IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 26, 2021, accepted July 11, 2021, date of publication July 15, 2021, date of current version July 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3097254

Deep Reinforcement Learning for Minimizing Tardiness in Parallel Machine Scheduling With Sequence Dependent Family Setups

BOHYUNG PAENG[1,2], IN-BEOM PARK[3], AND JONGHUN PARK[1,2]

[1]Department of Industrial Engineering, Seoul National University, Seoul 08826, South Korea
[2]Institute for Industrial Systems Innovation, Seoul National University, Seoul 08826, South Korea
[3]Department of Industrial Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: In-Beom Park (inbeom@skku.edu)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by Ministry of Science and ICT (MSIT) under Grant NRF-2015R1D1A1A01057496, and in part by the Institute of Engineering Research, Seoul National University.

ABSTRACT Parallel machine scheduling with sequence-dependent family setups has attracted much attention from academia and industry due to its practical applications. In a real-world manufacturing system, however, solving the scheduling problem becomes challenging since it is required to address urgent and frequent changes in demand and due-dates of products. To minimize the total tardiness of the scheduling problem, we propose a deep reinforcement learning (RL) based scheduling framework in which trained neural networks (NNs) are able to solve unseen scheduling problems without re-training even when such changes occur. Specifically, we propose state and action representations whose dimensions are independent of production requirements and due-dates of jobs while accommodating family setups. At the same time, an NN architecture with parameter sharing was utilized to improve the training efficiency. Extensive experiments demonstrate that the proposed method outperforms the recent metaheuristics, rule-based, and other RL-based methods in terms of total tardiness. Moreover, the computation time for obtaining a schedule by our framework is shorter than those of the metaheuristics and other RL-based methods.

INDEX TERMS Deep reinforcement learning, unrelated parallel machine scheduling, sequence-dependent family setups, total tardiness objective, deep Q-network.

I. INTRODUCTION

As the competition among enterprises intensifies, production scheduling becomes one of the essential decision-making problems in modern manufacturing systems. Specifically, manufacturers should fulfill production orders under the sequence-dependent family setup time (SDFST) requirement that occurs when two products belonging to different families are consecutively processed on a machine [1]. Furthermore, since customer demands frequently and unpredictably change, it is required to deal with the variabilities associated with the production requirements and due-dates of the products [2]. Accordingly, there is a challenge in developing a scheduling method that is able to obtain high-quality schedules while accommodating the variabilities.

The associate editor coordinating the review of this manuscript and approving it for publication was Hao Shen.

We focus on the unrelated parallel machine scheduling problem (UPMSP) with SDFST, which has attracted a great deal of attention in various domains such as semiconductor [3]–[5], chemical [6], and food industries [7]. A UPMSP aims to allocate each job to one of the machines where the processing time of a job on different machines is not related. This scheduling problem is known to be NP-hard for minimizing the total tardiness [8].

Metaheuristics have been successfully adopted for solving UPMSPs with SDFST under due-date constraints [9]–[12]. Unfortunately, it is not guaranteed for them to find a high-quality schedule for large-scale scheduling problems within a specific time limit. As an alternative, manufacturers have actively employed rule-based methods due to their short computation time, and ease of implementation [13]. However, schedules obtained by the rule-based methods may not be satisfactory since their decisions are made in a myopic manner [14].

To overcome the drawbacks of rule-based methods, reinforcement learning (RL) approaches have been actively investigated from decades ago [15], [16]. The purpose of RL is to learn an adaptive policy that maximizes the expected sum of cumulative rewards. In recent years, due to the remarkable success in deep reinforcement learning (DRL) that utilizes deep neural networks (DNNs) [17], several studies have shown promising results on the scheduling problems in manufacturing systems [18]–[21]. Yet, there are still two challenges to solve UPMSPs based on a DRL-based method while addressing SDFST as well as the variabilities in terms of production requirements and due-dates. First, the size of the state space might become large when accommodating due-dates and sequence-dependent setups in a state representation of a neural network, which leads to difficulties in function approximation and DNN generalization [21]. Second, since learning complexity grows quickly as the numbers of jobs and machines increase, it is intractable to re-train a DNN whenever such variabilities occur in large-scale manufacturing systems.

To this end, we propose a DRL-based method for minimizing tardiness for UPMSP with SDFST to address the above challenges. It is worth noting that learning DNN parameters robust to changes in production requirements and due-dates is the primary concern of our work. The contributions of this paper are summarized as follows:

- We design a novel state representation whose dimensionality is independent of production requirements and due-dates of jobs while accommodating SDFST. Given a state, an action is executed by periodically determining a setup status of a machine and a family of a job.
- To reduce the size of the network and increase the training efficiency, we proposed a DNN architecture in which network parameters are shared among several hidden layers [22]. In the experiments, the effectiveness of the proposed state representation and parameter sharing was demonstrated.
- To validate the performance of the proposed method, we tested our method on large-scale datasets. Experimental results showed that the proposed method outperforms recent metaheuristics, rule-based, and other RL-based methods in terms of the total tardiness for all datasets. Moreover, the computation time taken by the proposed method was shorter than those of other RL-based methods and metaheuristics considered. Finally, the robustness of the proposed method was investigated by solving scheduling problems with stochastic processing and setup time.

The rest of this paper is organized as follows. Section II introduces the previous approaches for solving the UPMSP with SDFST and the implications of studies on RL-based scheduling methods. Section III defines the scheduling problem considered in this paper. Details of the proposed method and its training algorithm are presented in Section IV. Performance comparisons with considered alternatives are carried out in Section V. Finally, Section VI concludes this paper.

II. LITERATURE REVIEW

A. UPMSPs WITH SDFST UNDER DUE-DATES CONSTRAINTS

UPMSPs with due-date related objectives are classical scheduling problems that have been comprehensively researched for decades [23]. In particular, several studies have addressed SDFST as main constraints [24]–[27]. To reduce the number of setups in a scheduling problem, batch scheduling heuristics were popularly adopted by forming batches of jobs processed on a machine without a setup [28]. To minimize the weighted tardiness, two-level batch heuristics were proposed under the assumption of common due-dates [29], and identical jobs [30]. Additionally, the batch apparent tardiness cost with setup (BATCS) heuristic was suggested by incorporating the processing time, slack time, and family setup time [31]. For the total tardiness objective, an improved simulated annealing heuristic was developed through incorporating a batch-based repair method [9].

On the other hand, metaheuristics have been adopted for solving UPMSPs with SDFST to minimize the total tardiness. They employed the batch formation technique to restrict schedules with a small number of setup changes in the solution space. The scheduling problem with primary customer constraints was successfully solved by the iterated greedy (IG) algorithm [10], which enhances an initial solution through the iterative neighborhood selection and exhaustive local search. More recently, Pinheiro *et. al.* [11] developed an improved IG by designing six local search operations such as the batch swap and job insertion. They showed competitive results in solving scheduling problems with six machines and 50 jobs. In [12], the artificial bee colony algorithm was proposed through applying crossover operations to exchange job sequences.

B. REINFORCEMENT LEARNING ON UPMSPs

Markov processes have been broadly adopted to solve optimal control problems in the presence of abrupt changes in system dynamics [32]. To the recent, several Markov random processes were employed with advantages of modeling fuzzy systems, such as Markov chaotic systems [33] and semi-Markov jump nonlinear systems [34]. For solving scheduling problems, a manufacturing system is formulated as Markov decision process (MDP) [35], and RL is utilized to learn a policy of MDP. RL aims to train an agent by interacting with an environment that consists of everything outside the agent. In particular, Q-learning (QL) [36], one of the representative model-free RL approaches, has been widely adopted to solve scheduling problems. Given a state observed from the environment that corresponds to a manufacturing system, the agent makes scheduling decisions by predicting the estimated value of an action called the Q-value.

For solving UPMSPs by utilizing RL-based methods, Zhang *et. al.* adopted QL to minimize the weighted tardiness [37], [38]. They employed a linear basis function to approximate Q-values for given state features indicating the status

of all jobs and machines. After six heuristics are designed as actions, QL-based adaptive rule selections outperformed individual rules. Although [37] dealt with sequence-dependent setups, the agents in [37] were only validated on the same scheduling problems as those for training. The method in [38] was tested on various scheduling problems without considering setups.

Yuan *et. al.* [39], [40] addressed ready time constraints and machine breakdown for minimizing the total tardiness and number of tardy jobs, respectively. They adopted a tabular method that stores Q-values by exploring state-action pairs. To represent the state in a limited size of the table, the values of continuous attributes should be discretized into several groups. For instance, in [40], the mean lateness of jobs was classified as being greater than or less than zero. As alternative methods for representing the state, unsupervised learning methods, such as self-organizing map [41], and k-means nearest neighbor [42], [43], have also been adopted for solving scheduling problems.

C. DEEP REINFORCEMENT LEARNING FOR SOLVING SCHEDULING PROBLEMS

Compared to the traditional RL methods, DRL aims to approximate Q-values from high-dimensional state features through DNNs [17]. This helps an agent to learn a nonlinear policy from large and continuous state spaces. Besides, [17] proposed a target network to enhance learning stability, and experience replay for reducing correlations among state-action pairs. With the extensive application of DRL to various decision-making problems such as energy supply control [44], vehicle routing [45], and electricity market pricing [46], it has gradually attracted prominence in the field of scheduling. Until recently, a lot of studies widened the scope of DRL to scheduling problems in computer resource management [47], [48] and distributed system [49], [50]. Those studies encoded the state as a 2D matrix of resources and upcoming timesteps. Since they employed fully connected networks, the matrix was flattened into a one-dimensional vector.

In addition, DRL-based methods have been employed for solving scheduling problems in manufacturing systems. In a hybrid flow shop scheduling problem, an agent allocates jobs from a given state that indicates whether the machine status is idle or busy or finished [51]. Among various shop scheduling problems, several researchers have investigated DRL-based methods for minimizing the makespan in job shop scheduling problems. Lin *et. al.* [52] represented machine status and statistics of processing time in the state to infer dispatching rules for each machine. In [20], the setup time was considered by utilizing the setup history and setup status of all machines. Recently, convolutional neural networks were employed to address the states represented in 2D matrices indicating relationships between jobs, operations, and machines [19], [53].

On the other hand, a few DRL approaches were developed for solving scheduling problems with due-date related objectives. In [54], due-dates of all waiting jobs were represented in the state for maximizing throughput. To minimize the total tardiness in a single machine scheduling problem, [55] utilized the slack time, which refers to the difference between the processing time and remaining time until due-dates of a job. Washneck *et. al.* [18] accommodated setup constraints in a semiconductor production scheduling problem for minimizing due-date deviations. Since their state included the setup status of all machines and due-dates of all jobs, DNNs in [18] should be trained again when solving new scheduling problems whose number of jobs is different from those of the training. Luo *et. al.* [21] focused on new job insertions in the flexible job shop scheduling problem under ready time constraints with the total tardiness objective. For a state that consists of seven features, an action is determined by selecting one of the heuristics. Yet, they did not consider setup constraints.

III. PROBLEM DESCRIPTION

In this section, we describe UPMSP with SDFST considered in this paper. There are N_J jobs where the j^{th} job is denoted as J_j. Each job belongs to one of N_F families from the set $\mathcal{F} = \{1, 2, \ldots, N_F\}$. Each job can be processed by one of any N_M machines where the i^{th} machine is denoted as M_i. The processing time is denoted as $p_{i,j}$ when the job J_j is performed on M_i. A job is finished after being processed once on one of the machines. Let $P(f)$ be the total number of the jobs that belong to family f, which indicates the production requirements of family f. As a result, the following equation holds:

$$\sum_{f=1}^{N_F} P(f) = N_J \tag{1}$$

At the beginning of the scheduling, due-dates of J_j, denoted as d_j, are given. Let G_i denote the current setup status of M_i. G_i is an element of $\mathcal{F}$ and equivalent to g if the job of family g can be processed on the machine without a setup change. If a job of family f is assigned on M_i whose G_i is g, the setup time, denoted as $\sigma_{f,g}$, is incurred before the job is processed on the machine.

The goal of the scheduling is to allocate each job to one of the machines in order to minimize the total tardiness, denoted as TT, which is defined as

$$TT = \sum_{j=1}^{N_J} \max(0, c_j - d_j) \tag{2}$$

where c_j is the completion time of J_j. When there is no setup time, the scheduling problem considered becomes equivalent to the problem in [8], which is proven to be NP-hard. Finally, the assumptions made in this paper are listed below.

- At the beginning of the scheduling, all jobs are ready to be processed, and all machines have been set up.
- There is no machine breakdown.
- After the setup change of a machine is finished for processing a job, the machine immediately starts to process the job.

TABLE 1. Five matrices and one vector that compose a state.

Notation	Name	Dimension
$\mathbf{S}_w$	Waiting job status	$N_F \times 2H_w$
$\mathbf{S}_p$	In-progress job status	$N_F \times H_p$
$\mathbf{S}_s$	Machine setup time	$N_F \times N_F$
$\mathbf{S}_u$	Utilization history	$N_F \times 3$
$\mathbf{S}_a$	Action history	$N_F \times 2$
$\vec{S}_f$	Family-independent history	3

- Machine can only process one job at a time.
- The preemption is not allowed.
- The moving time for each job is zero.

IV. PROPOSED METHOD

We propose a DRL-based scheduling method in this section which is divided into four subsections. First, we describe the MDP to solve UPMSP with SDFST constraint. Second, the proposed parameter sharing architecture is introduced. Finally, a training algorithm and the flowchart are described.

A. MDP FORMULATION

For employing DRL, the scheduling problem considered in this paper is formulated as MDP. We denote the state, action, and reward at timestep k as s_k, a_k, and r_k, respectively. After the agent executes an action a_k, the state transition takes place from s_k to s_{k+1}. Then, reward r_k and next state s_{k+1} are observed. We denote the time interval from s_k until s_{k+1} as the period k. In this paper, we model that the time spent in each period, denoted as T, is constant. We define in detail the action, state, reward, and state transition in the following.

1) ACTION

When an action is defined for each pair of a job and a machine, the size of the action space grows quickly as the numbers of jobs and machines increase. To reduce the size of action space, we define an action as a tuple of a job family and a machine setup status. The action set $\mathcal{A}$ is defined as follows.

$$\mathcal{A} = \{(f, g) | f = 1, \ldots, N_F, g = 1, \ldots, N_F\} \tag{3}$$

Then, we denote the feasible action set at s_k as $\mathcal{A}_k \subset \mathcal{A}$. An action $a_k = (f_k, g_k)$ indicates that a job of family f_k will be assigned on a machine whose setup status is g_k during the period k, where $a_k \in \mathcal{A}_k$. We note that a_k is feasible only if there is a waiting job of family f_k and an idle machine whose setup status is g_k in the period k.

2) STATE

When adopting RL-based methods to solve scheduling problems, the state is usually represented by utilizing observations on the current status of machines and jobs. If the dimension of a state varies as production requirements and due-dates of jobs change, the DNN is required to be re-trained whenever such changes occur. To solve the scheduling problems without re-training DNNs even when such changes occur, we propose a family-based state representation whose dimensionality is independent of the production requirements and due-dates of jobs.

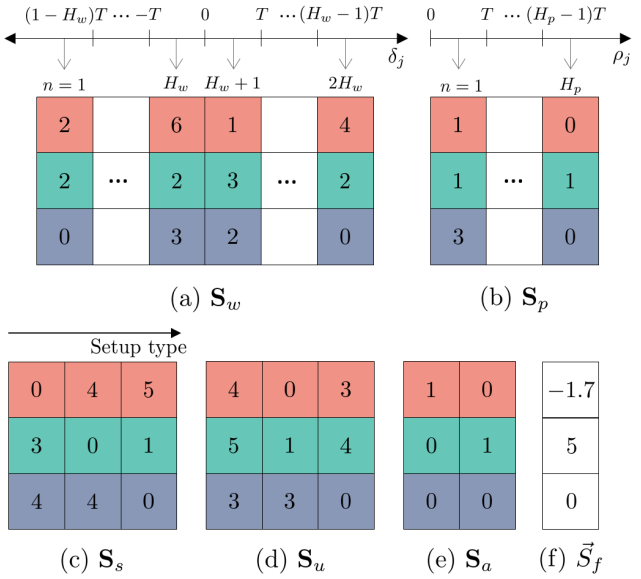

FIGURE 1. The 2D matrix representations of s_k. Red, green, and blue colors indicate family 1, 2, and 3, respectively. The numbers inside the matrices correspond to the values in s_k.

The proposed state consists of five 2D matrices and one vector. Table. 1 describes them in terms of notations, names, and dimension. To help the understanding of s_k, we present an example of s_k in a scheduling problem with three families, as depicted in Fig. 1. Next, we define the details of s_k. It is noted that the index k is omitted in each of the five matrices and vector for the sake of conciseness.

- $\mathbf{S}_w$: To accommodate the due-dates of waiting jobs, the value of the f^{th} row in this matrix refers to the number of waiting jobs for family f where their due-dates belong to one of periods. For separately counting each waiting job J_j with respect to their remaining time until due-dates, denoted as $\delta_j = d_j - kT$, $\mathbf{S}_w(f, n)$ is defined as follows.

$$\mathbf{S}_w = \begin{cases} |\{J_j \in \mathcal{W}_k(f) \,|\, \delta_j \leq (1-H_w)T\}| & n = 1 \\ |\{J_j \in \mathcal{W}_k(f) \,|\, \delta_j > (H_w-1)T\}| & n = 2H_w \\ |\{J_j \in \mathcal{W}_k(f) \,|\, \left\lceil \frac{\delta_j}{T} \right\rceil = n - H_w\}| & \text{otherwise} \end{cases} \tag{4}$$

where $\mathcal{W}_k(f)$, $\lceil \cdot \rceil$, and H_w refer to the set of waiting jobs that belong to family f at s_k, ceiling function, and a positive integer smaller than $\max_j d_j/T$, respectively. As indicated by Eq. (4) and the vertical line on the top of Fig. 1(a), $\mathbf{S}_w(f, 1)$ and $\mathbf{S}_w(f, 2H_w)$ refer to the number of waiting jobs for family f where their δ_j are smaller than $(1-H_w)T$ and larger than $(H_w-1)T$, respectively. This representation aims to restrict the number of columns in $\mathbf{S}_w$ into $2H_w$ while capturing due-dates of all waiting jobs in the matrix. The practical rationale behind Eq. (4)

is that jobs whose due-dates are quite far from the current period have a smaller impact on executing an appropriate action than the other jobs. In Fig. 1(a), $\mathbf{S}_w(1, H_w)$ is equal to 6, which indicates that there are six waiting jobs whose family is 1 and $\left\lceil \frac{\delta_j}{T} \right\rceil = 0$.

- $\mathbf{S}_p$: This matrix contains the number of in-progress jobs for each family of which their remaining processing time is included in one of periods. In-progress jobs refer to the ones that are currently being processed on a machine. Since the tardiness of in-progress and finished jobs are already determined, due-dates of in-progress jobs are not relevant to minimizing the tardiness. Meanwhile, the remaining processing time of in-progress jobs affects the completion time of the waiting jobs that will be assigned after in-progress jobs are completed. Therefore, each job J_j that belongs to $\mathcal{P}_k(f)$ is classified according to the remaining processing time, denoted as ρ_j, where $\mathcal{P}_k(f)$ is the set of in-progress jobs of the family f at s_k. Then, we define $\mathbf{S}_p(f, n)$ as below.

$$\mathbf{S}_p = \begin{cases} |\{J_j \in \mathcal{P}_k(f) \,|\, (n-1)T < \rho_j \leq nT\}| & n < H_p \\ |\{J_j \in \mathcal{P}_k(f) \,|\, (n-1)T < \rho_j\}| & n = H_p \end{cases} \tag{5}$$

where H_p is a positive integer less than $\max_j \rho_j/T$. We note that the dimension of the column in $\mathbf{S}_p$ is constrained to H_p in a similar way to Eq. (4). For example in Fig. 1(b), $\mathbf{S}_p(2, 1)$ is equal to 1, which indicates that there exists one job satisfying the following two conditions: the job belongs to $\mathcal{P}_k(2)$, and its remaining processing time is shorter than T.

- $\mathbf{S}_s$: To capture the family setup time for predicting the tardiness that will be incurred after s_k, $\mathbf{S}_s(f, g)$ represents the required setup time to process a job of family f on a machine whose setup status is g. Specifically,$\mathbf{S}_s(f, g)$ is defined as follow.

$$\mathbf{S}_s(f, g) = \begin{cases} \sigma_{f,g} & (f, g) \in \mathcal{A}_k \\ \sigma_{\max} & \text{otherwise} \end{cases} \tag{6}$$

where $\sigma_{\max}$ refers to the maximum setup time. If $(f, g) \notin \mathcal{A}_k$, $\mathbf{S}_s(f, g)$ is set to $\sigma_{\max}$ to consider the worst case.

- $\mathbf{S}_u$, $\mathbf{S}_a$, and $\vec{S}_f$: These are devised to incorporate the history of the job, machine, and agent status until s_k, which has been known to be effective for solving scheduling problems based on DRL [20]. First, $\mathbf{S}_u(f, 1)$ and $\mathbf{S}_u(f, 2)$ respectively refer to the amounts of processing and setup time until s_k that have been spent for processing jobs of the family f. $\mathbf{S}_u(f, 3)$ is the number of finished jobs whose family is f. Next, $\mathbf{S}_a$ represents the last action a_{k-1}. By using one-hot encoding [56], $\mathbf{S}_a(\cdot, 1)$ and $\mathbf{S}_a(\cdot, 2)$ denote N_F-dimensional vectors that indicate the family of a job and the setup status of a machine, respectively. Finally, $\vec{S}_f$ consists of the three historic features that cannot be grouped into a specific family. $\vec{S}_f(1)$, $\vec{S}_f(2)$, and $\vec{S}_f(3)$ are respectively equal to r_{k-1}, k, and a binary value that is set to 1 if s_k is terminal, otherwise 0. Figs. 1(e) and (f) provide the following four information: $k = 5$, $r_4 = -1.7$, $a_4 = (1, 2)$, and s_5 is not terminal.

Algorithm 1 State Transition by an Action

Input: $a_k = (f_k, g_k)$, $\mathcal{W}_k = \bigcup_{f'=1}^{N_F} \mathcal{W}_k(f')$
Output: State s_{k+1}, reward r_k, $\mathcal{W}_{k+1}$

```
M ← {M_i | G_i = g_k}, i = 1, ..., N_M
M_* ← arg min_{M_i ∈ M} (Σ_{j'} p_{i,j'}), where J_{j'} ∈ W_k(f_k)
while W_k ≠ ∅ and min_{i'} E_{i'} < (k+1)T do
    M_i ← arg min_{i'} E_{i'}
    g ← G_i
    if M_* = M_i then
        f ← f_k
    else
        f ← arg min_{f'} σ_{f',g}, where |W_k(f')| > 0
    end if
    J_j ← arg min_{J_{j'} ∈ W_k(f)} d_{j'}
    Assign J_j on M_i
    E_i ← E_i + p_{i,j} + σ_{f,g}
    W_k ← W_k \ {J_j}
end while
Obtain transited state s_{k+1}
Calculate r_k from Eq. (8)
```

3) REWARD

The reward proposed in this paper is motivated by [37] in the sense that r_k is calculated by considering job delays which occurred only during the period k. However, since we assumed that the time spent in each period is always equal to T, the reward in [37] is redefined in this paper to accommodate such assumption by using the following clip function.

$$\lambda_k(t) = \max(kT, \min((k+1)T, t)) \tag{7}$$

Then, r_k is defined as follows.

$$r_k = \sum_{j=1}^{N_J} -\max(0, \lambda_k(c_j) - \lambda_k(d_j)) \tag{8}$$

Eq. (8) states that the reward is equivalent to the negative sum of the job tardiness clipped by Eq. (7). When computing r_k, we assumed that c_j is set to be infinite if J_j is waiting until s_{k+1}. As a result, the total sum of rewards is equal to TT, which was proven in [37].

4) STATE TRANSITIONS

After executing a_k, the state transition from s_k to s_{k+1} occurs. Algorithm 1 describes the procedure for the state transition. Let E_i be the time when the current job performed on M_i will be finished.

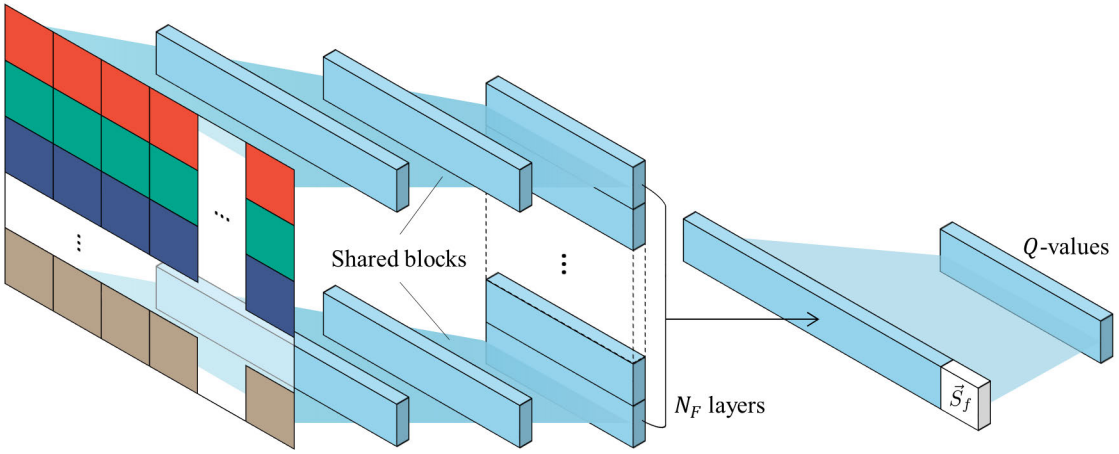

FIGURE 2. The proposed DNN architecture with parameter sharing. The red, green, blue, and brown colors represent different families. The sky color refers to the hidden layers and output layer.

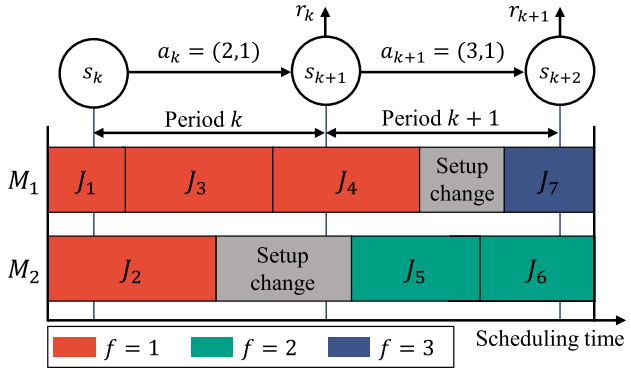

FIGURE 3. An example of a schedule obtained from state transitions. Red, green, and blue colors indicate family 1, 2, and 3, respectively.

In line 1, we obtain a machine set $\mathcal{M}$ that consists of the machines whose setup status is g_k. Among $\mathcal{M}$, we select the machine M_* where the sum of processing time for the jobs in $\mathcal{W}_k(f_k)$ is the shortest (line 2). Lines 3–15 continue until satisfying the following two conditions: there is at least one waiting job, and the minimum of E_i among all machines do not exceed the end time of the current period k. In line 4, the machine M_i whose E_i is the smallest among all machines is selected. Then, the setup status of the machine, called g, is obtained (line 5). In lines 6–10, we determine the family of a next job that will be performed on M_i, denoted as f. If M_* is equal to M_i, f is set to f_k by following a_k (line 7). Otherwise, f is selected to minimize the setup time incurred on M_i (Line 9). Among waiting jobs whose family is f, the job J_j with the earliest due-date is selected (line 11). After J_j is assigned on M_i (line 12), both E_i and $\mathcal{W}_k$ are updated (lines 13 and 14). Finally, s_{k+1}, $\mathcal{A}_{k+1}$, and r_k are obtained (lines 16 and 17).

5) EXAMPLE

Fig. 3 depicts a schedule that is built during the periods k and $k+1$. At s_k, M_1 and M_2 are respectively performing J_1 and J_2, and the jobs J_3 to J_7 are waiting for being assigned. Since a_k is equal to (2, 1), a job of the family 2 should be assigned on a machine whose setup status is 1. By following lines 2 and 11 in Algorithm 1, J_5 is assigned on M_2, and $\sigma_{2,1}$ is incurred before processing the job. Meanwhile, J_3 and J_4, which respectively belong to family 1, are consecutively allocated to M_1 whose G_1 is 1. At the end of the period k, s_{k+1} and r_k are obtained as the consequences of the state transition. After executing $a_{k+1} = (1, 3)$, the job J_7 of family 3 is allocated on M_1 whose setup status is 1. In summary, J_3, J_4, and J_5 are scheduled according to a_k and J_6 and J_7 according to a_{k+1}.

B. DNN ARCHITECTURE

A deep Q-network (DQN) was employed to estimate a Q-value given a state [17]. DQN takes a state s as an input and outputs Q-values for all possible actions, called $Q_\theta(s, a)$, where θ is the network parameters and $a \in \mathcal{A}$. Fig. 2. depicts the proposed fully-connected network architecture with parameter sharing that has been adopted for solving single lot-sizing [57], and job scheduling problems in computing platforms [22].

The input of DQN is equal to a matrix that is constructed by concatenating the five matrices $\mathbf{S}_w$, $\mathbf{S}_p$, $\mathbf{S}_s$, $\mathbf{S}_u$, and $\mathbf{S}_a$. Each row vector of the input is connected to a block that consists of several hidden layers. To reduce the network parameter size and increase the training efficiency, the parameters for each block are set to be the same. The last hidden layer is composed by concatenating the values in the last layers of N_F blocks and $\vec{S}_f$. Finally, the number of nodes in the output layer is equal to the number of all possible actions. The ReLU function [58] was adopted as an activation function except for the output layer to represent negative Q-values.

C. TRAINING DQN

Algorithm 2 describes the training procedure of the proposed DQN. Given a scheduling problem for training DQN, a scheduling process is repeated until there is no waiting job (lines 3–16). We refer to the completion of one scheduling

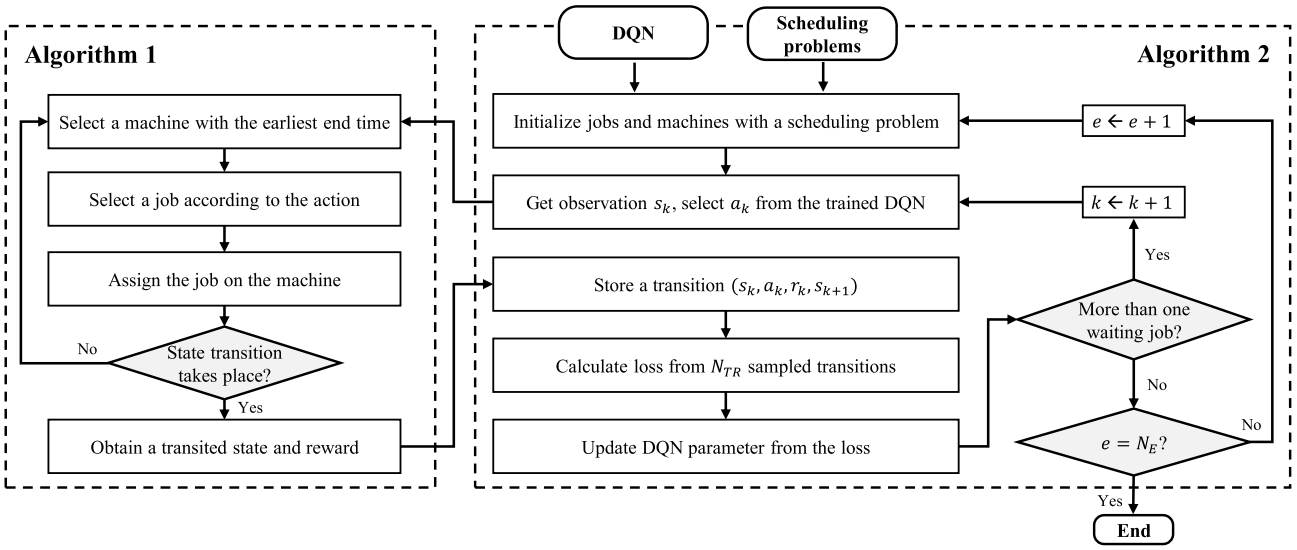

FIGURE 4. A flowchart of the proposed algorithms.

Algorithm 2 DQN Training Procedure

Input: Scheduling problem
Output: Q-network

1: **Initialization**: Set network Q_θ with random weight θ, target network $Q_{\hat{\theta}}$ with $\hat{\theta} = \theta$, and replay buffer B to size N_B.
2: **for** $e = 1, 2, \ldots, N_E$ **do**
3: $k \leftarrow 0$
4: Initialize F_j, d_j of N_J jobs, and status of N_M machines.
5: Observe s_k and $\mathcal{W}_k$
6: **while** $\mathcal{W}_k \neq \emptyset$ **do**
7: With probability ε select a_k randomly from $\mathcal{A}_k$
8: otherwise $a_k \leftarrow \arg\max_{a \in \mathcal{A}_k} Q(s_k, a)$
9: Get s_{k+1}, $\mathcal{A}_{k+1}$, r_k, $\mathcal{W}_{k+1}$ from **Algorithm 1**
10: Store transition (s_k, a_k, r_k, s_{k+1}) in B
11: Sample N_{TR} transitions $(s_u, a_u, r_u, s_{u+1}) \in B$
12: Calculate loss L from (9)-(11)
13: Perform a gradient descent step on L w.r.t. θ
14: $k \leftarrow k + 1$
15: **end while**
16: Synchronize $\hat{\theta}$ to θ at every N_U episodes
17: **end for**
18: **return** Q-network

process as an episode, and e indicates the index of the episode currently being performed. The scheduling processes continue until e reaches the number of training episodes, denoted as N_E.

At the start of an episode, k is set to 0, and N_J jobs and N_M machines are initialized (lines 3 and 4). In line 5, the agent initially observes the s_k with $\mathcal{W}_k$. For each timestep k, the agent selects a_k from the ε-greedy policy [36] presented in lines 7 and 8, where $\varepsilon \in [0, 1]$ is a probability to select an random action. After the state transition takes place from

TABLE 2. Dataset.

Dataset No.	N_M	N_J	N_F	τ
1	20	420	10	0.4
2	20	420	10	0.5
3	20	420	7	0.4
4	20	420	7	0.5
5	50	1050	10	0.4
6	50	1050	10	0.5
7	50	1050	7	0.4
8	50	1050	7	0.5

s_k to s_{k+1} (line 9), the transition, represented as a quadruple of state, action, reward, and next state, is stored in replay buffer (line 10). The replay buffer and its size are denoted as B and N_B, respectively. If the number of stored transitions in B exceeds N_B, the oldest ones are removed. Line 11 indicates that N_{TR} transitions are sampled to train DQN, where N_{TR} is the number of sampled transitions. Given a transition (s_u, a_u, r_u, s_{u+1}), the temporal difference error, called η_u, are calculated by the prediction $Q_\theta(s, a)$ and target Q-value [17] as follows.

$$\eta_u = r_u + \gamma \max_{a' \in \mathcal{A}_{u+1}} Q_{\hat{\theta}}(s_{u+1}, a') - Q_\theta(s_u, a_u) \tag{9}$$

where γ and $\hat{\theta}$ respectively indicate the discount factor [36] and the parameters of a target DQN which has the same network architecture as DQN for training. In Eq. (9), $r_u + \gamma \max_{a'} Q_{\hat{\theta}}(s_{u+1}, a')$ indicates a target Q-value which is set to r_u when s_{u+1} is terminal. We denote the temporal difference loss from sampled transitions as $L(\theta)$, which is defined as follows.

$$L(\theta) = \frac{1}{N_{TR}} \sum_{u=1}^{N_{TR}} h(\eta_u) \tag{10}$$

where h is the loss function, defined as below.

$$h(\eta) = \begin{cases} \frac{1}{2}\eta^2 & \text{if } \eta \leq \frac{1}{2}, \\ \frac{1}{2}(|\eta| - (\frac{1}{2})^2) & \text{otherwise.} \end{cases} \tag{11}$$

In Eq. (11), we adopted Huber loss [59] instead of mean-squared error for further enhancing the stability of DQN training [20]. By calculating $L(\theta)$, the network parameter θ is updated (lines 12 and 13). Line 16 shows that the target network parameter $\hat{\theta}$ is periodically replaced to the θ [17]. Finally, the trained DQN is acquired at the end of N_E episodes (line 18). Fig. 4 depicts the overall flowchart of the proposed algorithms.

After DQN training, a test procedure is implemented to solve the scheduling problems whose production requirements and due-dates change from those of the training problems. During the test procedure, random actions (line 8 in Algorithm 2) are not executed any more. The rest of the procedure is identical to Algorithm 2 except the lines 10–13 and 16 that are required to train a DQN.

V. EXPERIMENTS

A. DATASETS

We prepared 8 datasets that simulated the semiconductor wafer preparation facilities in South Korea. Table. 2 presents N_M, N_J, N_F, and due-date tightness, denoted as τ, for the scheduling problems in each dataset. Except for τ, datasets 1 and 3 are equivalent to datasets 2 and 4, respectively. Moreover, N_M and N_J of datasets 5–8 are 2.5 times larger than those of datasets 1–4, respectively.

Each dataset has 330 different scheduling problems which are divided into 300 problems for training the proposed DQN, and the other problems for validating the trained DQN. Fig. 5 depicts the distributions of production requirements for each family. In particular, across all datasets, production requirements were perturbed by at least 37%.

For each scheduling problem, d_j of N_J jobs were set to be uniformly distributed between $L(0.5 - \tau)$ and $L(1.5 - \tau)$, where L indicates an expected makespan adopted in several studies [9], [11], [30]. To simulate the real-world scenario, each machine performs a job at the beginning of the scheduling, where its remaining processing time was randomly generated.

B. EXPERIMENTAL SETTINGS

Experiments were conducted on a Xeon E5 2.2-GHz PC with 126-GB memory. When employing DRL-based methods, choosing appropriate values of hyperparameters plays a crucial role in the performance of DNN. Since it is challenging to determine optimal values of hyperparameters due to their huge search space, we implemented the random search [60], and found the values that achieved the best performance.

For performing a gradient descent step, we adopted RMSProp optimizer [61] where the learning rate is set to 2.5×10^{-3}. In the ε-greedy policy, the initial value of ε is set to 0.2. The value decays linearly to zero until e reaches $0.9 \times N_E$. Besides, N_B, N_U, N_{TR}, and γ is set to 10^5, 50, 64, and 1, respectively. Finally, we set $N_E = 10^5$, $T = \frac{2}{3}\bar{p}$ in datasets 1–4, and $N_E = 1.5 \times 10^4$, $T = \frac{1}{2}\bar{p}$ in datasets 5–8, respectively, where $\bar{p}$ is the average processing time over all pairs of jobs and machines. Each shared block consists of three hidden layers where the numbers of nodes in the first, second, and third layers are 64, 32, and 16, respectively. For each dataset, after the trained DQNs were stored when e is equal to $0.91 \times N_E$, $0.92 \times N_E$, …, $0.99 \times N_E$, and N_E, respectively, they were used for performance comparisons.

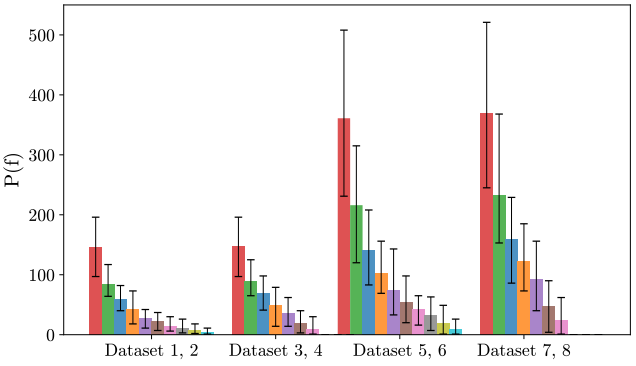

FIGURE 5. The ranges of variability in production requirements for 8 datasets.

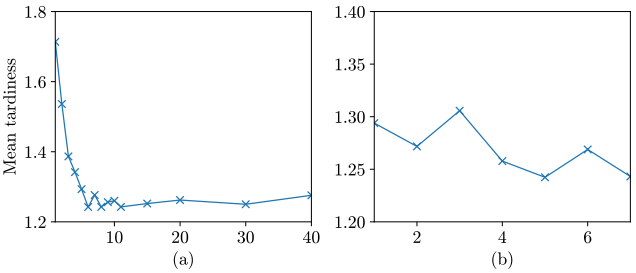

FIGURE 6. Mean tardiness results with changes in H_w and H_p on dataset 1. (a) H_w with $H_p = 5$. (b) H_p with $H_w = 6$.

Figs. 6(a) and (b) illustrate the mean tardiness results (in hours) on dataset 1 by varying H_w and H_p, respectively. In the experiments, the mean tardiness is calculated by dividing TT to N_J. We note that the values of H_w and H_p were chosen in the range described in Eqs. (4) and (5), respectively. In Fig. 6(a), TT significantly decreased until H_w reaches 6, which reveals the validity of $\mathbf{S}_w$. On the other hand, the performance changes were negligible when H_w exceeds 6. This implies that the advantage of classifying waiting jobs in terms of their due-dates is diminished due to the increase in the dimension of $\mathbf{S}_w$. As shown in Fig. 6(b), the performance improvement achieved by varying H_p was less significant than varying H_w. This can be attributed to the fact that $\mathbf{S}_w$ contains more observations than $\mathbf{S}_p$ since the number of waiting jobs is larger than those of in-progress jobs in most periods. As a result, the rest of the experiments were carried out with the best values of H_w and H_p, which were 6 and 5, respectively.

TABLE 3. Mean tardiness results of the proposed method and the other methods. Bold marks indicate the best results among all methods for each dataset.

Dataset No.	BATCS	SSTEDD	LSR	COVERT	IG	LBF-Q	TPDQN	Ours
1	9.716	8.026	78.116	28.105	1.744	9.079	26.849	**1.243**
2	10.660	9.013	80.516	29.952	2.867	11.339	29.265	**2.490**
3	9.339	7.236	77.589	24.218	1.585	10.191	27.021	**1.196**
4	10.300	8.242	79.989	25.967	2.719	11.340	27.916	**2.404**
5	8.625	7.942	78.925	29.056	2.440	8.763	29.533	**1.796**
6	9.635	8.887	81.325	31.181	3.520	9.527	32.557	**2.934**
7	8.151	7.040	78.006	23.701	2.440	7.322	30.752	**1.572**
8	9.178	8.027	80.406	25.445	3.669	10.486	24.402	**2.724**

C. PERFORMANCE COMPARISON

In order to show the effectiveness of the proposed method, we compared performances of IG in [11] and two RL-based methods, which are two-phase DQN (TPDQN) [18], and QL method with linear basis functions (LBF-Q) [37], respectively. The parameters of IG were the same as those in [11]. Since the production scheduling in the semiconductor manufacturing systems is usually carried out on an hourly basis [14], IG was terminated after an hour. For LBF-Q and TPDQN, ten models were stored as described in Section V-B for performance comparisons, respectively. The rest of the hyperparameters were the same as those in [18], [37], respectively.

Furthermore, we made comparisons between our method and four rule-based methods: BATCS [31], shortest setup time with earliest due date (SSTEDD), least slack remaining (LSR) [13], COVERT [62]. LSR and COVERT are widely adopted to minimize due-date related objectives, while BATCS is effective when solving scheduling problems with SDFST. In particular, SSTEDD selects the jobs which require the shortest setup time and decides a job with the earliest due-date among those jobs.

Table. 3 presents the mean tardiness results (in hours) of ours and the other methods. Among the rule-based methods, SSTEDD outperformed the other methods in all datasets. Meanwhile, LSR and COVERT yielded 3.2 times longer TT than the other rule-based methods in the best case and 11.3 times in the worst case. It can be said that addressing sequence-dependent setups is crucial for minimizing TT of the scheduling problems considered. It was observed that TT achieved by LBF-Q and TPDQN was longer than SSTEDD for all datasets. This may be due to the fact that the family setups were not accommodated in their state and action representations. Although the performances of IG were better than those of the rule-based and other RL-based methods for all datasets, TT achieved by IG was 13% to 55% longer than those of the proposed method. Based on these results, the proposed method appears to be effective for solving scheduling problems even when the production requirements and due-dates are changed from those of the training.

TABLE 4. Computation time results (in seconds) of SSTEDD, IG, LBF-Q, TPDQN, and the proposed method.

Dataset No.	SSTEDD	IG	LBF-Q	TPDQN	Ours
1	0.366	3600	12.428	37.845	4.331
2	0.370	3600	12.930	40.360	4.450
3	0.369	3600	12.270	30.870	3.260
4	0.354	3600	13.010	32.160	3.420
5	2.238	3600	65.417	238.633	17.603
6	2.347	3600	73.210	248.780	15.640
7	2.189	3600	70.000	198.620	14.570
8	2.069	3600	73.020	194.020	13.780

Table. 4 presents the average computation time taken by our method, TPDQN, LBF-Q, IG, and SSTEDD. We only presented the results of SSTEDD whose average computation time is the shortest among the four rule-based methods. The computation time results of LBF-Q and TPDQN were longer than those of the proposed method for all datasets. This might be related to the fact that Q-values are computed by the proposed method at each period, different from LBF-Q and TPDQN that compute Q-values whenever allocating a job to a machine. Moreover, the ratio of average computation time for the datasets 5–8 to that for datasets 1–4 was 5.56 for LBF-Q, 6.23 for TPDQN, and 3.98 for the proposed method, respectively. This observation can be attributed to the fact that the number of parameters for the proposed DQN is independent of N_M and N_J, different from LBF-Q and TPDQN.

Compared to the best rule, the computation time of the proposed method was increased by 8.8 to 12 times for the datasets 1–4, and 6.7 to 7.9 times for the datasets 5–8, respectively. Nevertheless, the results demonstrate that the proposed method built a schedule less than 20s for all datasets. Different from metaheuristics and rule-based methods, the proposed DRL-based method can quickly obtain a new schedule by using the trained DQN, which suggests the viability of the proposed method in terms of the computation time for real-world manufacturing systems with parallel machines.

To examine the robustness of the proposed method when both processing and setup time is stochastic, we carried out

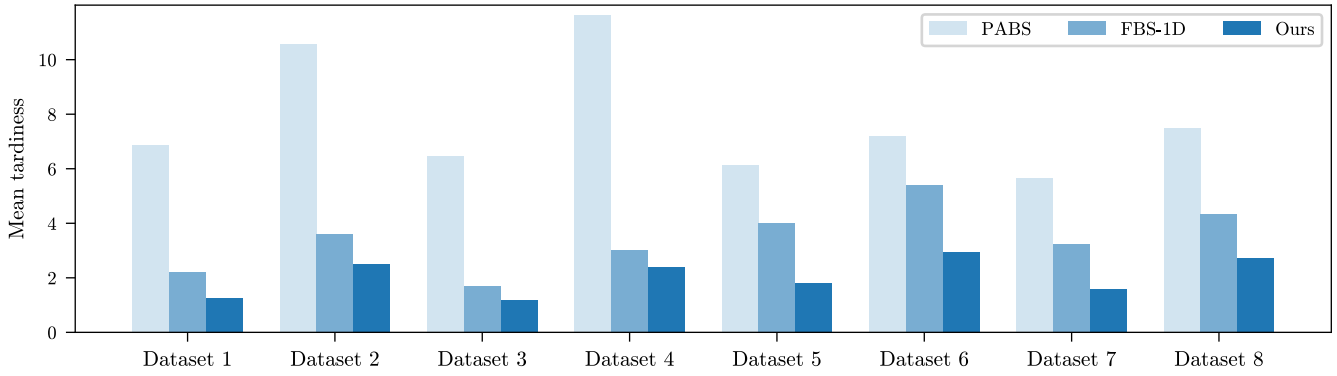

FIGURE 7. Mean tardiness results of PABS, FBS-1D, and ours across all datasets.

TABLE 5. The mean tardiness (in hours) of SSTEDD, IG, LBF-Q, and the proposed method on datasets 1 and 5.

Dataset No.	SSTEDD	IG	LBF-Q	Ours
1	8.013 ± 0.130	1.829 ± 0.118	9.014 ± 0.514	1.235 ± 0.082
5	7.929 ± 0.086	2.514 ± 0.074	8.816 ± 0.276	1.764 ± 0.078

additional performance comparisons. Specifically, both $p_{i,j}$ and $\sigma_{f,g}$ were not known in advance and set to be uniformly distributed between $[0.8p_{i,j}, 1.2p_{i,j}]$ and $[0.8\sigma_{f,g}, 1.2\sigma_{f,g}]$, respectively. Each test scheduling problems was solved 30 times with different random seeds.

Table. 5 shows the average and standard deviation of the mean tardiness (in hours) for SSTEDD, IG, LBF-Q, and our method. For datasets 1 and 5, both the average and standard deviation of TT yielded by the proposed method were the lowest among all methods. Based on these results, the proposed method seems to be robust even when processing and setup time are stochastic.

We further analyzed the effectiveness of the proposed state representation and parameter sharing architecture. To solely investigate the performance improvements achieved by the state representation and parameter sharing, we compared the proposed method with two modified baseline methods. First, we adopted production attributes-based state representation (PABS) proposed in [37]. Next, the proposed family-based state representation was utilized as one-dimensional vector (FBS-1D) by flattening and concatenating $\mathbf{S}_w$, $\mathbf{S}_p$, $\mathbf{S}_s$, $\mathbf{S}_u$, $\mathbf{S}_a$, and $\vec{S}_f$. For PABS and FBS-1D, we utilized the fully-connected network with three hidden layers where the number of nodes were the same as mentioned in Section. IV-B. The rest of the details are equivalent to the proposed method. Note that PABS and FBS-1D were the same except for the state representation.

Fig. 7 highlights the mean tardiness results (in hours) of PABS, FBS-1D, and ours in datasets 1–8. For all datasets, FBS-1D consistently outperformed PABS with respect to TT. Specifically, TT achieved by FBS-1D was 70% lower than those of PABS in datasets 1–4 and 34% lower for datasets 5–8, respectively, which demonstrate the superiority of the proposed state representation. Furthermore, compared to FBS-1D, the performances of the proposed method were 30% better for datasets 1–4 and 47% in datasets 5–8, respectively. Based on the above observations, parameter sharing appears to be more efficient for solving scheduling problems with large numbers of jobs and machines.

VI. CONCLUSION

In this paper, we proposed a DRL-based method for solving UPMSPs with SDFST constraint to minimize the total tardiness. To cope with the variabilities in production requirements and due-dates while addressing SDFST, we proposed a novel state representation whose dimension is invariant to such variabilities. Furthermore, we suggested the parameter sharing architecture to learn DQN parameters effectively and reduce the parameter size. As a result, the trained DQN was able to quickly solve unseen scheduling problems whose production requirements and due-dates are different from those considered in training.

To examine the performance of the trained DQN, the proposed method was compared to IG, four rule-based methods, LBF-Q, and TPDQN. The experimental results demonstrated that our method outperformed the existing methods in terms of the total tardiness for all datasets. Moreover, the computation time of the proposed method was shorter than those of IG and two RL-based methods. Through further experiments, it was verified that both the proposed state representation and parameter sharing architecture have contributed to the performance improvements.

Yet, the proposed DQN poses a limitation when the number of families changes since a re-training procedure is required. To address such limitation, we plan to develop the state and action whose dimensions are independent of the number of families. Furthermore, some assumptions made in this paper related to the ready time of jobs and machine breakdown will be relaxed in the future work. Finally, the network architecture can be improved by utilizing other deep learning models, such as recurrent DNNs, and convolutional neural networks.

REFERENCES

[1] A. Allahverdi, "The third comprehensive survey on scheduling problems with setup times/costs," *Eur. J. Oper. Res.*, vol. 246, no. 2, pp. 345–378, 2015.

[2] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, "A reinforcement learning approach to dynamic scheduling in a product-mix flexibility environment," *IEEE Access*, vol. 8, pp. 106542–106553, 2020.
[3] T. Yang, Y.-F. Wen, Z.-R. Hsieh, and J. Zhang, "A lean production system design for semiconductor crystal-ingot pulling manufacturing using hybrid taguchi method and simulation optimization," *Assem. Autom.*, vol. 40, no. 3, pp. 433–445, Jan. 2020.
[4] W. L. Pearn, S. H. Chung, and M. H. Yang, "The wafer probing scheduling problem (WPSP)," *J. Oper. Res. Soc.*, vol. 53, no. 8, pp. 864–874, Aug. 2002.
[5] L. Mönch, J. W. Fowler, and S. J. Mason, "Semiconductor manufacturing process description," in *Production Planning and Control for Semiconductor Wafer Fabrication Facilities*. New York, NY, USA: Springer, 2013, pp. 11–28.
[6] L. Shen, L. Mönch, and U. Buscher, "An iterative approach for the serial batching problem with parallel machines and job families," *Ann. Oper. Res.*, vol. 206, no. 1, pp. 425–448, 2013.
[7] C. A. Sáenz-Alanís, V. D. Jobish, M. A. Salazar-Aguilar, and V. Boyer, "A parallel machine batch scheduling problem in a brewing company," *Int. J. Adv. Manuf. Technol.*, vol. 87, nos. 1–4, pp. 65–75, 2016.
[8] J. Du and J. Y.-T. Leung, "Minimizing total tardiness on one machine is NP-hard," *Math. Oper. Res.*, vol. 15, no. 3, pp. 483–495, Aug. 1990.
[9] J.-F. Chen, "Scheduling on unrelated parallel machines with sequence- and machine-dependent setup times and due-date constraints," *Int. J. Adv. Manuf. Technol.*, vol. 44, nos. 11–12, pp. 1204–1212, Oct. 2009.
[10] S.-W. Lin, C.-C. Lu, and K.-C. Ying, "Minimization of total tardiness on unrelated parallel machines with sequence- and machine-dependent setup times under due date constraints," *Int. J. Adv. Manuf. Technol.*, vol. 53, nos. 1–4, pp. 353–361, Mar. 2011.
[11] J. C. S. N. Pinheiro, J. E. C. Arroyo, and L. B. Fialho, "Scheduling unrelated parallel machines with family setups and resource constraints to minimize total tardiness," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2020, pp. 1409–1417.
[12] K.-C. Ying and S.-W. Lin, "Unrelated parallel machine scheduling with sequence-and machine-dependent setup times and due date constraints," *Int. J. Innov. Comput., Inf. Control*, vol. 8, no. 5, 2012, pp. 3279–3297.
[13] Y. H. Lee, K. Bhaskaran, and M. Pinedo, "A heuristic to minimize the total weighted tardiness with sequence-dependent setups," *IIE Trans.*, vol. 29, no. 1, pp. 45–52, Jan. 1997.
[14] J. Lim, M.-J. Chae, Y. Yang, I.-B. Park, J. Lee, and J. Park, "Fast scheduling of semiconductor manufacturing facilities using case-based reasoning," *IEEE Trans. Semicond. Manuf.*, vol. 29, no. 1, pp. 22–32, Feb. 2016.
[15] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *Proc. IJCAI*, vol. 95, 1995, pp. 1114–1120.
[16] T. Gabel and M. Riedmiller, "Adaptive reactive job-shop scheduling with reinforcement learning agents," *Int. J. Inf. Technol. Intell. Comput.*, vol. 24, no. 4, pp. 14–18, 2008.
[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and S. Petersen, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
[18] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmuller, T. Bauernhansl, A. Knapp, and A. Kyek, "Deep reinforcement learning for semiconductor production scheduling," in *Proc. 29th Annu. SEMI Adv. Semiconductor Manuf. Conf. (ASMC)*, Apr. 2018, pp. 301–306.
[19] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, vol. 8, pp. 71752–71762, 2020.
[20] I.-B. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 3, pp. 1420–1431, Jul. 2020.
[21] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Appl. Soft Comput.*, vol. 91, Jun. 2020, Art. no. 106208.
[22] D. Zhang, D. Dai, Y. He, F. S. Bao, and B. Xie, "RLScheduler: An automated HPC batch job scheduler using reinforcement learning," 2019, *arXiv:1910.08925*. [Online]. Available: http://arxiv.org/abs/1910.08925
[23] A. Kramer and A. Subramanian, "A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems," *J. Scheduling*, vol. 22, no. 1, pp. 21–57, Feb. 2019.
[24] M. A. Bozorgirad and R. Logendran, "Sequence-dependent group scheduling problem on unrelated-parallel machines," *Expert Syst. Appl.*, vol. 39, no. 10, pp. 9021–9030, 2012.
[25] O. Shahvari and R. Logendran, "An enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes," *Comput. Oper. Res.*, vol. 77, pp. 154–176, Jan. 2017.
[26] A. Ekici, M. Elyasi, O. Ö. Özener, and M. B. Sarıkaya, "An application of unrelated parallel machine scheduling with sequence-dependent setups at vestel electronics," *Comput. Oper. Res.*, vol. 111, pp. 130–140, Nov. 2019.
[27] J. R. Zeidi and S. MohammadHosseini, "Scheduling unrelated parallel machines with sequence-dependent setup times," *Int. J. Adv. Manuf. Technol.*, vol. 81, nos. 9–12, pp. 1487–1496, 2015.
[28] C. N. Potts and M. Y. Kovalyov, "Scheduling with batching: A review," *Eur. J. Oper. Res.*, vol. 120, no. 2, pp. 228–249, 2000.
[29] R. H. Suriyaarachchi and A. Wirth, "Earliness/tardiness scheduling with a common due date and family setups," *Comput. Ind. Eng.*, vol. 47, nos. 2–3, pp. 275–288, Nov. 2004.
[30] D.-W. Kim, D.-G. Na, and F. F. Chen, "Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective," *Robot. Comput.-Integr. Manuf.*, vol. 19, nos. 1–2, pp. 173–181, Feb. 2003.
[31] S. J. Mason, J. W. Fowler, and W. M. Carlyle, "A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops," *J. Scheduling*, vol. 5, no. 3, pp. 247–262, 2002.
[32] O. L. V. Costa and M. D. Fragoso, "Discrete-time LQ-optimal control problems for infinite Markov jump parameter systems," *IEEE Trans. Autom. Control*, vol. 40, no. 12, pp. 2076–2088, Dec. 1995.
[33] J. Wang, "H_∞ synchronization for fuzzy Markov jump chaotic systems with piecewise-constant transition probabilities subject to PDT switching rule," *IEEE Trans. Fuzzy Syst.*, early access, Jul. 29, 2020, doi: 10.1109/TFUZZ.2020.3012761.
[34] H. Shen, M. Dai, Y. Luo, J. Cao, and M. Chadli, "Fault-tolerant fuzzy control for semi-Markov jump nonlinear systems subject to incomplete SMK and actuator failures," *IEEE Trans. Fuzzy Syst.*, early access, Jul. 24, 2020, doi: 10.1109/TFUZZ.2020.3011760.
[35] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.
[36] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA, USA: MIT Press, 2018.
[37] Z. Zhang, L. Zheng, and M. X. Weng, "Dynamic parallel machine scheduling with mean weighted tardiness objective by Q-learning," *Int. J. Adv. Manuf. Technol.*, vol. 34, nos. 9–10, pp. 968–980, Oct. 2007.
[38] Z. Zhang, L. Zheng, N. Li, W. Wang, S. Zhong, and K. Hu, "Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning," *Comput. Oper. Res.*, vol. 39, no. 7, pp. 1315–1324, Jul. 2012.
[39] B. Yuan, L. Wang, and Z. Jiang, "Dynamic parallel machine scheduling using the learning agent," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manage.*, Dec. 2013, pp. 1565–1569.
[40] B. Yuan, Z. Jiang, and L. Wang, "Dynamic parallel machine scheduling with random breakdowns using the learning agent," *Int. J. Services Oper. Informat.*, vol. 8, no. 2, pp. 94–103, 2016.
[41] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, "Real-time scheduling for a smart factory using a reinforcement learning approach," *Comput. Ind. Eng.*, vol. 125, pp. 604–614, Nov. 2018.
[42] 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, Aug. 2017.
[43] Y.-F. Wang, "Adaptive job shop scheduling strategy based on weighted Q-learning algorithm," *J. Intell. Manuf.*, vol. 31, no. 2, pp. 417–432, Feb. 2020.
[44] Y. Cheng, J. Peng, X. Gu, F. Jiang, H. Li, W. Liu, and Z. Huang, "Optimal energy management of energy internet: A distributed actor-critic reinforcement learning method," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2020, pp. 521–526.
[45] M. Nazari, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, Feb. 2018, pp. 9839–9849.
[46] Y. Ye, D. Qiu, J. Li, and G. Strbac, "Multi-period and multi-spatial equilibrium analysis in imperfect electricity markets: A novel multi-agent deep reinforcement learning approach," *IEEE Access*, vol. 7, pp. 130515–130529, 2019.
[47] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 50–56.

[48] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2019, pp. 505–513, doi: 10.1109/INFOCOM.2019.8737460.
[49] A. Mirhoseini, "Device placement optimization with reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2430–2439.
[50] Z. Cao, H. Zhang, Y. Cao, and B. Liu, "A deep reinforcement learning approach to multi-component job scheduling in edge computing," in *Proc. 15th Int. Conf. Mobile Ad-Hoc Sensor Netw. (MSN)*, Dec. 2019, pp. 19–24.
[51] D. Shi, "Intelligent scheduling of discrete automated production line via deep reinforcement learning," *Int. J. Prod. Res.*, vol. 58, pp. 1–19, Jun. 2020.
[52] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep Q network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4276–4284, Jul. 2019.
[53] B.-A. Han and J.-J. Yang, "Research on adaptive job shop scheduling problems based on dueling double DQN," *IEEE Access*, vol. 8, pp. 186474–186495, 2020.
[54] T. E. Thomas, J. Koo, S. Chaterji, and S. Bagchi, "MINERVA: A reinforcement learning-based technique for optimal scheduling and bottleneck detection in distributed factory operations," in *Proc. 10th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2018, pp. 129–136.
[55] S. Zheng, C. Gupta, and S. Serita, "Manufacturing dispatching using reinforcement and transfer learning," 2019, *arXiv:1910.02035*. [Online]. Available: http://arxiv.org/abs/1910.02035
[56] D. Harris and S. Harris, *Digital Design and Computer Architecture*. San Mateo, CA, USA: Morgan Kaufmann, 2010.
[57] H. Rummukainen and J. K. Nurminen, "Practical reinforcement learning-experiences in lot scheduling application," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1415–1420, 2019.
[58] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.
[59] P. J. Huber, "Robust estimation of a location parameter," *Ann. Math. Statist.*, vol. 35, no. 1, pp. 73–101, 1964.
[60] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.
[61] T. Tieleman and G. Hinton, "Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude," *COURSERA, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
[62] 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, 1987.

BOHYUNG PAENG received the B.S. degree in electrical engineering from KAIST, South Korea, in 2013. He is currently pursuing the Ph.D. degree with the Information Management Laboratory, Department of Industrial Engineering, Seoul National University, South Korea. His current research interests include scheduling, deep reinforcement learning, and deep learning applications.

IN-BEOM PARK received the Ph.D. degree in industrial engineering from Seoul National University, Seoul, South Korea, in 2020. He is currently a Postdoctoral Researcher in industrial engineering with Sungkyunkwan University. His current research interests include scheduling manufacturing systems, machine learning, and deep reinforcement learning.

JONGHUN PARK received the Ph.D. degree in industrial and systems engineering with a minor in computer science from the Georgia Institute of Technology, Atlanta, in 2000. He is currently a Professor with the Department of Industrial Engineering, Seoul National University (SNU), South Korea. Before joining SNU, he was an Assistant Professor with the School of Information Sciences and Technology, Pennsylvania State University, University Park, and the Department of Industrial Engineering, KAIST, Daejeon. His research interests include generative artificial intelligence and deep learning applications.