Smart Manufacturing Scheduling with Edge Computing Using Multiclass Deep Q Network

Chun-Cheng Lin, Der-Jiunn Deng*, Yen-Ling Chih, and Hsin-Ting Chiu

Abstract—**Manufacturing is involved with complex job shop scheduling problems (JSP). In smart factories, edge computing supports computing resources at the edge of production in a distributed way to reduce response time of making production decisions. However, most works on JSP did not consider edge computing. Therefore, this work proposes a smart manufacturing factory framework based on edge computing, and further investigates the JSP under such a framework. With recent success of some AI applications, the deep Q network (DQN), which combines deep learning and reinforcement learning, has showed its great computing power to solve complex problems. Therefore, we adjust the DQN with an edge computing framework to solve the JSP. Different from the classical DQN with only one decision, this work extends the DQN to address the decisions of multiple edge devices. Simulation results show that the proposed method performs better than the other methods using only one dispatching rule.**

Index Terms—**Smart manufacturing, job shop scheduling, deep Q network, edge computing, multiple dispatching rules.**

I. Introduction

SEMICONDUCTOR manufacturing is of high investment, high risk, and high payoff. Hence, it has been one of the most competitive industries in the world. Semiconductor manufacturing can be regarded as a job shop industry with long manufacturing process. Job shop scheduling problem (JSP) is a classical NP-hard problem [1] with numerous applications. Consider a set of *jobs*, each of which has a set of *tasks* needed to be processed by *machines* in a specific order. The JSP is to determine the starting time when each task of each job is processed by the corresponding machine so that some objectives (e.g., makespan and average flow time) are optimized. In practice, the factory staff base on their previous experiences to manually choose one of multiple commonly-used dispatching rules (e.g., FIFO, i.e., the job that comes first is served first) to handle JSPs of various manufacturing processes.

This work was supported in part by Grants MOST 106-2221-E-009-101-MY3 and MOST 105-2628-E-009-002-MY3.

C.-C. Lin, Y.-L. Chih, and H.-T. Chiu are with Department of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 300, Taiwan. E-mails: cclin321@nctu.edu.tw, sphere.c7@gmail.com, min850305@gmail.com

D.-J. Deng is with Department of Computer Science and Information Engineering, National Changhua University of Education, Changhua 500, Taiwan. E-mail: djdeng@cc.ncue.edu.tw

* D.-J. Deng is the corresponding author of this paper.

As the era of Industry 4.0 has arrived, smart factory can automatically/autonomously make decisions on complex and real-time manufacturing/maintenance/logistics decisions through advanced ICT technologies, including cyber-physical systems, human-machine collaboration, robotics, fog/edge computing, Internet of Things (IoT), big data analysis, artificial intelligence (AI), and so on. Among them, edge computing is a key technique to achieve smart factories, e-health [2], smart cities [3], and connected vehicles [4], [5]. Conventionally, to protect business secret or factory security, some manufacturing companies only allow to connect closed networks in the factories, and hence deploy a centralized cloud center within a factory to cope with a huge amount of manufacturing and logistics decisions so that a lot of response latency may be produced. Along the trend pushing computation from the network core to the edge where most manufacturing data (from an enormous number of mobile devices [6] and industrial IoT devices attached to facilities and robots in the factory) is generated, edge computing allocates a lot of computing resources to the edge of the production to reduce response time, lower bandwidth usage, improve energy efficiency, and ensure data safety and privacy [7].

This work proposes a smart semiconductor manufacturing factory based on an edge computing framework, as illustrated in Fig. 1. This factory has a large number of manufacturing machines, each of which is in charge of a manufacturing process. For instance, a factory in TSMC has at least one thousand machines, and a product generally involves about one hundred machines. Each machine is monitored and controlled by an edge device, which collects and preprocesses the information and system features of the controlled machine through IoT devices, and transmits them to a centralized cloud center. Then, the cloud center bases on the collected information, historical experiences, as well as the current whole conditions of the factory to make decisions, which are then sent to edge devices to make a detailed scheduling plan according to these decisions.

This work focuses on addressing the JSP under the smart factory framework as illustrated in Fig. 1. Most of the recent works on solving JSP adopted metaheuristic algorithms, e.g., genetic algorithm [8], particle swarm optimization [9], simulated annealing [10], and ant colony optimization [11]. Recently, advanced AI technique has been successfully applied in various fields with the success in machine learning, e.g., deep learning (DL) [12] and reinforcement learning (RL).

Previous works have adopted RL methods to solve the JSP [14]. RL methods have been shown to be a potential for efficiently finding high-quality solutions for scheduling problems [13]. In addition, neural networks (NN) [15] or the hybrid methods of NN and GA [16] were proposed to solve the JSP.

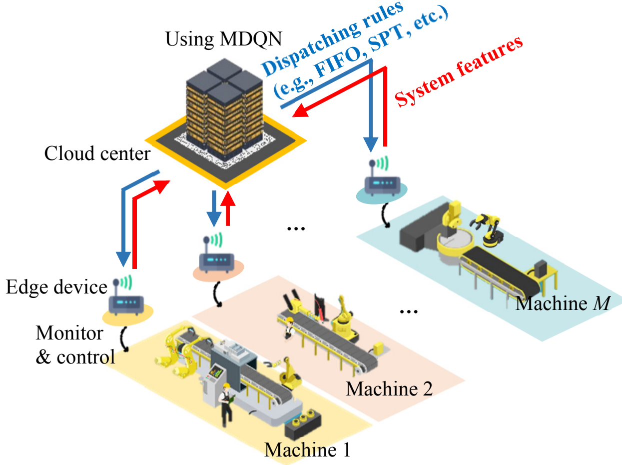

Fig. 1. Illustration of a smart semiconductor manufacturing factory based on an edge computing framework.

In recent years, with the mature of RL and NN techniques, deep Q network (DQN) [17], [18], which combines DL and RL, has successfully solved various practical problems. Especially, owing to the success of AlphaGo based on DQN [19], DQN has received a lot of attention. In the previous literature, DQN was mainly adopted to the applications in playing Atari games [20], and recently has been applied to solve JSPs [21]. Therefore, this work adopts DQN to solve the JSP under the proposed smart factory framework based on edge computing.

The main contributions of this work are listed as follows:

- To the best of our understanding, no works investigated the JSP under a smart factory framework with edge computing. This work is the first to adopt a DQN method to solve the JSP of a smart semiconductor manufacturing factory based on an edge computing framework.
- To make decisions on the dispatching rules for all edge devices in the distributed edge computing framework, different from the classical DQN that considers only one decision, this work adjusts the DQN to make multiple decisions to fit the requirement of the concerned problem.

II. System Framework

This work proposes a smart semiconductor manufacturing factory based on an edge computing framework (Fig. 1) with the following components:

- Machine: Each machine in the semiconductor manufacturing factory is assumed to process a task of each job, and is of various functions, e.g., wafer sorting, chip cutting, bonding, and so on.
- Edge device: Each machine is monitored and controlled by an edge device to avoid lowering the overall equipment effectiveness, delaying the delivery, and raising the cost. Each edge device collects and preprocesses the information and system features of the controlled machine, and transmits them to a centralized cloud center in the factory.
- Cloud center: A cloud center is connected with all edge devices, probably through gateways and fog devices. The cloud center receives the information from all edge devices, and then determines dispatching rules of all machines, which are then sent to all edge devices.

III. Problem Description

Consider a factory framework based on edge computing as illustrated in Fig. 1. Let M denote the number of machines in the factory. This work aims to solve the JSP under the smart factory framework. That is, given a number of customer orders each of which includes J jobs, each job has a set of tasks needed to be processed by M machines in a specific order. For simplification of the problem, suppose that each job must be processed by each machine once, i.e., each job has M tasks. In the JSP concerned in this work, after being acknowledged to fulfill a set of J jobs, the cloud center in the factory is required to find a method (i.e., DQN in this work) to determine the starting time when each task of each job is processed by the corresponding machine so that the makespan (i.e., the maximum completion time) is optimized. Then, each edge device receives the scheduling results computed by the cloud center, and executes the results in its controlled machine. JSP has been shown to be NP-hard, and been one of the most computationally stubborn combinatorial problems [1].

The real-world shop floor of a factory is complex, and hence, in practice, the factory staff follow one of several dispatching rules (e.g., FIFO) to determine the scheduling of jobs in each machine rather than a detailed scheduling of jobs. The reason why this work uses dispatching rules rather than directly assigning the scheduling result is that dispatching rules have simplicity, ease for implementation, and understandability that provide good managerial insights [22]. As the result, in practice, a lot of manufacturing factories rely on using dispatching rules to solve scheduling problems.

Previous works (e.g., [23]) provided two strategies to determine dispatching rules: a single dispatching rule (SDR) and multiple dispatching rules (MDR). The SDR strategy assigns one dispatching rule to all machines in the factory; whereas the MDR strategy assigns multiple dispatching rules to all machines in the factory. Most of the previous works indicated that using the MDR strategy performs better than using the SDR strategy. As the result, the DQN method proposed in this work adopts the MDR strategy.

In the JSP under the smart factory framework concerned in this work, to fulfill each customer order with J jobs, the cloud center obtains the information of the J jobs, including a sequence of M tasks in each job, the ordering of M machines that process M tasks of each job, and the processing time of each task. The mission of the cloud center is to assign a dispatching rule to each machine, and then to transmit the assignment information to the edge device that controls this machine. On the other hand, each edge device receives the information of jobs related to the controlled machine. The

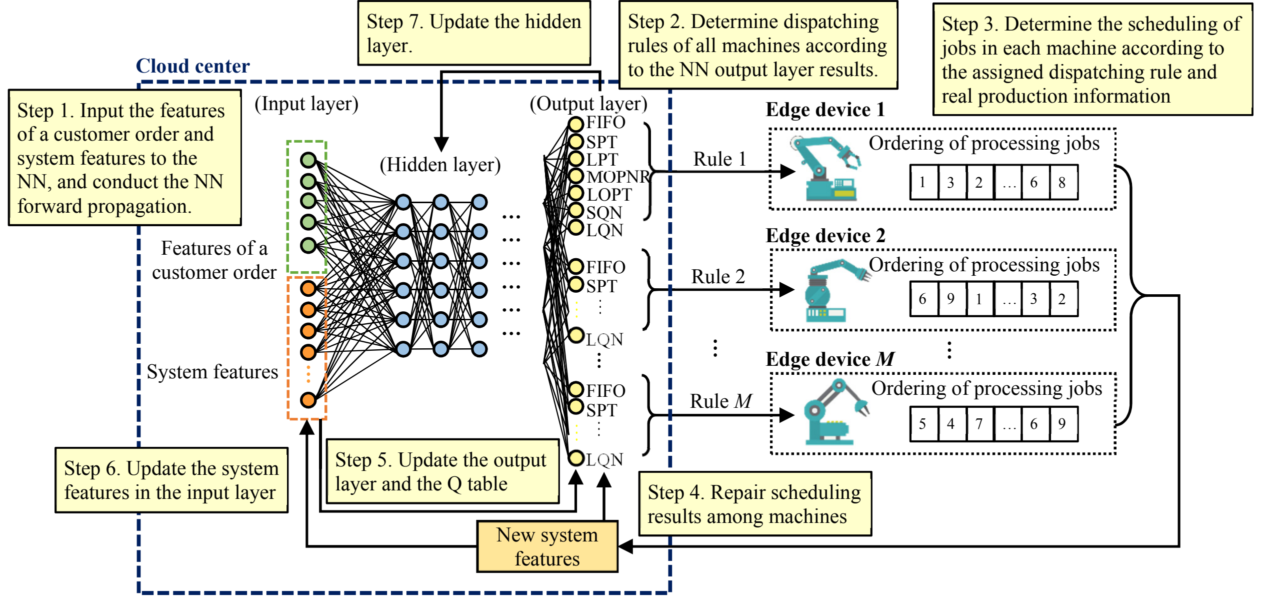

Fig. 2. The flowchart of the proposed MDQN method for the smart factory framework with edge computing.

mission of each edge device is to monitor and control the controlled machine, and to make a scheduling plan of job processed at the machines according to the dispatching rule assigned by the cloud center. After determining the scheduling result in the controlled machine, the edge device returns a reward message back to the cloud center, which will base on this reward message to adjust the NN for the next NN computation or later customer orders.

The notation used in this work is summarized as follows: M: number of machines; J: number of jobs; P_{jm}: processing time of job j at machine m; F_m: completion time of machine m; C_j: completion time of job j; Q_{jm}: queuing time of job j at machine m; μ_m: utilization ratio of machine m; I_m: idle time of machine m; O_m: number of WIPs of machine m; and K_{jm}: completion time of job j at machine m.

IV. Proposed Multiclass DQN Method

This work proposes a multiclass DQN (MDQN) method to solve the JSP with an MDR strategy under a smart factory environment with edge computing. MDQN is a multi-layered NN that, for a given state s, outputs a vector of action values $Q(s, \cdot\ ; \theta)$, where θ is a parameter of the network. For an a-dimensional state space and an action space containing b actions, the NN is a function from $\mathbf{R}^a$ to $\mathbf{R}^b$ [24]. To avoid complex computation, this work supposes that the transition time between machines is very short and can be neglected; machines never break down and are always available; aging damage to machine components is neglected; and each machine processes only one product at any time point.

The proposed MDQN method is given in Algorithm 1, and its flowchart is given in Fig. 2. Initially, the cloud center receives the information of a customer order with J jobs. This work reasonably supposes that the number of machines M is fixed in each customer order, and the number of jobs J is not. Note that for the case with less than M machines, it is simple to add dummy machines to make this case with exact M machines, and then to let the processing time of each job at dummy machines be zero. Then, we adopt the MDQN to decide the assignment of a dispatching rule to each machine. The NN in the proposed MDQN is divided into an input layer, a hidden layer, and an output layer. In this work, the input layer includes $8 + 2 \cdot M$ neurons consisting of 5 features of the customer order and $3 + 2 \cdot M$ system features. The 5 features of the customer order are listed as follows:

- M: Number of machines.
- J: Number of jobs.
- $\sum_j \sum_m P_{jm}$: Sum of all processing times.
- $\mathrm{Max}_{j,m}\ P_{jm}$: The maximum among all processing times.
- $\mathrm{Min}_{j,m}\ P_{jm}$: The minimum among all processing times.

And, the $3 + 2 \cdot M$ system features are listed as follows:

- $\bar{F} = (\sum_j C_j)/J$: Average completion time.
- $C_{\max} = \mathrm{Max}_j\ C_j$: The makespan.
- $\bar{Q} = (\sum_j \sum_m Q_{jm})/J$: Average queuing time.
- $\{\mu_m = (F_m - I_m)/F_m,\ \text{for each } m \in \{1, 2, ..., M\}\}$: Utilization ratio of machine m for each $m \in \{1, 2, \ldots, M\}$.
- $\{\bar{O}_m = (\sum_j K_{jm})/F_m,\ \text{for each } m \in \{1, 2, ..., M\}\}$: Average number of WIPs of machine m for each $m \in \{1, 2, \ldots, M\}$.

This work considers 7 common dispatching rules used in the JSP [25] (Table I). Hence, the output layer in the NN has $7 \cdot M$ neurons, which are divided into M groups. Each neuron implies a candidate dispatching rule, and each group of 7 neurons determines the dispatching rule of a machine.

Algorithm 1 MDQN(N customer orders)

1: **for** $i = 1, 2, \ldots, N$ **do**
2: Consider J jobs in customer order i. Assign a random or certain dispatching rule to each machine
3: Compute system features $\overline{F}, C_{\max}, \overline{Q}, \{\mu_m\}$, and $\{\overline{O}_m\}$
4: Create a state based on (J, $C_{\max}$, μ_1, μ_2, …, μ_M) in the Q table
5: Construct an NN with $8 + 2 \cdot M$ input neurons and $7 \cdot M$ output neurons
6: **for** epoch number $t = 1, 2, \ldots, T$ **do**
7: The cloud center uses Algorithm 2 to conduct NN propagation in the cloud center
8: The cloud center uses Algorithm 3 (with an ε-greedy strategy) to determine dispatching rules of all machines, and transmit the information to each edge device
9: Each edge device uses Algorithm 4 to determine a scheduling of jobs in the controlled machine
10: The cloud center oversees the scheduling results of all machines, and then uses Algorithm 5 to repair the scheduling results
11: The cloud center creates a state based on (J, $C_{\max}$, μ_1, μ_2, …, μ_M) in the Q table, and updates the Q table in the output layer using $q'_m = [q_m + (\mu'_m - \mu_m)/\mu_m] + \gamma \cdot \text{Max}_a\ Q[s', a]$
12: The cloud center updates the system features: $\overline{F}, C_{\max}, \overline{Q}, \{\mu_m\}$, and $\{\overline{O}_m\}$
13: The cloud center updates the hidden layer by the loss $E[(Q'_m - Q_m)^2]$
14: The cloud center decreases the probability value ε by ε-diff
15: **next for**
16: **next for**

TABLE I.
The dispatching rules considered in this work.

Dispatching rule	Content
First in first out (FIFO)	The first job is processed first.
Shortest processing time (SPT)	The job with the shortest processing time is process first.
Longest processing time (LPT)	The job with longest processing time is processing first.
Most operations remaining (MOPNR)	The job with most operations remaining at later tasks is processed first.
Longest operation processing time (LOPT)	The job with the longest processing time at later tasks is processed first.
Shortest next queue (SQN)	The job whose next task has the shortest processing time is process first.
Longest next queue (LQN)	The job whose next task has the longest processing time is process first.

In Algorithm 1, given a customer order i, the cloud center initially assigns a random or certain dispatching rule to each machine (Line 2). Then, the system features can be obtained according to these dispatching rules (Line 3), and a state based on these system features is created in a so-called *Q table* (used for recording the qualities of all output states), which will be explained later (Line 4). Then, all the input information is fed into the input layer of the NN (Line 5). After computation through the hidden layer, it comes out the values for the neurons in the output layer (Line 7). The output layer has M groups of neurons. According to the results in the output layer, the largest neuron value in each group of neurons determines a dispatching rule for each machine (Line 8). By doing so, multiple dispatching rules can be determined by considering the largest neuron value of each group of neurons in the output layer, which is different from the classical DQN method that generally outputs only one of the largest neuron value.

After each edge device receives the information on the dispatching rule determined by the cloud center, it further determines the scheduling result of processing jobs in the controlled machine (Line 9). Then, it sends the scheduling results back to the cloud center.

After collecting all scheduling results from edge devices, the cloud center oversees all scheduling results and repairs infeasible scheduling results (Line 10). Then, it uses these results to calculate new $3 + 2 \cdot M$ system features, and uses these system features to update the neurons values and the Q table in the output layer (Line 11). Then, it uses the new system features to overwrite the values of the input neurons for system features in the NN (Line 12). Then, it uses the updated neuron values in the output layer to update the weights of the NN hidden layer the fashion of backward propagation (Line 13).

Then, the cloud center repeats the above procedure until a maximal number of epochs T is achieved. Finally, the output neuron values in the NN determines the final decisions for multiple dispatching rules, by which all jobs of the concerned customer order are processed in the machines. Then, the weights of NN are reserved to process the next customer order.

Main steps of the proposed MDQN method are as follows:

A. NN forward propagation in the cloud center

After receiving the information of a customer order with J jobs and the system features, the cloud center adopts Algorithm 2 to conduct an NN propagation to obtain an NN output layer result (Line 7 of Algorithm 1). Recall that the input layer of the NN includes 5 features of customer orders and $3 + 2 \cdot M$ system features. In Algorithm 2, Line 1 only computes the first three features, because the other features have been known at this stage. Then, Line 2 feeds these features to the NN input layer, and Line 3 initializes the NN hidden layer if this is the first epoch. Then, Line 3 executes once of the NN propagation to obtain the NN output layer results, which are also called *Q-values* in this work.

Algorithm 2 NN propagation in the cloud center

Input: Information of J jobs
Output: Neuron values in the NN output layer (Q-values)
1: Compute $\sum_j \sum_m P_{mj}$, $\max_{m,j} P_{mj}$, and $\min_{m,j} P_{mj}$
2: The features M, J, $\sum_j \sum_m P_{mj}$, $\max_{m,j} P_{mj}$, $\min_{m,j} P_{mj}$ and the system features are fed into the neurons in the NN input layer
3: Randomly initialize the weights in the NN hidden layer if this is the first epoch
4: Execute once of the NN propagation

B. Determine dispatching rules

After obtaining an NN output layer result, the cloud center uses Algorithm 3 to determine the dispatching rules for all machines according to the result, and then sends the information to all edge devices (Line 8 of Algorithm 1). Recall that the NN has M groups of output neurons, each of which has 7 neurons to imply 7 candidate dispatching rules, respectively. To increase diversity of the solution, this work applies the ε-greedy policy [21] to determine the dispatching rules. Different from the classical DQN that always chooses the action with the largest Q-vale, the ε-greedy policy allows a probability ε to choose an arbitrary action at the early epochs, but reduces this probability as the number of epochs increases.

Algorithm 3 Determining dispatching rules
Input: NN output layer values
Output: Dispatching rules for all machines
1: **for** $m = 1$ to M **do**
2: **if** $P \le \varepsilon$ **then**
3: Machine m is assigned with a random dispatching rule
4: **else**
5: Machine m is assigned with the dispatching rule corresponding to the largest output neuron value in the m-th group of output neurons
6: **end if**
7: **next for**

C. Determining the scheduling result of a machine

After receiving the assigned dispatching rule of machine m, edge device m uses Algorithm 4 to determine a scheduling result of machine m (i.e., the starting time of processing each job at machine m) (Line 9 of Algorithm 1). However, not all dispatching rules can be adopted in machine m, because the information of processing jobs might be in conflict with this rule. Hence, if the generated schedule result is not feasible, Algorithm 4 sequentially adopts the MOPNR rule and the SPT rule to repair the repeated job numbers in the scheduling result. The reason why to use the MOPNR and SPT rules is that it has been shown in previous works (e.g., [26]) that the MOPNR and SPT rules perform well in terms of the makespan and machine utilization, which are consistent with the objective of this work. If the result is still infeasible, then a random scheduling result is adopted.

Algorithm 4 Determining the scheduling result of a machine
Input: The assigned dispatching rule of machine m
Output: Scheduling result of machine m
1: Edge device m bases on the assigned dispatching rule and the information of processing jobs at machine m to determine the scheduling result of machine m
2: **if** the scheduling result based on the assigned rule is infeasible **then**
3: **if** the assigned rule is not MOPNR **then**
4: Use the MOPNR rule to repair the repeated job numbers in the scheduling result
5: **end if**
6: **if** the scheduling result is infeasible **then**
7: **if** the assigned rule is not SPT rule **then**
8: Use the SPT rule to repair the repeated job numbers in the scheduling result
9: **end if**
10: **if** the scheduling result is infeasible **then**
11: A random schedule result is adopted
12: **end if**
13: **end if**
14: **end if**

D. Repairing scheduling results

Although Algorithm 4 generates a feasible scheduling result for each machine m, there is a problem when the cloud center oversees the scheduling results of all machines. That is, the starting time of a job processed at machine m could be earlier than the end time of the job processed at its earlier task, so that machine m needs to wait for a time period. Hence, when overseeing all scheduling results, the cloud center adopts Algorithm 5 to repair the starting and end times of the jobs with such a problem (Line 10 of Algorithm 1). In Algorithm 5, it checks if the end time of each job at each machine is greater than the starting time of the job's next task at a different machine. If true, then the two times are corrected; and then the cloud center corrects the starting and end times of the remaining jobs processed after this job at the same machine.

Algorithm 5 Repairing scheduling results
Input: The scheduling results of all machines
Output: A correct scheduling result for each machine
1: **for** $i = 1$ to J **do**
2: **for** $j = 1$ to $M - 1$ **do**
3: m_1 = index of the machine that processes task j of job i
4: m_2 = index of the machine that processes task $(j + 1)$ of job i
5: Let T^{e}_{im} (resp., T^{s}_{im}) denote the starting time (resp., end time) of job i at machine m according to the scheduling result of machine m
6: $\delta_1 = T^{e}_{im_1} - T^{s}_{im_2}$
7: **if** $\delta_1 > 0$ **then**
8: $T^{s}_{im_2} = T^{s}_{im_2} + \delta_1$ and $T^{e}_{im_2} = T^{e}_{im_2} + \delta_1$
9: Let j_2 denote the ordinal number in which job i is processed at machine m_2 according to the scheduling result of machine m_2, i.e., job i is the j_2-th job to be processed at machine m_2
10: **for each** job k processed after job i at machine m_2 except for the last job **do**
11: Let k' denote the index of the job processed next to job k at machine m_2
12: $\delta_2 = T^{e}_{km_2} - T^{s}_{k'm_2}$
13: **if** $\delta_2 > 0$ **then**
14: $T^{s}_{k'm_2} = T^{s}_{k'm_2} + \delta_2$ and $T^{e}_{k'm_2} = T^{e}_{k'm_2} + \delta_2$
15: **end if**
16: **next for**
17: **end if**
18: **next for**
19: **next for**

E. Updating the output layer

The proposed MDQN uses some features to characterize a system *state* of the NN, and maintains a *Q table* to record the best output qualities of all state transitions found so far. This work defines a state to be characterized by (J, $C_{\max}$, μ_1, μ_2, …, μ_M). Note that J is one of the input parameters of the NN, and indeed influences the performance. Hence, J is included in the state. Also note that M is not included in the state, because machine features have been captured by machine utilization rates μ_1, μ_2, …, μ_M. Because each of these features has too many possible values, the total number of states may become too large. To reduce the total number of states, after a lot of experimental trials, C_{max} is divided into five classes: [0, 0.5W), [0.5W, 0.7W), [0.7W, 0.9W), and [0.9W, 1.0W] where W is the sum of all processing times, i.e., $\sum_j \sum_m P_{jm}$. For each m = 1, 2, …, M, feature μ_m is divided into five classes: [0, 0.03), [0.03, 0.05), [0.05, 0.07), and [0.07, 1.00].

If h states have been found so far, and a Q table is defined as an $h \times h$ table in which the entry at position (s, s') is denoted by $Q[s, s']$ that records a vector with M elements representing the utilizations of M machines at state s', transited from state s. That is, this work supposes that high machine utilization would result in a high-quality solution, so is recorded in the Q table.

After the scheduling result is obtained, {J, $C_{\max}$, μ_1, μ_2, …, μ_M} can be obtained, and hence a state can be characterized (Line 11 of Algorithm 1). If this state is a new state, then the Q table is extended with this new state, i.e., a new row and a new

column for this state are added, and all the new entries are initialized to zero vectors. If the original state is s and the new state is s', then $Q[s, s'] = (q'_1, q'_2, \ldots, q'_M)$ where for each $m \in \{1, 2, \ldots, M\}$, element q'_m is calculated as follows:

$$q'_m = [q + (\mu'_m - \mu_m) / \mu_m] + \gamma \cdot \text{Max}_a\ Q[s', a] \quad (1)$$

where γ is the learning rate.

F. Updating the system features

Line 12 in Algorithm 1 computes new system features $\overline{F}, C_{\max}, \overline{Q}, \{\mu_m\}, \{\overline{O}_m\}$ according to the scheduling result, and feeds them to the neurons in the NN input layer, which will be used in the next run of the NN forward propagation.

G. Updating the hidden layer

At the end of an epoch of Algorithm 1 (Line 13), the cloud center calculates the loss of the prediction result in terms of Q values as follows: $E[(q'_m - q_m)^2]$, where q'_m is the Q value calculated by (1); and q_m is the NN output layer result (Line 8 of Algorithm 1). Then, the cloud center uses this loss to conduct a back propagation to update the weights in the hidden layer.

V. Experimental Results

This section evaluates the performance of the proposed MDQN for the concerned JSP in terms of the makespan of all jobs. Because no previous works proposed any JSP under a smart factory framework based on edge computing, the experiments focus on two parts: analyzing performance of the proposed method using different parameter settings; and comparing performance of the proposed method with the method using randomly chosen dispatching rules.

A. Experimental environment

The experiments in this work are conducted on a benchmark dataset of JSP from the [27] in which the instances entitled “Lawrence (la)” [28], “Demirkol, Mehta, and Uzsoy (dmu)” [29], “Taillard (ta)” [30], and “Applegate and Cook” [1] with different numbers of machines (M) and jobs (J) are used. The parameters used in this work are divided into three categories: 1) parameters for NN: number of neurons in the hidden layer of the NN is 10~160, number of epochs T is 16000, and learning rate is 0.01; 2) parameters for ε-greedy policy: ε = 0.9 and ε-diff = 0.02. Note that in ε-greedy policy, the value of parameter ε decreases by ε-diff for each epoch of the algorithm; 3) learning rate for updating Q table: γ = 0.7. This work sets the ratio of training data and testing data to be 2:1. The training data is used to train a forecast model, and then the testing data is used to evaluate the performance of the model.

Note that the experiments conducted in this work adopt fixed numbers of machines (M) and jobs (J). In practice, the number of machines in a factory is not changed frequently, and hence is considered to be fixed. On the other hand, this work uses the customer orders with a fixed number of jobs so as to obtain high forecast efficiency for experimental comparison. In fact, the proposed model can be applied to different numbers of machines and jobs.

B. Convergence analysis

This subsection analyzes the convergence of training a model using the proposed method with different settings of three factors: number of neurons in the hidden layer, combination of number of machines and number of jobs, and the parameter ε-diff in the ε-greedy policy.

Firstly, we analyze the convergence effect using different number of neurons in the hidden layer. Consider the problem instance “la01” with M = 5 and J = 10 in the “la” problem set. Fig. 3 shows the convergence effect when the number of neurons in the hidden layer is 10, 30, 60, 100, 130, and 150. From Fig. 3, the more the number of neurons is, the more quickly the convergence is achieved. When the number is no less than 100, the method converges after 150 epochs.

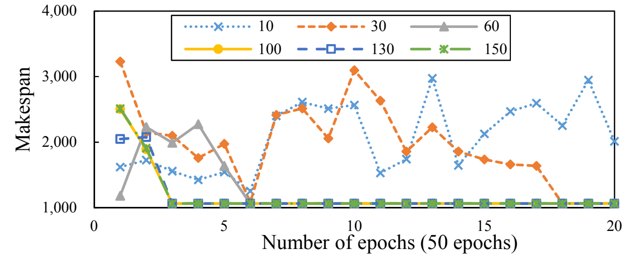

Fig. 3. Convergence analysis on the number of neurons in the hidden layer.

Secondly, we conduct the convergence analyses on different combinations of M and J. Fig. 4 shows the convergence analysis of the “la” instances with M = 5 and J = 10, the “dmu” instances with M = 15 and J = 20 or 15, and “ta” instances with M = 20 and J = 20. From Fig. 4, the more the number of machines or jobs is, the more slowly the convergence is achieved. Lastly, we conduct the convergence analysis on parameter ε-diff. Fig. 5 shows that the method using ε-diff = 0.00005 obtains the solution soon, but converges slowly.

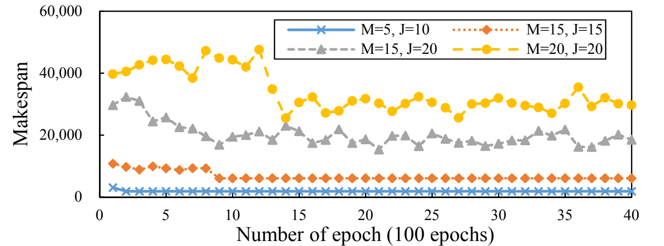

Fig. 4. Convergence analysis on different combinations of M and J.

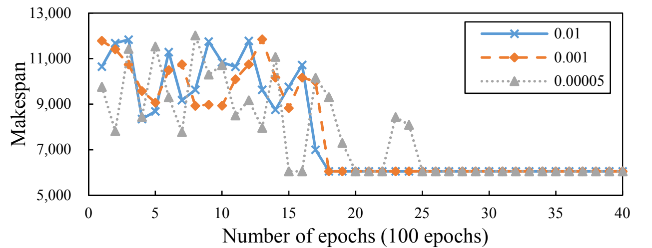

Fig. 5. Convergence analysis on difference values for parameter ε-diff.

In what follows, Fig. 6 shows the experimental comparison of the proposed MDQN method (labeled by "DQN") and the DQN method using random dispatching rules (labeled by "Random") on the problem instance "la08" with $M = 5$ and $J = 15$. Obviously, from Fig. 6, the proposed MDQN shows the effect of convergence.

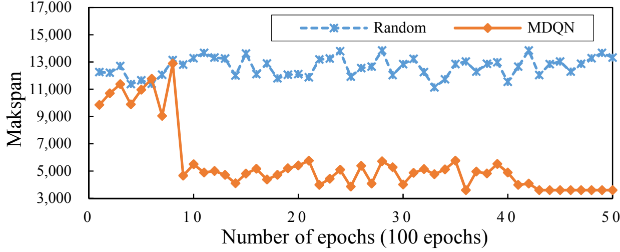

Fig. 6. Comparison between the proposed MDQN method and the DQN method using random dispatching rules.

C. Results under different settings

First, this subsection compares the experimental results using the models trained by only one data sample and multiple data samples, respectively (Fig. 7). Note that a data sample means a customer order with J jobs. From Fig. 7, the two models perform similar for the earlier epochs, but the model trained by multiple data samples performs better for the later epochs. Then, we compare the experimental results using the proposed MDQN method (applying the MDR strategy), the Random method (i.e., each machine is assigned to a random dispatching rule), and the methods applying the SDR strategy. According to the analysis in Fig. 7, the MDQN method trains the NN by multiple data samples. The SDR strategy means that all machines adopt the same dispatching rules. Because this work considers 7 dispatching rules, this experiment considers 7 methods applying the SDR strategy (i.e., each method uses one type of the 7 dispatching rules). The parameters in this experiment are set as follows: 1) parameters for NN: number of neurons in the hidden layer of the NN is 160, number of epochs T is 20000, and learning rate is 0.001; 2) parameters for ε-greedy policy: $\varepsilon = 0.9$ and ε-diff = 0.0001.

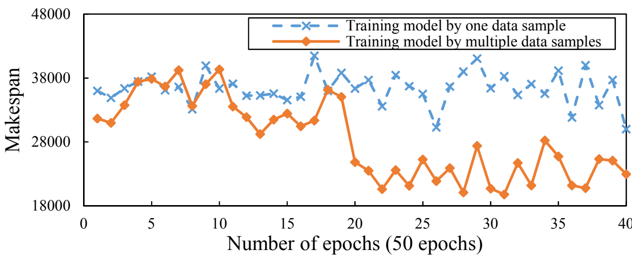

Fig. 7. Comparison of the results of the models trained by a single data sample and multiple data samples.

The experimental results using the above three types of methods on 10 problem instances with $M = 15$ and $J = 20$ are shown in Table II, in which the best processing time results are printed in boldface. From Table II, the proposed MDQN method performs best among all methods.

TABLE II.
Comparison of the experimental results using the proposed MDQN method, the random method, and the methods using the SDR strategy.

Instance	MDQN	Random	SDR						
			FIFO	SPT	LPT	MOPNR	LOPT	SQN	LQN
dmu01	**3520**	38594	4668	32364	28233	4618	4123	29029	27411
dmu02	**3765**	40359	4282	33442	28571	4720	4323	31365	31676
dmu03	**3953**	36372	5600	32125	34467	4560	4772	30028	31107
dmu04	**3521**	40594	4020	28414	34550	4752	4597	36116	30830
dmu05	**3790**	40476	4575	35099	33981	4456	4500	37235	30996
dmu41	**4881**	25624	5980	25095	23095	6397	5516	20223	23496
dmu42	**5895**	32651	6261	23942	25140	6459	6341	23102	23937
dmu43	**5559**	28479	6575	20003	22662	5911	6129	25026	21344
dmu44	**5502**	28584	6901	26281	22429	6055	7459	22641	23753
dmu45	**5189**	27941	6102	24615	23394	5268	6162	24987	21562
Average	**4557.5**	33967.4	5496.4	28138	27652.2	5319.6	5392.2	27975.2	26611.2

VI. CONCLUSION

This work has proposed a smart semiconductor manufacturing factory based on an edge computing framework, and investigated the JSP under the smart factory framework. In JSP, given a number of customer order each of which consists of multiple jobs, a cloud center and multiple edge devices are cooperated to dynamically determine one of seven dispatching rules of processing jobs to each manufacturing machine. This work further solved this problem by an MDQN, which combines DL and RL. To simultaneously determine multiple dispatching rules, the NN in the proposed MDQN includes multiple output neurons to make multiple decisions, which are different from the classical DQN. Experimental results showed the convergence effects under a variety of parameter settings. In addition, the results also showed that the model trained by multiple data samples performed well, and the MDQN with the MDR strategy performed better than the methods using the SDR strategy.

In the future, because practical manufacturing scheduling problems are much complex, it would be of interest to extend the proposed MDQN model with more scheduling considerations with a cloud-fog-edge framework. It is also of interest to improve the method, e.g., designing a different Q table scheme and a different reward scheme, adopting a different NN type, and integrating the MDQN with other machine learning methods. In addition, it is interesting to test the method on a larger number of problem instances. On the other hand, considering ease of implementation, this work used the MDR strategy. Although advantaged to practical applications, advanced smart techniques could allow implementation of detailed scheduling results. Therefore, a line of future research is to change the MDR strategy to directly finding the detailed scheduling result.

REFERENCES

[1] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA J. Comput.*, vol. 3, no. 2, pp. 149-156, 1991.

[2] X. Lyu, H. Tian, L. Jiang, A. Vinel, S. Maharjan, S. Gjessing, and Y. Zhang, "Selective offloading in mobile edge computing for the green Internet of things," *IEEE Netw.*, vol. 32, no. 1, pp. 54-60.

[3] G. Jia, G. Han, H. Rao, and L. Shu, "Edge computing-based intelligent manhole cover management system for smart cities," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1648-1656, 2018.

[4] K. Zhang, Y. Mao, S. Leng, A. Vinel, and Y. Zhang, “Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks,” in *Proc. of RNDM 2016*, IEEE Press, pp. 288-294.
[5] V. Sharma, F. Song, I. You, and M. Atiquzzaman, “Energy efficient device discovery for reliable communication in 5G-based IoT and BSNs using unmanned aerial vehicles,” *J. Netw. Comput. Appl.*, vol. 97, pp. 79-95, 2017.
[6] G. Jia, G. Han, J. Du, and S. Chan, “A maximum cache value policy in hybrid memory based edge computing for mobile devices,” *IEEE Internet Things J.*, in press.
[7] L. Liu, S. Chan, G. Han, M. Guizani, and M. Bandai, “Performance modelling of representative load sharing schemes for clustered servers in multi-access edge computing,” *IEEE Internet Things J.*, in press.
[8] Y. Wang, “A new hybrid genetic algorithm for job shop scheduling problem,” *Comput. Oper. Res.*, vol. 39, no. 10, pp. 2291-2299, 2012.
[9] H. W. Ge, L. Sun, Y. C. Liang, and F. Qian, “An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling,” *IEEE Trans. Sys., Man, Cybern. A: Syst., Humans*, vol. 38, no. 2, pp. 358-368, 2008.
[10] P. J. Van Laarhoven, E. H. Aarts, and J. K. Lenstra, “Job shop scheduling by simulated annealing,” *Oper. Res.*, vol. 40, no. 1, pp. 113-125, 1992.
[11] S.-P. Tseng, C.-W. Tsai, J.-L. Chen, M.-C. Chiang, and C.-S. Yang, “Job shop scheduling based on ACO with a hybrid solution construction strategy,” in *Proc. of FUZZ-IEEE 2011*, 2011, pp. 2922-2927.
[12] W. Yuan, C. Li, D. Guan, G. Han, and A. M. Khattak, “Socialized healthcare service recommendation using deep learning,” *Neural Comput. Appl.*, vol. 30, no. 7, pp. 2071-2082, 2018.
[13] W. Zhang and T. G. Dietterich, “A reinforcement learning approach to job-shop scheduling,” in *Proc. of IJCAI 1995*, 1995, vol. 2, pp. 1114-1120.
[14] M. E. Aydin and E. Öztemel, “Dynamic job-shop scheduling using reinforcement learning agents,” *Robot. Auton. Syst.*, vol. 33, no. 2-3, pp. 169-178, 2000.
[15] L. Tang, W. Liu, and L. Liu, “A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment,” *J. Intell. Manuf.*, vol. 16, no. 3, pp. 361-370, 2005.
[16] H. Yu and W. Liang, “Neural network and genetic algorithm-based hybrid approach to expanded job-shop scheduling,” *Comput. Ind. Eng.*, vol. 39, no. 3-4, pp. 337-356, 2001.
[17] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep q-learning with model-based acceleration,” in *Proc. of ICML 2016*, 2016, pp. 2829-2838.
[18] V. Mnih, et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529-533, 2015.
[19] D. Silver, et al., “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, pp. 484-489, 2016.
[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing Atari with Deep Reinforcement Learning*, arXiv preprint, arXiv:1312.5602, 2013.
[21] Y. Wei, L. Pan, S. Liu, L. Wu, and X. Meng, “DRL-scheduling: an intelligent QoS-aware job scheduling framework for applications in clouds,” *IEEE Access*, vol. 6, pp. 55112-55125, 2018.
[22] S. Nguyen, M. Zhang, and K. C. Tan, “Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules,” *IEEE Trans. Cybern.*, vol. 47, no. 9, 2951-2965, 2017.
[23] 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, 2018.
[24] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in *Proc. of AAAI 2016*, ACM Press, 2016, pp. 2094-2100.
[25] S. S. Panwalkar and W. Iskander, “A survey of scheduling rules,” *Oper. Res.*, vol. 25, no. 1, pp. 45-61, 1977.
[26] V. Subramaniam, G. K. Lee, G. S., Hong, Y. S., Wong, and T. Ramesh, “Dynamic selection of dispatching rules for job shop scheduling,” *Prod. Plan. Control*, vol. 11, no. 1, pp. 73-81, 2000.
[27] Jobshop Instance, available at: http://jobshop.jjvh.nl/index.php
[28] S. Lawrence, *An Experimental Investigation of Heuristic Scheduling Techniques*, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1984.
[29] E. Demirkol, S. Mehta, and R. Uzsoy, “Benchmarks for shop scheduling problems,” *Eur. J. Oper. Res.*, vol. 109, no. 1, pp. 137-141, 1998.
[30] E. Taillard, “Benchmarks for basic scheduling problems,” *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278-285, 1993.