Battery Management for Automated Warehouses via Deep Reinforcement Learning

Yanchen Deng[1], Bo An[1], Zongmin Qiu[2], Liuxi Li[2], Yong Wang[2], and Yinghui Xu[2]

[1] School of Computer Science and Engineering, Nanyang Technological University
{ycdeng,boan}@ntu.edu.sg
[2] Cainiao Smart Logistics Network
{zongmin.qzm,liuxi.llx,richard.wangy}@cainiao.com,renji.xyh@taobao.com

Abstract. Automated warehouses are widely deployed in large-scale distribution centers due to their ability of reducing operational cost and improving throughput capacity. In an automated warehouse, orders are fulfilled by battery-powered AGVs transporting movable shelves or boxes. Therefore, battery management is crucial to the productivity since recovering depleted batteries can be time-consuming and seriously affect the overall performance of the system by reducing the number of available robots. In this paper, we propose to solve the battery management problem by using deep reinforcement learning (DRL). We first formulate the battery management problem as a Markov Decision Process (MDP). Then we show the state-of-the-art DRL method which uses Gaussian noise to enforce exploration could perform poorly in the formulated MDP, and present a novel algorithm called TD3-ARL that performs effective exploration by regulating the magnitude of the outputted action. Finally, extensive empirical evaluations confirm the superiority of our algorithm over the state-of-the-art and the rule-based policies.

Keywords: Automated warehouses · Battery management · Deep reinforcement learning.

1 Introduction

With the rapid development of multi-robot systems, automated warehouses and Robotic Mobile Fulfillment Systems (RMFSs), such as the Kiva system [4], have emerged as a new category of automated order fulfillment systems. In such systems, order fulfillment is implemented by a fleet of battery-powered Automatic Guided Vehicles (AGVs) to transport movable shelves with Stock-keeping Units (SKUs). Due to their ability of reducing operational cost and improving throughput capacity, automated warehouses have been adapted in many e-commerce companies including Amazon, Cainiao, etc. Battery management is crucial to automated warehouses as recovering the depleted batteries is time-consuming and significantly affects the throughput capacity of the system by reducing the number of available AGVs. Moreover, since the number of charging poles is fixed,

inappropriate battery management would result in many low-power AGVs which cannot be recharged timely. As a result, these AGVs eventually cannot fulfill any job (including recharging) due to the extremely low battery and must be recovered manually, which makes battery management quite challenging.

A common way to implement battery management is by hand-crafted rules. For example, we could use a parameterized policy with two thresholds $\langle T_c, T_w \rangle$. The AGVs with battery lower than T_c are scheduled to charge and the AGVs with battery higher than T_w are scheduled to work. However, since most of them are built upon energy conservation, the rule-based policies are decoupled from the current workload and may schedule many AGVs to charge even during peak periods. As a result, the backlogs increase and the performance of the system degenerates significantly due to the lack of available AGVs. In other words, the rule-based policies are pre-defined and cannot be adjusted adaptively according to the real-time workload.

To maximize throughput capacity, it is necessary to explore alternative approaches to solve the battery management problem by explicitly considering the dynamic workload. Our first contribution is modeling the battery management problem as a Markov Decision Process (MDP) due to its capability in modeling long-term planning problems with uncertainty. The state of the formulated MDP includes the battery histograms of AGVs under different states, the number of working AGVs in different working areas and the number of backlogs. The action is defined as the upper bound of the AGVs in the working areas and the threshold for the charging AGVs. The reward is defined as the number of fulfilled orders in each time step.

Solving the MDP is challenging since both the state space and the action space are continuous. Traditional tabular-based reinforcement learning approaches [2, 8, 15, 20] cannot be applied due to their inability of modeling high-dimensional and complicated dynamics. Pioneered by DQN [13], deep reinforcement learning (DRL) [5] has achieved tremendous success in solving many sequential decision-making problems [14, 16, 18]. TD3 [6] is the state-of-the-art DRL algorithm for continuous control. However, it implements exploration by adding Gaussian noise to the outputted action, which could be inefficient in our problem where the permuted action may still has the same semantic if the magnitude of the outputted action is high. Therefore, our second contribution is proposing a novel algorithm called TD3 with action regulation loss (TD3-ARL) to enforce the state dependent exploration. In more detail, we regulate the magnitude of the outputted action by imposing a loss term on the objective function of the actor network to guarantee the diversity of exploration. Our extensive empirical evaluations have demonstrated the superiority of TD3-ARL over the state-of-the-art.

2 Related Work

Battery Management for Automated Warehouses. While battery management is crucial to large-scale automated warehouses, its influence on the

performance is usually omitted in automated warehouse studies [9]. McHANEY examined several charging schemes and pointed out the battery constraint can only be omitted when charging can be insured to take place without impacting system operation [11]. Recently, Zou et al. evaluated the performance of different recovering strategies including re-charging, swapping and inductive charging [22]. Several heuristics for dispatching low-power AGVs for battery swapping were proposed in [3]. However, these heuristics cannot be applied to our case since we focus on recovering depleted batteries via re-charging.

Deep Reinforcement Learning. Combining with high-capacity deep neural network approximators, DRL [5] has achieved great successes on challenging decision making problems [14, 16, 18]. Pioneered by DQN [13], many DRL algorithms such as A3C [12], DDPG [10], SAC [7] and TD3 [6] have been proposed. However, most of off-policy DRL algorithms use uncorrelated Gaussian noise to enforce exploration, which would perform poorly in our case where many actions are equivalent. Therefore, based on TD3 we propose a novel algorithm which performs effective exploration by regulating the magnitudes of the outputted actions.

3 Motivation Scenario

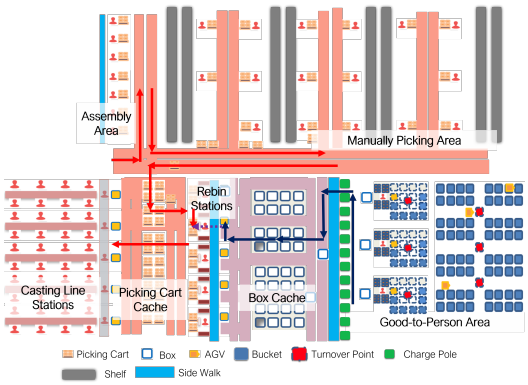


Fig. 1. The layout of a typical automated warehouse

Fig.1 gives the layout of a typical automated warehouse. Generally, the lifecycle of an order includes picking, consolidation and casting line. When an order comes in, the system assigns picking jobs for Good-to-Person Area to pick the required SKUs from buckets and store them to a turnover box. After finishing picking jobs, the box is sent to the Rebin Area for further consolidation. If the order does not contain any manually picking item, the system assigns an

assembly job for Assembly Area to assemble a picking-cart with empty parcels. Otherwise, the system assigns an assembly job for the Manually Picking Area to pick the SKUs from the shelves manually and assemble a picking-cart in which parcels are filled with the required SKUs. The picking-cart is sent to the Rebin Area as soon as the assembly job finishes. After both the turnover box and the picking-cart arrived at the Rebin Area, the consolidation is initiated to combine the SKUs from different areas by transferring SKUs in the box to the parcels in the picking-cart. After finishing consolidation job, the picking-cart is sent to the Casting Line Area for packaging and inspecting. The order is considered as fulfilled and sent outbound after the casting line job is finished.

Since the number of charging poles are fixed and all the transportation of buckets, boxes and picking-carts is fulfilled by battery-powered AGVs, inappropriate battery management would lead to the shortage of available AGVs and severely degenerates the throughput capacity of warehouses. Typically, battery management is implemented by thresholds. That is, the system schedules the AGVs with battery lower than a threshold to charge and schedules the charging AGVs with battery higher than another threshold to work. Besides, a working AGV will not execute charging job if the number of charging AGVs has already met the number of charging poles. Finally, an emergency charging mechanism that allows low-power AGVs to preempt charging poles is introduced to ensure that the battery of an AGV is higher than a safe threshold.

Although the rule-based policy is easy to be implemented and highly interpretable, it suffers from three major shortcomings. First, AGVs are scheduled to charge as long as their battery is lower than the charging threshold and there are available charging poles. As a result, it is possible to schedule a large number of AGVs to charge and cause the shortage of available AGVs, which is not desirable during peak periods. Second, the thresholds are decoupled from the current workload, which prohibits the system from improving throughput capacity proactively when needed. In fact, a common way to improve productivity in practice is to disable the charging poles manually to force AGVs to work for a longer time when the working areas are busy. However, this workaround is risky since reducing the number of charging poles would lead to a large number of low-power AGVs or even dead AGVs. Finally, the rule-based policy fails to exploit order structure. Specifically, since consolidation and casting line depend on picking, it is reasonable to schedule AGVs in picking areas to charge when most orders are in consolidation or casting line procedure. Given these limitations, we aim to take the current workload and the order structure into the consideration and propose a novel battery management scheme to maximize overall throughput capacity.

4 Problem Statement and MDP Formulation

In this section, we give the formal definition to battery management problem and formulate it as an MDP.

4.1 Battery Management Problem

We consider an automated warehouse with a fleet of AGVs G which are subject to the battery constraint, i.e., an AGV $g \in G$ cannot fulfill any job if its battery b_g is lower than the dead threshold T_{dead} . Orders arrive over time and each order includes picking jobs for Good-to-Person area, an assembly job for either Assembly area or Manual Picking area, and jobs for Rebin area and Cast Line area respectively. Given the number of charging poles $C < |G|$, the goal is to design a battery management scheme to maximize overall throughput capacity. More specifically, at each time step t, we aim to determine the set of charging AGVs $G_c^t \subset G$ and the set of AGVs for working areas $G_w^t \subseteq G$ such that $|G_c^t| \leq C, G_c^t \cap G_w^t = \emptyset$ and $G_c^t \cup G_w^t = G$ to maximize the number of fulfilled orders by the end of a day.

4.2 MDP Formulation

We propose to model the problem as a Markov Decision Process (MDP) due to its ability in modeling sequential decision-making problems with uncertainty. Formally, the MDP is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$ where $\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}$ are the state space, the action space, the reward function and the transition probability function, respectively. The definitions are given as follows.

- **State $s^t \in \mathcal{S}$.** The state we consider includes battery feature, AGV fleet feature and system feature. For the battery feature, we build a histogram with a bin size 5% for AGVs in charging state, working state and idle state, respectively. For the AGV fleet feature, we consider the number of working AGVs in each area. For the system feature, we consider the number of backlogs and the current time step t, where backlogs are the unfinished orders by the current time step.
- **Action $a^t \in \mathcal{A}$.** It is impossible to directly determine G_w^t and G_c^t as dividing a set into two disjoint subsets would result in a prohibitive large action space. Instead, we consider to schedule *anonymously*. Formally, a^t is defined by a tuple $\langle U_w^t, T_c^t \rangle$ where $|G| - C \leq U_w^t \leq |G|$ is the upper bound of the number of AGVs in working areas and T_c^t is the battery threshold for the charging AGVs. When realizing a^t, the system schedules $\max(|G_w^t| - U_w^t, 0)$ AGVs with the lowest battery to charge and schedules the charging AGVs with battery higher than T_c^t to working areas.
- **Reward function $\mathcal{R}$.** The reward is defined as the number of orders fulfilled in the time step t. We aim to find the policy $\pi^* : \mathcal{S} \to \mathcal{A}$ which maximizes the accumulated reward (i.e., the total number of fulfilled orders). That is,

$$\pi^* = \arg\max_{\pi} \sum_{t=1}^{T} \mathcal{R}(s^t, \pi(s^t))$$

.

5 Solving the MDP

Traditional RL algorithms cannot be applied to solve the MDP as the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are continuous and the transition probability function $\mathcal{P}$ does not have an explicit form. Therefore, we resort to deep RL and present a novel algorithm built upon TD3. We will first briefly introduce TD3 and show it would perform poorly in our problem due to inefficient exploration. Then we present our proposed algorithm.

5.1 TD3

Twin Delayed Deep Deterministic Policy Gradient (TD3) [6] is the state-of-the-art deep reinforcement learning algorithm for continuous control. TD3 is a deterministic policy gradient algorithm [17] and incorporates an actor-critic architecture where both the actor and the critics are parameterized by deep neural networks. The policy which is represented by the actor network π_ϕ is updated to maximize

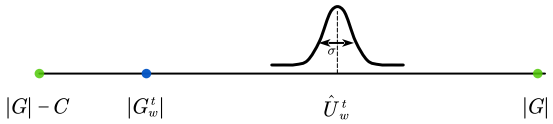
$$J(\phi) = \frac{1}{N} \sum_{s} Q_{\theta_1}(s, \pi_\phi(s)) \tag{1}$$

where N is the size of a mini-batch of experiences and Q_{θ_1} is a critic. To address the overestimation bias in critic learning, TD3 concurrently learns two critics $Q_{\theta_1}, Q_{\theta_2}$ and uses the smaller one to compute the targets for critic updates. The critics are updated to minimize the temporal difference error [19]. Besides, it adds noises to the target actions and enforces a delayed policy update to reduce variances in actor-critic methods.

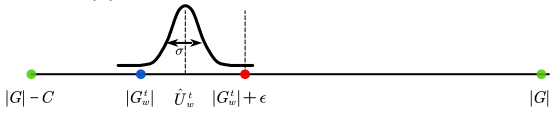
Unfortunately, TD3 would perform poorly when solving MDP $\mathcal{M}$ due to inefficient exploration. Since TD3 trains a deterministic policy in an off-policy way, a common approach to enforce exploration is to add uncorrelated mean-zero Gaussian noise $\mathcal{N}(0, \sigma)$ to the outputted action, which would perform poorly in our case. In fact, many actions in MDP $\mathcal{M}$ are equivalent and the conventional exploration scheme fails to exploit the fact effectively. Consider the conventional exploration scheme applied to U_w^t shown in Fig.2(a) where $\hat{U}_w^t$ is the deterministic action outputted by the actor. Since the system schedules $\max(|G_w^t| - U_w^t, 0)$ AGVs to charge, any $U_w^t \in [|G_w^t|, |G|]$ has the same semantic. As a result, TD3 could sample a lot of equivalent actions but rarely explore the region $[|G| - C, |G_w^t|]$ if the magnitude of the outputted action is close to $|G|$.

5.2 Enforcing State Dependent Exploration via Action Regulation Loss

Since all the actions in the range of $[|G_w^t|, |G|]$ are equivalent, we consider to tamp the magnitude of the outputted action such that $\hat{U}_w^t \leq |G_w^t| + \epsilon$ to ensure the diversity, where ϵ is a constant to achieve effective exploration (i.e., Fig.2(b)). Thus, the bound is dependent on the current state. A straightforward way would be rounding the outputted action to the bound directly. However, as



(a) conventional exploration scheme



(b) state dependent exploration scheme

Fig. 2. Different exploration schemes

shown in [1], directly clipping would incur a bias on the policy gradient. Another approach to enforce the bound is reward engineering [21], which uses L_2-norm of the outputted action as a penalty and injects it directly into the reward function. However, since the rewards in $\mathcal{M}$ are objective quantities with real-world interpretation (i.e., the number of orders fulfilled in each time step), it is inappropriate to impose a hand-crafted penalty term. Besides, directly injecting the penalties to the rewards can cause high variances in critic learning since the critics bootstrap both the actual rewards and the penalties. In fact, the penalties do not have a temporal structure and should not be bootstrapped since the out-of-bound actions in different time steps are independent from each other.

Instead, we tamp the magnitude of the outputted action by directly imposing a loss term on the objective function of the actor and refer it as TD3 with action regulation loss (TD3-ARL). Formally, the actor network π_ϕ in our algorithm is updated to maximize

$$J(\phi) = \frac{1}{N} \sum_s Q_{\theta_1}(s, a) - L(s, a)|_{a=\pi_\phi(s)}, \tag{2}$$

where $L(s, a)$ is the regulate loss defined by

$$L(s, a) = |G_w(s) + \epsilon - U(a)| + U(a) - G_w(s) - \epsilon. \tag{3}$$

Here, $G_w : \mathcal{S} \to \mathbb{N}$ is a function that returns the number of AGVs in the working areas and $U : \mathcal{A} \to \mathbb{N}$ is a function that returns the first component of an action (i.e., the upper bound of the number of AGVs in working areas). When $U(a)$ is higher than the state dependent bound $G_w(s)+\epsilon$, a loss of $2(U(a)-G_w(s)-\epsilon)$ is incurred to regulate the magnitude of the action. Otherwise the loss term cancels and the objective function is equivalent to the one in vanilla TD3. Combining with uncorrelated Gaussian noises $\mathcal{N}(0, \sigma)$, the regulated action can achieve effective state dependent exploration. Alg.1 presents the sketch of TD3-ARL.

Algorithm 1: TD3 with Action Regulation Loss

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$ and actor network π_ϕ with random parameters θ_1, θ_2 and ϕ
Initialize target networks $Q_{\theta'_1}, Q_{\theta'_2}, \pi_{\phi'}$ with $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay memory M
for *episode=1,. . . , E* **do**
  Reset the environment and get the initial state s
  for *t=1,. . . ,T* **do**
    Select an action $a = \pi_\phi(s) + \psi$ where $\psi \sim \mathcal{N}(0, \sigma)$
    Execute a in the environment and observe the reward $r = \mathcal{R}(s, a)$, new state s' and done signal d
    Store the experience (s, a, r, s', d) to M
    $s \leftarrow s'$
  for *i=1,. . . ,K* **do**
    Sample a mini-batch of N experiences $\{(s, a, r, s', d)\}$ from M
    $\widetilde{a}' \leftarrow \pi_{\phi'}(s') + \psi$ where $\psi \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), \alpha, -\alpha)$
    $y \leftarrow r + \gamma(1 - d) \min_{i \in \{1,2\}} Q_{\theta'_i}(s', \widetilde{a}')$
    Update critics Q_{θ_i} to minimize $\frac{1}{N} \sum (y - Q_{\theta_i}(s, a))^2, \forall i \in \{1, 2\}$
    if i mod *delay=0* **then**
      Update actor π_ϕ to maximize Eq. (2)
      $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
      $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

6 Simulator Design

To facilitate training and evaluating algorithms, we build an event-driven simulator to simulate order generation, battery changes and decision execution.

Data description. The data provided by Cainiao include the orders in the Wuxi warehouse for 25 days. Each order contains the releasing time and the detailed job allocation for each working area. For each job, we can infer its duration via its start time and end time.

Timeline design. We divide a day into 510 discrete time steps. During a time step, the following activities are executed sequentially.

- *Order generation.* Since orders in consecutive time steps are correlating in the real-world scenario, it is inappropriate to bootstrap from the real-world data for each time step individually. On the other hand, if we directly replay the real-world orders in a day, the number of different order pace patterns could be quite limited. As a result, the trained policy could overfit these patterns. Therefore, we propose to compromise by bootstrapping from long term patterns. Specifically, we divide a day into 24 slots (1h per slot) and uniformly select an input for each slot from the real-world data in the same time interval.
- *Interacting with policy.* The policy computes an action according to the current state and submits it to the simulator. The simulator executes the numerical action by casting it to the set of AGVs to be scheduled and repositioning

these AGVs to their destinations (i.e., charging poles or working areas) with random delays.
- *Processing orders.* The simulator processes orders by a first-in first-out (FIFO) strategy. Specifically, it iterates over the order queue and attempts to allocate AGVs for the remaining jobs of each order. The allocation for an order terminates if (1) there is no available AGVs in the working areas or (2) the prepositional jobs haven't been fulfilled (e.g., the casting line job depends on the consolidation job). An order is considered as fulfilled and is removed from the order queue if its casting line job has been finished.
- *Updating AGVs.* For each AGV, the simulator updates its battery according to the duration of each status it has experienced in the time step. If its battery is lower than the dead threshold T_{dead}, then the AGV is marked as dead and cannot be scheduled to fulfill any job.

After finishing these procedures, the simulator returns a feature vector that represents the current state and a reward that is the number of orders fulfilled in current time step.

7 Empirical Evaluation

In this section, we conduct extensive empirical evaluation to demonstrate the effectiveness of our proposed method.

7.1 Experimental Configurations

We consider the problems with $|G| = 640$ AGVs, the charging capacity $C = 120$, the emergency charging threshold 30% and the dead threshold $T_{dead} = 15\%$. The size of the training instances ranges from 3,500 orders to 4,500 orders per day. For each episode, we randomly choose a training instance as order inputs. The test instances we consider are categorized into low loads (∼3,500 orders per day), median loads (∼3,900 orders per day) and high loads (∼4,300 orders per day). For each of configuration, we generate 50 instances and report the means as the results. The performance metrics we consider are listed as follows.

- *Fulfilled ratio.* The metric measures the proportion of the fulfilled orders at the end of an episode, which directly reflects the productivity of a policy.
- *Number of bottleneck time steps.* We consider a time step as a bottleneck time step when there is no available AGVs. Intuitively, if a policy fails to provide more available AGVs when the working areas are busy, then bottleneck time steps are more likely to happen, especially when considering the effect of backlogs.
- *Latency per order.* We consider the latency of an order as the duration required to fulfill the order. If a policy fails to provide enough AGVs, then the system has to wait for available AGVs and the latency of each order will increase.

– *Average number of idle AGVs.* The metric considers the number of idle AGVs in each time step. Intuitively, a poor policy would provide less available AGVs in each time step and thus the average number of idle AGVs is low.

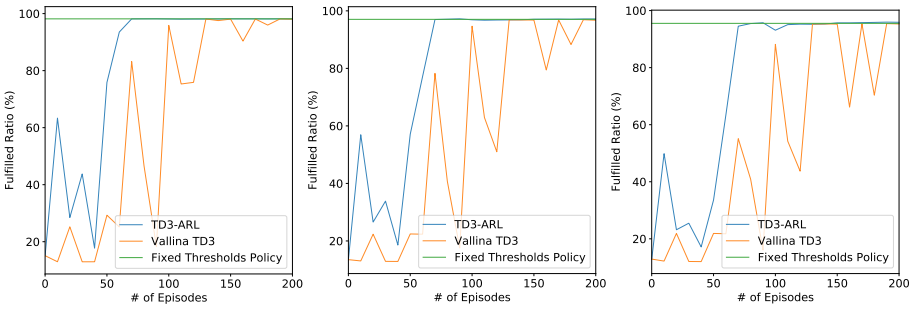
The competitors we consider are three rule-based policies parameterized by $\langle T_c^t, T_w^t \rangle$ where T_c^t is the charging threshold for AGVs in working areas and T_w^t is the working threshold for charging AGVs in time step t. The fixed thresholds policy we consider has static thresholds $T_c^t = 40\%$ and $T_w^t = 80\%$, which is widely used in real-world scenarios. The dynamic charging threshold policy has a fixed working threshold $T_w^t = 80\%$ and a dynamic charging threshold T_c^t which is computed by an upper bound T_c^{UB}, a lower bound T_c^{LB} and the number of charging AGVs in current time step. Formally,

$$T_c^t = T_c^{UB} - (T_c^{UB} - T_c^{LB}) \frac{|G_c^t|}{|G|}$$

In the experiments, we set $T_c^{UB} = 75\%$ and $T_c^{LB} = 35\%$. Finally, the dynamic working threshold policy has a fixed charging threshold $T_c^t = 40\%$ and a dynamic working threshold T_w^t which is computed by an upper bound T_w^{UB}, a lower bound T_w^{LB} and the number of working AGVs in current time step. Formally,

$$T_w^t = T_w^{UB} - (T_w^{UB} - T_w^{LB}) \frac{|G_{working}^t|}{|G|}$$

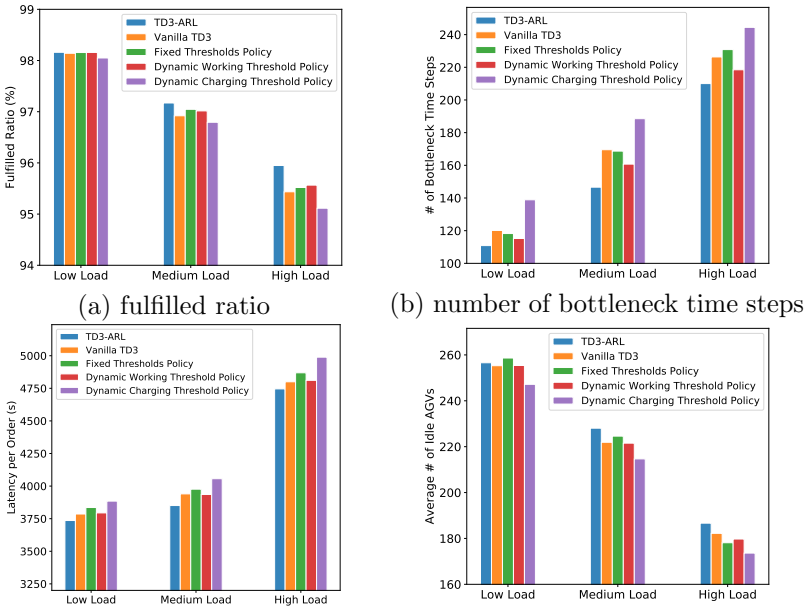
where $G_{working}^t \subseteq G_w^t$ is the set of working AGVs. In our experiments, we set $T_w^{UB} = 80\%$ and $T_w^{LB} = 60\%$.



(a) low Load (b) median Load (c) high load

Fig. 3. Learning curves of TD3-ARL and Vanilla TD3 under different loads

For the implementation of the TD3 algorithm, both the actor and the critics are parameterized by a four layer fully-connection network where the hidden layers include a 300 neuron and a 400 neuron layers. Relu activation functions are applied to layers other than the last layer. For the output layer of the actor, a sigmoid activation function is applied to bound the action space. To represent



(a) fulfilled ratio (b) number of bottleneck time steps

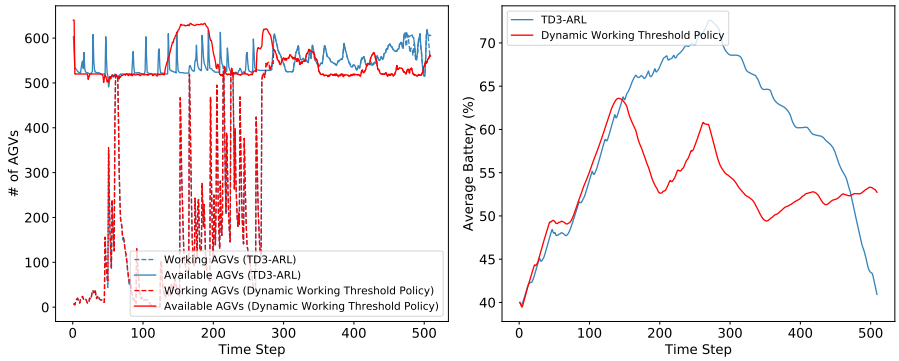
(c) latency per order (d) average number of idle AGVs

Fig. 4. Performance comparisons under different loads

the action component U_w efficiently, we squash the range $[|G| - C, |G|]$ to [0,1] and the action can be directly recovered from the raw output of the actor via a linear transformation. Finally, we set the size of replay memory to 10^6, the standard deviation of exploration noise $\sigma = 0.1$, the standard deviation of target smooth noise $\tilde{\sigma} = 0.05$, the noise range $\alpha = 0.1$ and the policy update frequency to 2.

7.2 Experimental Results

Learning curves. To evaluate the stabilities of vanilla TD3 and our proposed TD3-ARL, we test algorithms every 10 training episodes and present the fulfilled ratios in Fig.3. It can be seen that our proposed TD3-ARL not only outperforms the vanilla TD3 but also improves much stably. In fact, TD3-ARL outperforms the fixed charging threshold policy before the 100-th episode and its performance still improves slowly afterward. On the other hand, the performance of vanilla TD3 oscillates in a broad range and it can only compete after 180 episodes. That is because ARL can effectively regulate the magnitude of an output action to guarantee the diversity of the experiences. In contrast, vanilla TD3 uses state independent noise to enforce exploration, which performs poorly due to the existence of equivalent actions and significantly slows down the learning process.



(a) number of AGVs in working areas (b) average battery

Fig. 5. Behaviors of policies when solving the instance with high load

Performance Comparison. We benchmark the performance of different policies under different loads, and present results in Fig.4. It can be seen from the Fig.4(a) that TD3-ARL can fulfill more orders than the rule-based policies, especially when solving high load instances. That is due to the fact that low load instances are relatively trivial that even a naive policy with emergency charging mechanism can easily fulfill most of orders. As the scale of instances grows, the rule-based policies no longer can compete as they cannot adaptively adjust their actions to proactively improve productivity. Fig.4(b) demonstrates the superiority of TD3-ARL over the rule-based policies in terms of the number of bottleneck time steps. Compared to the rule-based policies, our TD3-ARL can adjust the number of charging AGVs according to the current workload, and has less bottleneck time steps. Fig.4(c) shows the superiorities of TD3-ARL over the rule-based policies in terms of order latency. Although the dynamic working threshold policy considers the number of working AGVs and results in a lower latency than the ones of the other rule-based policies, it is still inferior to TD3-ARL, which highlights the effectiveness of proactive scheduling. Finally, Fig.4(d) demonstrates our superiority in terms of idle AGVs. The results indicate that our TD3-ARL can provide more available AGVs than the rule-based policies, especially when solving the instances with high load.

Behavior analysis. To look deeper into the decisions made by policies, we analyze the behaviors of TD3-ARL and dynamic working threshold policy on a high load instance and present results in Fig.5. It can be seen that for the dynamic working threshold policy, the number of available AGVs in each time step is fairly stable. That is due to the fact that the policy is built upon energy conservation. As a result, it fails to take current workload into consideration and provide more available AGVs during the peak periods (i.e., after the 300-th time step). In fact, it schedules many AGVs to charge when the working areas are busy and can only provide about 520-540 available AGVs for working areas. In contrast, our TD3-ARL policy exhibits more deliberate behaviors, i.e., scheduling more AGVs

to charge at the beginning of an episode (and the average battery increases) and scheduling more AGVs to work when the working areas are busy (and the average battery decreases). Besides, it is worth noting that in TD3-ARL the average battery at the end of the episode is close to the one at the beginning of the episode, which indicates that TD3-ARL is able to take the full advantage of the accumulated energy to improve productivity.

8 Conclusion

In this paper, we investigated battery management problem for large-scale automated warehouses which employ battery-powered AGVs to fulfill orders. To cope with its dynamic nature, we formulate the problem as an MDP with continuous state and action spaces. Since there are many equivalent actions in the MDP, traditional state independent exploration scheme performs poorly. Therefore, we propose a novel algorithm TD3-ARL which regulates the magnitude of the outputted action and enforces state dependent exploration via imposing a regulation loss term on the objective function of the actor. Extensive evaluations show the superiorities over the state-of-the-art, as well as the rule-based policies.

Acknowledgements. This work was supported by Alibaba Group through Alibaba Innovative Research (AIR) Program and Alibaba-NTU Joint Research Institute (JRI), Nanyang Technological University, Singapore.

References

1. Chou, P.W., Maturana, D., Scherer, S.: Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In: ICML. pp. 834–843 (2017)
2. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: International Conference on Computers and Games. pp. 72–83 (2006)
3. Ebben, M.: Logistic Control In Automated Transportation Networks. Ph.D. thesis, University of Twente (2001)
4. Enright, J.J., Wurman, P.R.: Optimization and coordinated autonomy in mobile fulfillment systems. In: Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence (2011)
5. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning. Foundations and Trends in Machine Learning **11**(3-4), 219–354 (2018)
6. Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: ICML. pp. 1582–1591 (2018)
7. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: ICML. pp. 1856–1865 (2018)
8. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: ECML. pp. 282–293. Springer (2006)

9. Le-Anh, T., De Koster, M.: A review of design and control of automated guided vehicle systems. European Journal of Operational Research **171**(1), 1–23 (2006)
10. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: ICLR (2016)
11. McHANEY, R.: Modelling battery constraints in discrete event automated guided vehicle simulations. International Journal of Production Research **33**(11), 3023–3040 (1995)
12. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: ICML. pp. 1928–1937 (2016)
13. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
14. OpenAI: Openai five. https://blog.openai.com/openai-five/ (2018)
15. Rummery, G.A., Niranjan, M.: On-Line Q-Learning using connectionist systems. Tech. rep., Cambridge University (1994)
16. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484 (2016)
17. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.A.: Deterministic policy gradient algorithms. In: ICML. pp. 387–395 (2014)
18. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354 (2017)
19. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning **3**, 9–44 (1988)
20. Watkins, C.J.C.H.: Learning from Delayed Rewards. Ph.D. thesis, King's College, Cambridge, UK (May 1989)
21. Zhao, M., Li, Z., An, B., Lu, H., Yang, Y., Chu, C.: Impression allocation for combating fraud in e-commerce via deep reinforcement learning with action norm penalty. In: IJCAI. pp. 3940–3946 (2018)
22. Zou, B., Xu, X., De Koster, R., et al.: Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system. European Journal of Operational Research **267**(2), 733–753 (2018)