Dynamic Job-Shop Scheduling via Graph Attention Networks and Deep Reinforcement Learning

Chien-Liang Liu, *Senior Member, IEEE*, Chun-Jan Tseng, and Po-Hao Weng

***Abstract*—The dynamic job-shop scheduling problem (DJSSP) is an advanced form of the classical job-shop scheduling problem (JSSP), incorporating dynamic events that make it even more challenging. This article proposes a novel approach involving deep reinforcement learning and graph neural networks to solve this optimization problem. To effectively model DJSSP, we use a disjunctive graph, designing specific node features that reflect the unique characteristics of JSSP with machine breakdowns and stochastic job arrivals. Our proposed method can dynamically adapt to the occurrence of disruptions, ensuring that it accurately reflects the current state of the environment. Furthermore, we use the attention mechanism to prioritize crucial nodes while discarding irrelevant ones. This study proposes a model that applies graph attention networks to learn node embeddings, serving as input for the actor–critic model. The proximal policy optimization is then utilized to train the actor–critic model, which assists the model in learning the scheduling of job operations for machines. We conducted extensive experiments in static and public environments. Experimental results indicate that our method is superior to current state-of-the-art methods.**

***Index Terms*—Deep reinforcement learning (DRL), graph attention networks (GATs), job-shop scheduling problem (JSSP), proximal policy optimization (PPO).**

I. Introduction

SCHEDULING is crucial in smart manufacturing, as it can significantly improve profitability and optimize resource utilization during production [1]. The job-shop scheduling problem (JSSP) is a well-established problem in manufacturing scheduling. Addressing this problem effectively can significantly improve industrial productivity and equipment utilization.

Various solutions to JSSP have been proposed, and these can be broadly categorized into the following three types: exact algorithms, approximate algorithms, and heuristic methods [2]. JSSP can be formulated as an optimization problem with several constraints, so exact methods, such as mathematical programming [3] and branch-and-bound [4], can obtain optimal solutions. However, this combinatorial optimization problem is NP-hard [5], which means that problems become intractable with increasing problem size. Approximation algorithms aim to find solutions close to optimal in a short period. Although they do not always guarantee the optimal solution, they offer high-quality alternatives faster than exact algorithms. This makes them especially beneficial for addressing larger problem instances. On the other hand, heuristics generate solutions rooted in guiding principles or intuitive criteria, bypassing exhaustive exploration of all possible solutions. While they are efficient, heuristics do not guarantee an optimal result.

Most existing methods for JSSP assume that the production environment is stationary and the production environment states are predetermined. However, many unexpected events, such as machine failure, order cancelation, insertion of new jobs, and intervention schedules, usually occur in the production environment. The extension of JSSP that considers these dynamic events is known as the dynamic job-shop scheduling problem (DJSSP) [6], and effectively addressing it can significantly improve productivity in actual manufacturing. DJSSP is a challenging problem since uncertainty is involved in the production environment, making it impossible to consider future events in advance. The scheduling methods have to schedule the jobs according to current environment information. It is also impossible to arbitrarily alter the job sequences as historical sequences cannot be changed while future events are unknown. DJSSP is more challenging than JSSP due to uncertainty factors, but is worth investigating, as unexpected events are inevitable in a real-world environment.

This article proposes the use of deep reinforcement learning (DRL) to develop the scheduling method. DRL combines reinforcement learning (RL) with deep neural networks to learn complex decision-making strategies from data. RL uses the Markov decision process (MDP) to model the problem, in which

Manuscript received 9 January 2024; revised 18 February 2024; accepted 19 February 2024. This work was supported by the Ministry of Science and Technology, Taiwan, under Grant MOST 109-2628-E-009-009-MY3 and Grant 111-2221-E-A49-083-MY3. Paper no. TII-24-0139. *(Corresponding author: Chien-Liang Liu.)*

Chien-Liang Liu and Po-Hao Weng are with the Department of Industrial Engineering and Management, National Yang Ming Chiao Tung University, Hsinchu 300093, Taiwan (e-mail: clliu@nycu.edu.tw; s0713230.mg07@nycu.edu.tw).

Chun-Jan Tseng is with Management College, National Defense University, Taipei 112305, Taiwan, and also with the Department of Industrial Engineering and Management, National Yang Ming Chiao Tung University, Hsinchu 300093, Taiwan (e-mail: ottolan0728@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2024.3371489.

Digital Object Identifier 10.1109/TII.2024.3371489

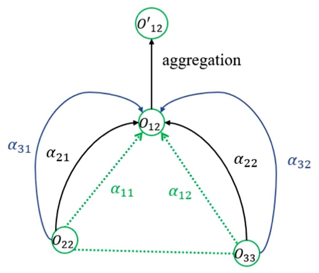

Fig. 1. Graph embedding via GATs with multihead attention, α_{ij} is the attention coefficient.

the future states and past states are independent when given the current state. The MDP assumption is appropriate for modeling DJSSP, as the agent only needs to consider the current environment information to make decisions without using the entire environment information. Furthermore, DRL can benefit from deep neural networks to learn discriminative feature representations, which can help to learn hidden information embedded in the data.

The disjunctive graph is effective in modeling JSSP [7]. Therefore, we propose to use the disjunctive graph to model the problem. In addition, attention is a mechanism that enables the model to focus on the important parts while ignoring the irrelevant ones. Therefore, this article proposes a novel model based on graph attention networks (GATs) [8] to address DJSSP. The proposed method exploits the underlying structure of the DJSSP using a graph to encode jobs and machines, as the goal is to learn the job-machine interactions and optimize scheduling decisions. Moreover, the attention mechanism can help the model pay attention to important nodes while ignoring irrelevant ones, as shown in Fig. 1. More specifically, we encode jobs and machines as graph nodes and incorporate an attention mechanism into the model to enable the model to learn node representations. Subsequently, we use the DRL framework, actor–critic, to learn the policy for the scheduling problem and use proximal policy optimization (PPO) [9] to train the model.

The main contributions of this article are as follows. First, we propose an effective graph embedding to extract features that can well represent the JSSP problem from the disjunctive graph directly. Second, we propose using GAT and DRL to model JSSP in dynamic environments. The architecture of the proposed method is size-agnostic; thus, we can train the model on instances of smaller size and test it directly on large instances. Finally, we evaluate the proposed method and comparison methods using numerous public benchmarks with different problem sizes and synthetic datasets with different processing time distributions. The proposed model exhibits excellent generalizability and outperforms the other alternatives in problems of large problem sizes.

The rest of this article is organized as follows. Section II provides a review of recent methodologies for JSSP, encompassing metaheuristic algorithms, DRL-based approaches, and scheduling techniques that utilize graph neural networks (GNNs). Section III elaborates on the proposed method to address JSSP in depth. The experimental results and subsequent discussion are presented in Section IV. Finally, Section V concludes this article.

II. Related Work

Mixed-integer linear programming and constraint programming are typical exact methods available to deal with JSSP [10]. Although these approaches can obtain optimal solutions, the computational effort scales exponentially with the size of the problem. On the other hand, metaheuristics are approximate methods that can use different strategies to guide the search process to obtain suboptimal solutions. They can maintain a balance between computational time and the quality of solutions. Therefore, many metaheuristics have been developed for JSSP over the past decades. For example, Ahmadian et al. [11] presented a metaheuristic approach for just-in-time JSSP using a variable neighborhood search algorithm. Dehghan-Sanej et al. [12] developed a new simulated annealing algorithm with a discrete harmony search for large-scale JSSP considering reverse flows under uncertainty. Kurdi [13] used a new island genetic algorithm model to deal with the Job Shop Scheduling Problem (JSSP). On the other hand, several soft-computing techniques have been developed to address scheduling problems using RL approaches. Han et al. [14] introduced a novel multistrategy, multiobjective differential evolutionary (DE) algorithm that employs a multistrategy and multicrossover DE optimizer to balance exploration–exploitation tradeoffs effectively. Hu et al. [15] presented an innovative method called CEDE-DRL, a coevolutionary DE framework assisted by DRL. Zhao et al. [16] proposed a hyperheuristic approach using Q-learning to address the energy-efficient distributed blocking flow-shop scheduling problem.

In recent years, the rapid advancement and successful implementation of artificial intelligence algorithms have sparked a growing interest in learning-based approaches to address JSSP [17]. Among learning-based methods, RL has emerged as a powerful method to tackle combinatorial optimization problems and has been successfully applied in various fields [18]. Q-learning, a well-known RL algorithm that selects actions based on the highest Q-value stored in the Q-table, has been demonstrated to improve multiobjective performance in JSSP and effectively handle unpredictable events [19]. Other RL methods, such as PPO [20] and deep deterministic policy gradient [21], have also been used to solve DJSSP and have achieved remarkable success.

DRL integrates RL with deep neural networks to acquire sophisticated decision-making strategies from data. Deep Q-network (DQN) is a well-known DRL algorithm, which has shown superior performance compared to popular dispatching rules in solving DJSSP. Waschneck et al. [22] developed a DQN-based algorithm for scheduling in the semiconductor industry, in which they trained the network with user-defined objectives. Luo [23] applied a DQN-based dynamic scheduling method for the flexible JSSP, considering the insertion of new jobs. The primary DRL methods utilized for JSSP have centered on DQN and its derivatives. Luo et al. [24] introduced a DRL approach based on the DQN algorithm to optimize scheduling decisions in a DJSSP. These studies have shown the potential of DRL to address JSSP and DJSSP.

However, these studies have some limitations. First, these methods are size-dependent, meaning that the training and test instances should be the same size. This constraint limits the generalizability of the developed methods. Second, they do not consider dynamic changes in job arrivals and machine breakdowns, critical factors in real-world scheduling problems. Third, they must be comprehensively compared with existing methods, making it difficult to assess their performance and generalizability [25].

The disjunctive graph is a prominent mathematical model for the comprehensive representation of JSSP [26]. In this graph, the nodes represent operations. Conjunctive edges show both precedence and subsequent constraints between operations, while disjunctive edges highlight machine-sharing constraints. Shakhlevich et al. [27] proposed an adaptive scheduling algorithm based on a mixed graph model. This algorithm incorporates a conflict resolution strategy during its learning and examination phases, showing adaptability in managing various practical constraints. Gholami and Sotskov [28] addressed parallel machine job-shop challenges using a weighted mixed graph paired with a conflict resolution approach.

GNNs are specialized artificial neural networks engineered to harness information from graph data structures. A typical GNN consists of nodes and edges that represent data points and their interrelationships, respectively. GNN learning employs message passing to derive node embeddings by exchanging messages and features between interconnected nodes through learnable parameters. The resulting node embeddings are aggregated and processed via a read-out function, often a dense layer, to produce the final prediction [29]. GNNs can determine node embeddings for various applications when provided with a graph. These embeddings encapsulate the structural nuances of the JSSP graph. Within scheduling, such embeddings empower the scheduling policy to formulate effective scheduling actions that respect the JSSP framework. Consequently, numerous studies have utilized GNN-based DRL techniques to address various scheduling challenges. These include JSSP [30], Distributed JSSP [31], flexible job-shop scheduling [32], and dynamic flexible job-shop [33], [34] problems.

III. Proposed Method

This section specifies the problem under investigation and summarizes the designs of the proposed method, including environment, state, action, reward, agent, and training algorithm. Specifically, we treat the DJSSP as a sequential decision-making process.

A. Problem Specification

JSSP aims to optimize one or more objectives, such as minimizing production costs or reducing makespan, by assigning n jobs to m machines with limited resources. DJSSP is an extension of JSSP, which considers unexpected events during production and is more frequently encountered in actual production than JSSP. The problem under consideration can be expressed as follows: Successive arrival of n jobs $J = \{J_1, \ldots, J_n\}$ that need to be processed on m machines $M = \{M_1, \ldots, M_m\}$. Each job J_i comprises multiple operations $O_i = \{O_{i1}, \ldots, O_{ij,\ldots}\}$. The jth operation of job J_i, O_{ij}, can only be processed on a specific machine. During production, a variety of dynamic events may take place. The unexpected events in this article are machine breakdowns and job arrival times are uncertain. Furthermore, the following assumptions must be satisfied in DJSSP.

1) Each machine can only process at most an operation at one time.
2) An operation can only be processed by one machine at a time.
3) Once an operation has begun, it cannot be interrupted unless there is a machine breakdown.
4) The processing time for each operation remains consistent throughout the entire scheduling process, except in the event of machine breakdown.
5) An operation can only be processed after its preceding operations have been completed.
6) *Machine breakdown:* During production, machines may fail, leading to a certain likelihood of breakdowns that require a repair period.
7) *Stochastic job arrival:* Tasks arrive incrementally on the shop floor, which means that they are not all available at the beginning of the scheduling but instead emerge progressively over time.

B. MDP Formulation

The entire scheduling process can be formulated as an MDP problem. The agent observes the current state s_t at each time step t and takes action a_t, which assigns an unscheduled operation to an idle machine. Subsequently, the environment transitions to the next time step, $t + 1$. This process will proceed iteratively until all operations are scheduled. The corresponding MDP can be defined as follows.

State: Since the agent takes action based on the current state s_t, the state should contain helpful information to help the agent decide. The state s_t in this article at each time step t is a disjunctive graph consisting of nodes and three types of arc, including disjunctive arcs, conjunctive arcs, and solved disjunctive arcs that represent the current status of the solution. Each node in the disjunctive graph has five node features that are listed as follows.

1) *Status:* The status of an operation can be unscheduled, scheduled, or completed and is represented by -1, 0, and 1, respectively.
2) *Processing time:* The duration required to process an operation, which is denoted as p_{ij}.
3) *The number of operations remaining:* The number of operations that have not yet been processed to complete the job. It gradually decreases as operations are completed for the same job.
4) *Degree of completion:* Progress made to complete the job when an operation is completed. For example, if a job comprises two operations with processing times of three

and two units, the degree of completion would be 0.6 and 0.4, respectively.

5) *Estimated completion time of operation*: If an operation is not scheduled, it is assumed that it will be processed at time 0 or immediately after its predecessor has completed.

In a static environment, key parameters, such as processing time, the number of remaining operations, the degree of completion, and the conjunctive arcs, remain constant over time. In contrast, in a dynamic setting, the status, estimated completion time, and resolved disjunctive arcs change in a time-dependent manner. However, with the introduction of uncertainties like stochastic job arrivals and random machine breakdowns, all parameters become time-dependent. This shift occurs because the graph state reflects only the operations from the arrived jobs, introducing variability. Moreover, in dynamic environments subject to machine breakdowns, both the processing time and the degree of completion are subject to change, as breakdowns prolong the processing time of ongoing operations.

Action: The agent selects an operation for an idle machine to process at each time step t, so the action a_t is the scheduling operation. Therefore, the action space comprises all operations. Due to the uncertainties of the environment, the agent needs to make the decision at every time step, both in the static and dynamic environments, to ensure that the condition of machine breakdown and the stochastic job arrivals are being considered. If no machines are ready to process the operation or if no operations are ready to be processed in the decision time step, the action will be null. To make the training process more efficient and ensure consistency of the decision-making process in both static and dynamic environments, transitions that contain null actions will be ignored in the training process.

Transition: Once our agent selects an operation a_t, we will update the directions of the solved disjunctive arcs of that machine based on current temporal relations and engender the new disjunctive graph as the new state s_{t+1}.

Reward: This work aims to train an agent to learn a policy that can make optimal decisions to minimize the makespan of scheduling. In this work, we define a reward function as the difference between the previous makespan $C_{\max}^{t-1}$ and the current makespan $C_{\max}^{t}$, as listed in (1). The rationale for this design is elucidated by considering the discount factor $\gamma = 1$. In this scenario, the return G is defined as $G = C_{\max}^0 - C_{\max}$ [7], as shown in (2). Under these conditions, maximizing G is equivalent to minimizing the makespan $C_{\max}$, which aligns with our objective

$$r_t = C_{\max}^{t-1} - C_{\max}^{t} \tag{1}$$

$$\begin{aligned} G &= r_1 + r_2 + \cdots + r_{|O|} \\ &= C_{\max}^0 - C_{\max}^1 + \cdots + C_{\max}^{|O|-1} - C_{\max}^{|O|} \\ &= C_{\max}^0 - C_{\max}. \end{aligned} \tag{2}$$

C. Disjunctive Graph

The JSSP can be formally described by a disjunctive graph $G = (V, C \cup D)$, where V is a set of nodes representing the operations of jobs, C is a set of directed arcs (conjunctions) representing the precedence constraints among operations, and D is a set of undirected arcs (disjunctions) representing operations processed on the same machines, as shown in Fig. 2. Fig. 2(a) shows the initial JSSP instance with conjunctive arcs displayed as solid lines and disjunctive arcs as dotted lines. Each machine clique is distinctly color-coded for better differentiation. The complete solution is presented in Fig. 2(b) with all disjunctive arcs directed, demonstrating the final job scheduling sequence. Solving an instance of JSSP is equivalent to determining the direction of each disjunction arc.

D. Graph Attention Networks

In a GNN architecture, the goal is to learn the node embeddings that can help subsequent tasks. GNN uses a message-passing scheme to iteratively update the node representation, in which the update of each node is based on the aggregation of messages from nearby nodes. Previous studies [35], [36] have used GNN variants, the graph convolution network (GCN), and the graph recurrent network, to deal with scheduling problems. These methods give equal weights to their neighboring nodes, but different nodes should receive different weights. For example, the operation that should be processed on the same machine (machine-sharing constraint) is more important than its previous operation (procedure constraint).

It is difficult to predetermine the node weights manually, so this work proposes to use attention to learn the weights during model training. Therefore, we propose to use GATs to learn graph embeddings, as GATs can automatically learn the importance of different nodes by applying the attention mechanism.

The entire embedding process can be defined as the following steps. First, the attention coefficient e_{ij} between node i and node j is obtained based on the following equation:

$$e_{ij} = \text{LeakyReLU}(f_\theta([\mathbf{W}x_i || \mathbf{W}x_j])). \tag{3}$$

In (3), x_i and x_j represent the node features of node i and node j, respectively. In addition, $\mathbf{W}$ is a linear transformation, and $||$ represents the concatenation operator. Subsequently, the concatenation of $\mathbf{W}x_i$ and $\mathbf{W}x_j$ becomes the input of a single-layer feed-forward neural network (FNN) f_θ, where θ is the weight of FNN and the activation function is LeakyReLU [37], which allows a small, nonzero gradient when the input is less than zero. Finally, the coefficients are normalized across its neighbors through Softmax function as listed in (4), in which $\mathcal{N}(i)$ denotes the neighbors of node i

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\Sigma_{q \in \mathcal{N}(i)} \exp(e_{iq})} \quad \forall j \in \mathcal{N}(i). \tag{4}$$

The coefficient α_{ij} denotes the importance of the neighbor x_j to the node x_i. Therefore, we can aggregate the features from its neighbors considering their importance, as listed in (5), in which σ is an activation function

$$x_i' = \sigma\left(\Sigma_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W} x_j\right). \tag{5}$$

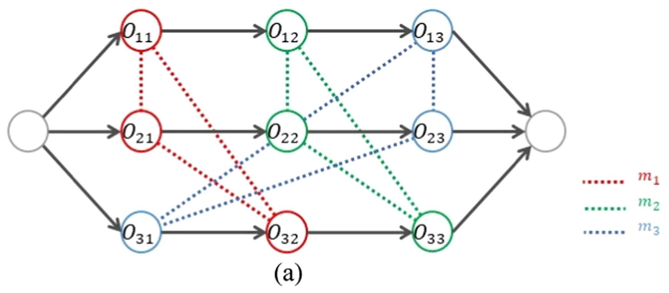

Fig. 2. Disjunctive graph representation for JSSP. (a) JSSP instance. (b) Complete solution.

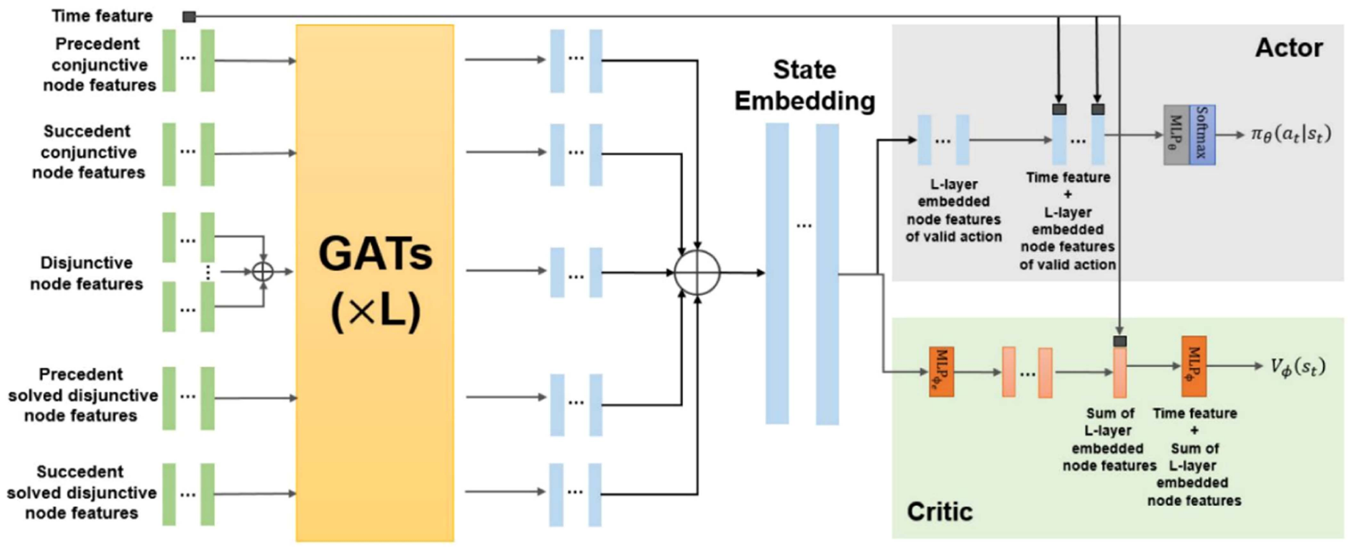
Time feature
Precedent conjunctive node features
Succedent conjunctive node features
Disjunctive node features
Precedent solved disjunctive node features
Succedent solved disjunctive node features
GATs (×L)
State Embedding
Actor
L-layer embedded node features of valid action
Time feature + L-layer embedded node features of valid action
MLP$_\theta$
Softmax
$\pi_\theta(a_t|s_t)$
Critic
MLP$_\phi$
Sum of L-layer embedded node features
Time feature + Sum of L-layer embedded node features
$V_\phi(s_t)$

Fig. 3. Network architecture.

E. Agent

The entire network architecture of our proposed agent is presented in Fig. 3. It comprises L layers of GATs and an actor–critic-based decision-making module. The agent makes the decision based on the observed state, which means that the agent takes the state as the input, and the output is the action. As mentioned in the previous section, we have designed several types of arcs in the state representation. Therefore, to take advantage of the graph state, we embed our graph nodes through different arcs through GATs to obtain the state embedding as shown in the following equation:

$$h_a^{(L)} = \left(\mathrm{ELU}\left(\mathrm{GAT}_\delta(h_{v_{\mathrm{pc}}}^{(L)})\right) \| \mathrm{ELU}\left(\mathrm{GAT}_\delta(h_{v_{\mathrm{sc}}}^{(L)})\right) \| \right.$$
$$\mathrm{ELU}\left(\mathrm{GAT}_\delta\left(\sum_{i\in V_d} h_i^{(L)}\right)\right) \| \mathrm{ELU}\left(\mathrm{GAT}_\delta(h_{v_{\mathrm{pd}}}^{(L)})\right) \|$$
$$\left.\mathrm{ELU}\left(\mathrm{GAT}_\delta(h_{v_{\mathrm{sd}}}^{(L)})\right)\right). \tag{6}$$

The proposed method uses multiple arcs to denote the relationship between nodes, so we use five different features as input to the model. In (6), $h_{v_{\mathrm{pc}}}$ and $h_{v_{\mathrm{sc}}}$ represent the features of the precedent and the succeeding conjunctive nodes of the node v. In contrast, $h_{v_{\mathrm{pd}}}$ and $h_{v_{\mathrm{sd}}}$ denote the features of the precedent and the succeeding unsolved disjunctive nodes of the node v, and h_i describe the sum of features of all disjunctive nodes while v_d is the set of disjunctive nodes of node v. The neighbors of these five different types of arcs of node v are calculated by GAT and the ELU function [38], which aims to improve the learning dynamics in the network by mitigating the vanishing gradient problem. Finally, the L layer embedding results of five different types of arc are horizontally concatenated to formulate the state embedding $h_a^{(L)}$. It should be mentioned that the number of layers L of GATs will significantly affect the embedding quality. When L becomes larger, the features of the embedded nodes will include messages transmitted to distant nodes. The hyperparameter L is set to 2 in this study.

After retrieving the state embedding from the graph state, these embeddings, known as the features of the graph nodes, are used by the actor–critic decision module to learn our policy and estimate the value of the policy. The entire decision process is listed in (7) and (8). In (7), each node embedding of feasible (valid) action, which is denoted as v, is fed into MLP to obtain the score of the operation for subsequent softmax. The MLP_θ is MLP with two hidden layers of 32 neurons followed by the tanh activation function. Furthermore, the time feature, denoted as t, can represent real-time steps in the environment and enhance our model's adaptability, allowing it to leverage up-to-the-moment information for more effective scheduling decisions. Our critic will evaluate the state value with the concatenation of state embedding and time feature, which is listed in (8). The MLP_{ϕ_e} and MLP_{ϕ} are MLP with two hidden layers of 64 neurons followed by the tanh activation function

$$\pi_\theta(a_t|s_t) = \frac{\exp\left(\mathrm{MLP}_\theta(t\,\|\,h_v^{(L)})\right)}{\sum_{v\in\mathscr{A}_{s_t}} \exp\left(\mathrm{MLP}_\theta(t\,\|\,h_v^{(L)})\right)} \tag{7}$$

$$V_\phi(s_t) = \mathrm{MLP}_\phi\left(t\,\Big\|\,\sum_{i\in V}\mathrm{MLP}_{\phi_e}(h_i^{(L)})\right). \tag{8}$$

F. Loss Function

Due to the high-dimensional features in the large-scale JSSP and the continuous change in the state space, we employ a promising policy gradient method, PPO, to train our proposed model. One of the main advantages of PPO is its simplicity and stability, as it does not require complex hyperparameter tuning or sophisticated optimization techniques and can be easily implemented. We propose a loss function that considers three elements to stabilize model training. The first is the clipped surrogate objective function, as shown in (9), where $\rho_t(\Theta) = \pi_\Theta(a_t|s_t)\,/\,\pi_{\Theta_{\text{old}}}(a_t|s_t)$ and G_t is the return, which means cumulative reward and $A_{\Theta_{\text{old}}}$ is the advantage of the old policy, which is derived from $G_t - V_\Theta(s_t)$ and both will be used in the following equations. The surrogate objective function can constrain the change in policy in a small range $[1-\epsilon, 1+\epsilon]$ to avoid the dramatic change in parameters between the previous and the new policy. Besides the surrogate objective function, the remaining two elements are the entropy bonus function and the error of the value function, as listed in (10) and (11), respectively. The entropy bonus function can motivate our agent to perform more exploration during the training process, and the error of the value function can enable the model to toward the correct direction to update the parameters. Based on these three elements, the proposed loss function is the sum of these elements, as shown in (12)

$$L^{\text{CLIP}}(\Theta) = \sum_{t=0}^{|O|}\min\left(\rho_t(\Theta)A_{\Theta_{\text{old}}}, \mathrm{clip}\left(\rho_t(\Theta), 1-\epsilon, 1+\epsilon\right)A_{\Theta_{\text{old}}}\right) \tag{9}$$

$$L^{\text{E}}(\Theta) = -\sum_{t=0}^{|O|}\sum_{a\in\mathscr{A}_{s_t}}\pi_\Theta(a_t|s_t)\log\pi_\Theta(a_t|s_t) \tag{10}$$

$$L^{\text{V}}(\Theta) = \sum_{t=0}^{|O|}\left(V_\Theta(s_t) - G_t\right)^2 \tag{11}$$

$$L^{\text{T}}(\Theta) = \sum_{n=1}^{N} c_p L_n^{\text{CLIP}}(\Theta) - c_v L_n^{\text{V}}(\Theta) + c_e L_n^{\text{E}}(\Theta). \tag{12}$$

IV. Experimental Results and Discussion

To demonstrate the effectiveness of our proposed method, we use static and dynamic environments in the experiments. In the static environment, the evaluations are performed based on generated instances and public JSSP benchmarks. In the dynamic environment, evaluations are performed based on the generated instances with the uncertainties involved in the environment. It is worth mentioning that our proposed method can be applied to static and dynamic environments without changing the model.

A. Comparison Methods

Our proposed method demonstrates the ability to solve various JSSP instances without retraining the model. To assess the generalization ability of the proposed method across different JSSP instances, we conducted several experiments and compared the proposed method with the following methods.

1) *Random:* Assign the job randomly/job priority is assigned randomly.
2) Priority dispatch rules (PDRs).
a) *First in first out (FIFO):* The first job is processed first.
b) *Shortest processing time (SPT):* The job with the SPT is processed first.
c) *Longest processing time (LPT):* The job with the LPT is processed first
d) *Shortest total processing time (STPT):* The job with the STPT is processed first.
e) *Most process sequence remaining (MPSR):* The job with the most processing sequence remaining is processed first.
3) DRL-based method: Zhang [7] and Chen [39] proposed to automatically schedule the JSSP through an end-to-end DRL agent. They both used a disjunctive graph to represent JSSP, so those methods are listed as one of the comparison methods.

B. Experimental Settings

In this article, we generate synthetic JSSP instances for training, validation, and testing. The problem size is presented in the $|J| \times |M|$ format. For training and validation, we generated 10 000 synthetic instances with a problem size of 10×10 to train our model and another 100 synthetic instances with a problem size of 10×10 to validate our model. The processing times of the operations follow the uniform distribution between 1 and 10, that is, $p_{ij} \sim U\,(1, 10)$. For testing in a static environment, we consider two well-known public benchmarks, including Taillard's instances [40] and Demirkol's (DMU) instances [41], to test the generalization ability of our model. These instances are of various sizes (ranging from 6×6 to 100×20), drawing from the distributions significantly different from the uniform distribution. Therefore, testing on those public benchmarks can also verify the generalizability of our model on instances with other distributions. For testing in the dynamic environment, we generate 500 synthetic instances with three different problem sizes, including 6×6, 8×8, and 10×10, to evaluate the performance of our model.

TABLE I
PPO HYPERPARAMETERS SETTING

Hyperparameters	Value
Number of training iterations (I)	5000
Number of rollout episodes (N)	10
Number of updating epochs (K)	2
Optimizer	Adam
Learning rate (η)	2×10^{-4}
Discount factor (γ)	1
Clipping ratio (ε)	0.2
Policy loss coefficient (c_p)	1
Value function loss coefficient (c_v)	0.5
Entropy loss coefficient (c_e)	0.01

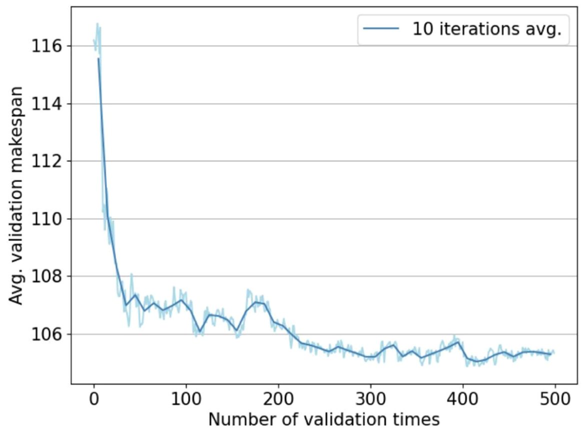

Fig. 4. Training curve on 10×10 instances.

The proposed method is implemented by PyTorch, based on the deep graph library[1] and the PyTorch Geometric library. Our hardware is a computer with Windows 10 64-bit OS, 32 GB RAM, Intel Core i7 4.0 GHz CPU, and NVIDIA GeForce GTX TITAN X GPU. The hyperparameters used in this article are summarized in Table I. We apply the grid search technique to small-scale examples to find the hyperparameters.

C. Performance on Public Benchmarks

In Fig. 4, we present the training curve for the problem size of 10×10, using 100 validation instances, where the x-axis represents the number of validation iterations, while the y-axis represents the makespan. Specifically, we conducted 5000 training iterations and recorded the makespan every 10 iterations. The results shown in Fig. 4 indicate a stable training process, with convergence achieved toward the end, which is demonstrated in the light blue line. Moreover, we also use the technique of moving average with a window size of 10 validation times to achieve the smoothing effect and also show the overall trend of training data, which is demonstrated in the dark blue line.

In addition to examining the training curve, we compared the performance of different parameter settings for the number of GAT layers, indicated as L. Two models were trained: one with two GAT layers and another with three. We tested on three sets of Ta benchmarks, with problem sizes ranging from 30×20 to 100×20. The results shown in Table II, presented as

TABLE II
RESULTS WITH DIFFERENT GAT LAYERS (L)

Problem size	$L = 2$	$L = 3$
30×20	2597 ± 83.6	2647.1 ± 117.6
50×20	3516.7 ± 142.6	3702.1 ± 151.5
100×20	6575.5 ± 225	6761.4 ± 255.1

mean makespan and standard deviation, indicate that the model with two GAT layers achieved a lower makespan and standard deviation. This finding motivates us to set L to 2 for better results and improved generalization ability. Two layers might be sufficient to capture complex relationships and features in the data. Moreover, fewer layers mean fewer parameters and less computation, leading to a more efficient training process.

Performance on Dmu benchmark: This section evaluates our model using the 16 Dmu benchmarks, including eight different problem sizes (ranging from 20×15 to 50×20). The performance metric is the makespan $C_{\max}$ and the variance. The experimental results are listed in Table III, indicating that the proposed method outperforms other alternatives in 13 of 16 instances. In the remaining three instances where our approach does not rank first, the relative scheduling error between our and the best-performing methods is within a narrow margin of 2.8%. Moreover, the result of variance shows that our proposed method showcases the ability to schedule unseen JSSP instances without being limited by instance sizes, as these instances differ significantly from the training instances in terms of problem size. This suggests that our proposed method performs effectively on JSSP instances characterized by varying problem scales and processing time distributions, underscoring its strong generalization capacity.

Performance on Ta benchmark: In addition to the abovementioned experiments, we also conducted experiments to compare our proposed method with two recent DRL-based methods. These two methods also use GNN to develop scheduling agents, explaining why they are listed as comparison methods in this section. The first method [7] used the disjunctive graph to represent JSSP and performed representation learning through GNN. Moreover, the JSSP scheduler was trained through synthetic instances, similar to our proposed method. The second method is called the disjunctive graph-embedded recurrent decoding transformer (DGERD), and the goal is to develop an agent to schedule JSSP. Similarly, the DGERD model was trained based on synthetic instances and evaluated using the public benchmark.

We conducted experiments using 16 Ta benchmarks, including eight different problems (ranging from 15×15 to 100×20). Still, the processing time distribution is quite different from the Dmu benchmarks used in the previous section. The purpose is to evaluate the generalization capacity using other unseen JSSP instances. The experimental results are presented in Table IV. This comparison has two performance metrics: the makespan $C_{\max}$ and the gap ratio. The gap ratio measures the gap between the makespan obtained from the method and the best solution from the literature. Equation (13) shows the definition of the gap

[1]Deep Graph Library: https://www.dgl.ai/

TABLE III
EXPERIMENTAL RESULTS ON DMU BENCHMARK, IN WHICH "UB" COLUMN DENOTES THE BEST SOLUTION FROM LITERATURE

Instance	Size	Random	LPT	SPT	FIFO	STPT	MPSR	Proposed	UB
Dmu03	20×15	3827	4592	3630	3807	4232	3435	**3303**	2731
Dmu04	20×15	3889	4047	3541	3689	4642	3355	**3321**	2669
Dmu08	20×20	4228	4551	4714	4183	4459	**3999**	4098	3188
Dmu09	20×20	4094	4511	4283	4433	4690	3869	**3753**	3092
Dmu13	30×15	5451	5580	4813	4962	5207	4759	**4708**	3681
Dmu14	30×15	5306	5591	4583	4978	4811	4238	**4124**	3394
Dmu18	30×20	5326	5810	6231	4957	5480	5003	**4800**	3844
Dmu19	30×20	5174	5787	5126	5216	5203	4930	**4837**	3768
Dmu23	40×15	5948	7045	6250	5828	6521	5383	**5240**	4668
Dmu24	40×15	6078	6484	5503	5921	6595	5358	**5319**	4648
Dmu28	40×20	6737	7322	6558	6454	7697	**5927**	5948	4692
Dmu29	40×20	6602	7386	6565	6354	7690	6107	**5824**	4691
Dmu33	50×15	6890	8779	7361	7657	7631	**6282**	6458	5728
Dmu34	50×15	7523	7991	7026	6673	7740	6359	**6284**	5385
Dmu38	50×20	7685	9051	7954	7777	8555	7604	**7275**	5713
Dmu39	50×20	8097	8514	7592	7167	8908	6953	**6776**	5747
Variance		1378.8	1627.9	1403.7	1306.6	1591.9	1257.8	**1226.4**	

The boldface entities denote the best results obtained in the respective experiments or analyses.

TABLE IV
EXPERIMENTAL RESULTS ON TA BENCHMARK, IN WHICH THE "UB" COLUMN DENOTES THE BEST SOLUTION FROM LITERATURE

Instance	Size	DGERD transformer	Zhang [7]	Proposed	UB
Ta01	15×15	1711 (0.39)	**1433 (0.16)**	1492 (0.21)	1231
Ta02	15×15	1639 (0.32)	1544 (0.24)	**1425 (0.15)**	1244
Ta11	20×15	1833 (0.35)	1794 (0.32)	**1752 (0.29)**	1357
Ta12	20×15	1765 (0.29)	1805 (0.32)	**1692 (0.24)**	1367
Ta21	20×20	2145 (0.31)	2252 (0.37)	**2097 (0.28)**	1642
Ta22	20×20	2015 (0.26)	2102 (0.31)	**1924 (0.20)**	1600
Ta31	30×15	2382 (0.35)	2565 (0.45)	**2277 (0.29)**	1764
Ta32	30×15	2458 (0.38)	2388 (0.34)	**2203 (0.24)**	1784
Ta41	30×20	**2541 (0.27)**	2667 (0.33)	2698 (0.35)	2005
Ta42	30×20	2762 (0.43)	2664 (0.38)	**2623 (0.35)**	1937
Ta51	50×15	3762 (0.36)	**3599 (0.30)**	3608 (0.30)	2760
Ta52	50×15	3511 (0.27)	**3341 (0.21)**	3524 (0.28)	2756
Ta61	50×20	3633 (0.27)	3654 (0.27)	**3548 (0.24)**	2868
Ta62	50×20	3712 (0.29)	3722 (0.30)	**3557 (0.24)**	2869
Ta71	100×20	6321 (0.16)	6452 (0.18)	**6289 (0.15)**	5464
Ta72	100×20	6232 (0.20)	**5695 (0.10)**	6002 (0.16)	5181
Average gap ratio		0.31	0.29	**0.25**	0

The boldface entities denote the best results obtained in the respective experiments or analyses.

ratio. The results show that our proposed method performs best in 11 of 16 instances. It is noted that the average gap ratio of the proposed method for all the instances is the lowest, indicating that the proposed method can make high-quality scheduling decisions in various types of JSSP, even though the problem size and the distribution of processing time are quite different from the training instances.

The results on the Dmu benchmarks show that the proposed method performs better than other comparison methods, including Random and PDRs. Furthermore, the results on the Ta benchmarks indicate that the proposed method can achieve promising results and perform better than other DRL-based methods on average. These two experimental results can empirically show that our proposed method is relatively robust in most benchmark instances compared to other methods. It is worth mentioning that the proposed method is size-agnostic, meaning that it can be applied to problems of different sizes from the training instances

$$\text{Gap Ratio (\%)} = \frac{\text{obtained makespan} - \text{best solution}}{\text{best solution}}. \quad (13)$$

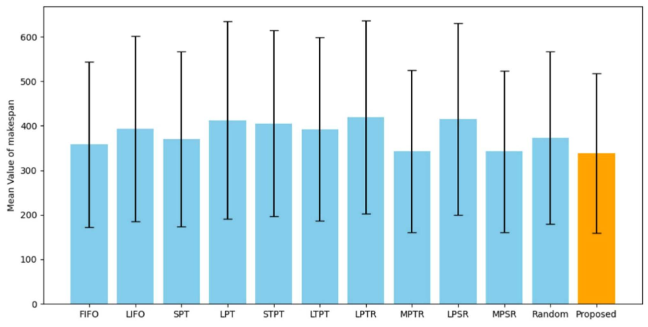

Fig. 5. Average makespan for simulation instances.

D. Performance on Static Simulation Instances

In addition to testing our model against public benchmarks, we performed experiments on synthetic instances in a static environment. We generated five sets of synthetic instances with varying problem sizes: 15×15, 20×20, 30×20, 50×20, and 100×20. Each set comprised 100 instances, with the processing times of operations following a uniform distribution between 1 and 10, denoted as $p_{ij} \sim U(1, 10)$. We performed experiments on these datasets and aggregated the results for comparative analysis.

Fig. 5 demonstrates that our proposed method achieves a lower makespan than other benchmark methods and exhibits superior stability. Furthermore, we performed a paired t-test, a statistical method comparing two related datasets, to determine the significance of the observed differences. The null hypothesis (H_0) states that there is no significant difference between the means of the two methods, with a significance level (α) set at 0.05. The p-values, listed in Table V, provide substantial evidence that our proposed method significantly outperforms all

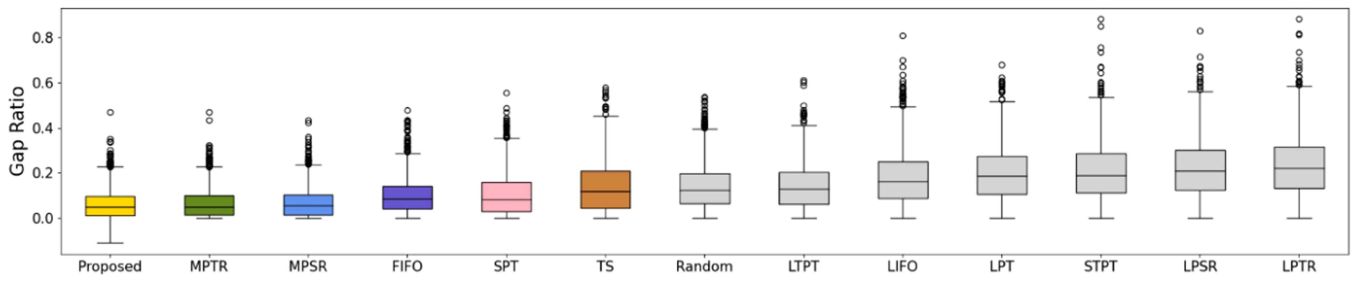

Fig. 6. Relative gap ratio for simulation instances.

TABLE V
RESULTS OF PAIRED SAMPLES T-TEST FOR SIMULATION INSTANCES

Methods	Random	MPSR	MPTR	FIFO	SPT
P-value	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001
Methods	FIFO	LIFO	LPT	STPT	LTPT
P-value	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001

TABLE VI
RESULTS OF ABLATION STUDY

Instance	Proposed method (GAT)	Proposed method replacing GAT with GCN
30×20	2628	2740
50×20	3547	3703

the PDRs. This robust statistical analysis reinforces our claims of improved generalization capability and superior stability.

E. Performance on Dynamic Environment

In this section, we conducted experiments in a dynamic environment in which the environment involves uncertainties owing to unexpected events. We use two commonly seen events, machine breakdown, and stochastic job arrival, to mimic the unexpected events occurring on the shop floor to assess further whether the proposed method can still work in environments where some unexpected events are unavailable during model training. Moreover, this can also verify the generalization capability of our model. We generate 500 synthetic instances for each problem scale, in which the problem scales are $6 \times 6, 8 \times 8$, and 10×10, respectively. The processing time of the operation follows the uniform distribution between 1 and 10, represented as $p_{ij} \sim U\,(1{,}10)$. Note that the proposed method is trained using the problem size of 6×6 and is evaluated using the instances of three different problem scales to prove that our proposed method can perform well on large-scale problems while training through small-scale problems.

Fig. 6 shows the box plot of the experimental results, in which the gap ratio is calculated according to (13). We also compare our proposed method with another well-known metaheuristic algorithm for scheduling problems, the tabu search (TS). The searching time for TS is the same as the solving time of our proposed method. As shown in Fig. 6, our proposed method performs better than the other comparison methods, including PDRs and TS. The relative gap ratio of the proposed method is the lowest and the maximum gap ratio of the proposed method does not exceed 0.2. These results demonstrate the effectiveness of our proposed method in achieving promising scheduling outcomes for both small- and large-scale problems, even within a dynamic environment. Furthermore, our agent exhibits robust performance and strong generalization abilities when dealing with unseen environments and uncertainties, which can closely resemble real-world conditions.

We conclude with the reasons why the proposed method can achieve promising results. First, the design of the states and the reward can help the subsequent GAT learn good feature representations. The agent can benefit from these to make a good decision about taking action. Second, the proposed method uses a graph representation that comprises different arcs and nodes to formulate the JSSP problem. Finally, we use the GAT attention mechanism to enable the model to pay attention to informative neighbors while ignoring the irrelevant ones.

F. Ablation Study

Our proposed approach uses an embedding network to map the input to its corresponding embedding. As outlined in this article, we employ a graph to represent the JSSP problem. Consequently, we substituted the GAT with a GCN during the ablation study. This allows for the evaluation of GAT's effectiveness. Furthermore, we assert that our proposed model benefits from the attention mechanism; this study validates this claim.

During the experimental phase, we tested instances from the Ta benchmark, which come in two distinct problem sizes: 30×20 and 50×20. Each problem size consists of 10 instances, and the chosen metric is the average makespan across these instances. As presented in Table VI, our proposed approach yields a reduced makespan compared to the proposed method that employs GCN for embedding.

In addition to comparing different embedding networks, we conducted another experiment to validate the effectiveness of our proposed strategies. This experiment determined whether our model benefits from incorporating solved disjunctive node features designed to reflect the current solution status more accurately. The study tested 500 synthetic instances in a dynamic environment across three problem sizes (6×6, 8×8, and 10×10). The evaluation metric, "WinRate," represents the proportion of instances in which the model achieved the best solution. Fig. 7 shows that including features from solved disjunctive arcs leads to superior outcomes compared to excluding

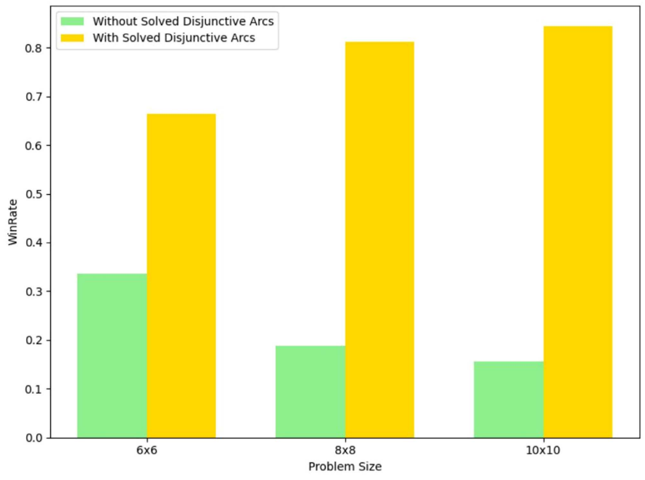

Fig. 7. Ablation study.

them. This finding confirms that our state design effectively enhances our model by providing additional informative details, thereby accurately reflecting the scheduling environment.

V. Conclusion

This article proposes using GAT and DRL to model JSSP in dynamic environments. We use a graph to denote JSSP and use the characteristics of DRL to deal with dynamic events. In particular, the architecture of the proposed method is size-agnostic, so the model can be trained on instances of smaller size and tested directly on large instances. The proposed method can generate promising results across numerous simulation instances and public benchmarks, regardless of static or dynamic environments. In future research, our focus will involve exploring alternative deep neural network architectures and refining graph attention models to further enhance the performance of our model. Moreover, as part of our future work, we intend to train our model using instances of varying problem sizes. This approach aims to enhance the model's generalization ability and increase its adaptability to a diverse range of problems.

References

[1] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep Q network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4276–4284, Jul. 2019.

[2] P. Festa, "A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems," in *Proc. IEEE 16th Int. Conf. Transparent Opt. Netw.*, 2014, pp. 1–20.

[3] H. M. Wagner, "An integer linear-programming model for machine scheduling," *Nav. Res. Logistics Quart.*, vol. 6, no. 2, pp. 131–140, 1959.

[4] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discr. Appl. Math.*, vol. 49, no. 1–3, pp. 107–127, 1994.

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

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

[7] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 1621–1632, 2020.

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representation*, 2018, pp. 1–12.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[10] S. Fatemi-Anaraki, R. Tavakkoli-Moghaddam, M. Foumani, and B. Vahedi-Nouri, "Scheduling of multi-robot job shop systems in dynamic environments: Mixed-integer linear programming and constraint programming approaches," *Omega*, vol. 115, 2023, Art. no. 102770.

[11] M. M. Ahmadian, A. Salehipour, and T. Cheng, "A meta-heuristic to solve the just-in-time job-shop scheduling problem," *Eur. J. Oper. Res.*, vol. 288, no. 1, pp. 14–29, 2021.

[12] K. Dehghan-Sanej, M. Eghbali-Zarch, R. Tavakkoli-Moghaddam, S. Sajadi, and S. J. Sadjadi, "Solving a new robust reverse job shop scheduling problem by meta-heuristic algorithms," *Eng. Appl. Artif. Intell.*, vol. 101, 2021, Art. no. 104207.

[13] K. Mohamed, "An effective new island model genetic algorithm for job shop scheduling problem," *Comput. Operations Res.*, vol. 67, pp. 132–142, 2016

[14] Y. Han et al., "Multi-strategy multi-objective differential evolutionary algorithm with reinforcement learning," *Knowl.-Based Syst.*, vol. 277, 2023, Art. no. 110801.

[15] Z. Hu, W. Gong, W. Pedrycz, and Y. Li, "Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization," *Swarm Evol. Comput.*, vol. 83, 2023, Art. no. 101387.

[16] F. Zhao, S. Di, and L. Wang, "A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem," *IEEE Trans. Cybern.*, vol. 53, no. 5, pp. 3337–3350, May 2023.

[17] 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, pp. 1809–1830, 2019.

[18] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput. Operations Res.*, vol. 134, 2021, Art. no. 105400.

[19] B. M. Méndez-Hernández, E. D. Rodríguez-Bazan, Y. Martinez-Jimenez, P. Libin, and A. Nowé, "A multi-objective reinforcement learning algorithm for JSSP," in *Proc. 28th Int. Conf. Artif. Neural Netw. Mach. Learn.*, 2019, pp. 567–584.

[20] M. Zhang, Y. Lu, Y. Hu, N. Amaitik, and Y. Xu, "Dynamic scheduling method for job-shop manufacturing systems by deep reinforcement learning with proximal policy optimization," *Sustainability*, vol. 14, no. 9, 2022, Art. no. 5177.

[21] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, vol. 8, pp. 71752–71762, 2020.

[22] B. Waschneck et al., "Deep reinforcement learning for semiconductor production scheduling," in *Proc. 29th Annu. SEMI Adv. Semicond. Manuf. Conf.*, Saratoga Springs, NY, USA, 2018, pp. 301–306.

[23] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Appl. Soft Comput.*, vol. 91, 2020, Art. no. 106208.

[24] B. Luo, S. Wang, B. Yang, and L. Yi, "An improved deep reinforcement learning approach for the dynamic job shop scheduling problem with random job arrivals," *J. Phys. Conf. Ser.*, vol. 1848, 2021, Art. no. 012029.

[25] Z. Liu et al., "A graph neural networks-based deep Q-learning approach for job shop scheduling problems in traffic management," *Inf. Sci.*, vol. 607, pp. 1211–1223, 2022.

[26] B. Roy and B. Sussmann, "Scheduling problems with disjunctive constraints," *Note DS*, vol. 9, pp. 106–114, 1964.

[27] N. Shakhlevich, Y. N. Sotskov, and F. Werner, "Adaptive scheduling algorithm based on mixed graph model," *IEE Proc. Control Theory Appl.*, vol. 143, no. 1, pp. 9–16, 1996.

[28] O. Gholami and Y. N. Sotskov, "Solving parallel machines job-shop scheduling problems by an adaptive algorithm," *Int. J. Prod. Res.*, vol. 52, no. 13, pp. 3888–3904, 2014.

[29] J. Juros, M. Brcic, M. Koncic, and M. Kovac, "Exact solving scheduling problems accelerated by graph neural networks," in *Proc. IEEE 45th Jubilee Int. Conv. Inf. Commun. Electron. Technol.*, 2022, pp. 865–870.

[30] K.-H. Ho et al., "Residual scheduling: A new reinforcement learning approach to solving job shop scheduling problem," *IEEE Access*, vol. 12, pp. 14703–14718, 2024.

[31] J.-P. Huang, L. Gao, and X.-Y. Li, "An end-to-end deep reinforcement learning method based on graph neural network for distributed job-shop scheduling problem," *Expert Syst. Appl.*, vol. 238, 2024, Art. no. 121756.
[32] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1600–1610, Feb. 2023.
[33] M. Zhang, L. Wang, F. Qiu, and X. Liu, "Dynamic scheduling for flexible job shop with insufficient transportation resources via graph neural network and deep reinforcement learning," *Comput. Ind. Eng.*, vol. 186, 2023, Art. no. 109718.
[34] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, and L. Qian, "Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 20, no. 1, pp. 1007–1018, Jan. 2024.
[35] K. Lei et al., "A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem," *Expert Syst. Appl.*, vol. 205, 2022, Art. no. 117796.
[36] G. Gebreyesus, G. Fellek, A. Farid, S. Fujimura, and O. Yoshie, "Gated-attention model with reinforcement learning for solving dynamic job shop scheduling problem," *IEEJ Trans. Elect. Electron. Eng.*, vol. 18, pp. 932–944, 2023.
[37] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 2015.
[38] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *Proc. Int. Conf. Learn. Representation*, 2016, pp. 1–14.
[39] R. Chen, W. Li, and H. Yang, "A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1322–1331, Feb. 2023.
[40] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, 1993.
[41] E. Demirkol, S. Mehta, and R. Uzsoy, "Benchmarks for shop scheduling problems," *Eur. J. Oper. Res.*, vol. 109, no. 1, pp. 137–141, 1998.

Chien-Liang Liu (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science from National Chiao Tung University, Taipei, Taiwan, in 2000 and 2005, respectively.

He is currently a Full Professor with the Department of Industrial Engineering and Management, National Yang Ming Chiao Tung University, Taipei, Taiwan. His research primarily focuses on machine learning, deep learning, data mining, and big data analytics. His work contributes significantly to the advancement of these fields.

Chun-Jan Tseng received the M.S. degree in logistic management from the Graduate Institute of Logistics Management, National Defense University, Taoyuan City, Taiwan, in 2007. He is currently working toward the Ph.D. degree in industrial engineering and management with National Yang Ming Chiao Tung University, Taipei, Taiwan.

He is currently a Military Instructor with Management College, National Defense University, Taiwan. His research interests include machine learning, data mining, deep learning, and logistics management.

Po-Hao Weng received the B.S. degree in logistic management, in 2022 from the Department of Transportation and Logistics Management, National Yang Ming Chiao Tung University, Taipei, Taiwan, where he is currently working toward the master's degree in deep reinforcement learning and scheduling with the Department of Industrial Engineering and Management.

His research interests include machine learning, reinforcement learning, deep learning, and scheduling optimization.