IPSJ Transactions on Mathematical Modeling and Its Applications Vol.13 No.2 36–49 (Aug. 2020)

Regular Paper

Branching Deep Q-network Agent with Reward Allocation Mechanism for Joint Replenishment Policy

Hiroshi Suetsugu[1,a)] Yoshiaki Narusue[1,b)] Hiroyuki Morikawa[1,c)]

Received: November 11, 2019, Revised: January 6, 2020,
Accepted: January 27, 2020

Abstract: In a supply chain where multiple products need to be delivered via a container ship, we need to consider a joint replenishment problem in periodic-review system where transportation costs of these containers are shared among products. By proposing a Q-learning agent with function approximation, called the branching deep Q-network (DQN) with reward allocation, the purpose of this paper is to ease the strict assumptions, such as the zero lead time or the stationary demand, which are essential in the existing heuristic algorithms. Our numerical experiments demonstrate that the proposed agent learns the coordinated replenishment policy and outperforms the benchmark policy.

Keywords: joint replenishment policy, multi-product inventory, stochastic inventory control, reinforcement learning, reward allocation

1. Introduction

The maritime transportation costs depend on the required number of containers in a supply chain, where multiple products are required to be delivered via a container ship. Therefore, the transportation cost per product would decrease when several products are ordered simultaneously. In this case, the joint replenishment policy (JRP), which considers the multi-product situations, is required to achieve the minimum total cost.

With maritime transportation, replenishment opportunities are often restricted to, say, once a week, depending on the port operation constraints. Therefore, this problem setting is defined as a joint replenishment problem in a periodic-review system, that is, replenishment opportunity comes at a regular time intervals, where the container costs are shared among products. In addition, stochastic non-stationary demands rather than stochastic stationary demands, whose statistical properties such as mean, variance, and autocorrelation, are all constant over time, need to be considered because of seasonality, trend or other factors in the actual business setting.

With stochastic demands, the Markov decision processes (MDP) have been employed for the formulation of the problem. When the number of products is very small, the MDP can be only solved because the action spaces grow exponentially with the number of products because of its combinatorial nature. Some studies (Refs. [10] and [11]) have succeeded in finding the optimal policy with MDP formulation, but the number of products remains only two or four.

Heuristic algorithms have been proposed for a large number of products. However, these heuristic algorithms often require strong assumptions, such as stationary demand, or/and ignore some of the important logistic conditions, such as non-zero order-to-delivery lead time and minimum order quantities. In addition, most studies in JRP have assumed continuous-review system, where an order can be placed at any particular time. Therefore, the assumptions on the demands and/or logistic conditions of the existing approaches do not match our problem setting.

By proposing the reinforcement learning (RL) agent, the purpose of this paper is to find the dynamic multi-product replenishment policy based on the branching DQN (BDQN) [13] with the extension of the reward allocation mechanism. The BDQN was proposed for robot control, in which the n dimensional action branches follow the shared state representation in the neural network Q-function approximation, while enabling the linear growth of the total number of network outputs along with increasing the dimensionality of action. We focus on the similarity between the supply chain management and robot control in terms of challenging uncertainties, and came up with the idea of treating each product as an independent decision maker with a credit assignment policy that is a hybrid of the global and the local rewards, which we call reward allocation.

Considering the difference between the assumed application in Ref. [13], where only single global reward is available, and our problem setting, where all the costs except the transportation costs can be calculated with respect to each product, we add the reward allocation mechanism to the original BDQN agent. In our case, each branch consists of one product, and our objective variable is total cost; the transportation costs are calculated across multiple products whereas the holding costs and penalty costs are

[1] Graduate School of Engineering, The University of Tokyo, Bunkyo, Tokyo 113–0033, Japan
[a)] hsuetsugu@sglab.co.jp
[b)] narusue@mlab.t.u-tokyo.ac.jp
[c)] mori@mlab.t.u-tokyo.ac.jp

calculated independently of each product. Thus, we introduce the reward allocation function to determine how much reward is allocated to each branch in order for each branch to effectively learn the coordinated behavior while escaping from falling into a local solution.

We extend our previous paper [12] by conducting experiments on increased number of products and different parameters, and examining the performance comparison between two reward allocation strategies. Note that this paper does not prove that our proposed agent is the best among the possible BDQN extensions, and our attempt to apply the BDQN extension is only limited to the joint replenishment problem setting.

We evaluate our proposed agent, called the branching deep Q-network with reward allocation (BDQN-RA), by applying it to a multi-product inventory control problem and ran numerical experiments that varied the number of products and demand stationarity in consideration of non-zero order-to-delivery lead time and minimum order quantities.

By conducting several numerical experiments, we found that our proposed agent achieves a better performance than the benchmark policy and exhibits a robust learning process even with 50 products with a non-stationary demand. We also found that the reward allocation strategy considerably affects the agent's behavior and appropriate reward allocation strategy is crucial in learning the cooperative behavior among branches.

The reminder of this paper is as follows. Section 2 reviews the literature on both JRP and RL. Section 3 presents our proposed solution. Section 4 presents the results of the numerical experiment, and Section 5 provides the conclusions of the paper.

2. Related Works

In this section, we first review the literature on JRP with stochastic demands. Then, we proceed to the survey of RL particularly on how to overcome the problem of state and action dimensionalities.

2.1 Joint Replenishment Policy

If there is only one product and if the stock status is only determined at the time of the review (usually called the periodic-review system), then the (s, S) policy has to be widely employed. In the (s, S) policy, an order is placed if the inventory position is at or below the order point s. The order is supplied after a replenishment lead-time and is available to satisfy the customer demands. The (s, S) policy produced a low total cost under the assumption of the demand pattern and the cost factors. Ref. [14] extended this method into a well-known dynamic lot-sizing problem.

However, under a multi-product inventory system, the coordination across products should be taken into account, and Ref. [5] asserts that a coordinated ordering policy can achieve 13% of the cost savings as compared with the economic order quantity models for a single-item approach when a set of twenty products is considered.

A way to model this problem is to formulate it as a MDP, and use a value iteration algorithm to find the optimal joint replenishment policy. To find the optimal policy, an improved policy-iteration algorithm is presented in Refs. [10] and [11]. Although they succeeded in finding the optimal policy, they could not overcome the curse of dimensionality, and their approach can be only used for two or four products.

One of the most representative heuristic algorithms for JRP is the $(\boldsymbol{S}, \boldsymbol{c}, \boldsymbol{s})$ policy. The $(\boldsymbol{S}, \boldsymbol{c}, \boldsymbol{s})$ policy was proposed by Ref. [1], which is often referred to as a "can-order" policy, where an order is triggered by an item j when its inventory position falls to or below the reorder level s. Then, any item for which the inventory position is at or below its can-order level c is also included in the order and is raised to the order-up-to level S. Although a continuous-review system was first assumed in the $(\boldsymbol{S}, \boldsymbol{c}, \boldsymbol{s})$ policy, it was applied to a periodic-review system in the research of Ref. [4]. However, stationary demands need to be assumed with the $(\boldsymbol{S}, \boldsymbol{c}, \boldsymbol{s})$ policy, since these parameters are fixed. A number of studies have been conducted on the determination of these policy parameters and ease some of the required assumptions in the literature.

Certain studies (Refs. [7], [8]) have formulated MDP, and solved MDP with policy iteration for a small number of products. Policy iteration with MDP is computationally intensive; therefore, these studies proposed the heuristic algorithm to handle a large number of products. In Refs. [7] and [8], capacity constraints were introduced, whereas stationary demands were assumed and the replenishment lead time was assumed zero. In addition, these studies have assumed a continuous-review system. Thus, these approaches cannot be applied to our problem setting.

2.2 Reinforcement Learning for Large State and Action Spaces

The BDQN was proposed to overcome the large discrete action spaces based on DQN, which uses neural network function approximation in Q-learning; therefore, we first review Q-learning and DQN as a way to overcome the large state spaces, and proceed to the survey of the BDQN agent.

Q-learning is based on estimating the expected total discounted future rewards of each state-action pair under the policy π: $Q_\pi(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-t} r_T | \pi]$, where s_t, a_t, r_t, and γ denote the states, action, reward, and discount factor, respectively. The Q function can be computed recursively with dynamic programming as follows:

$$Q^\pi(s, a) = \mathbb{E}_{s'}\left[r + \gamma \mathbb{E}_{a' \sim \pi(s')}\left[Q^\pi(s', a')\right] | s, a, \pi\right]. \tag{1}$$

We define the optimal $Q^*(s, a) = \max_\pi Q^\pi(s, a)$. Then, the optimal Q function satisfies the Bellman equation: $Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right]$. Q-learning is an off-policy TD control algorithm, and the one-step Q-learning is defined by:

$$Q(s_t, a_t) = (1 - \alpha_t) Q(s_t, a_t) + \alpha_t (r_{t+1} + \gamma \max_a Q(s_{t+1}, a)), \tag{2}$$

where α is the learning rate. When the state-action space is small enough for the Q-values to be represented as a lookup table, this iterative approximation converges to the true Q-values. However, this tabular-type Q-value representation soon faces problems due

to the large state-action space. Q-learning with function approximation has been proposed to overcome this problem. In the function approximated Q-learning, the following loss function needs to be minimized in the training process:

$$L(\theta) = \mathbb{E}[(y - Q(s, a; \theta))^2], \tag{3}$$

where

$$y = r + \gamma \max_{a'} Q(s', a'; \theta'), \tag{4}$$

and θ is the parameter of the neural network. A target network and experience replay have been proposed to cope with the problems related to non-stationarity and correlation in the sequence of observations [9].

Although deep RL has achieved a remarkable success for a large state space as described above, the problem associated with a large action space remains unsolved. To manage the large discrete action spaces, Ref. [13] proposed BDQN for robot control. If a robot has multiple movable parts (like arms, legs, fingers, etc.) and each part has several action options, then its combination of actions grows exponentially with the number of movable parts.

In BDQN (see the left-hand side of Fig. 1 for illustration), function approximated Q-values are represented with individual network branches following a shared network module that encodes a latent representation of the input and helps with the coordination of the branches. This architecture enables the linear growth of the total number of network outputs with increasing dimensionality of actions. The results showed that this branching agent performed well against the state-of-the-art continuous control algorithm, deep deterministic policy gradient (DDPG).

In this architecture, by employing a single global reward, the temporal difference target and loss function are defined as follows:

$$y_d = r + \gamma \frac{1}{N} \sum_d Q_d^- \left(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d \left(s', a'_d \right) \right), \tag{5}$$

$$L = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\frac{1}{N} \sum_d \left(y_d - Q_d \left(s, a_d \right) \right)^2 \right]. \tag{6}$$

Here, $d \in \{1, \ldots, N\}$ denotes the action dimension with $|\mathcal{A}_d| = n$ discrete sub-actions; y_d denotes the temporal difference target for branch d; $a_d \in \mathcal{A}_d$ denotes the sub-action; Q_d denotes the Q-value at state s and sub-action a_d for the branch d; Q_d^- denotes the target network; $\mathcal{D}$ denotes the experience replay buffer; and a denotes the joint-action tuple $(a_1, a_2, \ldots, a_N)$. The researchers stated that Eq. (5) showed better results than the naive setting for the temporal difference target:

$$y_d = r + \gamma Q_d^- \left(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d \left(s', a'_d \right) \right). \tag{7}$$

They also applied a dueling network into their branching architecture by setting the common state-value estimator and sub-action advantage as:

$$Q_d \left(s, a_d \right) = V(s) + \left(A_d \left(s, a_d \right) - \frac{1}{n} \sum_{a'_d \in \mathcal{A}_d} A_d \left(s, a'_d \right) \right), \tag{8}$$

where $V(s)$ denotes the common state value and $A_d \left(s, a_d \right)$ denotes the corresponding sub-action advantage.

According to the survey paper [2], control theory is an important research avenue in the supply chain management from the perspective of challenging uncertainties, feedback cycles and dynamics. Furthermore, there have been several attempts for the application of the control theory for supply chain management in the literature. Therefore, it would not be a bad idea to apply the approach in robot control, which is one of the most representative application fields of control theory, to supply chain management. As in Ref. [13], where they treated each movable part as an independent decision maker, we come up with the idea of treating each product as an independent decision maker.

However, unlike [13], where only single global reward is available, all the costs except the transportation costs can be calculated with respect to each product in our problem setting. Therefore, we added the reward allocation mechanism to the original BDQN agent such that the agent could learn coordinated ordering policy across multiple branches, where the part of rewards can be calculated with respect to each branch. The explanation of the proposed agent is provided in Section 3.

3. Method

3.1 Problem Setting

We consider a multi-product inventory system between one supplier and one retailer in a periodic-review system, and products are delivered via maritime transportation. Our objective is to minimize the total retailer cost, which includes the holding, penalty, and transportation costs. We assumed a non-stationary demand and a demand forecast conditioned by the forecast error parameter, which means that the agent knows the expected demand forecast accuracy as a form of prior knowledge. We have used the following notations:

i : Item number, $i = 1, \ldots, N$,
t : Period, $t = 1, \ldots, T$,
LT : Lead time from supplier to retailer, (in weeks),
l_i : Lot size of item i, (in palette),
$d_{i,t}$: Demand for item i during period t, (in palette),
$f_{i,t}$: Forecast of the demand for item i during period t,(in palette),
$x_{i,t}$: Order quantity for item i made at time t, (in palette),
$p_{i,t}$: Replenishment for item i from supplier during period t, (in palette),
$\hat{p}_{i,t}$: Replenishment forecast for item i from supplier during period t, (in palette),
$I_{i,t}$: Inventory position of item i at the start of time t, (in palette),
$\hat{I}_{i,t,\hat{t}}$: Inventory position forecast for item i at time $\hat{t}$ forecasted at time t (in palette),
$u_{i,t}$: Unsatisfied demand of item i during period t, (in palette),
$s_{i,t}$: Shipment of item i from retailer during period t, (in palette),
E_i : Forecast error parameter of item i.

The demand forecasts are generated so that the proportion of standard deviation of forecast error $(d_{i,t} - f_{i,t})$ to the standard deviation of demand itself equals E_i. Thus, E_i represents the degree of demand forecast accuracy. $E_i = 0$ means a perfect forecast, whereas $E_i = 1$ means no effect of prediction. Let E_i be 0.5 for all items in our experiments. We permitted the lost sales. Replen-

ishment at time t can be used from time $t+1$. In this study, we do not take supplier stock-out or any supply delay into consideration. Thus, the relationship among inventory, replenishment, shipment, demand, unsatisfied demand and inventory position forecast can be formulated as follows.

$$\hat{p}_{i,t+LT_i} = x_{i,t}, \tag{9}$$

$$p_{i,t} = \hat{p}_{i,t}, \tag{10}$$

$$s_{i,t} = \min(d_{i,t}, I_{i,t}), \tag{11}$$

$$u_{i,t} = d_{i,t} - s_{i,t}, \tag{12}$$

$$I_{i,t+1} = I_{i,t} - s_{i,t} + p_{i,t}, \tag{13}$$

$$\hat{I}_{i,t,\hat{t}+1} = I_{i,t} - \sum_{t'=t}^{\hat{t}} f_{i,t'} + \sum_{t'=t}^{\hat{t}} \hat{p}_{i,t'}. \tag{14}$$

Cost is defined as follows:

$$C_{i,t}^{\text{hold}} = U^{\text{hold}} \times I_{i,t}, \tag{15}$$

$$C_{i,t}^{\text{pel}} = U^{\text{pel}} \times u_{i,t}, \tag{16}$$

$$C_{t}^{\text{trans}} = U^{\text{trans}} \times \left\lceil \frac{\sum_i x_{i,t}}{CAP} \right\rceil, \tag{17}$$

where CAP represents the container capacity (in palette) and $\lceil\cdot\rceil$ is the ceiling function. $C_{i,t}^{\text{hold}}$, $C_{i,t}^{\text{pel}}$, and C_{t}^{trans} represent the holding, penalty, and transportation costs of item i during period t, respectively, and U^{hold}, U^{pel}, and U^{trans} are the unit holding, shortage, and transportation costs respectively.

In this problem setting, we tried to set our unit costs and other logistic conditions to have realistic values as much as possible. When we deliver goods from China to Japan using a 20-ft container ship, its maritime cost would be approximately \$400. The holding cost in Japan is approximately \$1.5 per m^3 per day, and a palette is approximately 1.2 m^3 in its size. Each product demand fits onto a palette, which is the usual method of packing in a container ship, so that the cost calculation would be consistent with an actual business setting. A 20-ft container can accommodate approximately 15 to 20 palettes; therefore, our container capacity (CAP) is 20 palettes. The order quantity unit size, which we call the lot size, should be an integer in palette. Demand and forecast can be specified in decimals because a customer's order to the retailer would be stated in pieces rather than palettes. Throughout our study, we let LT (which is the time required from order to delivery) to be three weeks by assuming maritime transportation between China and Japan.

3.2 MDP Formulation for Multi-product Inventory System with Demand Forecasts

3.2.1 Observations and State Variables

We need to consider the partial observability for the inventory control with the demand forecast. The partially observable Markov decision process (POMDP) provides a framework for making decisions under uncertainties. A POMDP is defined as a tuple (S, A, O, T, Z, R), where S, A, and O are the state, action, and observation space, respectively. The state-transition function $T(s, a, s') = P(s' \mid a, s)$ is the probability of the agent being in the state s' after taking action a in the state s. The observation function $Z(s, a, o) = P(o \mid a, s)$ is defined as the probability of the agent receiving observation o after taking action a in the state s.

In a POMDP, the agent cannot know its exact state, and belief $b(s)$, which is the probability distribution over S, is used. We can define $b(s)$ as $b(s) = P(s \mid h)$, where h denotes the past observation and action. Although the belief states along with the updating rule form a completely observable MDP, its learning process is computationally intensive. Among the several heuristic approaches for POMDP, Ref. [6] proposed Q-MDP approximation, which defined the following expression: $Q(b, a) = \sum_s b(s) Q^{MDP}(s, a)$.

With the availability of demand forecast information, the order decision at time t has been made primary based on the future inventory position at time $t + LT$ to ensure that our inventory satisfies the future demands after order replenishment. At every time step, meaning the beginning of every week, the agent obtains information about the future inventory positions according to the demand forecast. The on-order quantity $OO_{i,t}$ of item i at time t (i.e., the items that have been ordered but have yet been received) can be defined by $\sum_t \hat{r}_{i,t}$. Let $[\cdot]^{st:T}$ be the summation of $[\cdot]$ from st to $st + T$. In each period, the agent has observations as $o_t = [(I_{i,t}, OO_{i,t}, \hat{I}_{i,t,t+LT}, f_{i,t}^{t:LT}, f_{i,t}^{t+LT:M})]_{i=1}^{N}$ and makes a decision based on o_t. Here, M is the parameter that decides how far the future demand needs to be considered and we allow M to be four weeks. The definition of MDP states that the next state and the next reward should be only decided by the current state and the action taken at time t; o_t cannot be defined as a state because the actual future inventory position or/and future demand can be different from the forecasted inventory and demand. However, true information on $I_{i,t+LT}$, $d_{i,t}^{t:LT}$, and $d_{i,t}^{t+LT:M}$ can be observed afterward by the use of the actual demand. Thus, we can define the state by $s_t = [(I_{i,t}, OO_{i,t}, I_{i,t+LT}, d_{i,t}^{t:LT}, d_{i,t}^{t+LT:M})]_{i=1}^{N}$.

Let us assume that the average demand forecast error and the variance of demand do not change from time to time and that the agent knows E_i and the variance of demand for item i (var_i) as a form of prior knowledge, e.g., through past historical observations. Then, our *belief* state $b(s)$ should be conditioned only on o_t and can be defined by $P(s_t|o_t)$ instead of $P(s_t|h)$ in a general POMDP. In addition, by assuming that the expected forecast error follows a normal distribution, we can infer state s_t from observation o_t, since $d_{i,t}$ follows $\mathcal{N}(f_{i,t}, \sqrt{var_i}E_i)$.

Here, we have several approaches for this problem setting. Since we can obtain the true state information, experience memory can consist of the true state so that the Markovian property holds (option T_1). Otherwise, we can use the observation while ignoring the partial observability (option T_2). For the selection of action, we have to take action based on our observation, and we have two options: to use the demand forecast itself (option V_1), or to estimate the true state by using our prior knowledge about the demand forecast accuracy (option V_2). Illustrations of these approaches are provided in **Table 1**.

Note that our belief state $b(s)$ can be calculated by the use of only o_t and E_i. While greedily selecting the action in option V_2, we used the following expression for the greedy policy:

$$a_t = \arg\max_a \mathbb{E}[Q(\hat{s}_t, a)], \tag{18}$$

where $\hat{s}_t$ are the estimated states that use the Monte Carlo sampling by using the abovementioned distribution of $d_{i,t}$. In our ex-

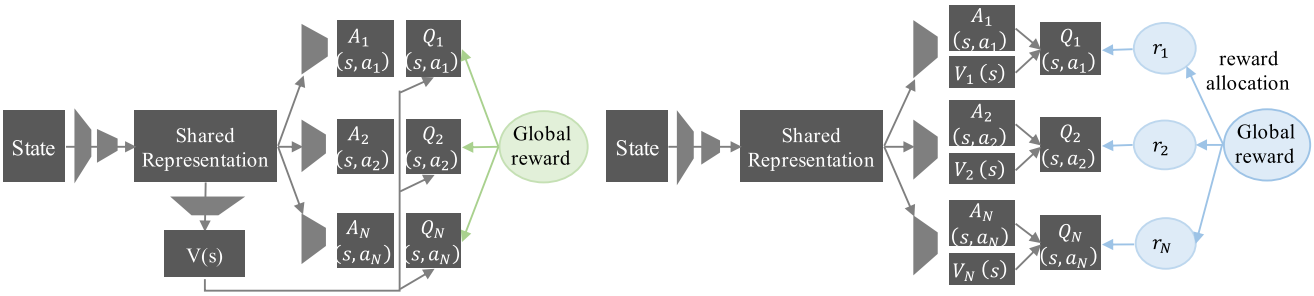

Fig. 1 Left-hand side: BDQN. Right-hand side: BDQN-RA.

Table 1 Options for experience memory and action selection.

Training	T_1	Select Action	$a_t \leftarrow \arg\max_a Q(s_t, a_t)$
		Memorize	$\mathcal{D} \leftarrow [s_t, a_t, r, s_{t+1}]$
	T_2	Select Action	$a_t \leftarrow \arg\max_a Q(o_t, a_t)$
		Memorize	$\mathcal{D} \leftarrow [o_t, a_t, r, o_{t+1}]$
Validation	V_1	Select Action	$a_t \leftarrow \arg\max_a [Q(o_t, a)]$
	V_2	Select Action	$a_t \leftarrow \arg\max_a \mathbb{E}[Q(\hat{s}_t, a)]$

Two options for memorizing the experience in the training period and selecting an action in validation period are represented.

periments, we generated 300 samples at each step. We assumed that the combination of option T_1 and V_2 would yield the best performance. Contrary to our initial expectations, the combination of T_2 and V_2 performed the best, where we treated $Q(o_t, a_t)$, that is obtained with option T_2, as if it represents $Q(s_t, a_t)$ in the validation phase. Therefore, we adopted this strategy for our experiments.

3.2.2 Action Space

In each period, an agent orders $x_{i,t} \in X_i$, which can be any multiples of lot sizes l_i. However, an infinite action space is not practical. In addition, taking a large number of orders as compared with the demand is unrealistic from a supply chain point of view. Therefore, we limited the possible order quantity for product i to $X_i = \{l_i a_i \mid a_i \in \mathcal{A}_i = \{0, 1, 2, 3\}\}$ where $\mathcal{A}_i$ denotes the action space for product i.

3.3 Branching Deep Q-network with Reward Allocation

In Ref. [13], they examined the action branching agent for an environment, which only had the availability of a global reward. In our case, each branch consisted of one product. Our objective variable was the total cost; the transportation cost was calculated across multiple products whereas the holding cost and penalty cost were calculated independently for each product, which means that the total cost included both global and local rewards.

After conducting several numerical experiments, our best result arrived from the architecture shown on the right-hand side in **Fig. 1**, which had the distinguishing feature of allocating rewards to each branch.

Our proposed agent can be considered as one of the multi-agent reinforcement learning approaches; i.e., each agent controls only one item independently. Our proposed agent and general multi-agent reinforcement learning are same in that an action for each item is taken independently. The difference lies in that our proposed agent has the shared representation, which can be regarded as a constraint from multi-agent perspective that all the agents have same parameters for the shared part in the Q-value function approximation, whereas each agent has completely separate parameters in general multi-agent reinforcement learning setting. In JRP, for some cases, even if the inventory of an item is large enough at particular time step, simultaneous order with other items would decrease the total costs. We assumed that the shared representation would enable for our agent to learn the aforementioned coordinated replenishment policy.

Since there are several ways for the reward allocation, and the appropriate reward allocation is not obvious, we examined the two reward allocation strategies; equal allocation strategy and quantity-based allocation strategy.

3.3.1 Reward Allocation

Unlike [13], we modified our temporal difference target equation as follows:

$$y_{i,t} = r_{i,t+1} + \gamma Q_i^- \left(s_{t+1}, \arg\max_{a_i' \in \mathcal{A}_i} Q_i\left(s_{t+1}, a_i'\right) \right), \tag{19}$$

where $r_{i,t+1}$ refers to the immediate reward for item i after taking an action at time t. Note that we use the item number i to represent the branch index. The transportation costs depend on the number of containers, and the remaining costs can be separately calculated for each product. There are several options available for the allocation of the total transportation cost to each product.

The most natural approach for the allocation of the transportation cost is proportionate to the order quantity of each product, which we call quantity-based allocation strategy, and is defined as follows:

$$r_{i,t+1} = -\left(C_{i,t}^{\text{hold}} + C_{i,t}^{\text{pel}} + C_t^{\text{trans}} \times \frac{x_{i,t}}{\sum_i x_{i,t}} \right). \tag{20}$$

However, the best results arrive from the following equal allocation strategy by which the total transportation cost is allocated equally to all the products even if a specific product is not ordered:

$$r_{i,t+1} = -\left(C_{i,t}^{\text{hold}} + C_{i,t}^{\text{pel}} + \frac{C_t^{\text{trans}}}{N} \right). \tag{21}$$

Intuitively, this allocation method would encourage each branch to put an order simultaneously.

Thus, the loss function should be defined for each branch, and all the back-propagation gradients of the branches are rescaled by $1/N$ for the shared part of our architecture.

$$L_i = \mathbb{E}_{(s,a_i,r_i,s') \sim \mathcal{D}} \left[L_\delta \left(y_i, Q_i\left(s, a_i\right) \right) \right], \tag{22}$$

where L_δ is the Huber loss function.

3.3.2 *n*-step TD Method

n-step TD method was devised to combine the merits of both the Monte Carlo and TD methods. The target in Monte Carlo backups is the return; whereas, the target in the one-step TD method is the first reward plus the discounted estimated value of the next state.

In our problem setting, there is obviously a reward delay because of the lead time from the supplier to the retailer. Finally, our target is defined as follows:

$$y_{i,t} = R_{i,t+1} + \gamma^n Q_i^- \left(s_{t+n}, \arg\max_{a_i' \in \mathcal{A}_i} Q_i \left(s_{t+n}, a_i' \right) \right), \tag{23}$$

where

$$R_{i,t+1} = \sum_{k=0}^{n-1} \gamma^{(k)} r_{i,t+k+1}. \tag{24}$$

Here, we let the step size n be the lead time LT.

3.3.3 State-value Estimator

As mentioned in Section 2, BDQN has a common state-value estimator. It is natural to set the branch-independent state-value estimator as an adaptation of dueling network into our proposed agent with reward allocation. Thus, the branch independent state value and advantage can simply be defined as follows:

$$Q_i \left(s, a_i \right) = V_i(s) + A_i(s, a_i), \tag{25}$$

where $V_i(s)$ denotes the branch independent state-value and $A_i \left(s, a_i \right)$ denotes the corresponding sub-action advantage.

4. Experiments

4.1 Experimental Setting

We conducted several numerical experiments to answer the following questions:

1) Can the proposed agent learn the coordinated order policy across multiple products?
2) Can the proposed agent converge with a large number of products?
3) Can the proposed agent converge with a non-stationary demand?
4) Can the proposed agent find an optimal policy as compared to the benchmark policy?
5) How the reward allocation strategy affect the agent's behavior?

For validation, we used the standard dueling double DQN, which we simply call DQN in this paper, and two benchmark policies, called a forecast-based economic order policy (F-EOP) and SIMPLE. DQN was employed to validate the learning convergence of the proposed agent as a RL-based approach, whereas two benchmark policies were employed to compare the result obtained by employing this proposed agent. Each episode consisted of 200 time steps, and the initial 20 steps were ignored from the evaluation so as to exclude the effect of initial inventory setting.

We tried three types of agents with respect to the branching architecture: BDQNs (BDQN with state-value estimator, proposed in Ref. [13]), BDQN-RA (BDQN with reward allocation), and BDQN-RAs (BDQN with reward allocation and state-value estimator).

In order to validate the abovementioned questions, we conducted four experiments by varying the number of products and the demand stationarity. Detailed explanations on experiments are provided in Section 4.6.

4.2 Benchmark Methodology

There has been no established JRP under the non-stationary demand and demand forecast; therefore, we selected a non-coordinated order policy based on Ref. [3] as our benchmark policy, which consisted of forecast-based order-point and an economic replenishment quantity based on Wagner-Whitin dynamic lot size model with extension to incorporate the demand forecasts.

4.2.1 Forecast-based Economic Order Policy

Based on the availability of demand forecasts, a replenishment order takes place when the forecasted inventory position at time $t + LT$ drops to the order-point or lower than that. Assuming that the forecast error follows a normal distribution, the order-point can be defined as $s = k \times \sigma \sqrt{LT}$ where k is the safety factor and σ is the standard deviation of forecast error.

At each time step, we choose the order quantity $x_{i,t} \in X_i$. When $x \in X_i$ is selected at time t, the expected unit time cost from $t+LT$ to the next replenishment timing is calculated by dividing the sum of the expected holding cost and the transportation cost by T:

$$C(x) = \frac{U^{\text{trans}} \times \lceil \frac{x}{CAP} \rceil + U^{\text{hold}} \times \sum_{\hat{t}=t+LT}^{t+LT+T} \hat{I}_{i,t,\hat{t}}}{T}, \tag{26}$$

where T is determined by estimating the timing for which the forecasted inventory position drops to or lower than the order-point on condition that x is replenished at time $t + LT$. Thus, the economic replenishment quantity with the demand forecasts can be derived by $\arg\min_x(C(x))$.

4.2.2 Simple Heuristics

Since with F-EOP, each product tries to enhance the fill-rate of container independently of other product order quantities, the order quantity would become too large and lead to the large inventory cost when the total demand per one time step is equal to or above the container capacity. Therefore, we also used the simple heuristic order policy, which we call it SIMPLE, as our benchmark policy, where each product puts an order with the minimum lot size when the inventory position drops to or lower than the order-point in F-EOP.

4.2.3 RL Agent Benchmark

In the BDQNs, the temporal difference target is defined as follows:

$$y_{i,t} = R_{t+1} + \gamma^n Q_i^- \left(s_{t+n}, \arg\max_{a_i' \in \mathcal{A}_i} Q_i \left(s_{t+n}, a_i' \right) \right), \tag{27}$$

where

$$Q_i \left(s, a_i \right) = V(s) + \left(A_i \left(s, a_i \right) - \frac{1}{|\mathcal{A}_i|} \sum_{a_i' \in \mathcal{A}_i} A_i \left(s, a_i' \right) \right), \tag{28}$$

$$R_{t+1} = \sum_{k=0}^{n-1} \gamma^{(k)} r_{t+k+1}, \tag{29}$$

$$r_{t+1} = -\left(\sum_i C_{i,t}^{\text{hold}} + \sum_i C_{i,t}^{\text{pel}} + C_t^{\text{trans}} \right). \tag{30}$$

Table 2 Summary of experiment settings and results.

ID	No. of Products	Demand Stationarity	F-EOP	SIMPLE	DQN	BDQNs	BDQN-RA	BDQN-RAs	Improved(%)
1	2	Stationary	(*) 133.2	141.4	105.3	108.0	(*) 106.7	107.3	19.9%
2	2	Upward Trend	(*) 216.5	237.6	205.2	266.3	(*) 192.0	195.2	11.3%
3	10	Upward Trend	447.8	(*) 376.7	-	817.2	346.8	(*) 341.3	9.4%
4	50	Upward Trend	1890.9	(*) 1580.9	-	-	(*) 1400.5	1404.1	11.4%

Results for DQN and BDQN families were derived by calculating the averaged total cost with a greedy policy using the trained model after 10,000 episodes over six runs. Improved(%) represents the decrease in total cost compared with the benchmark policy, and (*) denotes the item used for calculating Improved(%).

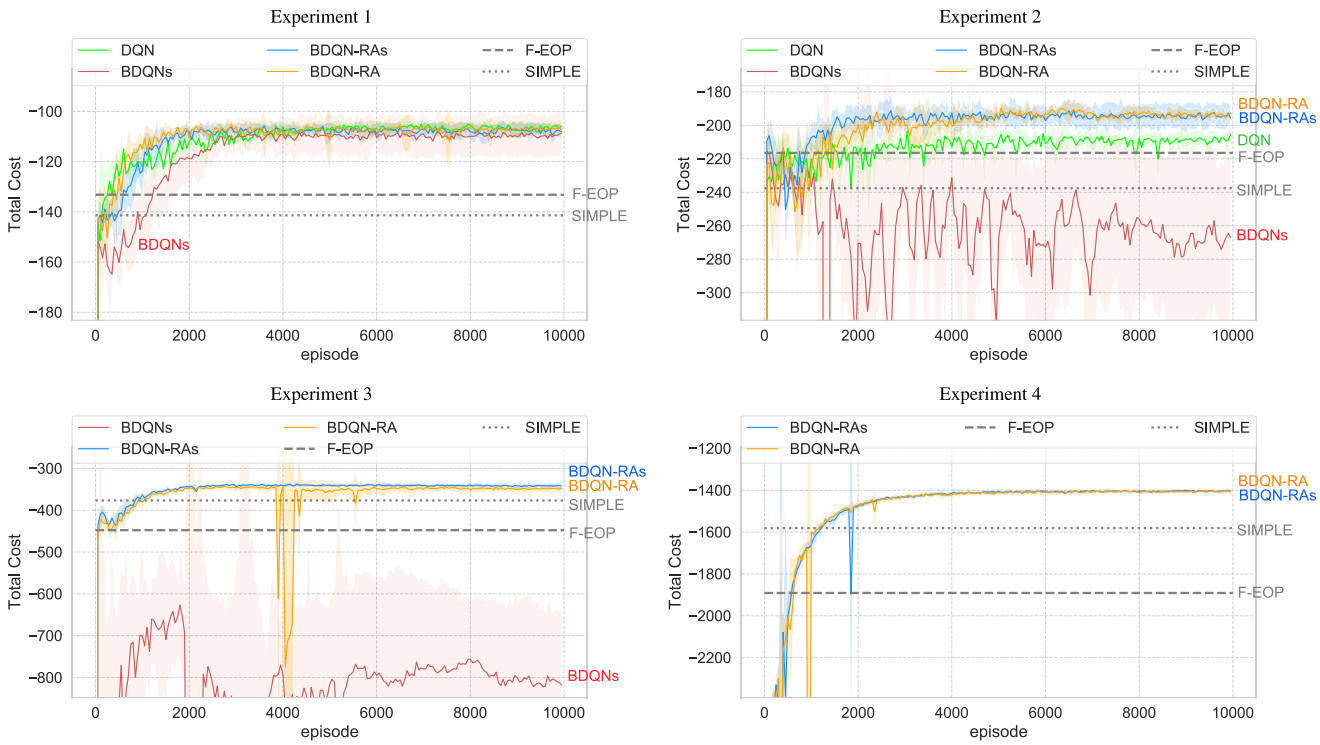

Fig. 2 Performance in total cost (multiplied by −1) during evaluation on the y-axis and training episodes on the x-axis. The solid lines represent the average over six runs with initialization seeds and the shaded areas represent the standard deviation. With the application of the greedy policy, evaluations were conducted every 50 episodes.

As shown, the immediate reward for branch i in the BDQNs is the total cost of all items incurred at particular time step.

4.3 Experiment Results

Table 2 and **Fig. 2** summarize the results and the learning curves of our experiments. Our proposed agent did perform better than did the benchmark policies. With the increase in the number of products, DQN and BDQNs (without reward allocation) did not converge whereas our proposed agent, BDQN-RA(s), performed well even with 50 products. DQN suffered from its combinatorial increase in the action space for a large number of products. When the number of products were equal to 10, its action space was around 10^6. As for BDQNs, the use of a single global reward made the convergence difficult because the feedback signal to each branch was considered too noisy. As shown in Eq. (30), each branch received the reward which is not only the transportation costs but also the inventory and penalty costs of other branches.

For the proposed agent, the reward allocation strategy worked better in stable learning while achieving the coordinated orders. In this case, as shown in Eq. (21), each branch received the reward which is branch independent holding and penalty costs plus allocated transportation cost. Thus, our proposed agent did not receive the irrelevant feedback to modify the replenishment policy for each item. The state-value estimator in the proposed agent did not demonstrate a considerable effect on the results.

We can see that with the increase in the number of products, the total cost from the benchmark SIMPLE is lower than another benchmark F-EOP as expected.

4.3.1 Experiment 1: Two-products with a Stationary Demand

One of the most important validation items about the action branching agent in JRP is to ascertain whether or not the coordinated order is possible across multiple products. We conducted a simple experiment to examine this possibility. In this setting, the total demand per unit time was much less than the transportation capacity, thus allowing room for coordinated orders to minimize the total cost.

As the learning process proceeded, our proposed agent learned to order these two products simultaneously, thereby minimizing the transportation cost. The left part of **Fig. 4** shows the result of the time series. Even if the inventory position of one item was relatively high (i.e., immediate order did not need to take place), the order took place in accordance with the other order. Through-

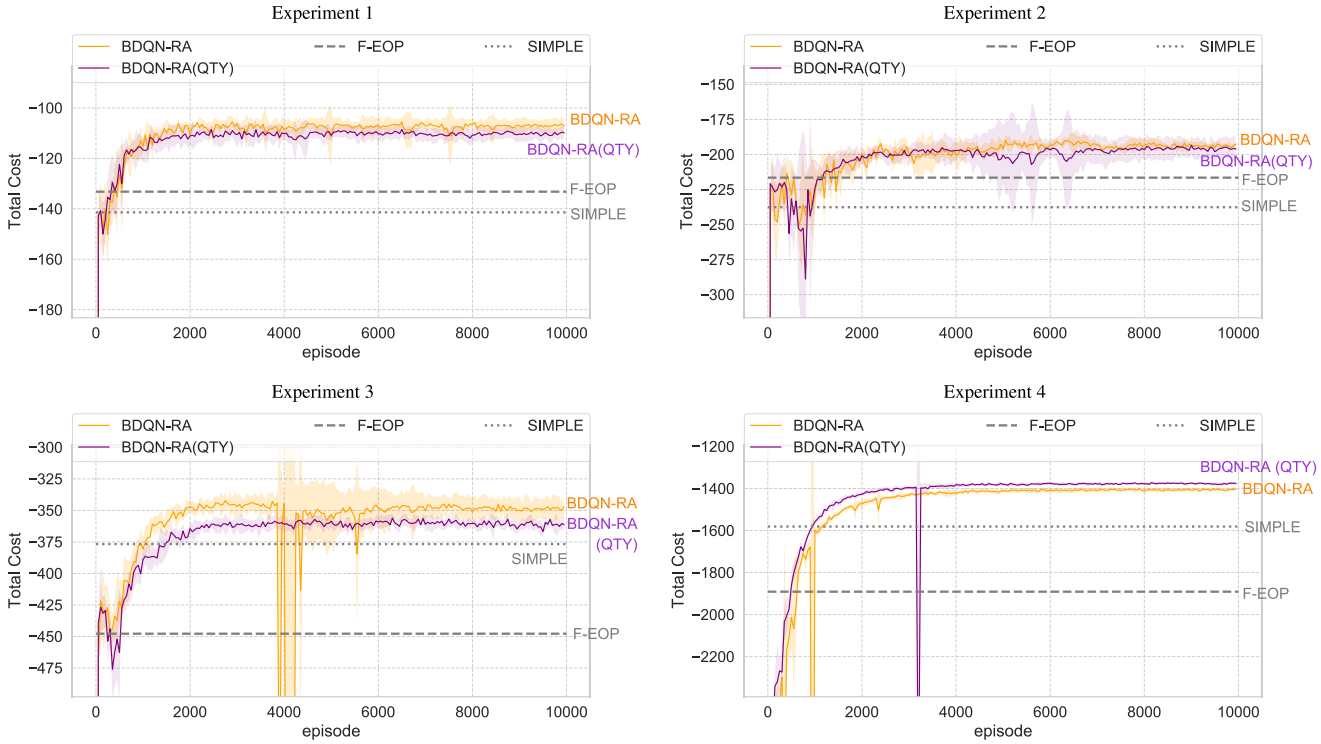

Fig. 3 Performance comparison in total costs of the reward allocation strategies : learning curve from BDQN-RA and BDQN-RA(QTY) agent. The solid lines represent an average over six runs with initialization seeds, and the shaded areas represent standard deviation. With the application of the greedy policy, evaluations were conducted every 50 episodes.

out these 200 time steps, most of the total order quantities per unit time equaled 16, whereas the order quantity of each product per unit time was 8. Considering the lot size of both products being 8, the coordinated orders occurred despite the fact that our proposed agent decided the order for each item independently. By doing so, our proposed agent achieved a low level of inventory while maintaining a high fill-rate for maritime transportation. As a result, the total cost decreased by 19.9% as compared with the benchmark result where the coordinated order did not take place, and the inventory level remained high as a result of independently minimizing the cost. The DQN agent performed best and it can be considered near the optimal JRP because the action space is small. Compared with DQN, our proposed agent demonstrated a slightly lower performance but still showed a considerable improvement over F-EOP. Despite this simple experimental setting, these results demonstrate the possibility of learning a coordinated ordering policy across multiple products with our branching architecture.

4.3.2 Experiment 2: Two-products with an Upward Demand

With upward demands, we expected that the ordering frequency should change from time to time with the increase in the total average demand per unit time. The bottom-right side of Fig. 4 shows the result of BDQN-RA and we see that the order timing is aligned between these two products in most of the time, with its frequency becoming smaller with increase in the average demand. On the contrary, the learning of the BDQNs agent was improper, which can be considered as the result of the agent been affected by the noisy joint-action selection under uncertain future demands. Although our agent performed better, a slight drop in performance and instability during the latter part of the training process was noticed. The agent is considered to have suffered from a non-stationary environment caused by branch-independent action selection, as observed in the general multi-agent RL setting.

4.3.3 Experiment 3: 10 Products with an Upward Demand

We extended our experiments to a more complicated setting with ten products, with the average unit time demand being much lower than the container capacity. The BDQNs agent failed to converge, whereas our proposed agent exhibited an efficient and stable learning process. The BDQN-RAs agent achieved a 9.4% cost reduction as compared to the benchmark policy.

4.3.4 Experiment 4: 50 Products with an Upward Demand

With 50 products, the BDQNs agent completely failed to converge (we have omitted the result for simplicity), whereas our proposed agent performed better than the benchmark policy. In this experiment, the average unit time demand is more than 38 palettes, which is far more than the container capacity (20 palettes per container). This means that we can ship products at every replenishment opportunity, and the resulting trade-off between the inventory costs and the transportation costs will be relatively small. Consequently, the learning process was more stable than the other three experiments.

4.4 Comparison of the Reward Allocation Strategy

Here we examined the reward allocation strategy. Let BDQN-RA(QTY) denote the agent with the order quantity-based allocation strategy, in which the transportation cost is allocated to each product in proportion to the order quantity as defined in Eq. (20). The rest of the setting is completely same as that of the BDQN-

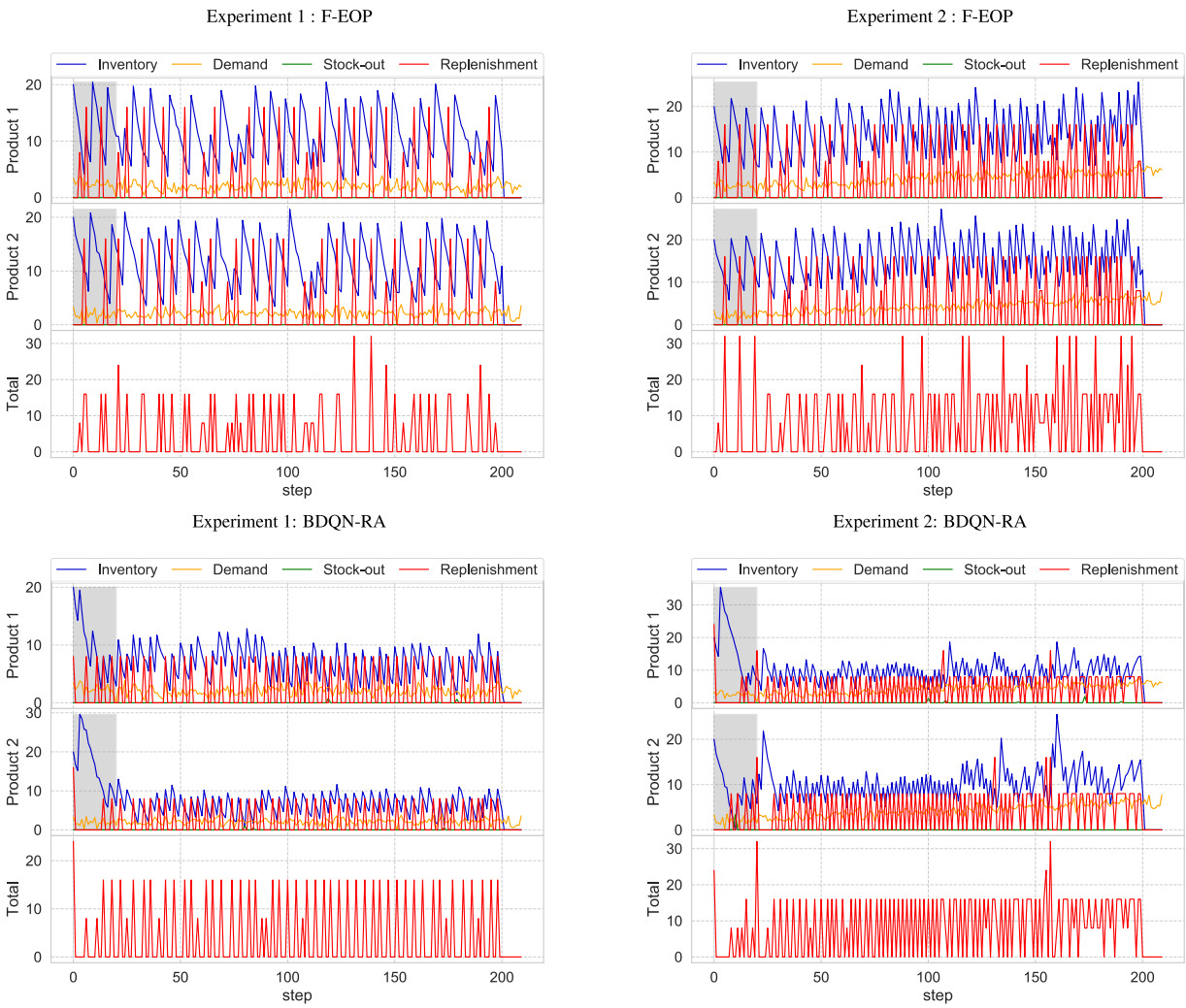

Fig. 4 Example of time series movement of each product derived from the validation results of benchmark (top) and BDQN-RA (bottom) for Experiments 1 (left) and 2 (right). Each figure represents the results of products 1 and 2, as well as the total replenishment quantity of both products (top to bottom). Orange, blue, green, and red lines represent demand, inventory, unsatisfied demand (stock-out), and replenishment quantity, respectively.

RA agent. **Figure 3** shows the learning curve from the BDQN-RA and BDQN-RA(QTY) agents. In all these experiments except for the Experiment 4, the BDQN-RA agent outperformed BDQN-RA(QTY).

The difference in performance may seem insignificant for the Experiment 1, but the resulting order behavior is quite different between these two agents. **Figure 5** shows the cost breakdown and the fill-rate of containers (on the left) and the average order quantity (on the right side) during the evaluation, and **Fig. 6** shows the example of time series movement. We observed that the inventory level from BDQN-RA is lower than that of BDQN-RA(QTY), whereas the fill-rate from BDQN-RA(QTY) is slightly higher. The BDQN-RA agent learned to order eight quantities every time, which is the minimum order quantity, whereas the average order quantity of BDQN-RA(QTY) agent is larger than eight. This considered to be affected by the reward allocation strategy: with order quantity-based allocation, each branch learned to enhance the fill-rate independently, whereas with equal allocation, each branch learned to order simultaneously with other branches.

The performance gap has been widened in the Experiment 3. **Figure 7** shows the cost breakdown and the fill-rate of containers during the evaluation, and **Fig. 8** shows the example time series movement. Note that the lot size of each product is 1, 1, 1, 1, 2, 2, 3, 3, 3, and 3, respectively and the demand has seen an upward trend in this setting. We see that the order quantity made by the BDQN-RA agent equaled the lot size (meaning the minimum order quantity) at almost every particular time and the inventory level was kept low, just by adjusting the order frequency with the increase in demand. On the other hand, the order quantity made by the BDQN-RA(QTY) agent was larger in the first half of the experiment. As a result, the inventory level was kept high.

On the other hand, if we look at the total order quantity (see the last row in Fig. 8), we can see that there are timings where no order has been put in the first half of the result from BDQN-RA(QTY) agent. By considering Eq. (20), we can interpret the result as follows: with quantity-based allocation, if the order for an item does not take place, no transportation cost is incurred for the item. Thus, during the training process, if the other item decides not to put an order, the incurred cost for an ordered item

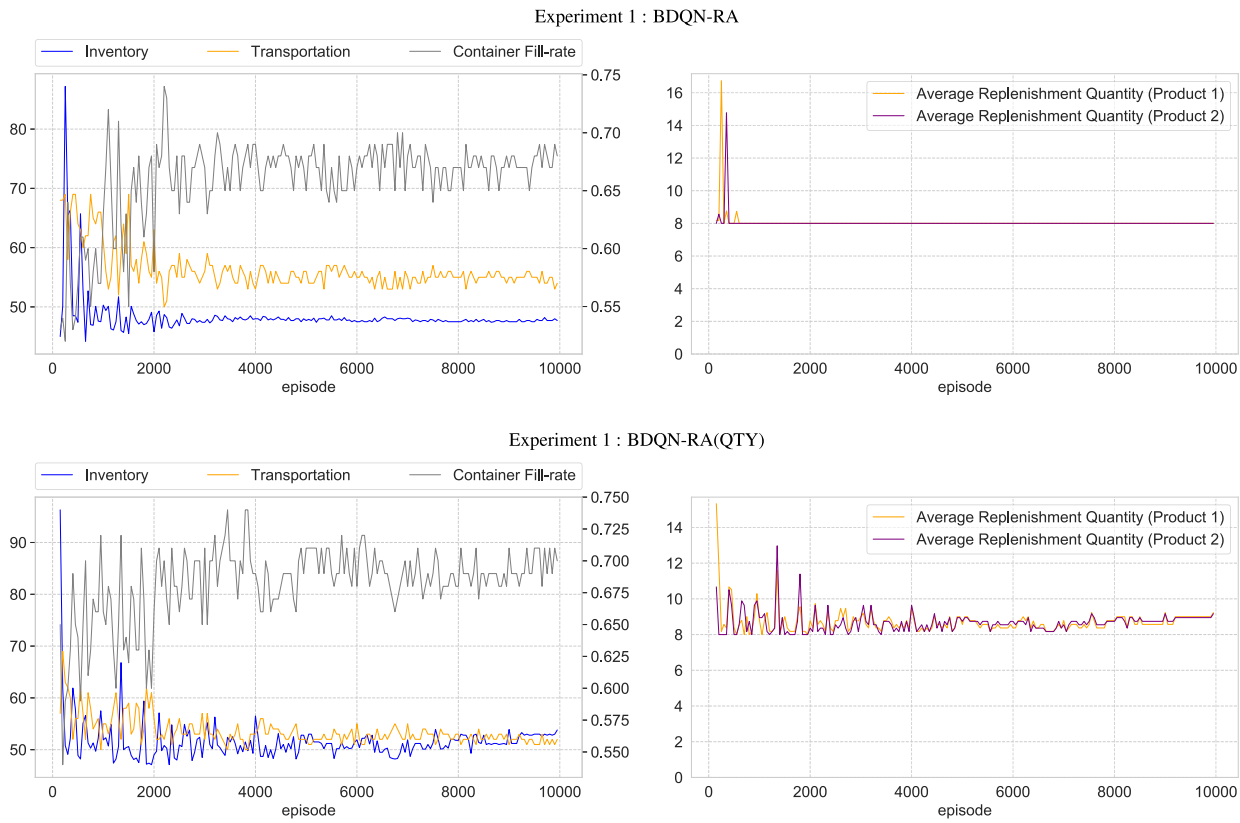

Fig. 5 Comparison of the reward allocation strategies for Experiment 1. The top part of the figure shows one of the results from the BDQN-RA. The bottom part of the figure shows one of the results from the BDQN-RA(QTY). Cost breakdown (on the left axis) and the fill-rate of containers (on the right axis) during evaluation (left part of the figure). Evaluations using the greedy policy were conducted every 50 episodes. Penalty cost is omitted for simplicity. Average replenishment quantity of each product during the evaluation (right part of the figure).

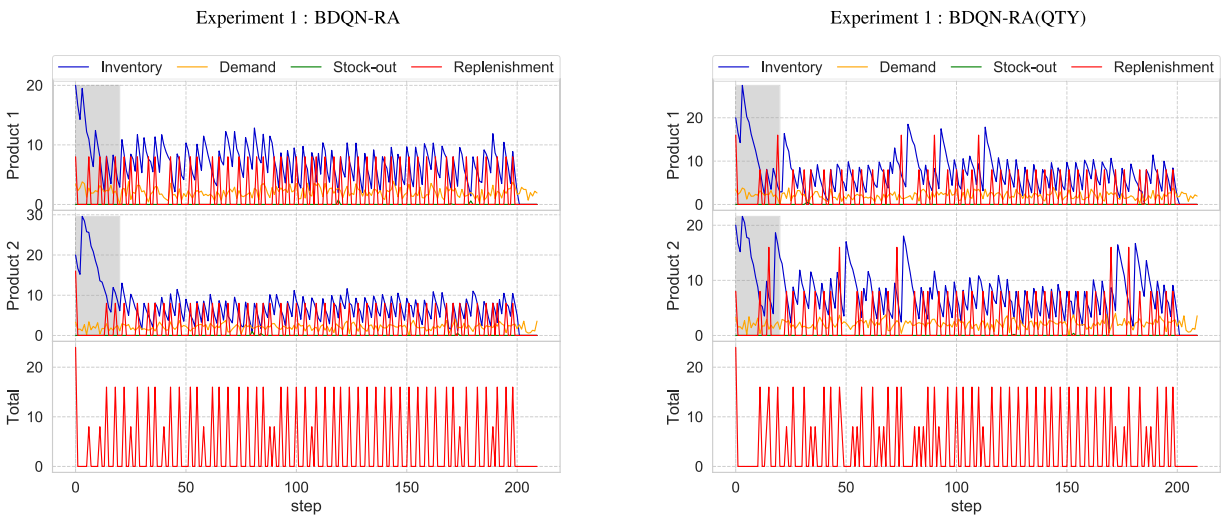

Fig. 6 Example of the time series movement of each product derived from the validation result of BDQN-RA (on the left) and BDQN-RA(QTY) (on the right) for Experiment 1. Each figure represents the result of product 1 and 2 as well as the total replenishment quantity of these two products, from top to bottom. Orange, blue, green, and red lines represent demand, inventory, unsatisfied demand (stock-out), and replenishment quantity, respectively.

increases even if the order quantity remains the same. This would eventually encourage all the items not to put an order simultaneously under some states. From these results, we can see that the reward allocation strategy had a large impact on the agent's behavior.

In the Experiment 4, the performance of the BDQN-RA(QTY) was slightly higher than the BDQN-RA agent. From Fig. 7, we can see that the transportation costs have not changed and the fill-rate is kept high during the training process. This implies a little room for improvement in the fill-rate, and the coordination among products is not required in this case. Therefore, quantity-based allocation strategy is considered effective in obtaining a more direct

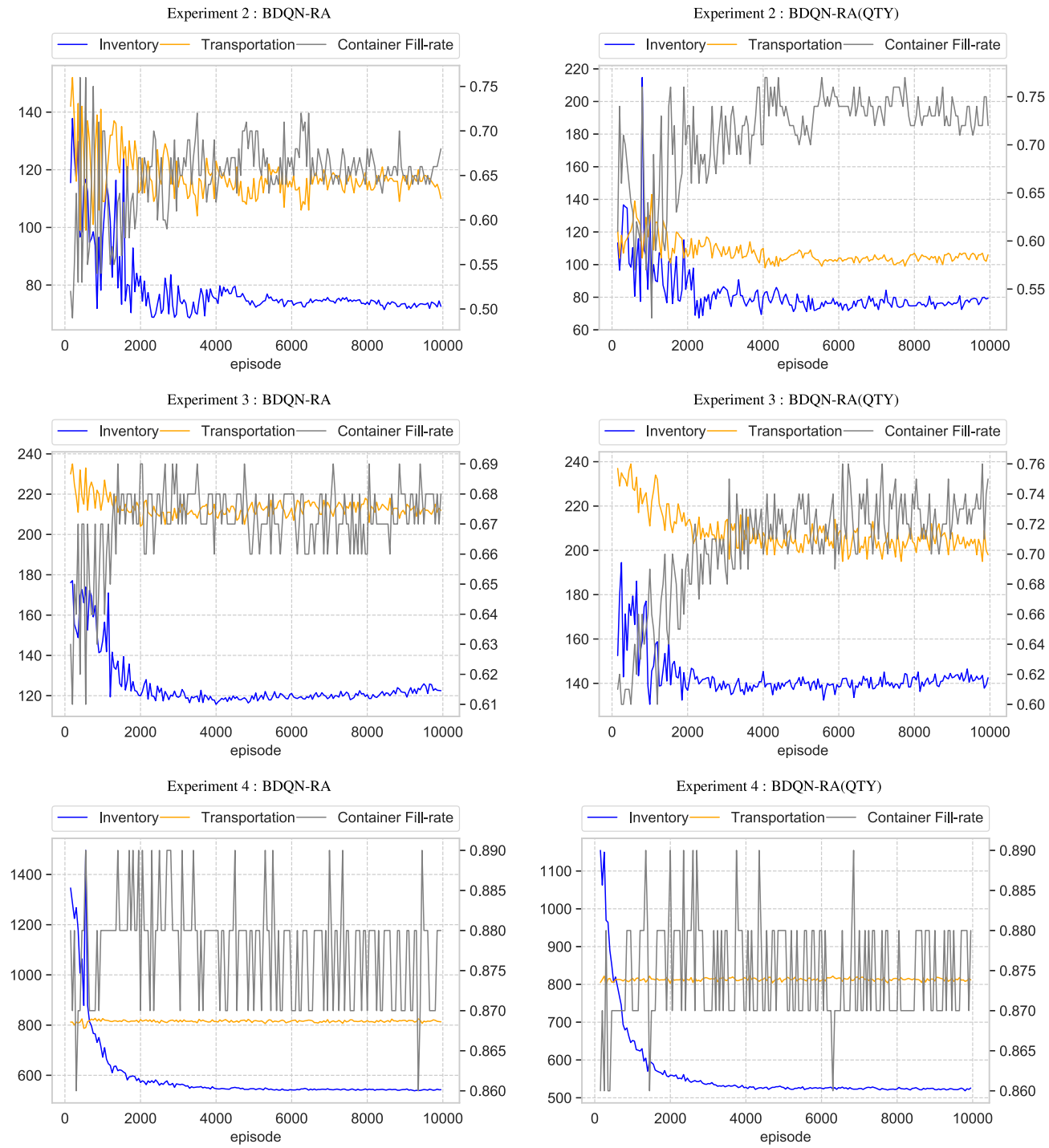

Fig. 7 Comparison of the reward allocation strategies for Experiments 2, 3, and 4. The left side of the figure shows one of the results from BDQN-RA, whereas the right side of the figure shows one of the results from BDQN-RA(QTY). Costs breakdown (on the left axis) and the fill-rate of containers (on the right axis) during evaluation are also shown. Evaluations using the greedy policy were conducted every 50 episodes. Penalty cost is omitted for simplicity.

feedback to optimize the inventory cost for each product.

4.5 Additional Experiments on Different Parameters

Based on the Experiment 3, i.e., the number of products is fixed to 10, we conducted additional experiments by considering the following situations: 1) higher/lower transportation unit cost, 2) different lot sizes. As shown in **Table 3**, BDQNs failed to converge for all cases, whereas our proposed agent outperformed the benchmark policy SIMPLE. In terms of the comparison between the two reward allocation functions, although the equal allocation outperformed the quantity-based allocation on average, we can see that the difference is not significant for cases on different U^{trans} settings. From these observations, the superiority of the reward allocation strategy may vary depending on the cost balance between the transportation and inventory costs along with other logistics conditions.

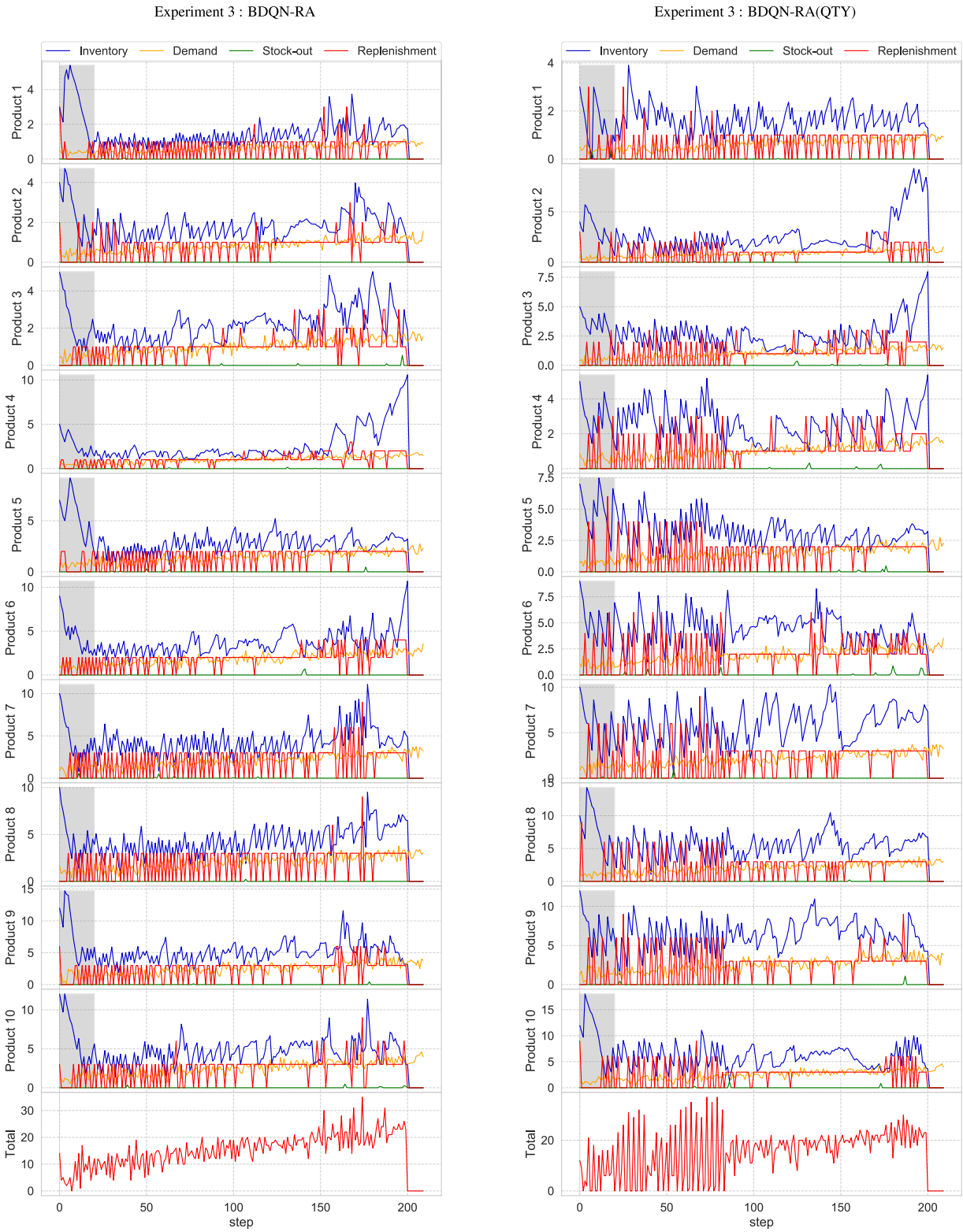

Fig. 8 Example time series movement of each product derived from the validation result of BDQN-RA (on the left) and BDQN-RA(QTY) (on the right) for Experiment 3. Each figure represents the result of product 1,2...10, and total replenishment quantity of all the products, from top to bottom. Orange, blue, green, and red lines represent demand, inventory, unsatisfied demand (stock-out), and replenishment quantity respectively.

4.6 Experiment Detail

4.6.1 Cost, Demand and Demand Forecast Setting

On the basis of the logistic condition described in Section 3, we let the cost parameters U^{hold}, U^{pel}, and U^{trans} be 0.02, 1.0, and 1, respectively. The demand and lot size of each product are presented in **Table 4**. The stationary demand was generated following $N(\mu, \sigma)$ and we let $\frac{\sigma}{\mu}$ be 0.4. Non-stationary data were generated by the simple addition of the linear upward trends until the tripling of demand at the end of the 200 time steps, defined by; $d_{i,t} = \hat{d}_{i,t} + 2\mu_i(t/200)$ where $\hat{d}_{i,t}$ denotes the stationary demand.

Table 3 Experiment results on different parameters.

Parameter		SIMPLE	BDQNs	BDQN-RA	(QTY)
U^{trans}	0.5	262.2	2,482.2 (1,909.9)	(*) 230.0 (3.6)	234.3 (1.7)
	2	593.7	1,284.4 (366.0)	(*) 581.2 (9.4)	592.5 (13.7)
Lot size (for all items)	2	365.1	2,199.7 (1,007.5)	(*) 347.8 (3.1)	371.8 (10.0)
	3	408.5	1,806.0 (969.4)	(*) 379.6 (4.3)	397.3 (2.8)

Results for BDQN families were derived by calculating the averaged total cost with a greedy policy using the trained model after 3,000 episodes over six runs. The value in (·) represents the standard deviation. F-EOP is omitted as the performance of SIMPLE is better for all cases. (*) represents the lowest cost.

Table 4 Demand and lot size settings in each experiment.

ID	μ	Lot size
1	[2, 2]	[8, 8]
2	[2, 2]	[8, 8]
3	[.3, .4, .5, .5, .7, .9, 1., 1., 1.2, 1.2]	[1, 1, 1, 1, 2, 2, 3, 3, 3, 3]
4	[.3, .4, .5, .5, .7, .9, 1., 1., 1.2, 1.2] ×5	[1, 1, 1, 1, 2, 2, 3, 3, 3, 3] ×5

Demand forecasts were generated so that the proportion of the standard deviation of the forecast error to the standard deviation of the demand itself equaled 0.5.

4.6.2 BDQNs and BDQN-RA(s)

The network had two hidden layers with 512 and 256 units for the Experiments 1, 2, and 3, whereas it had two hidden layers with 2048 and 1024 units for the Experiment 4 in the shared network module and one hidden layer per branch with 128 units. A gradient clipping of size 0.25 was applied. We used the Adam optimizer with a learning rate of 10^{-4}, $\beta 1 = 0.9$ and $\beta 2 = 0.999$. The target network was updated every ten episodes. A mini-batch size was 32 and a discount factor was 0.995. We used ReLu for all hidden layers and linear activation on the output layers. We adopted the ϵ-greedy policy with linear annealing.

4.6.3 DQN

We used the same parameters as for BDQN-family regarding gradient clipping, optimizer, learning rate, discount factor, mini-batch size, and ϵ-greedy policy.

5. Conclusion

We introduced the extended branching Q-learning agent with function approximation designed for combinatorial action dimension with global and local reward based on the cost structure of multi-product inventory system.

Our numerical experiments showed that with the increase in the number of products, both DQN and BDQNs failed to converge; however, our proposed agent performed better as compared with the benchmark policies. We also demonstrated that the reward allocation strategy played a key role in learning cooperative behavior with our branching agent.

Through our numerical experiments, although the equal allocation strategy outperformed the quantity-based allocation for all the experiments, we see that the reward allocation strategy can stand further improvement, and this should be investigated in future studies.

Our proposed agent only needed the demand forecast, which is usual in the real business setting; this result expands the possibility to adapt our approach in the real-world situations.

In this study, we used the same demand parameter μ for training and validation process; however, in practice, we need to prepare the unexpected demand level fluctuation. This generalization problem is one of the biggest challenges in the RL context, and we believe that this problem should be resolved for RL to be deployed in the real business applications. Future studies need to investigate the methods to enhance the generalization performance with our proposed agent.

References

[1] Balintfy, J.L.: On a Basic Class of Multi-Item Inventory Problems, *Management Science*, Vol.10, No.2, pp.287–297 (online), available from ⟨http://www.jstor.org/stable/2627299⟩ (1964).

[2] Dolgui, A., Ivanov, D., Sethi, S. and Sokolov, B.: Control Theory Applications to Operations Systems, Supply Chain Management and Industry 4.0 Networks, *Ifac-papersonline*, Vol.51, No.11, pp.1536–1541 (online), DOI: 10.1016/j.ifacol.2018.08.279 (2018).

[3] Ishigaki, A. and Hirakawa, Y.: Design of a Economic Order-Point System based on Forecasted Inventory Positions, *Journal of Japan Industrial Management Association*, Vol.59, No.4, pp.290–295 (2008).

[4] Johansen, S. and Melchiors, P.: Can-order policy for the periodic-review joint replenishment problem, Vol.54, No.3, pp.283–290 (online), DOI: 10.1057/palgrave.jors.2601499 (2003).

[5] Khouja, M. and Goyal, S.: A review of the joint replenishment problem literature: 1989–2005, *European Journal of Operational Research*, Vol.186, No.1, pp.1–16 (2008).

[6] Littman, M.L., Cassandra, A.R. and Kaelbling, L.: Learning policies for partially observable environments: Scaling up, pp.362–370 (1995).

[7] Minner, S. and Silver, E.A.: Multi-product batch replenishment strategies under stochastic demand and a joint capacity constraint, Vol.37, No.5, pp.469–479 (online), DOI: 10.1080/07408170590918254 (2005).

[8] Minner, S. and Silver, E.A.: Replenishment policies for multiple products with compound-Poisson demand that share a common warehouse, Vol.108, No.1-2, pp.388–398 (online), DOI: 10.1016/j.ijpe.2006.12.028 (2007).

[9] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. and Nature, V.J.: Human-level control through deep reinforcement learning, *Nature*, (online), DOI: 10.1038/nature14236 (2015).

[10] Ohno, K. and Ishigaki, T.: A multi-item continuous review inventory system with compound Poisson demands, *Math Method Oper Res*, Vol.53, No.1, pp.147–165 (online), DOI: 10.1007/s001860000101 (2001).

[11] Ohno, K., Ishigaki, T. and Yoshii, T.: A new algorithm for a multi-item periodic review inventory system, *Zeitschrift Für Operations Res*, Vol.39, No.3, pp.349–364 (online), DOI: 10.1007/bf01435462 (1994).

[12] Suetsugu, H., Narusue, Y. and Morikawa, H.: Joint Replenishment Policy in Multi-Product Inventory System Using Branching Deep Q-Network with Reward Allocation, *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, pp.115–121 (2019).

[13] Tavakoli, A., Pardo, F. and Kormushev, P.: Action branching architectures for deep reinforcement learning, *32nd AAAI Conference on Artificial Intelligence*, pp.4131–4138 (2018).

[14] Wagner, H.M. and Whitin, T.M.: Dynamic version of the economic lot size model, *Management Science*, Vol.5, No.1, pp.89–96 (1958).

Hiroshi Suetsugu received his B.E. and M.E. degrees from the University of Tokyo in 2006 and 2008, respectively. His current research interests are sequential decision making, time series analysis, and peer-to-peer electricity trading.

Yoshiaki Narusue received his B.E., M.E., and Ph.D. degrees from the Graduate School of Information Science and Technology, the University of Tokyo, Tokyo, Japan, in 2012, 2014, and 2017, respectively. Currently, he is a research associate with the Department of Electrical Engineering and Information Systems, the University of Tokyo, Tokyo, Japan. He received the second-best student paper award at IEEE Radio and Wireless Symposium in 2013, Hiroshi Harashima Academic Encouragement Award in 2013, TECO Green Tech Contest Bronze Award in 2016, and the best paper award at IEEE Consumer Communications and Networking Conference in 2018. His research interests include wireless power transfer, wireless communication, and the Internet of Things. He is a member of the IEEE, IEICE, and IPSJ.

Hiroyuki Morikawa received his B.E., M.E, and Dr. Eng. degrees in electrical engineering from the University of Tokyo, Tokyo, Japan, in 1987, 1989, and 1992, respectively. Since 1992, he has been in the University of Tokyo and is currently a full professor of School of Engineering at the University of Tokyo. From 2002 to 2006, he was a group leader of the NICT Mobile Networking Group. His research interests are in the areas of IoT/M2M/big data, sensor networks, wireless communications, and digital society design. He served as a technical program committee chair of many IEEE/ACM conferences and workshops, Vice President of IEICE, OECD Committee on Digital Economy Policy (CDEP) vice chair, Director of New Generation IoT/M2M Consortium, and he sits on numerous telecommunications advisory committees and frequently serves as a consultant to government and companies. He has received more than 70 awards including three IEICE best paper awards, IPSJ best paper award, JSCICR best paper award, Info-Communications Promotion Month Council President Prize, NTT DoCoMo Mobile Science Award, Rinzaburo Shida Award, and Radio Day Ministerial Commendation.