IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

State-dependent parameter tuning of the apparent tardiness cost dispatching rule using deep reinforcement learning

Byungwook Min, Chang Ouk Kim
Department of Industrial Engineering, Yonsei University, Seoul 03722, South Korea

Corresponding author: Chang Ouk Kim (kimco@yonsei.ac.kr)

This work was supported by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) and the Ministry of Trade, Industry & Energy (MOTIE) of the Republic of Korea (No. 20202020800290) and the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) under Grant NRF-2019R1A2B5B01070358.

ABSTRACT The apparent tardiness cost (ATC) is a dispatching rule that demonstrates excellent performance in minimizing the total weighted tardiness (TWT) in single-machine scheduling. The ATC rule's performance is dependent on the lookahead parameter of an equation that calculates the job priority index. Existing studies recommend a fixed value or a value derived through a handcrafted function as an estimate of the lookahead parameter. However, such parameter estimation inevitably entails information loss from using summarized job data and generates an inferior schedule. This study proposes a reinforcement learning-based ATC dispatching rule that estimates the lookahead parameter directly from raw job data (processing time, weight, and slack time). The scheduling agent learns the relationship between raw job data and the continuous lookahead parameter while interacting with the scheduling environment using a deep deterministic policy gradient (DDPG) algorithm. We trained the DDPG model to minimize the TWT through a simulation in a single-machine scheduling problem with unequal job arrival times. Based on a preliminary experiment, we verified that the proposed dispatching rule, ATC-DDPG, successfully performed intelligent state-dependent parameter tuning. ATC-DDPG also displayed the best performance in the main experiment, which compared the performance with five existing dispatching rules.

INDEX TERMS Deep deterministic policy gradient, dispatching rule, dynamic scheduling, parameter tuning, reinforcement learning.

I. INTRODUCTION

Reinforcement learning (RL) is a machine learning paradigm of a trial-and-error method that learns the actions of an agent such that the total reward acquired by the agent, the subject of learning, becomes optimal while interacting with the environment. In this study, we propose an RL-based dispatching rule combined with deep neural networks to solve the dynamic scheduling problem and analyze the experimental results to evaluate the applicability of the state-of-the-art RL algorithm in the scheduling field.

The dynamic scheduling problem is defined in a stochastic environment with random events [1]. Random events denote unavoidable constraints, such as unequal job arrival times, job preemption, and machine failures. As a simple example of dynamic scheduling, consider a single-machine problem where the job arrival times are unequal. Although only a single assumption was added, stating that the jobs arrive randomly, it results in an extremely difficult problem whose optimal solution is almost impossible to derive [2], [3].

The high complexity of dynamic scheduling has led to a wide variety of studies on solving problems using dispatching rules instead of optimization techniques. However, there is a critical issue in dispatching rules: because the performance of the rules varies by the job queue state, there is no single ideal dispatching rule that yields the best performance under all situations [4]. In other words, it cannot be guaranteed that the best dispatching rule at the current scheduling time will be the best for the next scheduling time. Accordingly, studies are actively being conducted on RL-based methods for selecting the best dispatching rule among multiple dispatching rules by extracting information from the state of the waiting jobs each time job processing is completed.

Q-learning is a representative classic RL model that observes the state (e.g., job data) of the environment (e.g., job queue in front of a machine), selects an action (e.g., the next job or dispatching rule), estimates the Q-value that corresponds to the expected total reward as a result of the selection, and finally records the result in a table. Aydin and Öztemel [5] proposed a Q-learning-based RL method (Q-III learning algorithm) that selects one dispatching rule among the shortest processing time (SPT) rule, cost over time, and critical ratio based on handcrafted features (i.e., the number of waiting jobs and their average slack time) of the shop in a dynamic job shop environment with unequal arrival times. Similarly, Wang and Usher [6] trained a single-machine agent in a job shop environment such that it would select one method from first-in first-out (FIFO), earliest due date (EDD), and SPT using Q-

learning. The authors used 10 handcrafted features based on the number of waiting jobs and estimation of the total lateness as the inputs of Q-learning. However, these studies have a limitation in that the scheduling effect is reduced because the raw states of the job queue are processed as discretized features and then used for Q-learning.

Chen et al. [7] proposed an RL-based composite dispatching rule for multiobjective dynamic scheduling. Their study considered the critical ratio, modified due date, and SPT for the component rules. Q-learning uses the discretized work-in-process ratio as input, learns the weight of each component, and finally composes the component rules into one rule based on the learned weights. Their study addressed a multiobjective problem that aimed to simultaneously minimize the mean flow time and mean tardiness to respond to changes in the work in process, and the experimental results revealed that the RL-based composite rule yielded better performance than the individual performances of the component rules.

Another approach to solving the dynamic scheduling problem is to estimate the optimal parameter value of the scheduling algorithm using RL. As a representative example, Shahrabi et al. [8] developed an improved algorithm by utilizing Q-learning to derive the optimal parameters of the variable neighborhood search (VNS). VNS, which is a heuristic method, generates schedules in a dynamic job shop problem with unequal job arrival times and machine failures. The inputs of Q-learning were set as the number of waiting jobs and their discretized average processing times, and the actions were defined as three discretized parameters required in VNS.

The abovementioned studies demonstrated that the performance can be improved if the existing dispatching rules are selected or synthesized based on the state of the shop via RL. In addition, methods using RL for parameter improvement in the scheduling algorithm have also demonstrated their effectiveness. However, existing studies employed discretized handcrafted features rather than using continuous raw data of waiting jobs for agent training; consequently, the agent may fail to generate a better schedule. Such a limitation is attributed to Q-learning. Classic Q-learning requires experiencing all actions in all states to derive the best actions; however, Q-learning is difficult to apply if the state-action space is continuous. Hence, we propose a dispatching rule based on a deep deterministic policy gradient (DDPG) algorithm embedded with deep convolutional neural networks. The DDPG can precisely estimate the Q-value even in a continuous search space. The proposed rule is based on apparent tardiness cost (ATC) dispatching and uses the DDPG algorithm to tune the lookahead parameter, which is the user parameter of ATC dispatching. Consequently, the lookahead parameter can be estimated precisely at each job dispatching time point by receiving the raw job data as input. Hereinafter, we abbreviate the proposed dispatching rule as ATC-DDPG.

ATC is a dispatching rule that combines the weighted shortest processing time (WSPT) and minimum slack (MS) rules; it is known to demonstrate the best performance in minimizing the total weighted tardiness (TWT) [9], [10]. Although the key to improving the performance of the ATC is the selection of an appropriate lookahead parameter, most existing studies used a fixed value between 1.5 and 4.5. Valente [11] proposed the ATC-vk1, 2, 3 rule to calculate the lookahead parameter using a handcrafted function for the first time. However, the corresponding study also used processed job features as inputs of the function instead of the original job data. Heger and Voss [12] also emphasized that the performance improvement of the ATC rule is in parameter selection and proposed an RL-based dispatching rule that selects two parameters, k_1 and k_2, of ATC with setups (ATCS), which is a variant of ATC. They reduced the mean tardiness by selecting two parameters according to product mix and system utilization, but there is still a limitation that learning proceeded in discretized state-action spaces. Table 1 summarizes studies related to this research.

The remainder of this paper is organized as follows. Chapter 2 introduces the ATC rule and the principles of RL. Chapter 3 describes the proposed ATC-DDPG algorithm in detail. Chapter 4 presents an analysis of the factors contributing to the outstanding performance exhibited by ATC-DDPG based on a preliminary experiment and the performance test of ATC-DDPG through the main experiment comparing five dispatching rules for three workload cases. Finally, Chapter 5 concludes this study and provides suggestions for possible future research topics.

TABLE 1. Summary of RL-based dynamic scheduling studies.

Author	Dynamic events	Input space	Input data type	Algorithm	Purpose of algorithm
Aydin and Öztemel [5]	Unequal arrival time	Discrete	Handcrafted	Q-III	Dispatching rule selection
Wang & Usher [6]	Unequal arrival time	Discrete	Handcrafted	Q-learning	Dispatching rule selection
Chen et al. [7]	Unequal arrival time	Discrete	Handcrafted	Q-learning	Dispatching rule aggregation
Shahrabi, Adibi, and Mahootchi [8]	Unequal arrival time, breakdown	Discrete	Handcrafted	Q-learning	Parameter tuning of VNS
Valente [11]	None	Continuous	Handcrafted	Handcrafted function	Parameter tuning of ATC
Heger and Voss [12]	Unequal arrival time, production mix change	Discrete	Handcrafted	Temporal difference	Parameter tuning of ATCS
Proposed Algorithm	Unequal arrival time	Continuous	Raw	DDPG	Parameter tuning of ATC

II. BACKGROUND

This study addresses the single-machine dispatching problem with unequal job arrival times. The purpose of scheduling is to minimize the TWT, and to achieve this goal, the state-dependent lookahead parameter of the ATC rule is determined using ATC-DDPG at each scheduling time. Before introducing the proposed ATC-DDPG, this chapter reviews the ATC rule and the principles of RL.

A. ATC RULE

The ATC rule defines the priority index for job i as follows:

$$I_i = \frac{w_i}{p_i} \exp\left(-\frac{\max(s_i,0)}{k\bar{p}}\right), \quad (1)$$

where w_i, p_i, and s_i denote the weight, processing time, and slack time of job i, respectively; $\bar{p}$ denotes the average processing time of waiting jobs; k represents the lookahead parameter. The larger k is, the more closely the ATC rule converges to the WSPT rule, whereas the smaller k is, the more the jobs in which s_i is positive are affected by the MS rule. Holsenback et al. [13] recommended a value of k between 0.5 and 2.0, whereas Vepsalainen and Morton [9] insisted that a value between 1.5 and 4.5 should be used if the slack time of jobs is extremely small.

ATC-vk2, a variant of ATC, determines the k value using the following simple rule instead of using a fixed value [11]. If p_{crit} is greater than 0.75, then k is defined as $k = k_H$; otherwise, it is defined as $k = 0.6$. Here, p_{crit} represents the ratio of critical jobs: if the slack time of a job is within the two-sided limit, the corresponding job is defined as a critical job; otherwise, it is defined as a noncritical job. The two-sided limits and k_H were determined through a simulation experiment. ATC-vk2 is used for a performance comparison in the main experiment.

B. ACTOR-CRITIC LEARNING

1) RL PRINCIPLE

In an RL system, an agent interacts with the environment. The agent observes the state s_t of the environment at timestep t and selects an action a_t according to policy μ in a deterministic or probabilistic manner. Consequently, immediate reward r_t is received, and the state is transitioned to s_{t+1}. Thus, the agent acquires an experience composed of $< s_t, a_t, r_t, s_{t+1} >$ at each timestep, and one episode is terminated when the interaction between the agent and environment reaches terminal timestep T. For the agent, policy μ serves as a learning target that determines the best action in each state. Although the agent uses a random policy during the initial stage of learning, it learns actions by optimizing the performance indicator while experiencing new episodes. The performance indicator is expressed as $Q_\mu(s,a) = \mathbb{E}_\mu[G_t|S_t = s, A_t = a]$, where $G_t = \sum_{i=t}^{T-1} \gamma^{(i-t)} r_i$ denotes the cumulative discounted reward obtained when one episode is completed after reaching the terminal timestep T by policy μ (γ denotes the discount rate). In addition, $Q_\mu(s,a)$ represents the average cumulative discounted rewards when the episode is iterated through an infinite loop, which can be expressed recursively using the Bellman equation as follows:

$$Q_\mu(s,a) = \mathbb{E}\left[r + \gamma Q_\mu(s',a')\right]|_{a'=\mu(s')}, \quad (2)$$

where s' represents the next state of state s.

If the episode is iterated according to policy μ, the Q-value $Q_\mu(s,a)$ can be accurately estimated for all pairs of $\langle s,a \rangle$ using (2); consequently, RL can derive the best action with the optimal Q-value for each state. However, because it is impossible to experience all pairs of $\langle s,a \rangle$ in a high-dimensional search space, the balance between exploitation to update the Q-value by revisiting a previously experienced $\langle s,a \rangle$ pair and exploration to experience a new $\langle s,a \rangle$ pair is important.

In terms of scheduling, a state represents the data (e.g., processing time, weight, and due date) of waiting jobs, and an action, i.e., an output of the policy, denotes the lookahead parameter value. Once this parameter is determined, the next job is selected by the ATC rule. The reward is determined based on the TWT after all jobs are completed rather than being assigned immediately when the next job is determined.

2) ACTOR-CRITIC METHOD

The actor-critic method is an RL algorithm composed of an actor and a critic. The actor includes a neural network $\mu_\theta(s)$ that learns the policy, and the critic involves a critic neural network $Q_w(s,a)$ that estimates the Q-value by receiving the state and action chosen by the actor as inputs. Here, w and θ denote the network weights. To estimate the best policy and best Q-value, the actor and critic update their weights based on experiences.

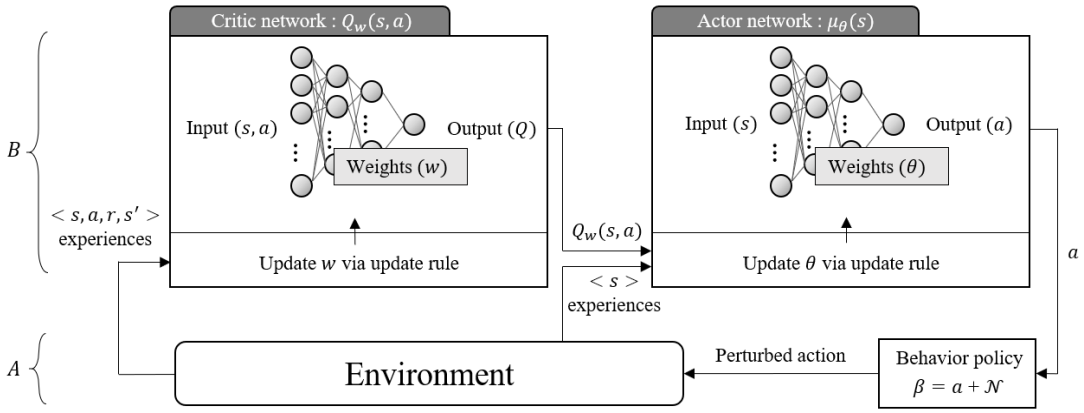

FIGURE 1. Learning paradigm of the actor-critic method.

As illustrated in Fig. 1, the learning process of the actor-critic method is composed of part A, where the actor executes action a determined by $\mu_\theta(s)$, and part B, where the network weights of the actor and critic are updated using the experiences $< s, a, r, s' >$ obtained from part A. This method conducts training by repeating parts A and B. In detail, the actor executes the behavior policy, $\beta = \mu_\theta(s) + \mathcal{N}$, containing action noise $\mathcal{N}$, on the environment; this policy reinforces exploration in the search space to execute a perturbed action and consequently induces it to visit a new state. In terms of scheduling, if the target policy $\mu_\theta(s)$, which outputs the lookahead parameter value, is reused in its original form, the opportunity to search for a better schedule will vanish. Accordingly, the action is occasionally perturbed by the amount of noise $\mathcal{N}$ during the learning process to induce an output value that differs from the lookahead parameter value determined by the target policy.

The critic is designed to update the weights w of $Q_w(s,a)$ using the experiences $< s, a, r, s' >$. The objective function is the minimization of the mean square error and is defined as follows:

$$\epsilon^2(w) = \frac{1}{2}\mathbb{E}\left[r + \gamma Q_w\left(s', \mu_\theta(s')\right) - Q_w(s,a)\right]^2, \qquad (3)$$

where $Q_w(s,a)$ denotes the previously trained Q-value. In addition, $r + \gamma Q_w(s', \mu_\theta(s'))$ is the Q-value reflecting experiences, and the weights w are updated such that $Q_w(s,a)$ becomes equivalent to the target $r + \gamma Q_w(s', \mu_\theta(s'))$. The equation for a weight update can be derived by calculating the gradient of $\epsilon^2(w)$ with respect to w [14]. The critic receives a new experience from the actor and updates the weight w of $Q_w(s,a)$; as this learning process continues, a value close to the true Q-value of (2) can be obtained.

The final objective of the actor is to obtain a policy that provides the best action that ensures the optimal Q-value in every state. Because the actor expresses the policy in the form of a neural network $\mu_\theta(s)$, the network weights θ should be trained to obtain the optimal policy. The optimization objective function is expressed as $\mathbb{E}_\mu[G_1] = \int_{s\in S} \rho^\mu(s) Q_\mu(s,a)\,\mathrm{ds}$, where $\rho^\mu(s)$ denotes the probability that an agent will exist in each state s when the behavior is executed infinitely from the starting state to the policy $\mu_\theta(s)$. Meanwhile, the actor uses the behavior policy β, which is $\mu_\theta(s)$ with added noise. Furthermore, because the Q-function $Q_\mu(s,a)$ is approximated by the critic network $Q_w(s,a)$, the objective function for updating $\mu_\theta(s)$ can be expressed as follows:

$$J(\theta) = \mathbb{E}_\mu[G_1] = \int_S \rho^\beta(s) Q_w(s,a)\,\mathrm{ds}\big|_{a=\mu_\theta(s)}. \qquad (4)$$

$J(\theta)$ is the average cumulative discounted reward and should therefore be maximized. The weight update equation can be defined by deriving the gradient of $J(\theta)$ with respect to θ [15].

In the actor-critic method, the weights of the actor and critic networks are updated each time an experience is acquired. If the actor repeatedly interacts with the environment, the actor learns an action that is more suitable for the state, whereas the critic learns a more accurate Q-value. In terms of scheduling, the role of the critic is to better estimate which TWT resulted from job selection according to the job state of the waiting line, whereas the role of the actor is to select the lookahead parameter value to derive a lower TWT (accordingly, the ATC rule determines the next job).

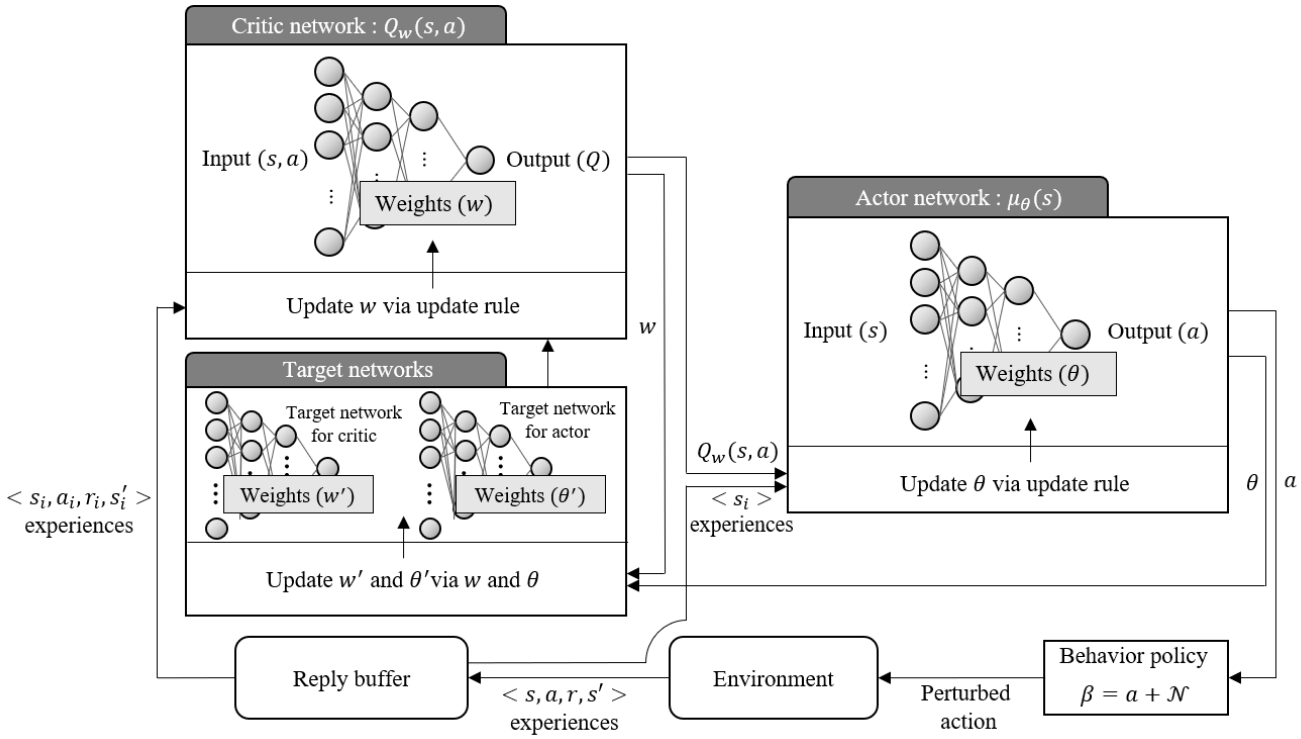

FIGURE 2. Learning paradigm of DDPG.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3152192, IEEE Access

3) DDPG

The actor-critic method suffers from the problems of nonstationary targets and correlation between experiences during the learning process. The former is a phenomenon in which not only the predicted value $Q_w(s,a)$ but also the target $r+\gamma Q_w(s',\mu_\theta(s'))$ become updated when the weight w is updated using (3). The latter is a phenomenon in which the agent becomes stuck in a local optimum in the search space owing to the continuous visits to similar states by the agent.

DDPG is an actor-critic-based RL algorithm; it implements the critic using a deep Q-network (DQN) [16] to solve nonstationary target problem. As illustrated in Fig. 2, a DQN is composed of a critic network and a target network. $Q_w(s,a)$ is trained using the critic network, whereas $Q_w\left(s',\mu_\theta(s')\right)$ is trained using the target networks. Specifically, the weights w' and θ' of the target networks are slowly updated incrementally compared to the weights of w and θ of the critic and actor networks to prevent $Q_w\left(s',\mu_\theta(s')\right)$ from moving together with $Q_w(s,a)$ whenever $Q_w(s,a)$ is changed [17]. Furthermore, the DQN uses a replay buffer to address the problem of correlation between experiences. The replay buffer serves as a place to accumulate experiences previously generated by the actor. The problem of high correlation between consecutive experiences is addressed by training the critic and actor based on the experiences randomly selected from the buffer at each weight update [18].

III. ATC-DDPG

The proposed ATC-DDPG method is an ATC rule embedded with DDPG. The pre-trained DDPG receives raw job data from the job queue and outputs the lookahead parameter k of the ATC, whereas the ATC rule applies the output k value to the rule and selects the job among the waiting jobs to minimize the TWT.

A. NETWORK COMPONENTS

Fig. 3 illustrates the DDPG network model used in ATC-DDPG. The inputs of the actor and critic networks are the processing time (p_i), weight (w_i), and slack time (s_i) of the waiting jobs $(i=1,\dots,q)$, and the size of the job queue is assumed to be q. DDPG is composed of an input layer that receives data, three convolutional layers (C1, C2, C3) that can extract low-dimensional features from the training data, two fully connected layers (FC1, 2) that apply predictions using the features, and an output layer (structurally, a layer that flattens C3 is added to connect C3 and FC1). The action, which is the output of the actor network, is fed into the middle of the critic network; consequently, the two networks become connected. The reason for this connection is because the critic network needs to estimate the Q-value for the state-action pair. The critic network outputs the Q-value, and the actor network outputs the lookahead parameter. The basic structures of the critic and actor networks are identical, except that the action, the output of the actor network, is entered into the critic network as an input.

In the convolutional layers, the kernel acts as a filter to extract features from the input data. It generates a feature map by extracting local features through a convolution operation with the input data while traveling along the input data by the stride width. In other words, one feature map is the result of a convolution operation obtained by one kernel by traversing all input data. In Fig. 3, F denotes the user-defined kernel number, which is equal to the number of generated feature maps because one feature map is generated for each kernel. The generated feature maps are provided as input data in the subsequent layer, and new feature maps are generated again via different kernels. ATC-DDPG employs the same number of kernels in all three convolutional layers.

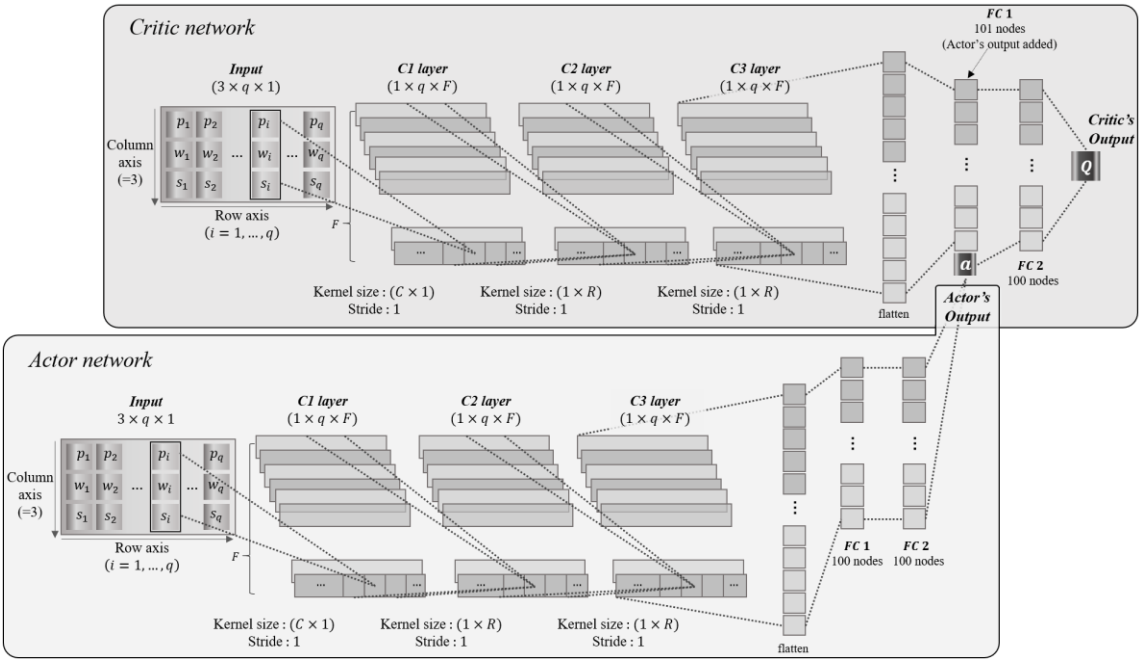

FIGURE 3. DDPG network model used in the ATC-DDPG dispatching rule.

In the case of a network that uses image data as input, a kernel size of $C \times C$ is commonly used in convolutional layer C1 to identify the correlation between pixels. However, in the case of a network that uses job data as input, a kernel size of $C \times 1$ is effective for improving the training performance because the basic information that must be identified in the job data is the correlations among the processing time, weight, and slack time of a single job rather than information between jobs. Thus, we configured the networks to extract information from single jobs using a kernel size of $C \times 1$ in C1 and to then identify the correlations with other jobs using a kernel size of $1 \times R$ in C2 and C3. The values of the network parameters (q, F, C, and R) were defined in the experiments.

B. RL COMPONENTS

The action in ATC-DDPG is the lookahead parameter k of the ATC rule; its range was limited to 0-10. Accordingly, a modified sigmoid function is used in the activation function for the output layer of the actor network.

$$\phi(x) = \frac{10}{1 + e^{-(x - ln9)}}. \tag{5}$$

The action noise $\mathcal{N}_i$ is defined as expressed in (6). Because the action noise $\mathcal{N}_i$ is employed to promote the exploration of k, its value should be large in the initial stage and decrease gradually as the number of training episodes i increases. Accordingly, we defined $\mathcal{N}_i$ such that in episode 1, $\mathcal{N}_i$ is extracted according to the normal distribution $N(0, 0.1)$, and in the last episode M, a very small value of $\mathcal{N}_i$ is extracted by reducing the variance of the normal distribution.

$$\mathcal{N}_i \sim \mathrm{N}\left(0, \frac{1}{10} \times \exp\left(-\frac{2(i-1)}{M}\right)\right), \quad i = 1, \dots, M. \tag{6}$$

A reward serves as an important factor in the convergence of RL. Based on how the reward is given, the agent may find a policy that minimizes the TWT, become stuck in a local minimum, or diverge. In terms of scheduling, the episode in which ATC-DDPG is generated corresponds to the job schedule. Thus, we can consider employing a method for giving a positive reward if the current episode achieves a TWT lower than the lowest among the recorded TWTs with the progress of episodes and a negative reward otherwise. However, because this method is extremely strict, a considerable amount of training time may be incurred for the agent to achieve its goal, and the training may not converge. To address this factor, we adapted the zone of proximal development (ZPD) concept of cognitive development theory to the reward method. The ZPD, proposed by the social psychologist Vygotsky, refers to a learning zone in which a student cannot learn alone but can succeed in learning with the help of an excellent teacher [19]. The learning method in the ZPD is a step-by-step process through a learning ladder, whereas the learning process using the lowest recorded TWT is a learning method that updates records without a learning ladder, which makes learning more difficult than that in the ZPD.

ATC-DDPG pursues an effect similar to providing a learning ladder to the agent using $TWT_{ZPD} = \{TWT_{ZPD^{(i)}} | i = 1, \dots, 10\}$, which is composed of the 10 lowest TWTs among the TWT records accumulated from learning, as the criterion for giving the rewards. Among the 10 TWT_{ZPD} values, the lowest TWT value is expressed as $TWT_{ZPD^{(1)}}$, and the largest TWT value is expressed as $TWT_{ZPD^{(10)}}$. Based on $TWT_{ZPD^{(10)}}$, a positive reward is given, and TWT_{ZPD} is updated when a TWT value less than or equal to $TWT_{ZPD^{(10)}}$ is recorded. The details of the rewarding method are as follows:

$$r = \begin{cases} -1 - \delta & if\ TWT_c > TWT_{ZPD^{(10)}}, \\ 1 + \delta & if\ TWT_c \le TWT_{ZPD^{(10)}}. \end{cases} \tag{7}$$

First, TWT_{ZPD} is initialized by performing 10 episodes using the initialized DDPG. Subsequently, for each episode, a reward r is given by comparing the current episode's TWT, i.e., TWT_c, with $TWT_{ZPD^{(10)}}$. In (7), δ is a relative reward expressed as $\delta = \frac{|TWT_c - TWT_{ZPD^{(10)}}|}{TWT_{ZPD^{(10)}}} \times 10$ that represents how far the current TWT_c is from $TWT_{ZPD^{(10)}}$. In an episode where $TWT_c > TWT_{ZPD^{(10)}}$, because the agent is unable to update $TWT_{ZPD^{(10)}}$, a negative reward $r = -1 - \delta$ is given. As the value of TWT_c increases, the reward δ also increases. Conversely, in an episode where $TWT_c \le TWT_{ZPD^{(10)}}$, a positive reward $r = 1 + \delta$ is given.

In addition, ATC-DDPG adapts the ZPD concept to the replay buffer as well. In the existing replay buffers, experiences are continuously updated as learning progresses; in turn, good experiences may disappear over time. In other words, good past records that can make an excellent teacher for the agent may disappear. Therefore, ATC-DDPG is designed to preserve the experiences that achieved TWT_{ZPD} in the replay buffer such that the experiences can serve as a learning ladder for the agent.

IV. EXPERIMENTS

This chapter introduces simulation-based experiments conducted to evaluate the performance of ATC-DDPG and analyzes the results. Section 4.1 describes the simulation and the setting of the DDPG hyperparameters. Section 4.2 analyzes how the proposed ATC-DDPG generates outstanding schedules in terms of the TWT through a preliminary experiment. Section 4.3 discusses the performance of ATC-DDPG by means of a main experiment considering three different workloads. The programs used during all experiments were implemented in Python and were executed on a personal computer equipped with an i5-7600 CPU (3.5 GHz).

A. SETUP

By referring to the studies of Wang and Usher [6] and Chen and Matis [20], we implemented a single-machine simulation with unequal job arrival times. The interval between the release time r_i of the jobs was assumed to follow an exponential distribution with mean λ; the weight w_i of the jobs was determined as an integer selected from the uniform distribution $U[1,5]$. The processing time p_i of jobs was determined according to the truncated normal distribution $N(\mu_i, \sigma_i^2)$, in which $3\sigma_i$ and higher values were not considered. Here, μ_i is an integer selected from the uniform distribution $U[1,10]$, and $\sigma_i = 0.1$. The due date d_i was determined as follows:

$$d_i = r_i + allowance\ factor \times p_i. \tag{8}$$

The allowance factor is a real number selected from the uniform distribution $U[3.0, 4.0]$. The performance indicator TWT can be expressed as (9) below. In (9), c_i denotes the completion time of job i, and tardiness is defined as $\max(0, c_i - d_i)$.

$$TWT = \sum_{i=1}^{N} w_i \times \max(0, c_i - d_i). \quad (9)$$

For the DDPG hyperparameter setting, we refer to the study by Lillicrap et al. [16]. To train the weights of the actor and critic networks, an Adam optimizer [21] with learning rates of 10^{-5} and 10^{-4} was used. During the experiment, three workload cases were considered, and the maximum queue length q in the input layer was set to 20, 30, and 50. The discount factor γ was set to 0.99. Furthermore, the actor network was composed of five hidden layers in addition to the input and output layers; in the first hidden layer, a sigmoid function was employed as the activation function, and ReLU was employed as the activation function for all other hidden layers [22]. For the activation function of the output layer, a modified sigmoid function was used to limit the scope of the action (see (5)). By contrast, for the critic network, the output layer used a linear activation function to output a real-valued Q-value. In both networks, convolutional layers were used up to the third hidden layer, the kernel number F was set to 40 in all layers, and the kernel size of each layer was $(3 \times 1), (1 \times 3)$, and (1×3) $(C = R = 3)$. Two fully connected layers comprised 100 nodes. The weights of the output layer were initialized based on the uniform distribution $U[-0.01, 0.01]$, and the weights of all other layers were initialized based on the He uniform [23]. The replay buffer size used in the experiment was 14,000.

B. PRELIMINARY EXPERIMENT

The preliminary experiment addressed a scheduling problem comprising 10 jobs. The analysis objective was to identify how ATC-DDPG can generate a better schedule than the ATC rule by tuning the lookahead parameter k. The arrival time of the jobs follows an exponential distribution with $\lambda = 7.5$ (light workload), and ATC-DDPG was trained 2,000 times using episodes composed of 1,000 jobs.

The Gantt chart in Fig. 4 shows the schedules for 10 jobs generated by ATC-bestK (fixed to k = 1.2) and by the trained ATC-DDPG. Schedule A was generated using ATC-bestK, whereas Schedule B was generated by ATC-DDPG. Schedules A and B exhibit different job selections from the sixth to eighth job selections: ATC-bestK processed the jobs on the order of 6-8-5, whereas ATC-DDPG processed the jobs on the order of 5-8-6.

Fig. 5 illustrates the changes in the cumulative weighted tardiness (CWT) of the two schedules from the sixth to eighth job selections. Until the sixth job selection, the CWT (CWT_A) of Schedule A and the CWT (CWT_B) of Schedule B were equal to 0. As shown in Fig. 5(a), although CWT_B increased to 4.39 owing to the sixth job selection, CWT_A remained at 0. As shown in Fig. 5(b), no change in CWT was observed from the seventh job selection. However, as shown in Fig. 5(c), CWT_A acutely increased from 0 to 28.62, whereas CWT_B increased further by 17.47 in the eighth job selection. Consequently, in this example, TWT_A displayed a higher value of TWT of 28.62 compared to the 21.86 of TWT_B. Here, the TWT_A generated by Schedule A was caused by Job 5 only. Although Schedule A set the weighted tardiness (WT) of Job 6 to 0 by processing Job 6 first, as a consequence, the WT of Job 5 increased excessively. By contrast, the TWT_B generated by Schedule B is composed of the WTs of Jobs 5 and 6. Although it temporarily displayed worse results than Schedule A by processing Job 5 first and increasing the WT of Job 5 to 4.39, the WT value of Job 6 was 17.47. Thus, Schedule B is better than Schedule A in terms of the TWT.

Subsequently, we verified whether the ATC-DDPG, which generated Schedule B, selected Job 5 by tuning the lookahead parameter k or by chance.

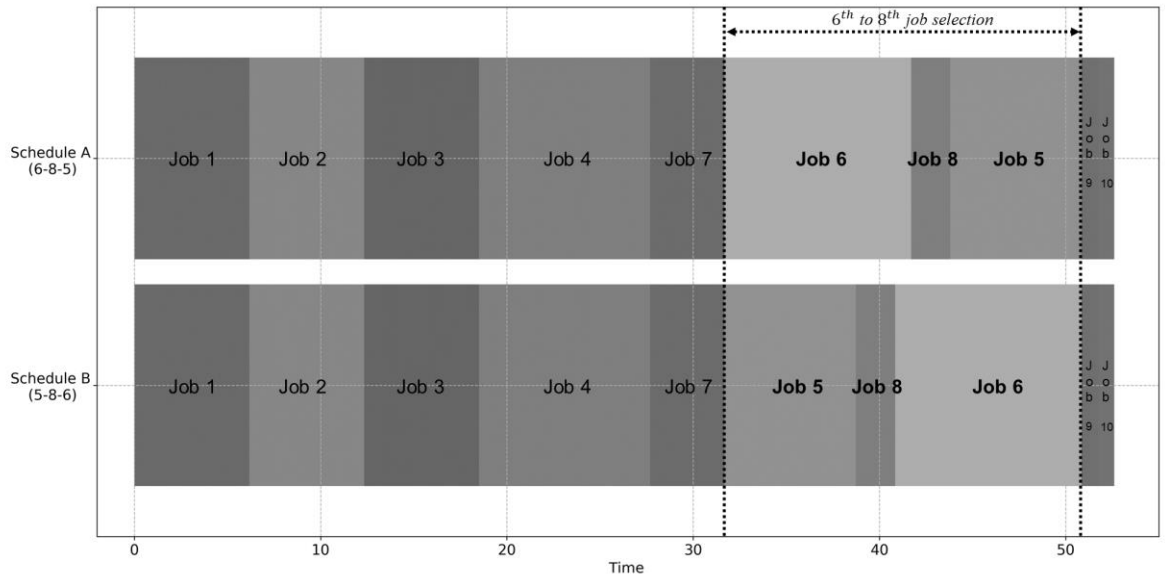

FIGURE 4. Schedules generated by ATC-bestK and ATC-DDPG (10 jobs and light workload case).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3152192, IEEE Access

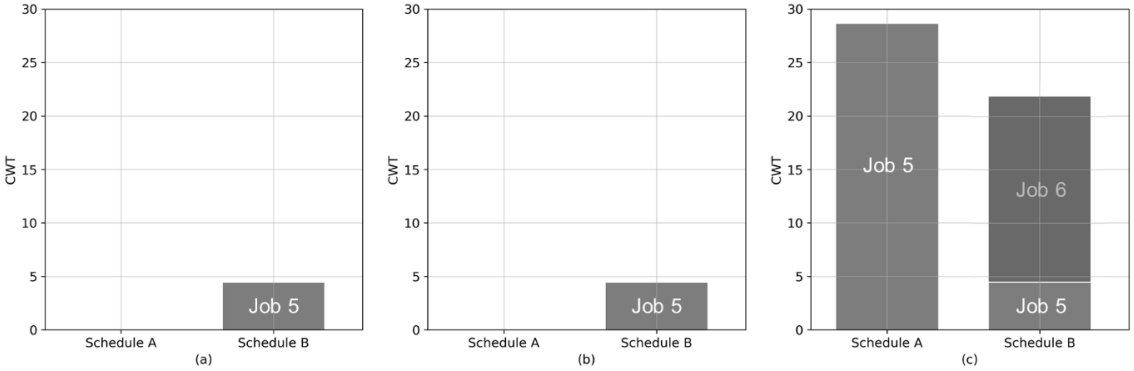

FIGURE 5. CWT: (a) 6^{th} job selection, (b) 7^{th} job selection, and (c) 8^{th} job selection.

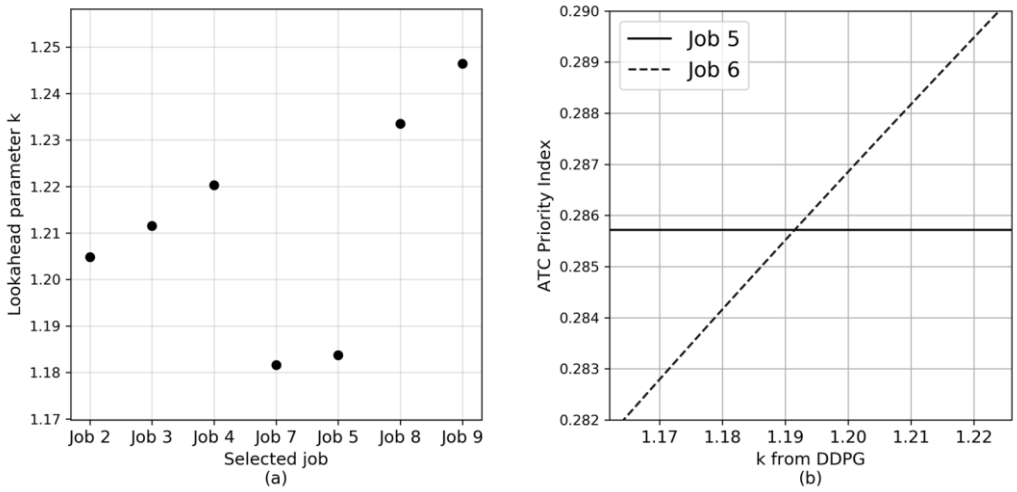

FIGURE 6. State-dependent lookahead parameter adjustment: (a) parameter values for schedule B generated by ATC-DDPG and (b) priority index values of Jobs 5 and 6 when Job 5 is dispatched.

Subsequently, we verified whether the ATC-DDPG, which generated Schedule B, selected Job 5 by tuning the lookahead parameter k or by chance. Fig. 6(a) illustrates the jobs selected during the seven job selections in Schedule B and the k values applied. Jobs 1, 6, and 10, which are not shown in Fig. 6(a), were not applied with a k value because they were the only jobs present in the job queue. ATC-DDPG provided a k value of 1.2 or more when selecting Jobs 2, 3, and 4. However, unlike the previous trend, it provided a low k value of approximately 1.18 when selecting Jobs 7 and 5 and then provided a k value of 1.2 or more again when selecting Jobs 8 and 9. Accordingly, the temporary drop in k observed in Fig. 6(a) is regarded as an intentional decrease induced by ATC-DDPG; in turn, the machine was able to eventually select Job 5.

The intentional drop in k is extremely interesting behavior. In fact, if k was maintained above 1.2, similar to the previous trend, the ATC-DDPG would have been unable to select Job 5 first. This scenario is shown in Fig. 6(b). Because the slack time s_5 of Job 5 was -2.15 (a negative value), the priority index was constant at $I_5 = w_5/p_5$ (solid line). By contrast, because the slack time s_6 was 5.67, $I_6 = \frac{w_6}{p_6}\exp\left(-\frac{5.67}{k\bar{p}}\right)$ changed according to k (dotted line). If the value of k was high, e.g., 1.2, ATC-DDPG would have selected Job 6 first because $I_5 < I_6$. However, DDPG tuned the k value to approximately 1.18, and Job 5 was selected first because $I_5 > I_6$. In other words, when a job with a positive s_i and a job with a negative s_i are present in the job queue, ATC-DDPG tunes the k value to a smaller value to increase the priority such that an urgent job with a negative s_i can be selected first. In summary, based on the preliminary experiment, we confirmed that ATC-DDPG can generate a schedule that can improve the TWT by providing an appropriate lookahead parameter according to the states of the waiting jobs.

C. MAIN EXPERIMENT

The main experiment considered three different workload cases (light, medium, and heavy) to measure the generalized performance of ATC-DDPG. The job data of the three cases were generated in an environment in which parameter λ of the exponential distribution, which determines the release time, was set to 7.5, 7.0, and 6.5. Consequently, the machine utilization rates of the three cases were 84%, 89%, and 94%, respectively.

One episode is composed of 1,000 jobs, and ATC-DDPG was repeatedly trained on the same episode 2,000 times. The training times for the three cases were approximately 16, 25, and 39 h. Fig. 7 illustrates the change in the TWT as the training of ATC-DDPG progresses. Although the TWT showed large fluctuations in the initial stage of training, it decreased stably as training progressed.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3152192, IEEE Access

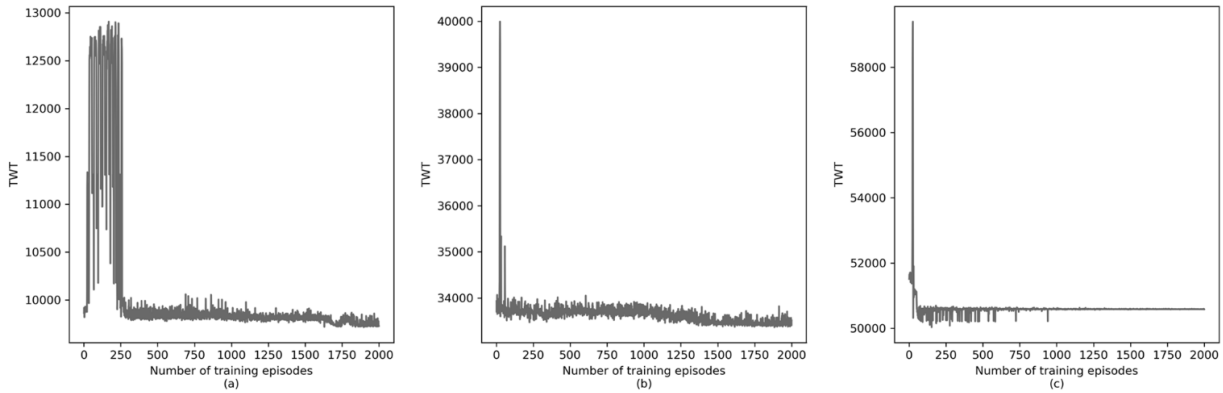

FIGURE 7. TWT according to the progress of ATC-DDPG learning: (a) light workload case, (b) medium workload case, and (c) heavy workload case.

As a performance comparison with other methods, we compared the proposed ATC-DDPG with FIFO, WSPT, MS, ATC-bestK, and ATC-vk2. In ATC-bestK, the best k value is derived experimentally. To determine the value of k, the TWT was calculated while increasing k by 0.1 intervals in 300 different episodes for each workload case. Subsequently, the k value that achieved the minimum TWT the most frequently in each case was selected as bestK. As a result, the bestK values of the three workload cases were derived as 1.2, 1.3, and 1.6.

The trained ATC-DDPG was compared with the other methods for 100 different episodes. Table 2 summarizes the number of wins of each algorithm in 100 tests. In all workload cases, FIFO, WSPT, and MS showed zero wins, and ATC-DDPG displayed the best performance. In the light workload case, ATC-DDPG recorded 59 wins, exhibiting better performance than ATC-bestK, which recorded 31 wins. Although ATC-vk2 was unable to demonstrate better performance than ATC-bestK or ATC-DDPG, its results are deemed reasonable considering the fact that ATC-vk2 is not designed for dynamic environments. In the medium workload case, ATC-DDPG exhibited better performance than ATC-bestK by recording 59 wins; however, the number of wins recorded by ATC-bestK increased to 38, indicating an improved performance over that in the light workload case. In the heavy workload case, ATC-DDPG displayed slightly better performance than ATC-bestK. This result is considered a phenomenon that occurs because the number of jobs with a slack time of close to zero increases as the workload increases; in turn, the opportunities to generate good schedules decrease.

TABLE 2. Number of wins in 100 tests for each algorithm.

Workload	FIFO	WSPT	MS	ATC-bestK	ATC-vk2	ATC-DDPG
Light	0	0	0	31	10	**59**
Medium	0	0	0	38	3	**59**
Heavy	0	0	0	49	0	**51**

TABLE 3. Mean TWT (standard deviation) of 100 tests for each algorithm.

Workload	FIFO	WSPT	MS	ATC-bestK	ATC-vk2	ATC-DDPG
Light	27,515.3 (11,865)	15,822.2 (3,398.1)	21,728.0 (11,279.2)	7,710 (3,278.6)	8,067.4 (3,466.1)	**7,708.3** (3,287.2)
Medium	63,267.3 (37,609.9)	19,012.2 (9,440.6)	55,797.0 (36,950.8)	17,073.1 (9,439.4)	17,942.3 (9,876.9)	**17,063.5** (9,433.6)
Heavy	197,180.8 (112,081.2)	49,101.6 (24,324.8)	187,804.8 (111,361.7)	46,990.0 (24,402.6)	49,159.5 (24,888.5)	**46,885.1** (24,273.2)

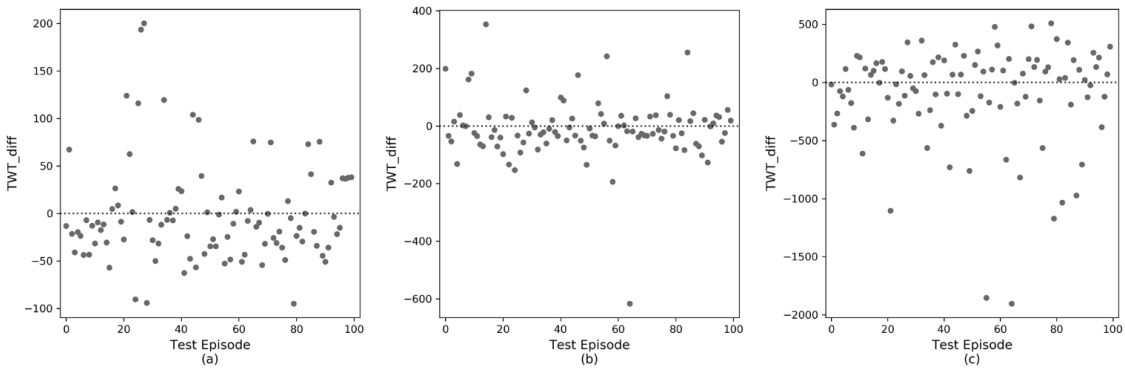

FIGURE 8. TWT performance improvement of ATC-DDPG in 100 tests: (a) light workload case, (b) medium workload case, and (c) heavy workload case.

Table 3 summarizes the mean and standard deviation of the TWT in 100 tests for each algorithm. In all workload cases, ATC-DDPG yielded the lowest mean TWT. The mean TWTs of the FIFO and MS algorithms, which did not reflect the job weight, were 1.5—4-times higher than those of the WSPT and ATC-based algorithms. In addition, ATC-bestK displayed outstanding performance compared to that of ATC-DDPG because it set the lookahead parameter close to the optimal value while adjusting it in steps of 0.1.

Finally, we verified the performance improvement of ATC-DDPG by comparing its TWT with that of ATC-bestK. Fig. 8 shows the value of TWT_{diff} (TWT of ATC-DDPG – TWT of ATC-bestK) in each test episode for each workload case. The dots presented below the dotted line indicate a negative TWT_{diff}, i.e., the test episodes in which ATC-DDPG is the winner. In the light workload case, ATC-DDPG won the most test episodes, as shown in Fig. 8(a). Although the TWT values worsened to +200 in a few episodes in which ATC-DDPG did not win, a TWT improvement of approximately -50 was observed in most episodes. In summary, ATC-DDPG was able to decrease the TWT in most cases by appropriately tuning the lookahead parameter based on the state of the waiting jobs, although in exceptional cases, it occasionally generated incorrect schedules owing to a change in k. This result is attributed to the limitations encountered by RL in searching the high-dimensional state-action space. Figs. 8(b) and 8(c) show much more notable improvements in the TWT. Significant improvements in the TWT were observed in many episodes, and remarkable improvements of -600 to -2000 were observed in some cases.

V. CONCLUSION

The ATC rule has been known to be the best dispatching rule for solving the TWT problem in single-machine scheduling. The performance of the ATC rule is determined by the lookahead parameter k. Although existing studies used a fixed value or a value calculated through a handcrafted function as the lookahead parameter, such an approach has a limitation that loss of information is inevitable because it uses summarized job data to estimate k. This study attempted to overcome this limitation of the existing ATC rules by proposing an RL-based ATC dispatching rule.

The contributions of this study can be summarized as follows. First, ATC-DDPG is the first dispatching algorithm that implements RL to estimate the k value of ATC. ATC-DDPG showed the applicability of the state-of-the-art RL algorithm in the manufacturing field by solving the dynamic scheduling problem, one of the most challenging problems in the manufacturing field. Second, ATC-DDPG algorithm could estimate k using only raw job data and achieved outstanding performance without explicit function which is required in existing studies. The RL components of ATC-DDPG were elaborately designed so that the agent could achieve a lower TWT, and the network components of ATC-DDPG effectively learned the relationship between raw job data and k. Third, this study introduced a replay buffer using a cognitive development concept known as ZPD and a new rewarding method for ATC-DDPG, thereby enabling effective parameter training. The preliminary experiment verified that ATC-DDPG could generate a schedule that can improve TWT by intelligently selecting k values that are dependent on the state of the job queue. Furthermore, in the main experiment, ATC-DDPG demonstrated outstanding performance in terms of the TWT compared to other dispatching rules in the three workload cases.

The ATC dispatching rule is a representative composite dispatching rule. In the future, the DDPG algorithm should be extended to the composition of more diverse dispatching rules according to the job states. In such a case, the action will serve as the weight that averages the priority scores of the dispatching rules. The extended DDPG is expected to be applicable to other scheduling performance indicators in addition to the TWT. Furthermore, ATC-DDPG can be extended to dynamic job shop environments composed of multiple machines. Although the ideal method is to regard the jobs in the job queue of every machine as an RL state, training can become extremely unstable as the number of machines increases; hence, a method of applying ATC-DDPG to only some machines with large workloads can also be considered.

REFERENCE

[1] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. Sched.*, vol. 12, no. 4, pp. 417–431, 2009.

[2] E. L. Lawler, "A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness," *Ann. Discret. Math.*, vol. 1, pp. 331–342, 1977.

[3] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Ann. Discret. Math.*, vol. 1, pp. 343–362, 1977.

[4] A. Atighehchian and M. M. Sepehri, "An environment-driven, function-based approach to dynamic single-machine scheduling," *Eur. J. Ind. Eng.*, vol. 7, no. 1, pp. 100–118, 2013.

[5] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Rob. Auton. Syst.*, vol. 33, no. 2, pp. 169–178, 2000.

[6] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Eng. Appl. Artif. Intell.*, vol. 18, no. 1, pp. 73–82, Feb. 2005.

[7] X. Chen, X. Hao, H. W. Lin, and T. Murata, "Rule driven multi objective dynamic scheduling by data envelopment analysis and reinforcement learning," *2010 IEEE Int. Conf. Autom. Logist.*, pp. 396–401, 2010.

[8] J. Shahrabi, M. A. Adibi, and M. Mahootchi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Comput. Ind. Eng.*, vol. 110, pp. 75–82, 2017.

[9] A. P. J. Vepsalainen and T. E. Morton, "Priority Rules for Job Shops with Weighted Tardiness Costs," *Manage. Sci.*, vol. 33, no. 8, pp. 1035–1047, Aug. 1987.

[10] C. N. Potts and L. N. V. Van Wassenhove, "Single Machine Tardiness Sequencing Heuristics," *IIE Trans.*, vol. 23, no. 4, pp. 346–354, 1991.

[11] J. M. S. Valente, "Improving the performance of the ATC dispatch rule by using workload data to determine the lookahead parameter value," *Int. J. Prod. Econ.*, vol. 106, no. 2, pp. 563–573, Apr. 2007.

[12] J. Heger and T. Voss, "Dynamically adjusting the k-values of the ATCS rule in a flexible flow shop scenario with reinforcement learning," *Int. J. Prod. Res.*, 2021.

[13] J. E. Holsenback, R. M. Russell, R. E. Markland, and P. R. Philipoom, "An improved heuristic for the single-machine, weighted-tardiness problem," *Omega*, vol. 27, no. 4, pp. 485–495, 1999.

[14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M.

Riedmiller, "Deterministic policy gradient algorithms," *31st Int. Conf. Mach. Learn.*, pp. 605–619, 2014.

[15] T. Degris, M. White, and R. S. Sutton, "Off-Policy Actor-Critic," *Proc. 29th Int. Conf. Mach. Learn.*, vol. 1, pp. 457–464, May 2012.

[16] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *4th Int. Conf. Learn. Represent.*, Sep. 2015.

[17] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *arXiv Prepr. arXiv*, Dec. 2013.

[19] L. S. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, 1978.

[20] B. Chen and T. I. Matis, "A flexible dispatching rule for minimizing tardiness in job shop scheduling," *Int. J. Prod. Econ.*, vol. 141, no. 1, pp. 360–365, 2013.

[21] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *3rd Int. Conf. Learn. Represent.*, Dec. 2014.

[22] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," *Proc. 14th Int. Conf. Artif. Intell. Stat.*, vol. 15, pp. 315–323, 2011.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 1026–1034, 2015.