IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 12, 2020, accepted April 1, 2020, date of publication April 14, 2020, date of current version April 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2987820

Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems

CHIEN-LIANG LIU, (Member, IEEE), CHUAN-CHIN CHANG, AND CHUN-JAN TSENG
Department of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 30010, Taiwan

Corresponding author: Chien-Liang Liu (clliu@mail.nctu.edu.tw)

This work was supported in part by Ministry of Science and Technology, Taiwan, under Grant no. MOST 107-2221-E-009-109-MY2.

ABSTRACT In the past decades, many optimization methods have been devised and applied to job shop scheduling problem (JSSP) to find the optimal solution. Many methods assumed that the scheduling results were applied to static environments, but the whole environments in the real world are always dynamic. Moreover, many unexpected events such as machine breakdowns and material problems may be present to adversely affect the initial job scheduling. This work views JSSP as a sequential decision making problem and proposes to use deep reinforcement learning to cope with this problem. The combination of deep learning and reinforcement learning avoids handcraft features as used in traditional reinforcement learning, and it is expected that the combination will make the whole learning phase more efficient. Our proposed model comprises actor network and critic network, both including convolution layers and fully connected layer. Actor network agent learns how to behave in different situations, while critic network helps agent evaluate the value of statement then return to actor network. This work proposes a parallel training method, combining asynchronous update as well as deep deterministic policy gradient (DDPG), to train the model. The whole network is trained with parallel training on a multi-agent environment and different simple dispatching rules are considered as actions. We evaluate our proposed model on more than ten instances that are present in a famous benchmark problem library – OR library. The evaluation results indicate that our method is comparative in static JSSP benchmark problems, and achieves a good balance between makespan and execution time in dynamic environments. Scheduling score of our method is 91.12% in static JSSP benchmark problems, and 80.78% in dynamic environments.

INDEX TERMS Job shop scheduling problem (JSSP), deep reinforcement learning, actor-critic network, parallel training.

I. INTRODUCTION

Scheduling can be viewed as a decision making process, in which the scheduling arranges the tasks that we should process sequentially. Moreover, scheduling is also an essential task in many application domains, including manufacturing [13], [43], [44], network [40] and supply chains [12], [36]. For example, Wang *et al.* [36] considered production and distribution in supply chain management, and focused on the permutation flow shop scheduling problem, which addressed the problem of batch delivery to multiple customers. Zheng and Wang [44] focused on a resource constrained unrelated parallel machine green manufacturing scheduling problem, which aimed to minimize the makespan and the total carbon emission. A typical scheduling comprises one or more objectives with several constraints, resulting in a complicated computational problem. Mathematical programming is a popular approach to deal with the scheduling problem by formulating it as an optimization problem with constraints, and using optimization techniques to seek the optimal solution.

A practical scheduling problem normally involves hundreds or thousands variables and enormous constraints, and the optimization is a NP-hard problem [15]. Consequently, meta-heuristic approach has become a popular method to obtain sub-optimal solutions of scheduling problem in recent years. However, real-world production systems always have many uncertainties, such as variation in process time and machine breaking down, so the optimal solution in the past may be unsuitable in current situation, and requiring a lot of time to reconstruct a new solution.

Reinforcement learning (RL) is a type of machine learning concerned with how agents should take actions in an environment so as to maximize the future reward. More specifically,

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

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

the main idea of RL is to let agents learn by trial and error throughout the execution of the actions [34]. Once the agent learns how to behave, it may generalize the model it has learned to other unseen problems. Although the training of RL might take a lot of time, a well-trained agent could give an immediate reaction for the problem encountered. It has been applied successfully to various problems, including robot control [9], manufacturing [33], and games [31]. Therefore, RL can be an alternative method for dealing with dynamic and complicated scheduling problems.

In the past, RL has several successful applications on job shop scheduling problem (JSSP), and could find competitive solutions for JSSP benchmark problems. However traditional RL needs to handcraft the features in the complex systems and requires domain knowledge to obtain useful information from raw statement. Nowadays, the progress of deep learning (RL) has accelerated the development of RL. Cunha *et al.* [10] presented a literature review on job shop scheduling, evolutionary algorithms, and deep reinforcement learning (DRL). Nazari *et al.* [25] developed an end-to-end framework for solving the Vehicle Routing Problem (VRP) using DRL, and their proposed method outperformed classical heuristics and Google's optimization tools (OR-Tools). More importantly, DRL has achieved remarkable success in many areas. AlphaGO [31] is one of the most successful DRL algorithms now. The success of AlphaGO inspires us to apply DRL to JSSP by using deep architecture to learn from low-level features of the statement and combine them to generate high-level features. The feature learning phase avoids the prior knowledge to be involved in the whole statement, making it more flexible when facing immediate events.

This work proposes to use actor-critic model to develop DRL model for scheduling problems, in which DRL agent is divided into two parts, critic network and actor network, to analyze JSSP from two aspects. In critic network, the agent evaluates the value of the statement and gives an approximation that allows the agent to know the state it faced is good or not. In contrast, actor network relies on the approximation given by critic network to have a corresponding behavior with respect to this statement. The whole process is just like a teaching scenario, in which critic network is the teacher and actor network is the student. When facing the problem, the student gives an answer, then the teacher checks student's work and gives the reward for the student to fix the answer according to the reward.

To evaluate the proposed model, we train the DRL agent on several JSSP benchmarks from the OR library and formulate JSSP as a decision making problem. This work purposes a parallel training method, combining asynchronous update [23] as well as deep deterministic policy gradient (DDPG) [20], to train the model. Moreover, we compare our proposed method with several alternatives, including commonly used dispatching rules, meta-heuristic approach, optimum solution obtained from mathematical programming, and traditional Q-learning. The experimental results indicate that the proposed model could achieve promising results and complete the scheduling task quickly. Furthermore, testing in dynamic environment is conducted in the experiments, in which randomness factors are injected into the original state. The results show that the proposed DRL agent can obtain a competitive solution of JSSP. To the best of our knowledge, this is the first work using DRL to cope with JSSP problems in a dynamic environment.

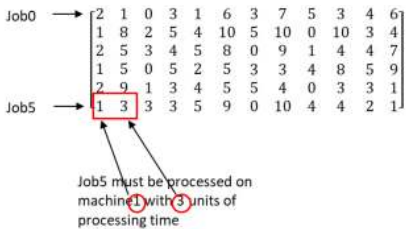

FIGURE 1. JSSP instance (ft06) in the OR-Library.

The rest of this paper is organized as follows. Section II presents related surveys and techniques. Section III introduces the proposed algorithm. Section IV shows the experimental results and Section V presents detailed discussion. The conclusions and future work are presented in Section VI.

II. RELATED WORK

This work focuses on JSSP, and proposes to use DRL to cope with this problem. Thus, this section briefly introduces JSSP, RL and DRL.

A. JOB SHOP SCHEDULING PROBLEM

Job shop scheduling problem has been a popular research issue for decades, and the main purpose of JSSP is to find an optimal solution of scheduling. This work focuses on simple JSSP, indicating that one machine can only do one job at a time and each operation of a job is only assigned to one machine. Setup time and due date are also ignored in this work.

OR-library [5] is a collection of test data sets for a variety of operations research (OR) problems, and it also includes many benchmark problems for scheduling. It has been used to evaluate the performance of scheduling algorithms in many research studies [22], [26]. Fig. 1 shows the JSSP representation in OR-Library, in which each row represents one job and each column is correspondent to the operation. Take job5 for example, job5 must first be processed on machine 1 for 3 units of processing time, then go to machine 3 for next operation. Traditional operation research (OR) has several methods dealing with JSSP, such as integer programming (IP) [21] and branch-and-bound [27]. Although OR methods can guarantee the solution to be optimal, they can only deal with small-scale problems. Thus, meta-heuristic methods, including tabu search (TS) [26], beam search (BS) [7], [29], simulated annealing (SA), genetic algorithms (GA) [4], have become an alternative choice to solve JSSP as they could efficiently find near optimal solutions on large-scale problems.

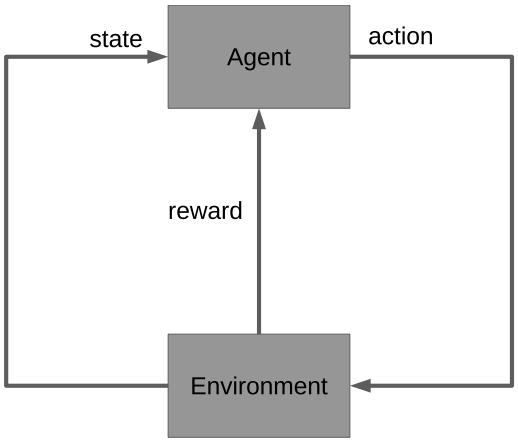

FIGURE 2. Agent and environment in the reinforcement learning.

B. REINFORCEMENT LEARNING

As compared with mathematical programming and meta-heuristic methods, RL-based approaches for JSSP will not give a static solution and react to the problem instantly without long computation time. In RL scenario as shown in Fig. 2, an agent takes an action in an environment, and then the environment transforms the action taken in the current state into the next state and a reward, which are fed back into the agent. Next, the agent transforms the new state and reward into the next action.

Typically, JSSPs can be modeled as a kind of decision making process called Markov decision process (MDP) [6]. In MDP, state $s(t)$ describes the situation of the agent encounters at time t. Then, agent takes action $a(t)$ based on $s(t)$ and receives the reward $r(t)$ from environment. The main goal of RL is to learn from a series of decisions, so that the agent could make actions to maximize the accumulation of future rewards. RL comprises two categories, model-free and model-based. Model-based RL makes the prediction to the next state and learns the whole MDP transition model. However the number of states in JSSPs is always enormous and even infinite, making it infeasible to learn the whole transition situations. Consequently, this work proposes to use model-free RL, which only evaluates the value of state instead of modeling the whole environment.

In RL algorithm, Q-learning [38] is a good choice to learn the value function when you take an action in a state by policy control to optimize the value function in an unknown MDP. The Q-value in Q-learning is a pair $Q(s, a)$ and the update of Q-value is required once taking an action and receiving a reward. The update equation is listed in Equation (1).

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)], \quad (1)$$

where r denotes immediate reward after taking action a, α is the learning rate of the algorithm, $\max_{a'} Q(s', a')$ represents maximum action value of next state s', $Q(s, a)$ is the estimate value conditional on state s and action a, and $0 \leq \gamma < 1$ denotes a discount factor, indicating that the reward is diminished over time.

At each iteration, Q-value updates after choosing a policy derived from Q, and traditionally one could use ϵ-greedy policy to provide a chance for the agent to perform exploration, so that the agent has a probability of ϵ to randomly choose an action in the next step.

To apply RL to JSSPs, Aydin and Öztemel [3] combined dispatching rules and RL in their proposed method. The action list used in their proposed method involved SPT (shortest processing time), COVERT (C over T), and CR (critical ratio). They proposed an algorithm called Q-III algorithm, which was an improved version of Q-learning, to solve the local optimum problem. Zeng and Sycara [42] applied RL to JSSP to find the solution of repair-based optimization when criteria were changing. They used case-based reasoning method to learn how to revise the schedule from past experience without observing the new statement. Instead of formulating JSSP as MDP, Gabel and Riedmiller [14] used multi-agent production scheduling (MAPS) to solve JSSP. They used neural value approximation to improve data efficiency and evaluated MAPS on many established JSSP benchmark problems. Similarily, Jiménez [22] used Multi-Agent Markov Decision Process (MMDP) to model JSSP, in which they simply used multi-agent Q-learning to adapt to different schedules. They tested their agent on not only JSSP, but also parallel machines job shop scheduling problem (JSSP-PM) and flexible job shop scheduling problem (FJSSP).

C. DEEP REINFORCEMENT LEARNING

Progress has been made over the years in DL, so the incorporation of DL and RL, also called DRL, has witnessed the great success in recent years [2], [39]. Mnih *et al.* [24] proposed a classical DRL architecture called deep Q-network (DQN) that successfully handles high dimensional input states like raw pictures and complicated action spaces. DQN can work on raw input video game picture and take an action by using convolutional neural network (CNN) to extract image features. The experimental results on various video games have shown that it is possible for DQN to achieve the human level when playing video games.

AlphaGO, proposed by Silver *et al.* [31], is probably one of the most famous DRL applications. The main idea behind AlphaGO is to combine rollout ability of Monte Carlo tree search (MCTS) and the learning ability of DRL. AlphaGO could normally find the best placement of GO by DRL network, while MCTS expanses the game from current place to find long-term inference. AlphaGO has to learn from the experience of past games and some self-play. The advanced version of AlphaGO Zero [32] can defeat AlphaGO without any human data and domain knowledge. Several training techniques have been devised to improve performance of DRL, such as training DRL in asynchronous scenario [23], combing KL-divergence into loss function [20] and replacing backpropagation to evolution strategy [30]. The success of DRL inspires many researchers to apply DRL to cope with complicated problems

that involve uncertainties and disturbances. For example, Waschneck *et al.* [37] employed DQN for RL and adopted a multi-agent method that considered a production stage and a job allocation stage in the semiconductor manufacturing scheduling. In addition, they applied a two-phase training approach for DQN agents to separately enhance scalability and stability. Vinyals *et al.* [35] proposed a multi-agent RL that used data from both human and agent games that were obtained from various leagues of continually adapting strategies and counter-strategies, each represented by deep neural networks. Their developed model, AlphaStar, was rated over 99.8% of ranked human players in StarCraft II. Ye *et al.* [41] devised a multi-agent deep reinforcement learning method to optimize the offering strategies of multiple self-interested generation companies. The aforementioned works indicated that multi-agent deep reinforcement learning has been applied to different application problems. Cui *et al.* [8] proposed a RL technique to achieve optimal trajectory tracking for autonomous underwater vehicles, in which the problem involves external disturbances, control input non-linearities and model uncertainties. Their proposed model comprised critic and action neural networks; the former is used to evaluate the long-time performance of the control, while the latter is used to compensate for the unknown dynamics. The JSSP problems comprise factorial explosion of possible solutions, and many unexpected events may be introduced to change the constraints, explaining why this work proposes to apply DRL to JSSP problems.

III. PROPOSED METHOD

This work proposes to use DRL with actor-critic architecture to let the agent interact with job shop scheduling environment. As mentioned above, JSSPs can be normally modeled as a MDP problem. We formulate the situation of the agent at time t as $s(t)$ and the action that agent takes at this situation as $a(t)$. Once the agent takes an action, it will receive an immediate reward $r(t)$, which could be viewed as a production cost. Therefore, our goal is to optimize the expected future reward over time.

The proposed model comprises two networks, actor network and critic network [18]. For the actor network, we propose to use CNN to learn the continuous behavior of the agent that can obtain maximum expected reward in the future as CNN is famous in learning discriminative features from data. On the other hand, critic network shares similar structure with actor network, but gives a value expectation of the statement depending on the action made by actor network.

A. ENVIRONMENT

The basic idea is to formulate the JSSPs with MDP, but when the number of machines becomes larger, the action space will be too large when using only one agent in the model, so it is expected that training of DRL will become a challenging task. As a result, this work proposes to model JSSP as an extension of MDP called Multi-Agent Markov decision (MMDP) [28] to model JSSPs.

The main difference between MDP and MMDP is that MDP has only one agent to deal with all machines, so the agent is allowed to view the whole picture of scheduling. In MMDP, m agents are available to associate with m machines, so that MMDP modeling is more flexible, and avoids to re-plan the schedule when machine breakdown occurs. Moreover, the central control of one agent will limit the computation. We briefly describe JSSP with MMDP as follows.

1) Agent: Each agent is associated with a specific machine.
2) State: The states in this work comprises process time matrix, Boolean matrix of the assigned job to each agent (machine) and Boolean matrix of completed job. The three Boolean matrices represent three different channels of a state that will become the input of a CNN. This representation is similar to the RGB channels of an image.
3) Action: The action in this work is corresponding to a dispatching rule, such as shortest processing time (SPT) and first in first out (FIFO).
4) Reward: This work uses process time of the selected job, remaining process time of the job and the comparison of the smallest makespan as our rewards. When the smallest makespan is unavailable, we replace it with a small number.

B. LEARNING ALGORITHM

In Q-learning, Q-value is the main element of the learning algorithm. For small-scale problems, the Q-value pair $Q(s, a)$ could be kept in computer memory. However, the number of Q-value pairs will be enormous or even infinite for large-scale problems. Thus, it is infeasible to keep all possible pairs in the memory, and estimation of Q-value is important when applying Q-learning. Besides, our goal is to learn the behavior instead of only using greedy policy, as used in ϵ-greedy [4], [21], explaining why we propose to use the Actor–Critic network to obtain both value estimation and behavior estimation.

Critic network is in charge of value estimation. When agents select an action and encounter a new state, critic network estimates possible action values conditional on current state to help agent make the action in the next step. We use CNN architecture to estimate Q-value pairs, $Q(s, a)$, and a typical CNN architecture comprises convolution layer, nonlinear activation layer and fully connected layer. The convolution layer we use is partial convolution filter $(1 \times n)$ rather than the original convolution filter $(n \times n)$ as the goal is to find the relation among operations rather than machines.

The actor network is similar to critic network, but actor network estimates the behavior distribution when the agent arrives at a new state and takes an appropriate action based on the critic that given by critic network in the previous step. Instead of using greedy policy, learning how to behave or

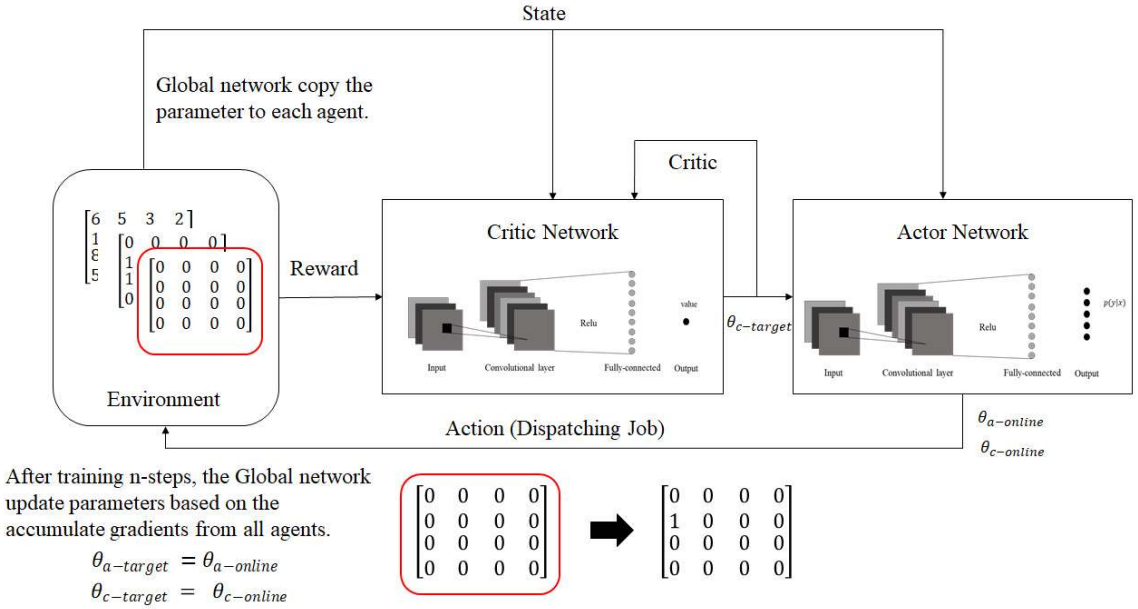

FIGURE 3. JSSPs on actor-critic DRL.

select actions in different environments can make the exploration of agents more efficiently.

C. DRL ON JSSP

Learning of DRL is to via continuous interaction with JSSP environment. Fig. 3 shows the complete JSSP training flow of DRL. This work proposes to use three matrices in the environment to represent the states of JSSPs, including process time matrix, assignment matrix, and activation matrix, respectively. Resource or each machine is assigned to one agent, so that the environment involves multiple agents. In the implementation, the action can be simple dispatching rules, such as SPT/LPT. Fig. 3 is an example that the agent selects SPT as the action and dispatches the shortest processing time job that the machine can process.

In the beginning, state and the weights of the two networks are initialized; meanwhile, initial state is sent to the actor network to let agent take an action, namely, dispatching the first job. Once dispatching the first job is completed, the state transits to next state, and the environment gives a reward to critic network depending on process time or remaining time of job at the same time. Then, critic network gives a critic of the action about whether this job dispatching performs well in this schedule. Critic is the most important part in the learning phase, since actor network modifies the dispatching policy based on critic. Once all jobs are completed, the environment gives a reward of maximum makespan in the end and resets the state to initial state. Subsequently the agent will restart from initial state. The training will not be terminated until the number of iterations exceeds the specified threshold.

Update rule in the Deep Learning is based on loss function. In DRL, the loss function involves two parts, actor loss and critic loss. Critic loss function is listed in Equation (2).

$$L_c(\theta_c) = (r + \gamma \max_{a'} Q(s', a', \theta_c) - Q(s, a, \theta_c))^2, \quad (2)$$

where θ_c is the parameters for critic network. Just like Q-learning, the goal is to make our estimation of state-action $Q(s, a, \theta_c)$ to be more close to the target value, $r + \gamma \max_{a'} Q(s', a', \theta_c)$, which includes immediate reward and the maximum value estimation of next state. On the other hand, the actor loss function is defined in Equation (3).

$$L_a(\theta_a) = \sum \log \pi(a|s, \theta_a) L_c(\theta_c), \quad (3)$$

where θ_a is the parameters for actor network. In Equation (3), the value of $L_a(\theta_a)$ could be viewed as the degree of surprising when the agent takes an action a. If an action a is less likely taken, but obtained a high reward, the value of $L_a(\theta_a)$ will increase, meaning that the reward is out of our expectation. It is obvious that the loss function of actor $\sum \log \pi(a|s, \theta_a) L_c(\theta_c)$ is correlated to critic loss function $L_c(\theta_c)$, so the update of critic network will be important in DRL.

D. PARALLEL TRAINING

When training RL agent in a dynamic environment, the strong correlation between state $s(t)$ and $s(t + 1)$ makes the immediate update of agent less efficient. Agent needs to execute more iterations in the environment to find more samples of states. Therefore training DRL for JSSPs will encounter the same problem. The DQN algorithm tackles this problem with a huge experience data pool known as experience replay. The experiences of agent at each step will be stored in a large dataset and agent will update by sampling from experience pool rather than using immediate situation.

Experience replay has been proven successfully to reduce the variance of the update of agents, but JSSPs involves more than one agent, namely machine in this work, making the experience of agent become enormous and hard to be stored. In this work, we propose a parallel training method, combining asynchronous update [23] and deep deterministic policy gradient(DDPG) [20]. Notably, DDPG is an extension of DQN on actor-critic [20], [23] and asynchronous update has empirically shown that it could efficiently and stably train the model in multi-agent DRL model, explaining why this work proposes to use asynchronous update mechanism.

When updating the network in DQN, we use a separate network, namely target network, to generate target Q-value from the parameters of previous network. Fixing target network in n-step can improve the stability of network. Therefore, avoiding to have a divergence policy, we can add a target actor network in actor-critic network. The parameter of target network is copied from the earlier version of online network, thus our critic loss function of online network will slightly modify to become Equation (4).

$$L_c(\theta_{c_{online}}) = (r + \gamma \max_{a'_{target}} Q(s', a', \theta_{c_{target}}) - Q(s, a, \theta_{c_{online}}))^2 \tag{4}$$

Note that in $\max_{a'_{target}} Q(s', a', \theta_{c_{target}})$, the agent will evaluate the action in state s' deriving from target network. Just like critic loss function, the actor loss function will modify to:

$$L_a(\theta_{a_{online}}) = \sum \log \pi(a|s, \theta_{a_{online}}) L_c(\theta_{c_{online}}) \tag{5}$$

In every n-step we will set that:

$$\theta_{a_{target}} = \theta_{a_{online}} \tag{6}$$

$$\theta_{c_{target}} = \theta_{c_{online}} \tag{7}$$

The idea to delay the update of target network is similar to continuous training on a label in supervised learning. Although changing target may immediately accelerate learning, a stable target can avoid divergence and help train the DDPG more stationary.

In MMDP, multiple agents are involved in the model, so when an agent takes an action, its state will change, which will influence other agents. Thus, how to update each agent and coordinate them will be the main challenge. Fig. 4 shows the concept of Asynchronous DDPG for the MMDP on JSSP [23], which indicates that each agent of machines will copy the parameter from global network and train their network in their own statement. The global network will update the parameters based on the accumulate gradients from all the agents. Notably, the agents explore different parts of the environment and this can avoid each agent being affected by other agents as each agent explores its own environment simultaneously. Algorithm 1 shows the entire update algorithm of Asynchronous DDPG.

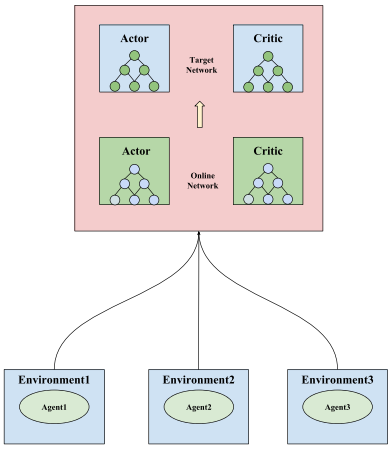

FIGURE 4. Asynchronous DDPG for the MMDP on JSSP.

Algorithm 1 Update Algorithm of Asynchronous DDPG

Input: $\theta_{c_{online}}$ and $\theta_{a_{online}}$
Output: $\theta'_{c_{online}}$ and $\theta'_{a_{online}}$

1 At time t
2 Step 1. Copy global network parameter θ to each agent.
3 Step 2. Let agents interact with their environments and dispatch a job.
4 Step 3. Once the job is finished, and a reward r_t is obtained, update their network.
5 Step 4. Accumulate gradients:

$$d\theta'_{c_{online}} \leftarrow d\theta'_{c_{online}} + \nabla_{\theta'_{c_{online}}}(L_{c_{online}}(\theta'_{c_{online}})) \tag{8}$$

$$d\theta'_{a_{online}} \leftarrow d\theta'_{a_{online}} + \nabla_{\theta'_{c_{online}}}(L_{c_{online}}(\theta'_{c_{online}})) \tag{9}$$

6 Step 5: Perform asynchronous update of θ using $d\theta'_{c_{online}}$ and $d\theta'_{a_{online}}$

IV. EXPERIMENTS

In the experiments, we evaluate our proposed DRL model on some benchmark problems of operations research. Traditionally we could dispatch all the jobs by optimization method or meta-heuristic approaches to obtain a complete scheduling, which are called dispatching rules of global view. However, many unexpected events or requirements may influence the jobs that have been scheduled, and these approaches that consider whole production line require to re-schedule to obtain new scheduling. In contrast, we view job shop problem as a decision problem that we can dispatch job at every moment and easily deal with any stochastic event happened.

A. ENVIRONMENT AND MODEL SETTING

The testing environments are initialized by the instances provided by OR-library. Table 1 shows the instances that are used in the experiments. Moreover, to simulate a dynamic environment, we include randomness factors in the environment. Once initialization is completed, randomness is injected to the environment, such as shuffling the order of jobs that assigned

TABLE 1. Test instance.

Test Instance	Source
Ft06 (6×6) , Ft10 (10×10)	Fisher [11]
La11 – 15 (20×5)	Lawrence [19]
Orb01 - 09 (10×10)	Applegate and Cook [1]

to machines and increasing or decreasing the process time of job. The purpose is to evaluate whether the proposed model could deal with static instances that have seen as well as unexpected instances.

The deep architecture we use in the experiments is actor-critic neural network with one convolution layer and one fully connected layer. In 10×10 and 20×5 instances, 16 filters with kernel size (1, 2) are used in convolution layer and 100 neurons are used in a hidden layer of fully connected part. As for the 6×6 instance, we use 8 filters with kernel size (1, 1) in convolution layer and 100 neurons in the hidden layer of fully connected part. Normalization and zero padding are both applied to all instances. The optimizer of the neural network is Kingma and Adam [17] and the hyperparameters beta is 0.9, epsilon is 1e-8 and the learning rate is 1e-4 to 1e-6 according to the size of instance. The number of epochs is 50 to 100 epochs based on the size of instances. In the model design, the model explores other solutions to avoid to converge to local optimum in the middle of training process.

B. COMPARISON WITH DISPATCHING RULES

Our proposed DRL model uses simple dispatching rules of local view as our actions, so the experiments compare our model with local dispatching rules of local view and global optimum, which is obtained by python library, OR-Tools.[1] The evaluation criteria in the experiments is the makespan and the goal is to minimize makespan. Two rewards that we design for DRL training on JSSPs will be tested in our reinforcement environment. The first reward is called fixed reward as shown in Equation (10).

$$r_i = 1 - \lambda \times t_p$$
$$r_f = \frac{\gamma}{|y^* - y_f|}, \tag{10}$$

where t_p is process time, λ is a constant, y^* is the optimal result, y_f is the final makespan obtained from DRL. Immediate reward returns once DRL makes actions, and fixed reward will return when all jobs are finished. The second reward is called combined reward as listed in Equation (11).

$$r_c = 1 - \lambda \times t_p + \frac{\gamma}{|y^* - \hat{y}|}, \tag{11}$$

where $\hat{y}$ is the predicted makespan. Combined reward fixes reward every time when DRL makes actions. The learning curve for DRL is presented in Fig. 5, in which instance la11 is

[1]Google Optimization Tools: https://github.com/google/or-tools

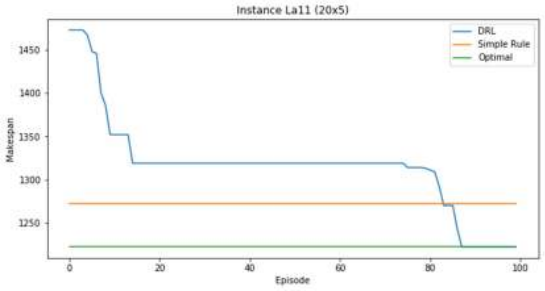

FIGURE 5. Learning phase of DRL.

used in the experiments. Our DRL model smoothly advances the solution and achieves the global optimum after 80 episodes. Besides, we compare the proposed model with several typical algorithms of JSSPs. The comparison algorithms are listed as follows. To the best of our knowledge, the proposed work is the first study attempting to apply DRL to a dynamic environment on JSSPs problem, explaining why we only compare with traditional RL method.

1) Simple dispatching rules: SPT/LPT (Shortest/Longest process rule) and FIFO(First in first out) [14]
2) Meta-heuristic: GA: Combining ant colony optimization algorithm with the taboo search algorithm [16]
3) Traditional RL: Generic Multi-Agent Q-learning (QL) [22]
4) Optimal solution (OPT)

Table 2 shows the experimental results, which point out that our proposed model with two different rewards are competitive to optimum solution of static benchmark and outperforms dispatching rules of local view, in which "Proposed 1" uses combined reward, while "Proposed 2" uses fixed reward. It is noted that optimum solution is the best one, so it could be viewed as the upper bound of any approach. We conclude that the proposed DRL model is training on a stochastic environment, giving the base to improve the generalization ability of our proposed model.

We further investigate the impact of different rewards on our proposed method, and we summarize the performances for these two rewards on all the tasks in Fig. 6, which indicates that using combined reward in our DRL model normally leads to better results on makespan as compared with using fixed reward in the end of training episode. The combined reward uses the expectation of future makespan that scheduled by the current parameters of our model. Continuously making a comparison to optimal solution can ensure that our model takes the action that affects future performance. Nevertheless, in dynamic environment the optimal solution is difficult to be found, so we replace the optimal solution by the smallest makespan that our model has met in the environment. It is apparent that the design of reward is the key to the success of DRL, and this work considers makespan as it is the most important part in manufacturing scheduling. Different requirements would lead to different rewards, and the design of other rewards is considered as one of our future works.

TABLE 2. Experimental results.

Instance	Metric	OPT	FIFO	LPT	SPT	ACO with TS	QL	Proposed 1	Proposed 2
Ft06	Makespan	55	65	77	88	55	57	57	58
(6 × 6)	Scheduling score (%)	-	84.6	71.43	62.50	100	96.49	96.49	94.83
Ft10	Makespan	930	1184	1295	1074	960	960	1041	1097
(10 × 10)	Scheduling score (%)	-	78.55	71.81	86.59	96.88	96.88	89.34	84.78
La11	Makespan	1222	1272	1467	1473	1222	1222	1222	1222
(20 × 5)	Scheduling score (%)	-	96.07	83.30	82.96	100	100	100	100
La12	Makespan	1039	1039	1240	1203	1039	1039	1111	1134
(20 × 5)	Scheduling score (%)	-	100	83.79	86.37	100	100	93.52	91.62
La13	Makespan	1150	1199	1230	1275	1150	1150	1181	1239
(20 × 5)	Scheduling score (%)	-	95.91	93.50	90.20	100	100	97.38	92.82
La14	Makespan	1292	1292	1434	1427	1292	1292	1292	1292
(20 × 5)	Scheduling score (%)	-	100	90.10	90.54	100	100	100	100
La15	Makespan	1207	1587	1612	1339	1207	1259	1288	1339
(20 × 5)	Scheduling score (%)	-	76.06	74.88	90.14	100	95.87	93.71	90.14
Orb01	Makespan	1059	1369	1410	1478	1059	1154	1211	1211
(10 × 10)	Scheduling score (%)	-	77.36	75.11	71.65	100	91.77	87.45	87.45
Orb02	Makespan	888	1007	1293	1175	888	931	1002	1002
(10 × 10)	Scheduling score (%)	-	88.18	68.68	75.57	100	95.38	88.62	88.62
Orb03	Makespan	1005	1405	1430	1179	1005	1095	1150	1181
(10 × 10)	Scheduling score (%)	-	71.53	70.28	85.24	100	91.78	87.39	85.10
Orb04	Makespan	1005	1325	1415	1236	1005	1068	1132	1176
(10 × 10)	Scheduling score (%)	-	75.85	71.02	81.31	100	94.10	88.78	85.46
Orb05	Makespan	887	1155	1099	1152	887	976	1045	1046
(10 × 10)	Scheduling score (%)	-	76.80	80.71	77.00	100	90.88	84.88	84.80
Orb06	Makespan	1010	1330	1474	1190	1010	1064	1106	1190
(10 × 10)	Scheduling score (%)	-	75.94	68.52	84.87	100	94.92	91.32	84.87
Orb07	Makespan	397	475	470	504	397	424	468	460
(10 × 10)	Scheduling score (%)	-	83.58	84.47	78.77	100	93.63	84.83	86.30
Orb08	Makespan	899	1225	1176	1107	899	956	1022	1105
(10 × 10)	Scheduling score (%)	-	73.39	76.45	81.21	100	94.04	87.96	81.36
Orb09	Makespan	934	1189	1286	1262	934	996	1082	1100
(10 × 10)	Scheduling score (%)	-	78.55	72.63	74.01	100	93.78	86.32	84.91
Average	Makespan	936	1132	1213	1135	938	978	1026	1053
	Scheduling score (%)	-	83.27	77.29	81.18	99.80	95.59	91.12	88.94

Exploration is an important technique in RL. A commonly used approach is ϵ-greedy strategy. Given a threshold value for exploration, say ρ, we draw a random number x from a uniform distribution, and check whether the condition $x > \rho$ holds. If the condition is true, we determine the action at time t, namely a_t, using the following equation to perform exploration.

$$a_t \leftarrow \text{uniform}(0, n),$$

where n is the size of action space. It is worth mentioning that when $a_t = 0$, the agents do noting, indicating that no job is available for them. Otherwise, we use the following equation to determine a_t based on exploitation strategy.

$$a_t \leftarrow \arg\max_a b(.|s_t),$$

where $b(.|s_t)$ is the behavior policy (e.g., output of the actor network), $\forall a \in A, s \in S$.

We conduct experiments to investigate the effectiveness of ϵ-greedy algorithm in the proposed method. Fig. 7 shows the experimental results. The results indicate that discarding exploration in training time of DRL model will lead DRL model to find a local optimal solution. In RL, exploration means trying the action that agent does not select before, whereas exploitation is to make the best decision

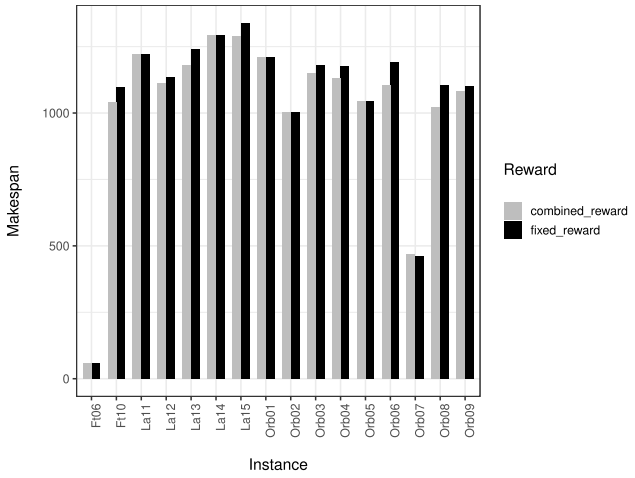

FIGURE 6. Combined reward vs. fixed reward.

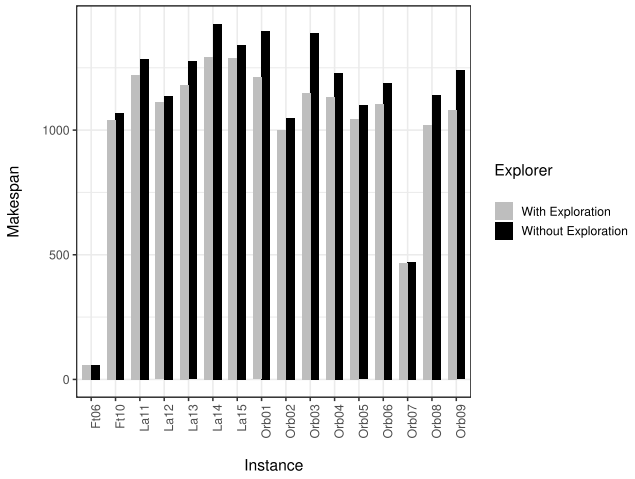

FIGURE 7. With exploration vs. without exploration.

given current information. The experimental results show that applying exploration strategy to JSSP to find more different scheduling solutions can avoid our model to fall into a local optimal solution.

C. FLEXIBILITY

In a real-world environment, many unexpected conditions such as machine breakdown or accident events may occur, making it possible to change processing time or permutation of machine ordering. Mathematical programming and meta-heuristic approaches have to re-schedule to deal with these problems, and it is expected that re-scheduling is a computation-intensive task. In contrast, the proposed DRL model can train on stochastic environments and act rapidly when accident events occur.

It is noted that a RL model designed for a dynamic environment is unavailable in the previous works. Moreover, the algorithmic operation for existing studies that focused on static environment is not really designed for dynamic environments. To cope with a dynamic environment, these methods

TABLE 3. Comparison of execution time.

	SPT	LPT	Proposed Method	OPT
10×10	0.12 sec	0.12 sec	0.9 sec	>5 hrs
5×20	0.4 sec	0.4 sec	1.1 sec	>5 hrs
6×6	0.015 sec	0.015 sec	0.03 sec	2.5 sec

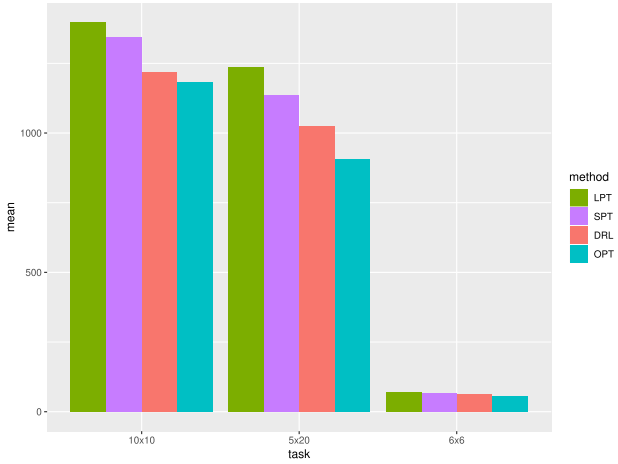

FIGURE 8. Performance comparison under stochastic events.

have to decompose the dynamic environment into different static environments; therefore the designed algorithms have to be retrained once the new environment is observed. Therefore, these methods are not listed as the comparison method, explaining why the experiments compare the proposed DRL model with simple dispatching rules and the optimal solution.

To evaluate the flexibility of our proposed DRL, we set some random breakdowns and changing some process time of job during dispatching. Each size (6×6, 10×10, 5×20) of instance will repeat 100 times from different initial parameters. The execution times and experimental results are presented in Table 3 and Fig. 8, respectively.

Table 3 and Fig. 8 show that small-size instance (6×6) can get the optimal solution in short run, but when the number of machines or the jobs increases, the optimal solution will be hard to be found. The re-schedule times for 10×10 and 5×20 instances are both more than 5 hours. It is impossible to take more than 5 hours to fix the immediate changing, not to mention that the size of job shop problems in real world is much bigger than this.

In contrast, our proposed DRL model achieves smaller makespan and tightness variation than simple dispatching rules. Furthermore, the time for the proposed DRL model to dispatch all the jobs is also competitive as compared with simple dispatching rules. The proposed DRL learns how to combine these simple dispatching rules together based on different situations. Although one needs to train a DRL model on an instance of fixed size for a while, a trained model can easily finish scheduling in a few seconds or minutes and execute on similar instances.

TABLE 4. DRL on 20 × 15 instance.

Instance	Metric	SPT/LPT	Proposed Method	Optimal
Abz7	Makespan	849	757	656
	Scheduling score (%)	77.27	86.66	-
Abz8	Makespan	929	821	645
	Scheduling score (%)	69.43	78.56	-
Abz9	Makespan	887	857	661
	Scheduling score (%)	74.52	77.13	-
Average	Makespan	888	812	654
	Scheduling score (%)	73.74	80.78	-

We also evaluate our model on the largest instance in OR–library (20 × 15) and the experimental results are presented in Table 4. The proposed DRL still yields better performances than dispatching rules even though the search space becomes huge.

V. CONCLUSION

In this work we propose an actor–critic architecture with DRL to cope with classic job shop problems. The action space we chose is a combination of simple dispatching rules which can easily be executed in any complex environment. We also propose a parallel training technique to ensure the convergence of the model. The testing environment is setting on OR library instances and many random elements are included in the evaluation. The final results demonstrate that our DRL model could deal with unexpected incidents, such as machine breakdown and sudden additional order. Besides, the quality of the solutions that are found by our model is also comparative. Our model outperforms traditional dispatching rules and executes almost as fast as simple dispatching rules. Therefore the complexity of our model is also insured.

Several future works are possible. First, the design of reward in this work is based on handcraft. We need to adjust the reward when facing different environments. Thus learning of reward might be another interesting issue. Second, we only focus on JSSP in this work, and transferring our DRL model to other scheduling problems is a possible research direction. Finally, the input of the neural network has to be fixed size in our proposed model. When facing larger instances, our method needs to increase parameters in the network and re-train the network to fit the new instances. It is a challenging task in deep learning that could accept variable sizes of the inputs, and this is another future work. Third, it is a valuable research to consider the factors or patterns of scheduling problems to generate the simulation. Just like Atari learning environment that has been widely used by researchers to develop DRL algorithms on games, this can be a fundamental environment for the training and evaluation of DRL algorithms on scheduling problems.

REFERENCES

[1] D. Applegate and W. Cook, "A computational study of the job-shop scheduling instance," *ORSA J. Comput.*, vol. 3, no. 2, pp. 149–156, 1991, doi: 10.1287/ijoc.3.2.149.

[2] F. Abdullah Asuhaimi, S. Bu, P. Valente Klaine, and M. A. Imran, "Channel access and power control for energy-efficient delay-aware heterogeneous cellular networks for smart grid communications using deep reinforcement learning," *IEEE Access*, vol. 7, pp. 133474–133484, 2019.

[3] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robot. Auto. Syst.*, vol. 33, nos. 2–3, pp. 169–178, Nov. 2000.

[4] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA J. Comput.*, vol. 6, no. 2, pp. 154–160, 1994.

[5] J. E. Beasley, "OR-library: Distributing test problems by electronic mail," *J. Oper. Res. Soc.*, vol. 41, no. 11, p. 1069, Nov. 1990.

[6] C. Boutilier, "Sequential optimality and coordination in multiagent systems," in *Proc. IJCAI*, vol. 99, 1999, pp. 478–485.

[7] Y.-L. Chang, H. Matsuo, and R. S. Sullivan, "A bottleneck-based beam search for job scheduling in a flexible manufacturing system," *Int. J. Prod. Res.*, vol. 27, no. 11, pp. 1949–1961, 1989.

[8] R. Cui, C. Yang, Y. Li, and S. Sharma, "Adaptive neural network control of AUVs with control input nonlinearities using reinforcement learning," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 6, pp. 1019–1029, Jun. 2017.

[9] Y. Cui, T. Matsubara, and K. Sugimoto, "Pneumatic artificial muscle-driven robot control using local update reinforcement learning," *Adv. Robot.*, vol. 31, no. 8, pp. 397–412, Apr. 2017.

[10] B. Cunha, A. M. Madureira, B. Fonseca, and D. Coelho, "Deep reinforcement learning as a job shop scheduling solver: A literature review," in *Proc. Int. Conf. Hybrid Intell. Syst.*, A. M. Madureira, A. Abraham, N. Gandhi, and M. L. Varela, Eds. Cham, Switzerland: Springer, 2020, pp. 350–359.

[11] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *Industrial Scheduling*, J. F. Muth and G. L. Thompson, Eds. Upper Saddle River, NJ, USA: Prentice-Hall, 1963, pp. 225–251.

[12] E. M. Frazzon, A. Albrecht, M. Pires, E. Israel, M. Kück, and M. Freitag, "Hybrid approach for the integrated scheduling of production and transport processes along supply chains," *Int. J. Prod. Res.*, vol. 56, no. 5, pp. 2019–2035, Mar. 2018.

[13] Y. Fu, J. Ding, H. Wang, and J. Wang, "Two-objective stochastic flow-shop scheduling with deteriorating and learning effect in industry 4.0-based manufacturing system," *Appl. Soft Comput.*, vol. 68, pp. 847–855, Jul. 2018.

[14] T. Gabel and M. Riedmiller, "On a successful application of multi-agent reinforcement learning to operations research benchmarks," in *Proc. IEEE Int. Symp. Approx. Dyn. Program. Reinforcement Learn.*, Apr. 2007, pp. 68–75.

[15] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.

[16] K.-L. Huang and C.-J. Liao, "Ant colony optimization combined with taboo search for the job shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 4, pp. 1030–1046, Apr. 2008.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[18] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[19] S. Lawrence, "Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," Graduate School Ind. Admin., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 1984.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971

[21] A. S. Manne, "On the job-shop scheduling problem," *Oper. Res.*, vol. 8, no. 2, pp. 219–223, 1960.

[22] Y. M. Jimenez, "A generic multi-agent reinforcement learning approach for scheduling problems," Ph.D. dissertation, Vrije Univ. Brussel, Brussels, Belgium, 2012, p. 128.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

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

[25] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9839–9849.
[26] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. New York, NY, USA: Springer, 2016.
[27] E. Pinson, "The job shop scheduling problem: A concise survey and some recent developments," in *Scheduling Theory and Its Applications*, P. Chretienne, E. G. Coffman, Jr., J. K. Lenstra, and Z. Liu, Eds. New York, NY, USA: Wiley, 1995, pp. 277–294.
[28] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.
[29] I. Sabuncuoglu and M. Bayiz, "Job shop scheduling with beam search," *Eur. J. Oper. Res.*, vol. 118, no. 2, pp. 390–412, Oct. 1999.
[30] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, *arXiv:1703.03864*. [Online]. Available: http://arxiv.org/abs/1703.03864
[31] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
[32] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.
[33] N. Stricker, A. Kuhnle, R. Sturm, and S. Friess, "Reinforcement learning for adaptive order dispatching in the semiconductor industry," *CIRP Ann.*, vol. 67, no. 1, pp. 511–514, 2018.
[34] R. S. Sutton, A. G. Barto, and F. Bach, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[35] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019.
[36] K. Wang, H. Luo, F. Liu, and X. Yue, "Permutation flow shop scheduling with batch delivery to multiple customers in supply chains," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 10, pp. 1826–1837, Oct. 2018.
[37] 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.
[38] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[39] C. Wu, B. Ju, Y. Wu, X. Lin, N. Xiong, G. Xu, H. Li, and X. Liang, "UAV autonomous target search based on deep reinforcement learning in complex disaster scene," *IEEE Access*, vol. 7, pp. 117227–117245, 2019.
[40] W. Yang, Y. Zhang, C. Yang, Z. Zuo, and X. Wang, "Online power scheduling for distributed filtering over an energy-limited sensor network," *IEEE Trans. Ind. Electron.*, vol. 65, no. 5, pp. 4216–4226, May 2018.
[41] 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.
[42] D. Zeng and K. Sycara, "Using case-based reasoning as a reinforcement learning framework for optimisation with changing criteria," in *Proc. 7th IEEE Int. Conf. Tools with Artif. Intell.*, 1995, pp. 56–62.
[43] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under industry 4.0," *J. Intell. Manuf.*, vol. 30, no. 4, pp. 1809–1830, Apr. 2019.
[44] X.-L. Zheng and L. Wang, "A collaborative multiobjective fruit fly optimization algorithm for the resource constrained unrelated parallel machine green scheduling problem," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 5, pp. 790–800, May 2018.

CHIEN-LIANG LIU (Member, IEEE) received the M.S. and Ph.D. degrees from the Department of Computer Science, National Chiao Tung University, Taiwan, in 2000 and 2005, respectively. He is currently a Full Professor with the Department of Industrial Engineering and Management, National Chiao Tung University. His research interests include machine learning, data mining, deep learning, and big data analytics.

CHUAN-CHIN CHANG received the M.S. degree in computer science from National Chiao Tung University, Taiwan, in 2018. His research interests include machine learning and data mining.

CHUN-JAN TSENG received the M.S. degree from the Graduate Institute of Logistics Management, National Defense University, Taiwan, in 2007. He is currently pursuing the Ph.D. degree with the Department of Industrial Engineering and Management, National Chiao Tung University, Taiwan. His research interests include machine learning, data mining, and logistics management.