge proportional to the time that the AGV spends at the station. Constraints (25)–(26) set the lower and upper limits ($Q = 100$) for an amount of battery discharge. Constraints (28)–(29) ensure valid domains for the remaining decision variables.

$$\tau_r, \delta_{rv} \geq 0 \qquad \forall r \in T, \forall v \in V \tag{28}$$

$$z_r, x_{rr'v}, y_{rr'vv'} \in \{0, 1\} \qquad \forall r, r' \in T, \forall v \in V \tag{29}$$

Appendix C. Layout in the case study

The layout shown in Fig. 6 is based on the KMWE's production facility. Each blue square represents a pickup-and-delivery (PD) node, each orange square represents a parking (P) node, and each green square represents a charging (C) node. In total, there are 61 PD nodes (divided over 5 departments), 9 parking nodes, and 9 charging nodes. For the AGV movements within this layout, we use the Manhattan distance metric.

In our case study environment, a flexible flow-shop manufacturing system in a high-mix-low-volume production environment, AGVs are responsible for transporting products between work centers (containing CNC machines) as well as from a manufacturing preparation center (MPC) to different work centers. In Fig. 6, the AGV can be seen moving from its parking location at the MPC towards the material handling point of a work center, from where the product needs to be transported to another work center. In this system, human operators still play a vital role in preparing materials (at the MPC) and overseeing the loading and unloading activities (at the work centers) in coordination with the AGVs. Currently, our industry partner is in the process of adopting and deploying AGVs in the production facility.

Appendix D. Hyperparameters

We tuned the hyperparameters using Bayesian Optimization. Whereas tuning techniques such as grid search and random search use a relatively high computational time, bayesian optimization technique typically requires fewer iterations to get to a high performant set of hyperparameter values. The reason for this relatively low optimization time is that Bayesian Optimization focuses on hyperparameters that yield the most promising results and tends to neglect hyperparameters that do not make a substantial difference in performance. The open-source hyperparameter optimization framework Optuna (which used for the tuning) provided us with hyperparameter importances. According to Optuna, the most important hyperparameters were epsilon start, experience buffer capacity, gamma, and the number of neurons in the hidden layer. Plugging the hyperparameter values of the best performing models found by Optuna lead to an increase of 7.89% compared to the model using the initial hyperparameter values. Final

Table 6
ND3QN's hyperparameters.

Hyperparameter	Description	Value
	Number of hidden layers in policy network	2
	Number of hidden layers in autoencoder network	2
	Number of neurons per hidden layer in policy network	512
	Number of neurons in first hidden layer in autoencoder network	25
	Number of neurons in second hidden layer in autoencoder network	20
	Gradient optimizer	Adam
U	Code dimensions	5
L	Number of tiles	25
lr	Learning rate	0.0001
γ	Discount Factor	0.95
n	n-step learning	5
N_{replay}	Experience buffer capacity	250 000
β_0	Beta start	0.4
β_{steps}	Linearly anneal from β_0 to 1 in training steps	100 000
α	Alpha	0.4
N_{steps}	Number of steps after which to replace the target net	100
N_{batch}	Batch size	1024

hyperparameter values are shown in Table 6. For the Bayesian Optimization, we used a Tree-structured Parzen Estimator (TPE) algorithm recommended by Optuna in situations with limited parallel computing power and a not low-dimensional search space.

References

Bank, D., Koenigstein, N., & Giryes, R. (2020). Autoencoders. arXiv preprint arXiv:2003.05991.

Bilge, U., Esenduran, G., Varol, N., Öztürk, Z., Aydin, B., & Alp, A. (2006). Multi-attribute responsive dispatching strategies for automated guided vehicles. *International Journal of Production Economics*, *100*(1), 65–75.

Brainport Industries Campus (2020). The evolution of the high-tech manufacturing industry. https://www.brainportindustriescampus.com/en/innovating/factory-of-the-future. Accessed 21 January 2023.

Confessore, G., Fabiano, M., & Liotta, G. (2013). A network flow based heuristic approach for optimising AGV movements. *Journal of Intelligent Manufacturing*, *24*, 405–419.

De Koster, R. B., Le-Anh, T., & Van Der Meer, J. R. (2004). Testing and classifying vehicle dispatching rules in three real-world settings. *Journal of Operations Management*, *22*, 369–386.

De Ryck, M., Versteyhe, M., & Debrouwere, F. (2020). Automated guided vehicle systems, state-of-the-art control algorithms and techniques. *Journal of Manufacturing Systems*, *54*, 152–173.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., et al. (2017). Noisy networks for exploration. arXiv preprint arXiv:1706.10295.

Froger, A., Mendoza, J. E., Jabali, O., & Laporte, G. (2017). A matheuristic for the electric vehicle routing problem with capacitated charging stations. In *[Research report] centre interuniversitaire de recherche sur les reseaux d'entreprise, la logistique et le transport (CIRRELT)*.

Guan, X., & Dai, X. (2009). Deadlock-free multi-attribute dispatching method for AGV systems. *International Journal of Advanced Manufacturing Technology*, *45*, 603–615.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., et al. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Holler, J., Vuorio, R., Qin, Z., Tang, X., Jiao, Y., Jin, T., et al. (2019). Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In *Proceedings - IEEE international conference on data mining, ICDM* (pp. 1090–1095).

Hu, H., Jia, X., He, Q., Fu, S., & Liu, K. (2020). Deep reinforcement learning based AGVs real-time scheduling with mixed rule for flexible shop floor in industry 4.0. *Computers & Industrial Engineering*, *149*, Article 106749.

Kamoshida, R., & Kazama, Y. (2017). Acquisition of automated guided vehicle route planning policy using deep reinforcement learning. In *6th IEEE international conference on advanced logistics and transport, ICALT 2017* (pp. 1–6). IEEE.

Keskin, M., & Çatay, B. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation Research Part C (Emerging Technologies)*, *65*, 111–127.

Kullman, N. D., Cousineau, M., Goodson, J. C., & Mendoza, J. E. (2022). Dynamic ride-hailing with electric vehicles. *Transportation Science*, *56*(3), 775–794.

Le-Anh, T., & De Koster, R. M. B. M. (2004). Multi-attribute dispatching rules for agv systems with many vehicles. *ERIM Report Series Research in Management*.

Le-Anh, T., & De Koster, M. B. (2005). On-line dispatching rules for vehicle-based internal transport systems. *International Journal of Production Research*, *43*(8), 1711–1728.

Le-Anh, T., & De Koster, M. B. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, *171*(1), 1–23.

Lin, K., Zhao, R., Xu, Z., & Zhou, J. (2018). Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1774–1783).

Liu, R., & Zou, J. (2019). The Effects of Memory Replay in Reinforcement Learning. In *2018 56th annual Allerton conference on communication, control, and computing, Allerton 2018* (pp. 478–485). Institute of Electrical and Electronics Engineers Inc..

Maciejewski, M., Bischoff, J., & Nagel, K. (2016). An assignment-based approach to efficient real-time city-scale taxi dispatching. *IEEE Intelligent Systems*, *31*(1), 68–77.

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

Pettit, J. F., Glatt, R., Donadee, J. R., & Petersen, B. K. (2019). Increasing performance of electric vehicles in ride-hailing services using deep reinforcement learning. arXiv preprint arXiv:1912.03408.

Rhazzaf, M., & Masrour, T. (2021). Deep learning approach for automated guided vehicle system. In *Artificial intelligence and industrial applications: smart operation management* (pp. 227–237). Springer.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In *4th international conference on learning representations, ICLR 2016 - conference track proceedings*. International Conference on Learning Representations, ICLR.

Sewak, M. (2019). Deep q network (dqn), double dqn, and dueling dqn. In *Deep reinforcement learning* (pp. 95–108). Springer.

Shi, J., Gao, Y., Wang, W., Yu, N., & Ioannou, P. A. (2019). Operating electric vehicle fleet for ride-hailing services with reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 1–13.

Singh, N., Dang, Q.-V., Akcay, A., Adan, I., & Martagan, T. (2022). A matheuristic for AGV scheduling with battery constraints. *European Journal of Operational Research*, *298*(3), 855–873.

Singh, N., Sarngadharan, P., & Pal, P. K. (2011). AGV scheduling for automated material distribution: a case study. *Journal of Intelligent Manufacturing*, *22*, 219–228.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9–44.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT Press.

Sweda, T. M., Dolinskaya, I. S., & Klabjan, D. (2017). Adaptive routing and recharging policies for electric vehicles. *Transportation Science*, *51*(4), 1326–1348.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103).

Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, *170*(3), 677–709.

Vuksanović, D., Ugarak, J., & Korčok, D. (2016). Industry 4.0: the future concepts and new visions of factory of the future development. In *Paper presented at sinteza 2016 - international scientific conference on ICT and e-business related research* (pp. 293–298).

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Frcitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In *33rd international conference on machine learning, ICML 2016, Vol. 4* (pp. 2939–2947). (9).

Wei, Q., Yan, Y., Zhang, J., Xiao, J., & Wang, C. (2022). A self-attention-based deep reinforcement learning approach for AGV dispatching systems. *IEEE Transactions on Neural Networks and Learning Systems*.

Wiering, M., & van Otterlo, M. (2012). *Reinforcement Learning State-of-the-Art, Vol. 12*.

Zamiri Marvizadeh, S., & Choobineh, F. (2014). Entropy-based dispatching for automatic guided vehicles. *International Journal of Production Research*, *52*(11), 3303–3316.

Zhang, X.-j., Sang, H.-y., Li, J.-q., Han, Y.-y., & Duan, P. (2022). An effective multi-AGVs dispatching method applied to matrix manufacturing workshop. *Computers & Industrial Engineering*, *163*, Article 107791.