Deep Multi-Agent Reinforcement Learning using DNN-Weight Evolution to Optimize Supply Chain Performance

Taiki Fuji†, Kiyoto Ito†, Kohsei Matsumoto†, and Kazuo Yano‡
† Center for Exploratory Research, Research & Development Group, Hitachi, Ltd.
‡ Research & Development Group, Hitachi, Ltd.
Email: {taiki.fuji.mn, kiyoto.ito.kp, kohsei.matsumoto.aa, kazuo.yano.bb} @hitachi.com

Abstract

To develop a supply chain management (SCM) system that performs optimally for both each entity in the chain and the entire chain, a multi-agent reinforcement learning (MARL) technique has been developed. To solve two problems of the MARL for SCM (building a Markov decision processes for a supply chain and avoiding learning stagnation in a way similar to the "prisoner's dilemma"), a learning management method with deep-neural-network (DNN)-weight evolution (LM-DWE) has been developed. By using a beer distribution game (BDG) as an example of a supply chain, experiments with a four-agent system were performed. Consequently, the LM-DWE successfully solved the above two problems and achieved 80.0% *lower total cost than expert players of the BDG.*

1. Introduction

As globalization has progressed, businesses have needed to develop efficient supply chains (SCs). An SC can be defined as a network of autonomous business entities collectively responsible for activities such as procurement, manufacturing, and distribution [1]. Although different entities in an SC operate subject to different sets of environmental constraints and objectives, they are highly interdependent when it comes to improving the total SC performances relating to objectives such as on-time delivery and cost minimization. Therefore, optimizing the performance of a part of the chain does not necessarily contribute to optimizing the entire SC performance, and problems regarding supply chain management (SCM) become more and more difficult as the scale and structure of the SC become larger and more complicated. As a result, information technology (IT) systems that support decision making about various entity activities by gathering information about SC have become essential.

To develop efficient IT systems for SCM, multi-agent-system (MAS) architecture has conventionally attracted attention [2]. In the MAS for SCM, an autonomous system called an *agent* decides each entity's operations to optimize the performance. All agents also simultaneously cooperate to optimize performances of the entire SC. Since such a distributed architecture is suitable for the network structure of an SC, a MAS-based SCM system is desired to optimize complicated SC performances.

The one of the most important challenges to develop such MAS-based SCM systems is designing agent *policies*. A policy is a set of rules for each agent to decide how to control entity operations with respect to certain constraints, environments, and objectives and has conventionally been designed on the basis of human experiences or theories delivered from simple SC models. However, designing the policies by these conventional schemes has become difficult because entities have different constraints, environments, and objectives and interdependencies among entities become more and more complex as the SC scale becomes larger.

To address problems in designing agent policies, automatic policy designing by machine learning (ML) has drawn attention. Among various ML techniques, policy learning using reinforcement learning (RL) is considered as an especially promising approach because RL does not require prepared datasets about entity operation whereas other ML techniques require datasets covering all environments, constraints, operations, and results of the entity operation. More specifically, RL is a technique for an agent to learn a policy by correcting necessary data itself during trial-and-error on the content of operations [3], [4]. Thus, RL is considered as an optimal solution addressing challenges where a huge number of factors must be taken into account like

URI: http://hdl.handle.net/10125/50044
ISBN: 978-0-9981331-1-9

SCM.

The objective of this research is to develop a multi-agent RL (MARL) technique [5] that enables agents to learn policies that optimize SC performance. There are two challenges to achieve an MARL technique for SCM: the first is to build an MARL environment of an SC that is regarded as Markov decision processes (MDPs) for every agent, and the second is to avoid learning stagnation among agents in learning processes.

To apply RL to a certain problem, all processes concerning the problem must satisfy a Markov property (MP). That is, environmental change for a certain agent action must be determined by a combination of the previous state of the environment and the agent action. For each agent in an MAS, however, other agents are part of the environment. Since all agents act independently and mutually affect each other, an environmental change for an agent depends on not only the previous state for the agent but also the other agents actions. In short, it is impossible to assume the MP. To solve this problem, Nash-Q [6] and Team-Q [7], which maintain the MP by observing the states and actions of other agents, have been developed. In addition to these observations, a model-based method such as AWESOME [8], which uses prior knowledge about the task, is presented to maintain MP while suppressing the RL environmental change. However, these methods cannot be applied to the MAS-based SCM because it is difficult to observe the states and actions of other agents and to preliminarily model tasks in accordance with customers' demands, which change from moment to moment (an imperfect information problem). Therefore, in the MAS-based SCM, it is necessary to solve learning when information is imperfect.

The second is a problem similar to the prisoner's dilemma. In the learning process of MARL, other agents sometimes cannot learn better policies unless an agent (agent A) changes its policy. When the policy change temporarily lowers the performance of agent A, agent A does not change the policy in the learning process, so other agents cannot learn a better policy. This problem is called learning stagnation. To overcome learning stagnation, studies that learn the communication protocol between agents have been presented for sharing the information about the cooperation among agents [9], [10], [11], [12]. However, in the MAS-based SCM, these related studies cannot be applied because the agents do not communicate directly but mutually affect each other through the environment. Hence, the agents have to communicate not directly but indirectly.

In this paper, we present a leaning management with deep-neural-network (DNN)-weight evolution (LM-DWE) to achieve a MARL technique for SCM. The LM-DWE uses a learning management method to reduce the non-stationary property problem of the MARL environment and introduces an evolutionary computation (EC) to solve the learning stagnation problem of MARL. To determine whether learning stagnation is avoided, we apply the method to a beer distribution game (BDG), which is a simple example of SCM. Specifically, by evolving the DNN-weights, which approximate each agent's policy, the agents act cooperatively and the reward of the entire SC increases.

2. Problem settings of MAS-based SCM

In this section, we describe the problem settings of the MAS-based SCM. We set the following problem. Most SCM is usually performed in imperfect-information situations. For example, even companies that are cooperating do not completely share the same business goals or technical capabilities and often only have incomplete knowledge about consumer preferences. Moreover, the SCM situations are repeated. Therefore, in this paper, the BDG [13], which is a famous role-play simulation game of SCM, is taken as the MAS-based SCM problem.

2.1. Beer distribution game: BDG

The objective in the BDG is to minimize inventory costs of a supply chain system while preventing shortages of beer as requested by customers through the multi-stage supply chain. The BDG has three game theory characteristics [14].

- Cooperation and a little competition (non-zero-sum game)
- Imperfect information
- Repeated dilemmas

When entities cooperate, a low cost is accomplished for the whole system. However, when entities compete such as by placing excess orders and making excess deliveries, the cost of the whole system becomes high. In addition, the BDG is an imperfect information game in which strictly limited information can be observed with respect to other agents (entities) regarding states, actions (operation), and rewards. Additionally, the BDG is a repeated-inventory management game containing dilemmas in which other agents' policies need to be considered.

There are three BDG rules in this paper:

- A standard chain system consisting of four entities connected in series is utilized.
- Delays of the orders and deliveries between connected entities are introduced.

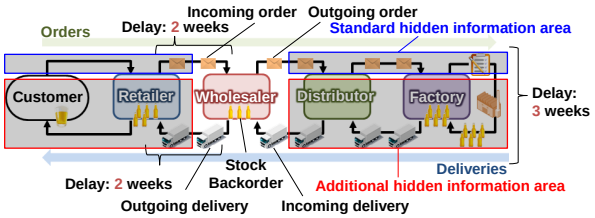


Figure 1. Beer distribution game (BDG). The standard BDG system consists of four entities: Retailer, Wholesaler, Distributor, and Factory. Order delays and delivery delays are set as two weeks, and production delay of Factory is set as three weeks. There are some hidden information areas including an entitys own constraints. This figure indicates Wholesaler's observation.

- Each agent can observe only its own entity's information.

Figure 1 shows the standard chain system: Retailer, Wholesaler, Distributor, and Factory. The standard BDG rules allow each entity's agent to observe the other entities' stocks and deliveries. In this paper, we impose a stricter rule: each agent can observe only a connected entity's information (stocks, backorders, incoming deliveries, outgoing deliveries, incoming orders, and outgoing orders). Moreover, in the BDG, the orders and deliveries between connected entities are always delayed. Each agent needs to decide the size of orders by considering the state values obtained at several past turns.

The each cost of a turn is calculated as

$$C_{i,t} = 0.5S_{i,t} + B_{i,t}, \tag{1}$$

where $C_{i,t}$, $S_{i,t}$, and $B_{i,t}$ stand for the cost, stock, and backorder of i-th entity at the turn t being equal to a week, respectively. Therefore, the total cost Z of the whole system at the end of an episode is calculated as

$$Z = \sum_{i=0}^{N-1} \sum_{t=0}^{W-1} C_{i,t}. \tag{2}$$

That is, each agent should learn the order policy for reducing Z.

2.2. Non-stationary environments

In the MARL, each agent learns a cooperative task by individually performing it in a trial-and-error manner and working in an RL environment, which is non-stationary. Therefore, the MP cannot be assumed.

For simplicity, game simulations are performed by simulating situations in which trial-and-error of RL is performed in the BDG where agents are connected to two of the four entities. In this case, the entities' agents are connected to Retailer and Wholesaler. Meanwhile, Factory and Distributor have a fixed policy that orders an incoming order quantity that is the same as the outgoing order quantity. Here, we design a simple policy function for each agent to maintain an arbitrary amount of stock shown in the following action function

$$a_{i,t} = x_1 RO_{i,t} + p_{i,t}(S_{i,t}, x_2) \tag{3}$$

$$p_{i,t}(S_{i,t}, x_2) = \begin{cases} (12 + x_2) - S_{i,t} & (\geq 0) \\ 0 & (otherwise) \end{cases}, \tag{4}$$

where x_1 and x_2 are parameters to be optimized and $RO_{i,t}$ stands for the received order of i-th entity at the turn t. By changing these parameters, the policy is updated. Figure 2 shows the order transition of two game simulations in which two agents perform trial-and-error. In this game simulation, trial-and-error of the Wholesaler agent leads to the same parameter change in both games. On the other hand, trial-and-error of the Retailer agent leads to different parameter changes in both games. As a result, we found that agents were repeating totally different orders. As shown in Table 1, the two games were almost completely different. Therefore, when multiple agents perform trial-and-error, the environment becomes non-stationary and MP cannot be assumed. This phenomenon occurs particularly conspicuously at the beginning of learning.

However, it is possible to learn by creating a situation where weak MP can be assumed. In the next simulation, we fix the Retailer agent's policy to prevent two agents from performing trail-and-error at the same time. As a result, the agent whose policy is fixed returns constant output for certain input like that from Distributor and Factory, so weak MP can be assumed. Figure 3 shows the simulation results when the parameters of the Retailer agent are x_1 and x_2, which are 0.5 and 0, respectively.

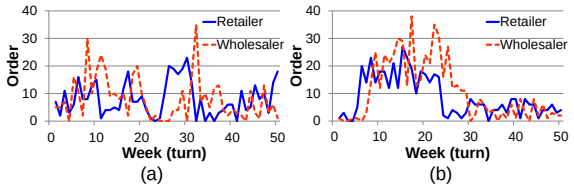


Figure 2. Order transitions in non-stationary environments. (a) Game 1 and (b) Game 2

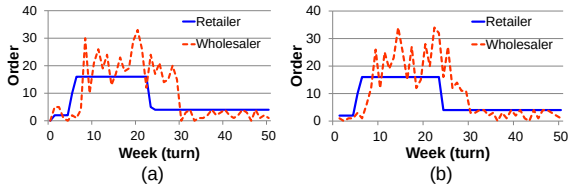


Figure 3. Order transitions in weak Markov-property environments. (a) Game 1 and (b) Game 2

Table 1. Degrees of similarity and cost differences between non-stationary and weak MP environments in two games.

	Non-stationary		Weak MP	
Entity type	R	W	R	W
DoS of orders	0.12	0.54	0.99	0.88
Cost difference	1279.5	2192.5	209	114.5

DoS: Degree of similarity, R: Retailer, W: Wholesaler

As shown in Figure 3 and Table 1, the two games had a high degree of similarity. Moreover, the difference in total costs is small. The results reveal that the environment became more stable than when two agents performed trial-and-error at the same time. On the basis of these results, we introduce an M-agent that switches on/off the learning of each entity agent and repeats the learning for selecting a well evaluated story. Here, a story is defined as a collection of multiple episodes.

2.3. SCM dilemma

For the second problem, we consider situations similar to the prisoner's dilemma in the BDG by using the Retailer and Wholesaler agents as shown in Table 2. For Wholesaler to minimize cost, it needs to handle order quantity so that it can deliver on-time while estimating the order policy of Retailer. However, perfect estimation is difficult, leading to Wholesaler either holding excess stock or causing backorders. Equation (1) indicates that the backorder cost is higher than the stock cost in the BDG cost calculation. The rational (selfish) Retailer agent cannot accommodate the demand increase from the customer, and as backorders occur, Retailer tries to decrease the backorders quickly, so demand can increase drastically with the number of orders. However, Wholesaler cannot respond to this drastic demand increase, so backorders occur and cost increases. This is a betrayal of Retailer.

On the other hand, a cooperative Retailer agent does not drastically increase order quantity, even if a backorder occurs. In other words, Retailer does not increase the cost for Wholesaler unnecessarily (Retailer increases order quantity gradually, which does not cause excess stock or shortages). However, as a penalty for this operation, the cost for the Retailer agent increases. In fact, such dilemmas occur between all connected agents.

Table 2. SCM dilemma.

	W: Coordination		W: Betrayal	
R: Coordination	R:○	W:○	R:×	W:⊙
R: Betrayal	R:⊙	W:×	R:△	W:△

R: Retailer, W: Wholesaler
Cost ⊙: Very low, ○: Low, △: Moderate, ×: High

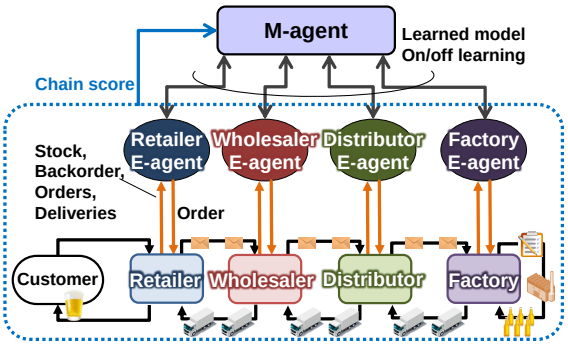


Figure 4. M-agent as a weak centralized control agent that manages learning of each E-agent. Specifically, each E-agent shares learned NN model and supply chain costs with M-agent.

To solve this problem, we introduce evolutionary computation techniques that force the entities' agents to change their policies.

3. Method: DNN-weight evolution for deep MARL

In this paper, we present a leaning management with DNN-weight evolution (LM-DWE) for a deep MARL. Our proposed method utilizes two techniques.

- Actor-critic deep RL is applied to the learning algorithm of each entity agent (E-agent) for handling continuous values in the complex system.
- The learning management agent (M-agent) with evolutionary computation (EC) is introduced to manage an E-agent's learning. Each E-agent shares abstract data such as the learned DNN model and supply chain costs with the M-agent.

The relationship between the M-agent and E-agents is as shown in Figure 4. Each E-agent connects to one entity but cannot communicate with other E-agents.

3.1. E-agent's learning algorithm: Actor-critic deep reinforcement learning

In this section, we describe the E-agent learning using deep RL. In the following, we use "action" as a substitute for "operation" in accordance with RL. In single-agent RL, the environment of the agent is described by a Markov decision process [5].

Definition: A Markov decision process is a tuple $\langle \boldsymbol{S}, \boldsymbol{A}, f, \rho \rangle$ where $\boldsymbol{S}$ is the finite set of environment states, $\boldsymbol{A}$ is the the finite set of E-agent actions, $f : \boldsymbol{S} \times \boldsymbol{A} \to \boldsymbol{S}$ is the state transition function, and $\rho : \boldsymbol{S} \times \boldsymbol{A} \to \mathbb{R}$ is the reward function.

The state vector $\boldsymbol{s}_k \in \boldsymbol{S}$ is observations from the environment at each discrete turn k. The E-agent can

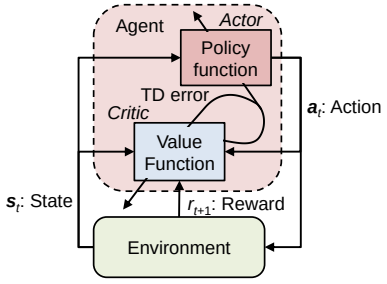


Figure 5. Actor critic method.

alter the state at each turn by taking action $\boldsymbol{a}_k \in \boldsymbol{A}$. After the action vector $\boldsymbol{a}_k$, the environment change its state from $\boldsymbol{s}_k$ to $\boldsymbol{s}_{k+1} \in \boldsymbol{S}$ according to the state transition probabilities. The E-agent receives a scalar reward $r_{k+1} \in \mathbb{R}$, according to the reward function $\rho : r_{k+1} = \rho(\boldsymbol{s}_k, \boldsymbol{a}_k)$.

The E-agent's goal is to maximize, at each turn k, the expected discounted reward of an episode

$$R_k = E\left\{\sum_{t=0}^{\infty} \gamma^t r_{k+t+1}\right\} \tag{5}$$

where γ represent a discount factor at the present state, respectively. The value R_k represents the reward accumulated by the E-agent in the long run. In this paper, an actor-critic deep RL [15] is applied to the learning algorithm of each E-agent. In this method, a probabilistic policy function $\boldsymbol{\mu}(\boldsymbol{a}|\boldsymbol{s})$, where $\boldsymbol{a}$ and $\boldsymbol{s}$ are action vector and state vector, respectively, with the expected reward as the evaluation value is defined, and the policy is updated along the direction of the policy gradient.

The actor-critic, which has an *actor* model as the policy function and a *critic* model as the value function, is utilized for handling continuous values. The basic concept of the method is to regard the policy function as independent from the value function as shown in Figure 5. The actor model generates an action in a certain state, the critic model calculates the temporal-difference (TD) error, and then these models are updated on the basis of the TD error.

Similar to the deep-Q-network (DQN) [3], experience replay is used. It is said that initial value dependency and the need for a dropout technique [16] can be alleviated. Furthermore, to calculate the TD error, a target model is set and is updated after a certain period of time. Target $\boldsymbol{y}_t$ is used together to define a loss error function, and this error is improved. The target $\boldsymbol{y_t}$ is expressed by the following formula:

$$\boldsymbol{y}_t = r_{t+1} + \gamma \boldsymbol{Q}'\left(\boldsymbol{s}_{t+1}, \boldsymbol{\mu}'\left(\boldsymbol{s}_{t+1} \mid \boldsymbol{\theta}^{\boldsymbol{\mu}'}\right) \mid \boldsymbol{\theta}^{\boldsymbol{Q}'}\right), \tag{6}$$

where $\boldsymbol{\theta}^{\boldsymbol{\mu}'}$ and $\boldsymbol{\theta}^{\boldsymbol{Q}'}$ are weights of the actor and critic models, respectively, which take an action when the best evaluation value is obtained in the present state. The TD error is calculated as $\boldsymbol{y}_t - \boldsymbol{Q}\left(\boldsymbol{s}_t, \boldsymbol{a}_t \mid \boldsymbol{\theta}^{\boldsymbol{Q}'}\right)$.Then, the actor and critic models are updated with the gradient calculated by the loss function:

$$\boldsymbol{\mathcal{L}} = \frac{1}{N}\sum_{\boldsymbol{y}}\left(\boldsymbol{y}_t - \boldsymbol{Q}\left(\boldsymbol{s}_t, \boldsymbol{a}_t \mid \boldsymbol{\theta}^{\boldsymbol{Q}'}\right)\right)^2. \tag{7}$$

The probabilistic policy gradient is given as

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}^{\boldsymbol{\mu}}} \boldsymbol{\mathcal{J}} \approx \frac{1}{N} \boldsymbol{E}_{\boldsymbol{\mu}'}\left[\boldsymbol{\nabla}_{\boldsymbol{a}} \boldsymbol{Q}\left(\boldsymbol{s}, \boldsymbol{a} \mid \boldsymbol{\theta}^{\boldsymbol{Q}}\right)|_{\boldsymbol{s}=\boldsymbol{s}_t, \boldsymbol{a}=\boldsymbol{\mu}(\boldsymbol{s}_t)} \boldsymbol{\nabla}_{\boldsymbol{\theta}^{\boldsymbol{\mu}}} \boldsymbol{\mu}(\boldsymbol{s} \mid \boldsymbol{\theta}^{\boldsymbol{\mu}})\right]. \tag{8}$$

The DNN-weights are updated with $\tau(\ll 1)$ as follows,

$$\boldsymbol{\theta}^{\boldsymbol{\mu}'} \leftarrow \tau \boldsymbol{\theta}^{\boldsymbol{\mu}} + (1-\tau)\boldsymbol{\theta}^{\boldsymbol{\mu}'} \tag{9}$$

$$\boldsymbol{\theta}^{\boldsymbol{Q}'} \leftarrow \tau \boldsymbol{\theta}^{\boldsymbol{Q}} + (1-\tau)\boldsymbol{\theta}^{\boldsymbol{Q}'}. \tag{10}$$

3.2. M-agent's algorithm: Learning management with DNN-weight evolution

The M-agent is introduced as a weak centralized control for multi-agent learning, as shown in Figure 6. The M-agent addresses the two MARL problems mentioned in Section 2. The M-agent copes with the problem that the environment is destabilized by trial-and-error of multiple agents using multi-point-search learning management. For the SCM dilemma, we apply the weight generation method of DNN using EC, which extends the multi-point search method.

3.2.1. Multi-point-search learning management The M-agent can switch on / off weight updates of each E-agent's DNN-weights and decides which E-agent to learn. In addition, as shown in Figure 6, we define a fixed number of learning episodes in the BDG system as a story and repeat this M times to make E-agents learn M stories. The initial DNN-weights of M trials repeatedly use the predetermined values.

During the learning of each story, the DNN-weights are saved every time the total cost is updated as a lower cost. After completing a story trial, E-agents send the value of total cost when the lowest cost is achieved and the weight of the DNN-weights to the M-agent in the story. Then, the M-agent selects the DNN-weights in accordance with criteria based on the total cost received from all E-agents. The selected DNN-weights are sent to each E-agent and set as the initial DNN-weights. Then, relearning is performed.

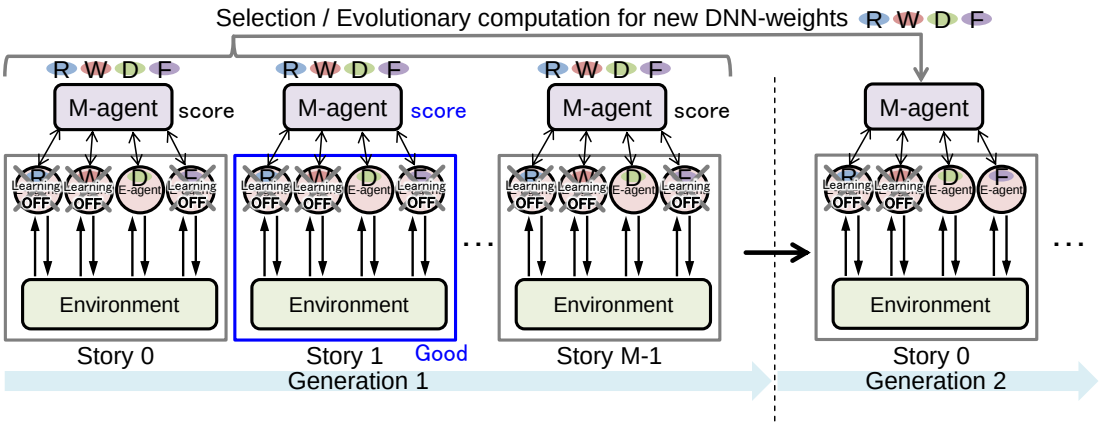


Figure 6. Learning management by M-agent. M-agent can switch on / off weight updates of each E-agent's DNN-weights and decides which E-agent to learn. M-agent selects the DNN-weights in accordance with criteria based on the total cost received from all E-agents. Selected DNN-weights are sent to each E-agent and set as initial DNN-weights.

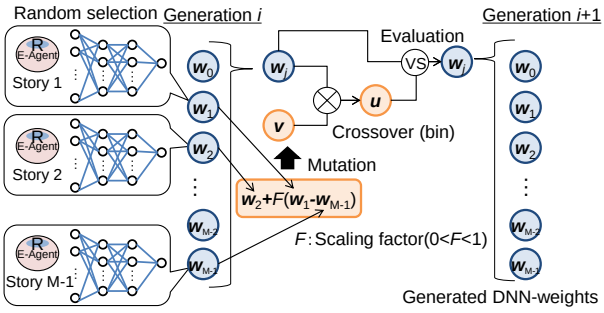


Figure 7. DNN-weight evolution using evolutionary computation (EC). Differential evolution is utilized as EC. Crossover and mutation are applied to weights of learned models

3.2.2. DNN-weight evolution Next, we describe the selection method of DNN-weights to be used for the next generation on the basis of the total cost received from all E-agents. In this paper, we propose using two methods. The first is to simply set the DNN-weights of a previous generation, which had the lowest cost, as the initial values. It is a so-called elite strategy and has relatively good learning efficiency. However, the dilemma situations are difficult to break down because only an elite individual's range is searched, so the evaluation value cannot be expected to be further improved.

Therefore, we propose applying EC to generate the new DNN-weights as shown in Figure 7. This method has the effect of creating a new policy on the basis of diversity when the learning stagnation occurs and forcibly changing it to a policy that makes the total cost value lower. In this paper, we apply a differential evolution (DE) algorithm [17], which has a high search performance even though it is simple, to the DNN-weight evolution. In this method, a story corresponds to an individual of the DE.

Algorithm 1 DE for DNN-weight evolution

for story $j = 0$ to $M-1$ **do**
  Generate random numbers a, b, $c \in [0, M-1]$
  for E-agent $i = 0$ to $N-1$ **do**
    Select three weights $\boldsymbol{w}_{a,i}$, $\boldsymbol{w}_{b,i}$, $\boldsymbol{w}_{c,i}$ as parents
    Calculate mutator $\boldsymbol{v}_i = \boldsymbol{w}_{a,i} + F(\boldsymbol{w}_{b,i} - \boldsymbol{w}_{c,i})$
    Compute crossover $\boldsymbol{u}_i = [u_{i,0}, ..., u_{i,P-1}]$ as follows:
    for parameter $k = 0$ to $P-1$ **do**
      Generate uniform random $rnd \equiv U(0,1)$
      if $rnd < CR$ **then**
        Set $u_{i,k} = v_{i,k}$
      else
        Set $u_{i,k} = w_{j,i,k}$
      end if
    end for
  end for
  if $f(\boldsymbol{u}_0, ..., \boldsymbol{u}_{N-1}) < f(\boldsymbol{w}_{j,0}, ..., \boldsymbol{w}_{j,N-1})$ **then**
    Replace $\boldsymbol{w}_{j,i}$ with $\boldsymbol{u}$
  end if
end for

Algorithm 1 formally describes DE for DNN-weight evolution. First, three DNN-weights ($\boldsymbol{w}_{i,a}$, $\boldsymbol{w}_{i,b}$, and $\boldsymbol{w}_{i,c}$) are randomly selected as parents. Then, the mutator $\boldsymbol{v}$ is calculated on the basis of the following equation:

$$\boldsymbol{v} = \boldsymbol{w}_{i,a} + F(\boldsymbol{w}_{i,b} - \boldsymbol{w}_{i,c}), \tag{11}$$

where F is a scaling factor $(0 < F < 1)$. Next, a child weight $\boldsymbol{u}$ is generated by doing crossover between $\boldsymbol{w}_{i,j}$ and $\boldsymbol{v}$. Finally, $\boldsymbol{x}_i$ and $\boldsymbol{u}_j$ are evaluated using the fitness function f of the BDG test, and then better DNN-weights are selected.

Table 3. Parameter settings of BDG.

Parameters	Values
Unit type, Num. of Units	R, W, D, F: 4 units
Num. of turns	1 episode : 50 turns
Num. of product types	1
Max of order number	100
Initial Stock	12
Customer order type	Constant (0-3 turn: 4, 4-49 turn: 8)

R: Retailer, W: Wholesaler, D: Distributor, F: Factory

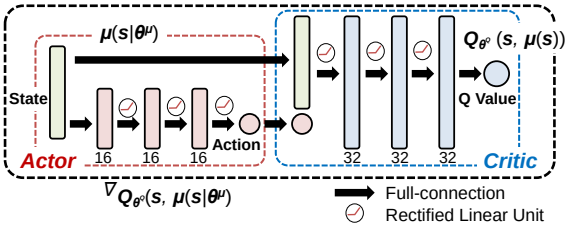


Figure 8. DNN Model design of all E-agents. DNN model consists of actor model and critic model. Both models have three layers. Output of actor model will be input to critic model with a state.

4. Experiments

In this section, we evaluate LM-DWE on the BDG. We connect E-agents individually to entities constituting a SCM system. There is no direct communication between E-agents. Instead, each E-agent is weakly controlled by the M-agent while communicating abstract data, which are the E-agent's total costs and DNN-weights.

4.1. Implementation

Table 3 shows parameter settings of the BDG that were set on the basis of the standard BDG rules. In addition to the rules, we imposed a severe limitation: each E-agent can observe only its own entity's information (stock, backorder, incoming orders, outgoing orders, incoming deliveries, and outgoing deliveries) in the past 10 turns as shown in Figure 1.

Figure 8 shows the DNN model architecture for the learning. The learning parameter settings are shown in Table 4. The actor model and critic model both had three middle layers. Each middle layer of the actor and critic models had 16 and 32 units, respectively. The Rectified Linear Unit (Relu) function was applied to the activation function. These parameter settings were determined empirically.

4.2. Preliminary experiments

First, we conducted a single-agent RL (SARL) experiment to determine whether convergence of SARL is possible. Next, we verified the effect of learning management using multiple stories in MARL and performed MARL experiments using two E-agents.

Table 4. Parameter settings of MARL.

Parameters	Values
Target model update rate τ	10e-3
Batch normalization size	32
Discount rate γ	0.99
Experience memory size	100k
Ornstein-Uhlenbeck process θ, μ, σ	0.15, 0., 0.3
Num. of actor middle layers	3 (16/16/16)
Num. of critic middle layers	3 (32/32/32)
Num. of sequential turn data (=State)	10
Num. of learning episodes (arbitrary)	0.1 k–2 k
Num. of learning stories (arbitrary)	1–40
Scaling factor F of DE	0.5
Crossover rate CR of DE	0.5
Reward for each E-agent	$-C_{i,t}$

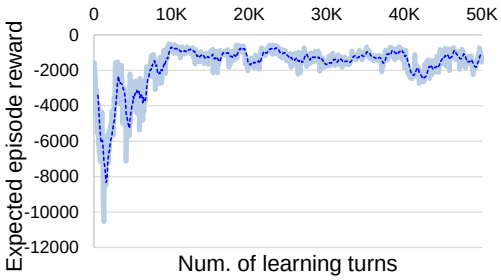


Figure 9. SARL result for Retailer E-agent.

4.2.1. Single-agent RL One entity was connected to a learning E-agent, and the other three entities' policies were set as "No policy" in which the quantities of outgoing orders and incoming orders are the same.

Figure 9 shows an example result of SARL for the Retailer E-agent. This result revealed that SARL is possible in the problem dealing with multi-dimensional state number and continuous values.

4.2.2. Learning management using multiple stories in MARL In this experiment, two learning E-agents connect to Wholesaler and Distributor individually. The initial DNN-weights of both E-agents are set as the DNN-weights obtained by each SARL. Moreover, the policies of Retailer and Factory are set to "No Policy" as a stable policy.

Figure 10 shows the learning results for two E-agents using five stories. The x-axis indicates the expected episode reward, and the y-axis indicates the number of learning turns: 50 trial-and-error turns $\times$ 1000 episodes. The results reveal that the expected reward value of story 4 was relatively low and reduced rapidly. Meanwhile, stories 2 and 4 maintained relatively high expected reward values.

In this experiment, by making two E-agents indi-

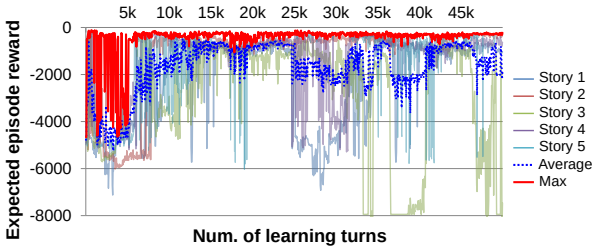


Figure 10. Results for MARL in which two agents connect to Wholesaler and Distributor, individually. Policies of Retailer and Factory are set as "No Policy." Blue dotted line is average value transition of all stories, and max line is maximum value transition of all stories.

vidually connected to the two entities learn and the other entities use "No Policy," the non-stationarity of the environment and learning processes were reduced. Moreover, even if the same initial values of weights were set to each E-agent's model for each story, MARL diversified. Thus, in the MARL environment, multiple stories should be utilized.

4.3. Evaluation of LM-DWE

To evaluate LM-DWE, we performed MARL experiments in which four E-agents connect to four SC entities. By comparing LM-DWE with the LM with an elite strategy (LM-ES), we verify the performance of our proposed method. The initial weights of all E-agents are also set as the weights obtained by each SARL. There are 300 episodes in a generation and 40 stories (individuals). In the LM-DWE method, 20 stories are set as elite stories and the remaining 20 stories are generated by DE processes including the 20 elite stories. Moreover, the M-agent switches on/off the learning of each E-agent and the DE processes every three generations. The learning E-agents are randomly selected under the constraint of reducing the number of learning E-agents at an early stage of learning. At the beginning of the learning, learning of two E-agents is performed to make the environment stable. Then, the learning E-agents are added one by one.

Figure 11 shows comparison results for cost transitions between LM-ES and LM-DWE. The results were average cost transitions of six experiments. In the experiments, the same changes were applied for the mode of each entity. The intermediate gray and light gray areas indicate two-agent learning and three-agent learning stages, respectively. From the results, both total costs were increased by adding the learning E-agents and were reduced by the learning. The total cost for LM-DWE was lower than that for LM-ES. Specifically, the total cost for LM-DWE gradually

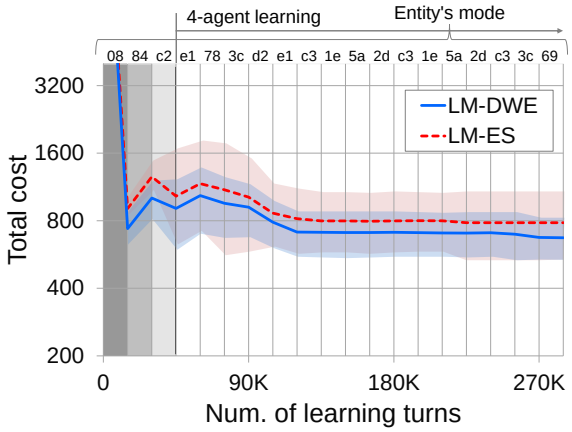


Figure 11. Comparison results for total-cost transitions between LM-ES and LM-DWE (average of six experiments). Each entity's mode which is "No policy" mode (no agent), "learning on" mode, or "learning off" mode is represented by bit flags with two hexadecimal numbers. Left and right hexadecimal numbers indicate fixed policy mode and learning mode positions respectively for Retailer(R), Wholesaler(W), Distributor(D), and Factory(F). (e.g. "08"→ R: learning on, W: No policy, D: No policy, F: No policy, "c3" → R: learning off, W: learning off, D: learning on, F: learning on)

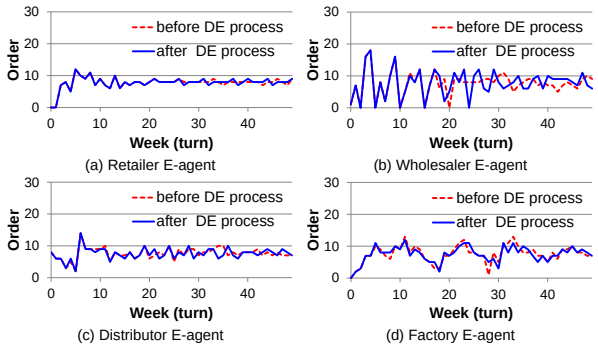
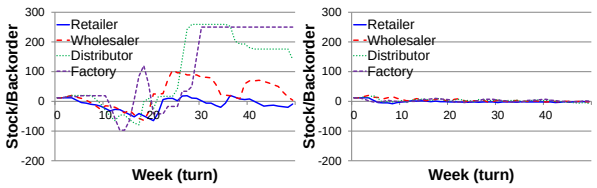


Figure 12. Example of order transition changes between before and after DE process in BDG test. DE processes were applied only to learning E-agents. Total cost for the 50th turn was reduced from 542.5 to 505. Average (Ave.) and standard deviation (Std.) of all E-agents' orders → Before: [Ave. 7.65, Std. 2.63], After: [Ave. 7.62, Std. 2.64]

reduced whereas that for LM-ES did not change from the learning turns near 120k turns.

Figure 12 shows the examples of order transition changes by the DE processes in the BDG test. The total cost for the 50th turn was reduced from 542.5 to 505. However, the averages and standard deviations of all E-agents's orders differed little.

Figure 13 shows the stock/backorder transition of the BDG test before and after LM-DWE-based MARL. Figure 13(a) shows results of the BDG test using the learned models obtained by SARL of each E-agent. Figure 13(b) shows results obtained by using



(a) Before LM-DWE based MARL (b) After LM-DWE based MARL

Figure 13. Comparison results for the stock/backorder transition before and after application of LM-DWE (best) in BDG test.

Table 5. Average and standard deviation of orders before and after LM-DWE based MARL.

	Before		After(best)	
Entity	Ave.	SD	Ave.	SD
Retailer	7.78	7.02	7.76	**1.99**
Wholesaler	8.28	13.89	7.72	**4.17**
Distributor	10.28	26.24	7.56	**1.88**
Factory	14.96	32.49	7.44	**2.54**

Ave.: Average, SD: Standard deviation

Table 6. Total cost for 35th week.

	Human[13]	Before LM-DWE	After LM-DWE
Total cost	2028	4823	406 (best)

the learned models that were finally acquired by the LW-DWE-based MARL. From the results, the stock management of LM-DWE-based MARL was greatly improved. Table 5 also indicates the improvement of order policy of each E-agent in response to the customer's demands. Finally, we compared the proposed LM-DWE's cost with human-level cost [13] as shown in Table 6. From the results, LM-DWE's cost was 80.0 % lower than human-level cost.

4.4. Discussions

The purpose of this paper was to solve two problems of MARL in the MAS-based SCM by using LM-DWE. Indeed, we demonstrated results for reducing non-stationary by the learning management and results for braking down the learning stagnation of MARL by using the DNN-weight evolution.

The results in Figure 11 show that the learning progresses with the elite strategy using multiple stories. However, the problem of learning stagnation occurred in the latter half of the learning. Thus, no matter how much it was learned, the total cost was not improved. Meanwhile, by changing the policies of the learning E-agents simultaneously to policies that were able to obtain better costs by using the DNN-weight evolution, it was possible to advance learning while reducing the occurrences of the dilemma situations. Moreover, the results in 12 show that slight changes of each entitys ordering policy have a large impact on total costs. When the cost of this experiment improve, the entity mode pattern was "c3", and Distributor and Factory were the learning entities. The details of cost changes for Retailer, Wholesaler, Distributor, and Factory were $+12.0$, -6.0, -22.5, and -21.0, respectively. That is, the cost for Retailer only increased. If Retailer was a learning entity, trial and error would be done to improve the entity's cost. In this situation, there is a high possibility that the cost improvement did not occur. Therefore, it can be said that LM-DWE contributed to the total cost improvement when the stagnation of MARL under imperfect information (dilemma situation).

Figure 13(a) shows an interesting phenomenon in the SCM. In the MAS-based SCM, if the agent's action at the head of the supply chain affects the adjacent agent, the problem is that the effect become larger the further towards the back of the supply chain the agent is. As a specific example, the Bullwhip Effect (BE) is a typical problem in SCM [18]. The BE amplifies the demand fluctuation on the downstream side towards the upstream side. In this SCM system, Retailer selling the beer to the customer was set as the down-stream side and Factory producing beers was set as the upstream side. As a result of each entity trying to respond to the down-stream side's requests individually, an enormous shortage of items and excessive beer delivery were caused on the upstream side, which was more affected by the demand fluctuation. Thus, stable supply could not be performed in the whole system. Meanwhile, as shown in Figure 13(b), LM-DWE based MARL succeeded in suppressing the BE by learning the coordination.

The learning management using multiple stories could be performed in parallel for speeding up MARL. In related work, A3C [19] succeeded in efficient learning by parallelized SARL. Similarly, the parallelized MARL could improve the learning speed.

We acknowledge there are several limitations in this study. First, in our experiments, the SC entities were connected in series. Thus, the effectiveness of our proposed method was not verified in branched systems, which are more realistic. However, the proposed LM-DWE may also work well in branched systems because it is applied the same way. Second, the customer order policy and the delays between entities were fixed. In the real system, these values usually change from moment to moment. Third, in the experiments, the number of turns was fixed as 50. Thus, if the number of sequential turn data as input data for the DNN was fixed, each E-agent was able to learn the order policy. However, if the number of turns increases, the size of input data should be increased. Therefore, we need to consider

combining our method with an algorithm that can deal with arbitrary-length data such as long short-term memory (LSTM) [20].

5. Conclusion

In this paper, we presented a LM-DWE that enables all E-agents controlled by a M-agent to learn a coordination task. The proposed LM-DWE used a learning management method for reducing the non-stationary property problem of the MARL environment, and an EC for the learning stagnation problem of MARL. We used a stricter BDG, in which each entity's observation was more restricted than under the standard BDG rules, as a problem of MAS-based SCM. By using the stricter BDG, experiments of four-agent learning were performed. The results revealed that LM-DWE could successfully learn the coordinated order policy in an imperfect information game. Moreover, LM-DWE's total cost was 80.0 % less than the human-level total cost at the 35th turn in the BDG test.

In future work, to make it closer to a more realistic problem, we will extend the standard BDG problem to more complicated and realistic SC problems, in which more entities, branches, and delays are added. Moreover, the sequential turn data used as input data were fixed as 10-turn data in the experiments although the length of meaningful time series fluctuates from time to time. Therefore, we will apply a recurrent NN using an LSTM [20] to LM-DWE to make it able to handle data of arbitrary lengths.

Acknowledgments

We would like to thank team members, Masahiro Kato, Kanako Esaki, and Ryusei Akita.

References

[1] J. M. Swaminathan, S. F. Smith, and N. M. Sadeh, "Modeling supply chain dynamics: A multiagent approach," *Decision sciences*, vol. 29, no. 3, pp. 607–632, 1998.

[2] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, N. Kalchbrenner, J. Nham, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews*, vol. 38, no. 2, pp. 156–172, 2008.

[6] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: theoretical framework and an algorithm," in *ICML*, vol. 98. Citeseer, 1998, pp. 242–250.

[7] M. L. Littman, "Value-function reinforcement learning in markov games," *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.

[8] V. Conitzer and T. Sandholm, "Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents," *Machine Learning*, vol. 67, no. 1-2, pp. 23–43, 2007.

[9] J. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 2137–2145.

[10] S. Sukhbaatar and R. Fergus, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems*, 2016, pp. 2244–2252.

[11] J. Foerster, N. Nardelli, G. Farquhar, P. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," *arXiv preprint arXiv:1702.08887*, 2017.

[12] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games," *arXiv preprint arXiv:1703.10069*, 2017.

[13] J. D. Sterman, "Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment," *Management science*, vol. 35, no. 3, pp. 321–339, 1989.

[14] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[18] H. L. Lee, V. Padmanabhan, and S. Whang, "The bullwhip effect in supply chains," *Sloan management review*, vol. 38, no. 3, pp. 93–102, 1997.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.