A Reinforcement Learning Approach for Joint Replenishment Policy in Multi-Product Inventory System

Hiroshi Suetsugu [1] **Yoshiaki Narusue** [1] **Hiroyuki Morikawa** [1]

Abstract

This study proposes a reinforcement learning approach to find the near-optimal dynamic ordering policy for a multi-product inventory system with non-stationary demands. The distinguishing feature of multi-product inventory systems is the need to take into account the coordination among products with the aim of total cost reduction. The Markov decision process formulation has been used to obtain an optimal policy. However, the curse of dimensionality has made it intractable for a large number of products. For more products, heuristic algorithms have been proposed on the assumption of a stationary demand in literature. In this study, we propose an extended Q-learning agent with function approximation, called the branching deep Q-network (DQN) with reward allocation based on the branching double DQN. Our numerical experiments show that the proposed agent learns the coordinated order policy without any knowledge of other products' decisions and outperforms non-coordinated forecast-based economic order policy.

1. Introduction

Own-brand goods play an important role for retailers in terms of their profits. At their own risk, many retail companies have tried to purchase their goods in the production country and deliver these goods directly to selling countries. In this case, the joint replenishment policy (JRP), which takes multi-product situations into account, is required to achieve the minimum total cost. In a supply chain where multiple products need to be delivered via a container ship, the maritime transportation cost depends on the required number of containers. Thus, the transportation cost per product would decrease when several products are ordered simultaneously. However, JRP is considered to be non-deterministic polynomial-time (NP)-hard because of its combinatorial nature. There has been much research on JRP. According to (dos Bastos et al., 2017), JRP research can be divided into three categories depending on the demand assumptions: deterministic, dynamic or stochastic. In the real-world business setting, almost all the businesses fall into the stochastic demand setting. Some studies use the Markov decision process to solve the JRP with the stochastic demand. The problem is complex in the stochastic demand setting; therefore, a number of studies were conducted on the assumption of the stationary demand which follows a Poisson or normal distribution.

On the other hand, many researchers have started using reinforcement learning (RL) for the supply chain management setting because of recent advances in RL. A typical problem setting is the Beer Game ((Oroojlooyjadid et al., 2017), (Chaharsooghi et al., 2008)), where a multi-echelon supply chain problem is considered under uncertain future demand and supply. (Oroojlooyjadid et al., 2017) used a DQN agent, which was presented in (Mnih et al., 2015), to obtain an order policy in the Beer Game. Although the existing literature showed positive results in multi-echelon supply chains, only one product has been considered so far. This is because of the problem of large discrete action spaces, which appear in the general RL approach. If each product has four discrete ordering options: zero (means not ordering), one, two and three (which are the number of the order, lot sizes), a combination of all possible actions would be 4^N (where N is the number of products). In this case, only 10 products would be intractable because its combination reaches 1,048,576. To take full advantage of RL in the supply chain management, these exponentially increasing action space problems with multiple products need to be resolved.

(Tavakoli et al., 2017) proposed the branching DQN (BDQN), in which function approximated Q-values are represented with individual network branches followed by a shared decision module that encodes a latent represen-

[1]The Graduate School of Engineering, The University of Tokyo, Japan. Correspondence to: Hiroshi Suetsugu <hsuetsugu@mlab.t.u-tokyo.ac.jp>.

Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 36^{th} International Conference on Machine Learning, Long Beach, California, USA, 2019.

tation 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 action dimensionality.

In this paper, we propose the RL agent with function approximation to find the near-optimal dynamic multi-product ordering policy under non-stationary demand based on BDQN combined with credit assignment. The proposed agent is based on the idea of treating a single agent as a multi-agent learner with a credit assignment policy that is a hybrid of the global and the local rewards with the aim of coping with the combinatorial increase in the action space. From several numerical experiments, we found that the branching deep Q-network with reward allocation (BDQN-RA) is a promising approach for solving the aforementioned problem.

We evaluate our proposed agent by applying it to a multi-product inventory control problem and ran numerical experiments that varied the number of products and demand stationarity. Our proposed agent achieves better performance than the forecast-based economic order policy (F-EOP) and exhibits a robust learning process even on a large number of products with non-stationary demand.

The reminder of this paper is as follows. In Section 2, we review the literature on both JRP and RL. In Section 3, we present our proposed solution. In Section 4, we present the results of the numerical experiment, and we provide our conclusions in Section 5.

2. Related work

2.1. Joint replenishment policy

If we consider only one product and if the stock status is determined only at the time of the review (usually called the periodic review system), then we see that the (s, S) policy has been widely used. In the (s, S) policy, if the inventory position is at or below the order point s, an order is placed. 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. (Wagner & Whitin, 1958) extended this method to well-known dynamic lot-sizing problem.

However, under a multi-product inventory system, the coordination across products should be taken into account, and (Khouja & Goyal, 2008) 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 considering a set of twenty products. The $(\boldsymbol{\sigma}, \boldsymbol{S})$ policy was proposed by (Johnson, 1967) for multi-product inventory problem where an order would be placed when the inventory position falls to or below σ and the inventory would be raised up to S. The $(\boldsymbol{S}, \boldsymbol{c}, \boldsymbol{s})$ policy was proposed by (Balintfy, 1964), which is often referred to as a "can-order" policy where an order is triggered by the 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. A number of studies have been conducted on how to find these policy parameters with some assumptions on the demand distribution or the lead time. Recent studies let the assumption be in more realistic settings. Certain studies (Minner & Silver, 2005), (Minner & Silver, 2007) have assumed stochastic demand, formulated the Markov decision process (MDP), and solved MDP with policy iteration. In these cases, the transition probabilities were explicitly described using the assumed demand distribution. Policy iteration with MDP is computationally intensive; therefore, the existing study ended up with the heuristic algorithm to handle a large number of products.

As (Larsen, 2009) pointed out, the effectiveness of JRP depends on the correlation between the demands across multi-products; the more negative the correlation, the less advantageous it is to coordinate the replenishment decisions by using JRP. In this setting, the decision maker should select whether or not they should use the coordinated order policy in advance.

2.2. Reinforcement learning for large discrete action spaces

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

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

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

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

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 because of the large state-action space.

Q-learning with function approximation has been proposed to overcome this problem. In 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 relating to non-stationarity and correlation in the sequence of observations (Mnih et al., 2015).

Whereas deep reinforcement learning has achieved 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, (Tavakoli et al., 2017) proposed BDQN in which n dimensional action branches followed the shared state representation in the neural network Q-function approximation (see Fig. 1 left-hand side). 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, the temporal difference target using a single global reward and loss function are defined as follows:

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

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

Here, $a_d \in \mathcal{A}_d$ denotes the action for the branch d; Q_d denotes the Q-function for the branch d; Q_d^- denotes the target network; $\mathcal{D}$ denotes the experience replay buffer; and $\mathcal{A}$ denotes the joint-action tuple $(a_1, a_2, ..., a_N)$. The researchers stated that Equation (6) showed better results than the naive setting for the temporal difference target:

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

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

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

where $V(s)$ denotes the common state value and $A_d(s, a_d)$ denotes the corresponding advantage. In this study, we attempted to extend this BDQN agent such that the agent could learn coordinated ordering policy across multiple products. The explanation of the proposed agent is provided in Section 3.

2.3. Reinforcement learning for partially observable Markov decision process

Regarding the inventory control with the demand forecast, we need to consider the partial observability. 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 state s' after taking action a in state s. The observation function $Z(s, a, o) = P(o \mid a, s)$ is the probability of the agent receiving observation o after taking action a in state s.

In a POMDP, the agent cannot know its exact state, and belief $b(s)$, which is probability distribution over S, is used. We can define $b(s)$ as $b(s) = P(s \mid h)$, where h denotes past observation and action. Although 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, (Littman et al., 1995) proposed Q-MDP approximation, in which $Q(b, a) = \sum_s b(s) Q^{MDP}(s, a)$ was defined.

3. Method

3.1. Problem setting

We consider a multi-product inventory system between one supplier and one retailer. 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 the agent knows the expected demand forecast accuracy as prior knowledge. We have used the following notations:

i : Item number, $i = 1, ..., N$,
t : Period, $t = 1, ..., T$,
LT_i : Lead time of item i from supplier to retailer,
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),
$r_{i,t}$: Replenishment for item i from supplier during period t, (in palette),

$\hat{r}_{i,t}$: 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 to the standard deviation of demand itself equals E_i. Thus, E_i represents the degree of demand forecast accuracy. $E_i = 0$ means perfect forecast whereas $E_i = 1$ means no effect of prediction. Let E_i be 0.5 for all items in our experiments. We permit lost sales. Replenishment 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 between inventory, replenishment, shipment, demand, unsatisfied demand and inventory position forecast can be formulated as follows.

$$\hat{r}_{i,t+LT_i} = x_{i,t}, \tag{9}$$
$$r_{i,t} = \hat{r}_{i,t}, \tag{10}$$
$$s_{i,t} = min(d_{i,t}, I_{i,t}), \tag{11}$$
$$u_{i,t} = d_{i,t} - s_{i,t}, \tag{12}$$
$$I_{i,t+1} = I_{i,t} - s_{i,t} + r_{i,t}, \tag{13}$$
$$\hat{I}_{i,t,\hat{t}+1} = I_{i,t} - \sum_{t'=t}^{\hat{t}} f_{i,t'} + \sum_{t'=t}^{\hat{t}} \hat{r}_{i,t'}. \tag{14}$$

Cost is defined as follows:

$$C_{i,t}^{hold} = U^{hold} \times I_{i,t}, \tag{15}$$
$$C_{i,t}^{pel} = U^{pel} \times u_{i,t}, \tag{16}$$
$$C_{t}^{trans} = U^{trans} \times \lceil \frac{\sum_i x_{i,t}}{CAP} \rceil, \tag{17}$$

where CAP represents container capacity (in palette) and $\lceil \cdot \rceil$ is ceiling function. $C_{i,t}^{hold}$, $C_{i,t}^{pel}$, 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 as realistic a value 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 is fit 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 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_i (which is the time required from order to delivery) be three weeks, 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

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 future demands after order replenishment. At every time step, 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 $o_t = [(I_{i,t}, OO_{i,t}, \hat{I}_{i,t,t+LT}, f_{i,t}^{t:LT}, f_{i,t}^{t+LT:M})]_{i=1}^{N}$ and makes a decision based on o_t. Here, M is the parameter which decides how far the future demand needs to be considered and we let M be four weeks. The definition of MDP states that the next state and the next reward should be decided only 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 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 does not change from time to time and the agent knows its degree of forecast error as prior knowledge. 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. Also assuming that expected forecast error follows normal distribution, we can infer state s_t from observation o_t by using E_i (see Section 4 for further explanation).

Here, we have several approaches for this problem setting. Since we can obtain true state information, experience

memory can consist of the true state so that the Markovian property holds. Otherwise, we can use observation while ignoring partial observability. For action selection, we have to take action based on our observation, and we have two options to: 1) use demand forecast itself, or 2) estimate the true state by using our prior knowledge about demand forecast accuracy. 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. When selecting action greedily in validation with option 2, we used the following for the greedy policy:

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

where $\hat{\boldsymbol{s}}_t$ are estimated states that use the Monte Carlo sampling. In our experiments, we generated 300 samples at each step. Contrary to our initial expectations, the combination of option 2 in experience memory and option 2 in action selection performed the best. 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, infinite action space is not practical and also taking a large number of orders compared with the demand is unrealistic from a supply chain point of view. Therefore, we limited action space X_i to $X_i = \{l_i a \mid a \in \{0, 1, 2, 3\}\}$.

3.3. Branching deep Q-network with reward allocation

In (Tavakoli et al., 2017), they examined the action branching agent for environment in which only a global reward was available. In our case, each branch consisted of one product. Our objective variable was total cost; the transportation cost was calculated across multiple products whereas the holding cost and penalty cost were calculated independently of each product, which means that the total cost included both global and local rewards.

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

3.3.1. Reward allocation

Unlike (Tavakoli et al., 2017), we modified our temporal difference target equation as follows:

$$y_d = r_d + \gamma Q_d^- \left(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d\left(s', a'_d\right) \right), \tag{19}$$

where r_d refers to the reward per product. The transportation costs depend on the number of containers; the remaining costs can be calculated separately for each product. There are several options for allocating the total transportation cost to each product. The best results come from the following allocation by which the total transportation cost is allocated equally to all the products even if a specific product is not ordered:

$$r_d = -(C_{i,t}^{hold} + C_{i,t}^{pel} + \frac{C_t^{trans}}{N}). \tag{20}$$

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 branches backpropagation gradients are rescaled by $1/N$ for the shared part of our architecture.

$$L_d = \mathbb{E}_{(s,a_d,r_d,s')\sim\mathcal{D}}\left[L_\delta\left(y_d, Q_d\left(s, a_d\right)\right)\right], \tag{21}$$

where L_δ is the Huber loss function.

3.3.2. n-step TD method

n-step TD method was devised to take 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 the n-step TD method, the target for the n-step backup is as follows:
$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_t\left(S_{t+n}\right), \quad \forall n \geq 1.$

In our problem setting, there is obviously a reward delay because of the lead time from the supplier to the retailer. Thus, we let step size n be lead time in our learning setting.

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 be simply defined as follows:

$$Q_d\left(s, a_d\right) = V_d(s) + A_d(s, a_d), \tag{22}$$

where $V_d(s)$ denotes the branch independent state-value and $A_d\left(s, a_d\right)$ denotes the corresponding advantage.

4. Experiments

4.1. Experimental settings

We conducted numerical experiments to examine the following questions:
1) Can the proposed agent learn the coordinated order policy across multiple products?

Table 1. Training and validation approach.

Training	Select Action	$a_t \leftarrow \arg\max_a Q(o_t, a_t)$
	Memorize Experience	1: $\mathcal{D} \leftarrow [s_t, a_t, r, s_{t+1}]$
		2: $\mathcal{D} \leftarrow [o_t, a_t, r, o_{t+1}]$
Validation	Select Action	1: $a_t \leftarrow \arg\max_a [Q(o_t, a)]$
		2: $a_t \leftarrow \arg\max_a [\sum_{s_t} b(s_t) Q(s_t, a)]$

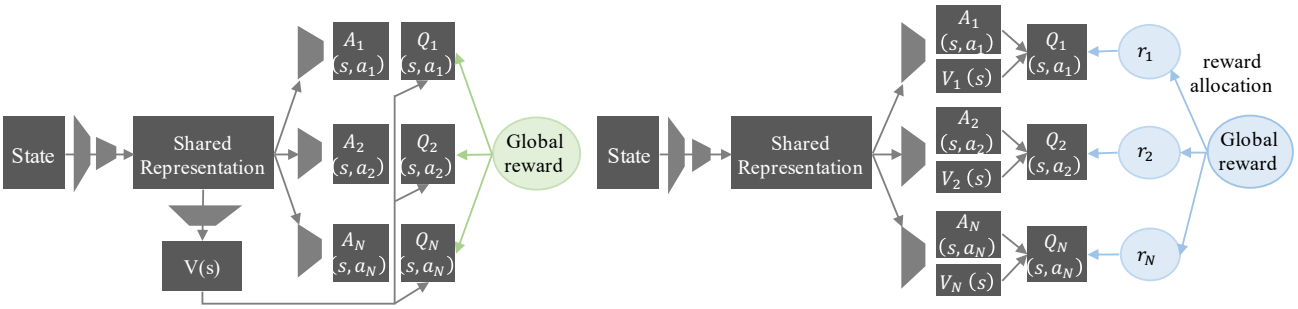

Figure 1. Left: BDQN. Right : BDQN-RA

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 with the benchmark policy?

For validation, we used the standard Dueling Double DQN, which we simply call DQN in this paper, and a forecast-based economic order policy (F-EOP) as the benchmark approach. DQN was used to validate the learning convergence of this proposed approach, whereas F-EOP was used to validate the optimality of the result obtained by using this proposed approach. Each episode consisted of 200 time steps, and initial 20 steps were ignored from the evaluation so as to exclude the effect of initial inventory setting.

Regarding the branching architecture, we tried three types of agents: BDQNs (BDQN with state-value estimator), BDQN-RA (BDQN with reward allocation), and BDQN-RAs (BDQN with reward allocation and state-value estimator).

In order to validate above-mentioned questions, we conducted three experiments by varying the number of products and the demand stationarity. Detailed explanations on experiments are given in Section 4-D.

4.2. Benchmark methodology: Forecast-based economic order policy

There has been no established JRP under non-stationary demand and demand forecast; therefore, we selected a non-coordinated order policy based on (Ishigaki & Hirakawa, 2008) 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 demand forecasts.

4.2.1. Forecast-based order-point

When demand forecasts are available, a replenishment order takes place when the forecasted inventory position at time $t + LT$ drops to order-point or lower. Assuming that forecast error follows normal distribution, 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. k can be determined so that sum of the expected penalty cost and expected holding cost for the safety stock are minimized.

4.2.2. Economic replenishment quantity with demand forecasts

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:

$$C(x) = \frac{U^{trans} \times \lceil \frac{x}{CAP} \rceil + U^{hold} \times \sum_{\hat{t}=t+LT}^{t+LT+T} \hat{I}_{i,t,\hat{t}}}{T}, \quad (23)$$

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 $argmin_x(C(x))$.

Table 2. Summary of experiment settings and results

ID	# of Products	Demand Stationarity	F-EOP	DQN	BDQNs	BDQN-RA	BDQN-RAs	Improved(%)
1	2	Stationary	133.2	105.3	108.0	(*) 106.7	107.3	19.9%
2	2	Upward Trend	216.5	205.2	266.3	(*) 192.0	195.2	11.3%
3	10	Upward Trend	447.8	-	817.2	346.8	(*) 341.3	23.8%

The final results for DQN and BDQN-family were derived by calculating the averaged total cost with a greedy policy using the trained model after 10,000 episodes over 6 runs. Improved(%) represents the decrease in total cost compared with F-EOP. (*) denotes the item used for calculating Improved(%).

Figure 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 average over 6 runs with initialization seeds and shaded areas represent standard deviation. Evaluations using the greedy policy were conducted every 50 episodes.

4.3. Experiment result

Table.2 and Fig.2 summarize the experiments' results and the learning curve. Our proposed agent performed better than did the non-coordinated F-EOP policy. As the number of products increased, DQN and BDQN (without reward allocation) suffered from convergence whereas our proposed agent, BDQN-RA(s), performed well even with 10 products. DQN suffered from its combinatorial increasing action space for a large number of products. When the number of products equaled 10, its action space was around 10^6. As for BDQN, using a single global reward made the learning convergence unstable because the feedback signal to each branch was considered to be too noisy. For the proposed agent, the reward allocation strategy worked well in stable learning while achieving coordinated orders. The state-value estimator in the proposed agent did not show a significant impact on the results.

4.3.1. Experiment 1: Two-products with stationary demand

One of the most important validation items regarding the action branching agent in JRP is 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, 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 such that the transportation cost was minimized. The left part of Fig. 3 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. Throughout 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, 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% in comparison 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. DQN agent performed best and it can be considered near-optimal JRP because the action space is small. In comparison with DQN, our proposed agent showed a slightly lower performance but still showed a significant improvement over F-EOP. Despite this simple experimental setting, these results show the possibility to learn a coordinated ordering policy across multiple products with our branching architecture.

4.3.2. Experiment 2: Two-products with upward demand

With upward demands, we expected that the ordering frequency should change from time to time as the total average demand per unit time increased. The bottom-right of Fig. 3 shows the result of BDQN-RA and we see that the order timing is aligned among these two products in most of

the time, and its frequency becomes shorter as the average demand increases. On the other hand, learning of BDQN agent was ill-behaved, which can be considered to be the result of the agent's having been affected by 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 observed; it is considered to have suffered from a non-stationary environment caused by branch-independent action selection, as seen in the general multi-agent reinforcement learning setting.

4.3.3. Experiment 3: Ten-products with upward demand

We extended our experiments to a more complicated setting with 10 products, and an average unit time demand was still much less than the container capacity. BDQN failed to converge, whereas our proposed agent exhibited an efficient and stable learning process. BDQN-RAs agent achieved a 23.8% cost reduction as compared with F-EOP.

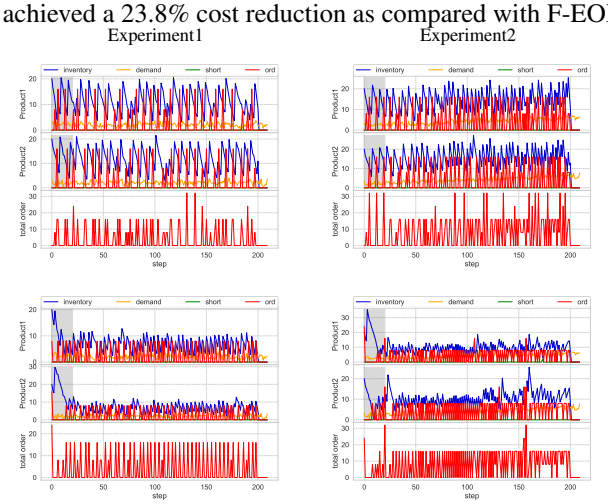

Figure 3. Time series movement of each product derived from the validation result of benchmark (top) and BDQN-RA (bottom) for Experiment 1(left) and 2(right). Each figure represents results of product 1 and 2, as well as total order quantity of both products (top to bottom). Orange, blue, green, and red lines represent demand, inventory, unsatisfied demand, and order quantity respectively.

4.4. Experiment detail

4.4.1. Cost, demand and demand forecast setting

On the basis of the logistic condition described in Section 3, we let 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 provided in Table.3. 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 demand at the end of the 200 time steps tripled, defined by; $d_{i,t} = \hat{d}_{i,t} + 2 * \mu_i * (t/200)$ where $\hat{d}_{i,t}$ denotes stationary demand. 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.

Table 3. Demand setting 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]

4.4.2. BDQNs and BDQN-RA(s)

The network had two hidden layers with 512 and 256 units 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 10 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 ϵ-greedy policy with linear annealing.

4.4.3. 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 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 as the number of products increased, both DQN and BDQN suffered from convergence; however, our proposed agent performed better when compared with F-EOP. Instability in the latter part of the learning process caused by the branch-independent action selection using our proposed agent should be investigated in future studies.

Our proposed agent needed only the demand forecast, which is usual in the real business setting; this expands the possibility to adapt our approach in real-world situations. In future studies, we also need to investigate how to extend this by including multi-layer and multi-retailer supply chains with realistic constraints.

References

Balintfy, J. L. On a basic class of multi-item inventory problems. *Management Science*, 10(2):287–297, 1964. ISSN 00251909, 15265501. URL http://www.jstor.org/stable/2627299.

Chaharsooghi, K. S., Heydari, J., and Zegordi, H. S. A reinforcement learning model for supply chain ordering management: An application to the beer game. 45(4): 949–959, 2008.

dos Bastos, L., Mendes, M., de Nunes, D., Melo, A., Carneiro, M., do de Janeiro, B., Vargas, B., and do do Pará, B. A systematic literature review on the joint replenishment problem solutions: 2006-2015. *Prod*, 27 (0), 2017. ISSN 0103-6513. doi: 10.1590/0103-6513.222916.

Ishigaki, A. and Hirakawa, Y. Design of a economic order-point system based on forecasted inventory positions. *Journal of Japan Industrial Management Association*, 59(4):290–295, 2008.

Johnson, E. L. Optimality and computation of (s,s) policies in the multi-item infinite horizon inventory problem. 13 (7):475–491, 1967.

Khouja, M. and Goyal, S. A review of the joint replenishment problem literature: 1989–2005. *European Journal of Operational Research*, 186(1):1–16, 2008.

Larsen, C. The Q(s,S) control policy for the joint replenishment problem extended to the case of correlation among item-demands. *Int J Prod Econ*, 118(1):292–297, 2009. ISSN 0925-5273. doi: 10.1016/j.ijpe.2008.08.025.

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

Minner, S. and Silver, E. A. Multi-product batch replenishment strategies under stochastic demand and a joint capacity constraint. 37(5):469–479, 2005. ISSN 0740-817X. doi: 10.1080/07408170590918254.

Minner, S. and Silver, E. A. Replenishment policies for multiple products with compound-Poisson demand that share a common warehouse. 108(1-2):388–398, 2007. ISSN 0925-5273. doi: 10.1016/j.ijpe.2006.12.028.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., and Nature, V. J. Human-level control through deep reinforcement learning. *Nature*, 2015. doi: 10.1038/nature14236.

Oroojlooyjadid, A., Nazari, M., Snyder, L., and Takáč, M. A deep Q-Network for the beer game: A reinforcement learning algorithm to solve inventory optimization problems. 2017.

Tavakoli, A., Pardo, F., and Kormushev, P. Action branching architectures for deep reinforcement learning. *CoRR*, abs/1711.08946, 2017. URL http://arxiv.org/abs/1711.08946.

Wagner, H. M. and Whitin, T. M. Dynamic version of the economic lot size model. *Management science*, 5(1):89–96, 1958.