A Deep Reinforcement Learning Framework Based on an Attention Mechanism and Disjunctive Graph Embedding for the Job-Shop Scheduling Problem

Ruiqi Chen, Wenxin Li, and Hongbing Yang

***Abstract*—The job-shop scheduling problem (JSSP) is a classical NP-hard combinatorial optimization problem, and the operating efficiency of manufacturing system is affected directly by the quality of its scheduling scheme. In this article, a novel deep reinforcement learning framework is proposed for solving the classical JSSP, where each machine has to process each job exactly once. This method based on an attention mechanism and disjunctive graph embedding, and a sequence-to-sequence pattern is used to model the JSSP in the framework. A disjunctive graph embedding process based on node2vec is used to learn the disjunctive graph representations containing JSSP characteristics, thereby generalizing the model considerably. An improved transformer architecture based on a multi-head attention mechanism is used to generate solutions. Containing a parallel-computing encoder and a recurrent-computing decoder, it is adept at learning long-range dependencies and effective at solving large-scale scheduling problems. Experimental results verified the effectiveness of the proposed method.**

***Index Terms*—Attention mechanism, deep reinforcement learning (DRL), graph embedding, job-shop scheduling, transformer model.**

I. Introduction

THE job-shop scheduling problem (JSSP), which is a classical NP-hard problem in production scheduling, is prevalent in numerous discrete-manufacturing industries [1]. It plays a core role in the production management of discrete manufacturing systems and can help industrial enterprises improve productivity and equipment utilization. It has been intensively studied over the past decades.

Manuscript received 12 November 2021; revised 18 March 2022; accepted 5 April 2022. Date of publication 14 April 2022; date of current version 13 December 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 52075354. Paper no. TII-21-5024. *(Corresponding author: Hongbing Yang.)*

Ruiqi Chen and Hongbing Yang are with the School of Mechanical and Electrical Engineering, Soochow University, Suzhou 215031, China (e-mail: rui.ie@outlook.com; yanghongbing@suda.edu.cn).

Wenxin Li is with the School of Management, Shanghai University, Shanghai 200436, China (e-mail: liwenxinn@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3167380.

Digital Object Identifier 10.1109/TII.2022.3167380

Exact algorithms (e.g., mathematical-programming ones [2] and branch-and-bound ones [3]) are classical methods for solving JSSP. They can obtain optimal solutions after sufficient runs. JSSP also can be solved by approximate algorithms [e.g., local-search ones [4], meta-heuristic ones [5], constructive ones [6], and artificial intelligence (AI) ones [7]]. They can yield appropriate solutions after a period of running. Most exact algorithms cope with small-scale problems effectively, but when the problem scale expands, they often need to consume a lot of computing time to find high-quality solutions. Therefore, when dealing with a large-scale problem, it is more reasonable to use an approximate algorithm.

The disjunctive (mixed) graph has been effectively used in JSSP modeling [8]. Many novel algorithms (e.g., the fast branch and bound algorithm [3], the shifting bottleneck procedure [9], etc.) were developed based on it. Some job-shop schedules should be adaptive to deal with various dynamic changes in real-world production scenarios. As a powerful tool, disjunctive graph is often employed to develop adaptive algorithms for flexible schedule. Shakhlevich *et al.* [10] proposed an adaptive scheduling algorithm based on mixed graph model. They used the conflict resolution strategy in the learning and examination stages. The adaptive approach can adapt to various practical constraints. Similarly, Gholami and Sotskov [11] modeled parallel machines job-shop problem by a weighted mixed graph with a conflict resolution strategy, and an adaptive algorithm was proposed to solve larger scale problems similar to the samples in the learning stage.

In recent years, with the rapid development and successful application of AI algorithms, AI-based methods for solving the JSSP have attracted increasing attention [12]. In 1988, Foo and Takefuji [13] used a 2-D Hopfield artificial neural network (NN) to solve the JSSP, optimizing the model by minimizing the sum of the starting times of the last operation for each job. To solve the dynamic JSSP, Adibi *et al.* [14] proposed a hybrid method based on variable neighborhood search and an NN to cope effectively with the arrival of new jobs and machine breakdowns.

As one of the most active areas in AI research, deep reinforcement learning (DRL) combines the perceptual abilities of deep learning with the decision-making abilities of reinforcement learning. In Atari games and the game of Go, DRL has

shown intelligence beyond that of the leading human experts. DRL has also been successfully applied to solve combinatorial optimization problems. Bello *et al.* [15] presents a DRL framework to tackle combinatorial optimization problems. This method achieved close to optimal results on traveling-salesman problems (TSPs) with up to 100 nodes and also performed well on knapsack problems. Kool *et al.* [16] proposed a neural sequence transduction model for TSP, vehicle routing problems, and orienteering problems, and then introduced a reinforcement learning method to train the model. Production-scheduling methods based on DRL require a considerable time to be trained, but after training, they respond quickly and can be applied to daily rolling scheduling or even real-time scheduling in the factory. To date, the main DRL methods used for the JSSP have involved the deep Q network (DQN) and its derivatives. To solve the adaptive JSSP, Han and Yang [17] proposed a model based on dueling double DQN. The information, such as processing time and machine utilization rate, is input into the model in matrix form, and a convolutional NN is used to approximate the state-action values; this method gives optimal solutions to small-scale problems. Lin *et al.* [18] proposed a smart manufacturing factory framework based on edge computing and used a multiclass DQN to investigate the JSSP under such a framework. Palombarini and Martinez [19] stored rescheduling knowledge in DQN and learned rescheduling strategies directly from high-dimensional inputs. Waschneck *et al.* [20] designed a DQN algorithm for scheduling in the semiconductor industry and trained the network through user-defined objectives. Luo [21] designed a DQN-based dynamic scheduling method for flexible JSSP, which considers the insertion of new jobs. Liu *et al.* [22] proposed a multiagent actor–critic model for solving the JSSP and used the deep deterministic policy gradient (DDPG) algorithm to train the model.

Although research on DRL-based JSSP algorithms has achieved a certain degree of success, careful analysis reveals two major drawbacks in that approach. 1) These scheduling models always have poor generalizability: The input dimensions are highly correlated with the problem structure, and once the model is trained, the NN structure is fixed and the model can only deal with problems with a specific structure. 2) These models perform well when dealing with small-scale scheduling problems, but when the problem scale expands, their solutions deteriorate seriously in quality.

A good strategy for feature extraction can make NNs work better when solving the JSSP. Some researchers have used raw data (e.g., processing time matrix or machine utilization matrix) as input [17], [22] or have used one-hot vectors (a vector that has 0s in all its cells except one that contains a 1) to encode state features [23], [24]. However, those approaches lead to a lack of generalization, and one-hot vectors are sparse and do not contain state characteristics. Adopting well-designed composite dispatching rules as agent's action is another way to achieve good scheduling results [21].

Graph embedding is a process in which graph data (usually high-dimensional dense matrices) are mapped to low-dimensional dense vectors, and it deals with the problem that it is difficult to input graph data efficiently into machine learning models. Graph embedding has been applied successfully to social relationship prediction [25] and product recommendation systems [26], among other problems, but it is yet to be applied to the JSSP. Herein, we propose a representation learning method for disjunctive graphs to extract JSSP features. These representations are taken as the input of the decision model, and the output of the model acts directly on specific tasks rather than the heuristic dispatching rules, thereby giving full play to the end-to-end learning advantages of DRL.

In the case of large-scale scheduling, there is too much information to consider; so we introduce an attention mechanism to allocate the limited information processing resources to the most valuable parts. Attention mechanisms can be applied in a wide range of scenarios. To solve combinatorial problems in which the size of the output dictionary depends on the length of the input sequence, Vinyals *et al.* [27] designed a pointer network model based on an attention mechanism; this takes the attention as a pointer to the input elements and achieves better result than do traditional models in solving convex-hull problems, Delaunay triangulations, and the TSP. Vaswani *et al.* [28] proposed a transformer attention architecture that has an encoder–decoder structure and is based on a multiheaded self-attention mechanism; this is one of the most powerful natural language processing architectures currently available. Kool *et al.* [16] improved the transformer architecture to solve routing problems and trained the model using reinforcement learning; their study showed the potential of the transformer architecture to deal with combinatorial optimization problems.

To improve model generalizability and the performance of solving a large-scale JSSP, herein, we set minimizing the maximum completion time $C_{\max}$ as the optimization object and propose a sophisticated disjunctive graph embedded recurrent-decoding transformer model (hereinafter referred to as DGERD transformer), which integrates a graph embedding method and a transformer architecture with recurrent-encoding process. It guarantees good generalizability and obtains a high-quality scheduling scheme in a short time.

The main contributions of this article are as follows.

1) A novel DRL framework for solving the JSSP is designed, which overcomes the problem of the poor generalizability of previous models and can solve problems of any scale after being trained once.

2) A graph-embedding method is proposed to extract the JSSP features directly from the disjunctive graph, and these features are effective in helping agents to make scheduling decisions.

3) A JSSP solution model based on an attention mechanism is designed, which can help agents to learn long-range dependencies better. Benefiting from this attention mechanism, the DGERD transformer is more suitable for solving large-scale scheduling problems than other methods.

The rest of this article is organized as follows. Section II defines the JSSP and describes its disjunctive-graph representation. Section III proposes the DGERD transformer scheduling framework and introduces the disjunctive graph embedding process, the scheduling model based on an attention mechanism, and the reinforcement learning training strategy. Section IV details the training and hyperparameter adjustment and reports a series of comparative experiments to test the performance of the DGERD transformer. Finally, Section V concludes this article.

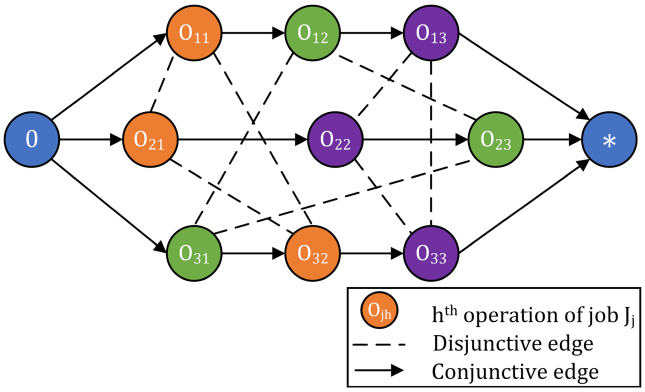

Fig. 1. JSSP instance represented by the disjunctive graph.

II. Disjunctive Graph Description of Job-Shop Scheduling Problem

The JSSP can be described as the problem of scheduling N jobs on M machines to pursue some optimal production objectives. Processing of the jth job consists of a set of operations, which must be performed in a given order. The hth operation $o_{jh}(h \in \{1, 2, \ldots, M\})$ of job j must be processed on a specific machine. Its starting time and processing time are t_{jh} and p_{jh}, respectively. $O_m(m \in \{1, 2, \ldots, M\})$ is a set of all operations processed by machine m. A job can only be processed on one machine at a time. Once an operation is started on a machine, the processing is not interrupted until completion [29]. The objective of scheduling is to minimize the maximum completion time $C_{\max}$. The JSSP can be formulated as follows:

$$\min C_{\max} = \max_j \{t_{jM} + p_{jM}\} \tag{1}$$

$$\text{s.t.} \quad t_{jh} + p_{jh} \le t_{j(h+1)} \quad \forall\, j,\ 1 \le h \le M-1 \tag{2}$$

$$t_{jh} + p_{jh} \le t_{j'h'} \quad \text{for } o_{jh},\ o_{j'h'} \in O_m \text{ and } t_{jh} \le t_{j'h'} \tag{3}$$

$$\sum_{h=1}^{M} R_{j,h,m} = 1 \qquad \forall\, j, m \tag{4}$$

$$t_{jh} \ge 0 \qquad \forall\, j, h \tag{5}$$

where $R_{j,h,m}$ is binary variable, if operation o_{jh} is processed on the machine m, $R_{j,h,m} = 1$, and 0 otherwise. Precedence constraint in (2) guarantees the processing sequence of operations in the same job by processing route. Overlap constraint in (3) ensures that each machine can only process one operation at a time. Constraint in (4) provides that a job is processed only once by the same machine.

A disjunctive graph is a generalized graph some of whose arcs are disjunctive. Balas [30] transformed disjunctive graphs into machine scheduling problems. The JSSP can be modeled as a disjunctive graph $G = (V,\ C \cup D)$, as shown in Fig. 1, where V is a set of nodes that each represents an operation. Additionally, there are two special nodes: a source 0 and a sink *. C and D represent conjunctive arcs and disjunctive arcs, respectively. A conjunctive arc is directed, connects two successive operations in a job, and reflects the precedence constraint. A disjunctive arc is undirected, and two operations connected by a disjunction arc can be processed by the same machine. In Fig. 1, operations that can be processed by the same machine are identified by the same color. The procedure of solving the JSSP can be regarded as transforming a disjunctive graph into a directed subgraph by removing redundant arcs and determining the directions of disjunctive arcs.

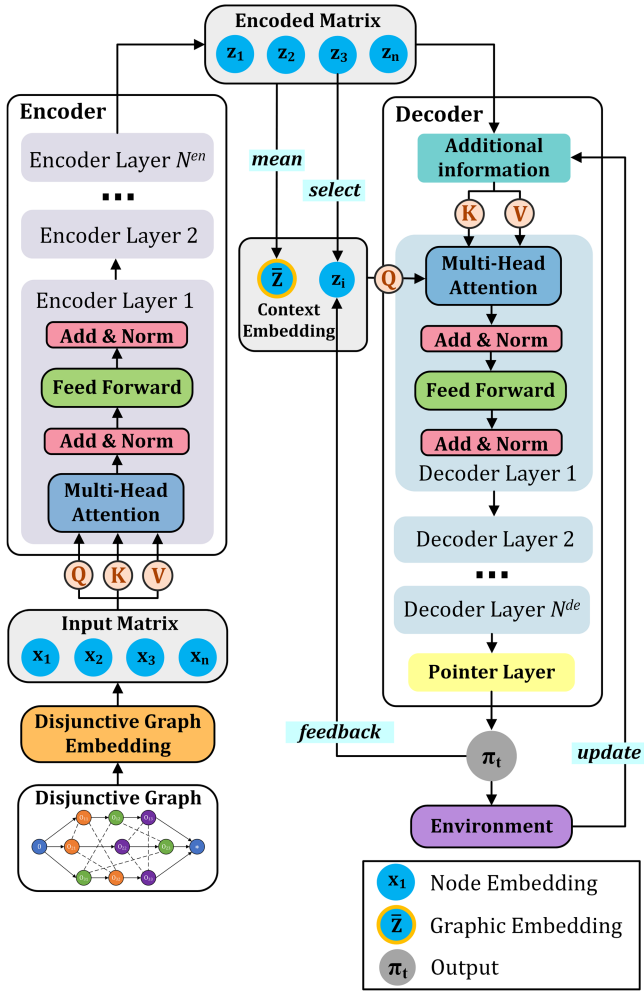

Fig. 2. DGERD transformer architecture.

III. JSSP Scheduling Framework: DGERD Transformer

Herein, we propose a novel DRL framework, i.e., the DGERD transformer, for solving the JSSP based on an attention mechanism and disjunctive graph embedding, as shown in Fig. 2. The DGERD transformer runs in a sequence-to-sequence (seq2seq) manner and uses an encoder–decoder structure, just like the most competitive neural sequence transduction models [16], [28]. Different from other combinatorial optimization problems, JSSP has complex precedence constraints and overlap constraints among items that need to be arranged, which makes it difficult to directly feed the raw data (e.g., coordinates in TSP) into the model. Therefore, this article first preprocesses the raw data using a disjunctive graph feature extraction method. When given a job-shop instance, each operation of jobs is mapped to a feature

vector x_i via the feature extraction, and the input sequence $X = \{x_1, \ldots, x_n\}$ is formed after gathering all x_i. The input sequence is encoded in parallel by the encoder and mapped to a sequence of continuous representations $Z = \{z_1, \ldots, z_n\}$. Through m decoding processes (where m is the total number of operations), the decoder generates an output sequence $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_m\}$ (one element at a time). $\boldsymbol{\pi}$ represents the processing order of all operations. For example, if $\pi_a = o_{jh}$, $\pi_b = o_{j'h'}$ $(a < b,\ j \neq j')$, and o_{jh}, $o_{j'h'}$ are processed by the same machine, then o_{jh} is assigned before $o_{j'h'}$. At each time step t $(t \in \{1, \ldots, m\})$, the DGERD transformer estimates once the probability $p_\theta(\pi_t = o_{jh}|\pi_{1:t-1}, X; \theta)$ of selecting candidate operation o_{jh}. The conditional probability p_θ is computed by a deep neural network (DNN) with parameter θ. The probability of obtaining a solution $\boldsymbol{\pi}$ can be estimated by the probability chain rule

$$p(\boldsymbol{\pi}|X;\theta) = \prod_{t=1}^{m} p_\theta(\pi_t|\pi_{1:t-1}, X;\theta). \tag{6}$$

Essentially, the role of the DGERD transformer is to construct a functional mapping from operation characteristics to a process order.

A. JSSP Disjunctive Graph Feature Extraction

Grover and Leskovec [25] proposed the node2vec algorithm for learning continuous representations of the nodes in a network. This algorithm framework can be applied effectively to multilabel classification and link prediction. We improve the node2vec algorithm to extract the disjunctive graph features. A feature extraction network independent of the scheduling model is trained. Its optimization objective is not directly related to the minimize the $C_{\max}$ objective but is related to extract the operation features into vectors and reduce information loss.

Given a disjunctive graph $G = (V,\ C \cup D)$, the goal is to transform the nodes V into the d_{emb}-dimensional feature representations $\mathbb{R}^{d_{\text{emb}}}$ through a mapping function $f: V \to \mathbb{R}^{d_{\text{emb}}}$. Let $u \in V$ be one node corresponding to the operation o_{ih}. Define $N_C(u) \in V$ as the network neighborhood of node u, which is connected by a conjunctive arc and corresponds to the next operation of o_{ih}. Define $N_C(u) \in V$ as the other type of network neighborhoods of node u, corresponding to the operations processed by the same machines as o_{jh}. The objective of disjunctive graph feature extraction is to maximize the log-probability of correctly predicting the two types of neighbor nodes

$$\max_f \sum_{u \in V} \log\left[Pr\left(N_C(u)|f(u)\right) \times Pr\left(N_D(u)|f(u)\right)\right]. \tag{7}$$

The first step of disjunctive graph embedding is to randomly generate some walk-tracks from each source node u. A walk-track example (o_{31} as the source node u) is shown in Fig. 3, and we can transform it into a clear sequence form $\{o_{31},\ o_{23},\ o_{12},\ o_{13},\ o_{33},\ *\}$.

Node2vec suggests that a biased random-walk strategy, which explores graphs in the breadth-first or depth-first sampling fashion, can extract the features of two types of similarities, i.e., homophily or structural equivalence, respectively. Herein, we

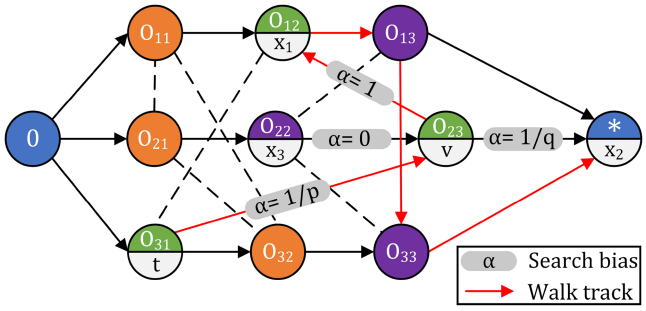

Fig. 3. Walk-track generation in disjunctive graph embedding.

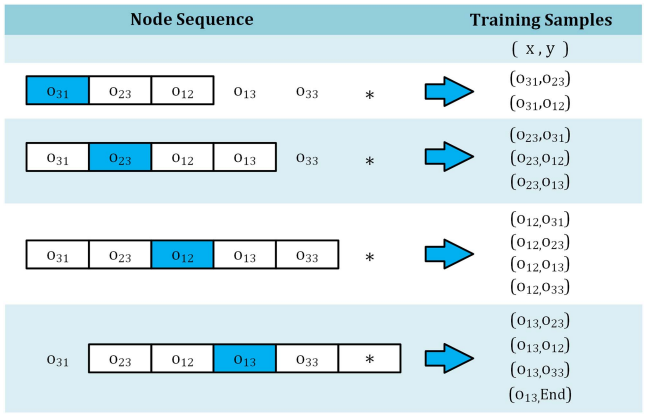

Fig. 4. Use sliding window to generate training samples.

improve node2vec to explore disjunctive graphs. Two parameters, i.e., the return parameter p and the in–out parameter q, are used to guide the exploration. Fig. 3 shows a random walk that has just passed node t and now resides at node v, where it must decide on the next step. The search bias $\alpha_{pq}(t,x)$ is the unnormalized probability of taking strategy S_{vx} (from node v transfer to node x). If node x corresponds to the previous operation of node v, then $S_{vx} = \alpha_{pq}(t,x) = 0$. If node x corresponds to the next operation of node v or e_{vx} is a disjunctive arc, then

$$S_{vx} = \alpha_{pq}(t,x) = \begin{cases} \frac{1}{p}, & \text{if } d_{tx} = 0 \\ 1, & \text{if } d_{tx} = 1 \\ \frac{1}{q}, & \text{if } d_{tx} = 2 \end{cases}. \tag{8}$$

Here, d_{tx} denotes the shortest-path distance between nodes t and x. The feature-learning process is based on the skip-gram algorithm [31]. Training samples can be obtained by using a sliding window over the consecutive sequence, as shown in Fig. 4. Finally, stochastic gradient descent is used to optimize the objectives [as shown in (7)] to obtain the node features x_i^G for each operation. As another crucial factor for the JSSP, the normalized operation processing time p_i constitutes the input $X = \{x_1, \ldots, x_n\}$ together with $x_i^G : x_i = [x_i^G,\ p_i]$. Here, $[\cdot,\cdot]$ is the horizontal concatenation operator.

B. Multihead Attention Mechanism

By improving the Ptr-Net algorithm and transformer architecture, we design an attention scheduling model for solving

the JSSP. After feature extraction, some of the correlations and dependencies that exist in the disjunctive graph (e.g., precedence constraints and overlap constraints) have been extracted into the input sequence. The scheduling model analyzes these correlations and dependencies and then formulates a schedule through them. This model overcomes the disadvantages of traditional neural sequence transduction models (such as recurrent and convolutional NNs) and can better learn the long-range dependencies in a sequence, which makes it superior when solving large-scale problems. At the same time, it also makes no assumptions about the temporal/spatial relationships across the data, in line with the independence requirement of the operation features.

The attention mechanism can be described as mapping queries Q and key–value pairs K-V (the dimension of the keys is d_k) to an output matrix through an attention function

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{9}$$

where $\text{Softmax}(x_i) = e^{x_i}/\sum_C e^{x_j},\ x_i \in C$. This output can be seen as a weighted sum of the values V, where the weight assigned to each value is the compatibility between the query and the corresponding key. Following Vaswani *et al.* [28], we use the multihead attention instead of the single-head one

$$\begin{aligned}\text{MutiHead}(Q, K, V) &= [\text{head}_1, \ldots, \text{head}_{\mathcal{M}}]\, W^O \\ \text{where head}_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)\end{aligned} \tag{10}$$

where W_i^Q, W_i^K, W_i^V, and W^O are learnable matrices. This allows our model to jointly attend to information from different representation subspaces, which reflect different aspects of JSSP characteristics.

C. Parallel Encoding

The encoder structure of the DGERD transformer (shown in Fig. 2, left) is similar to that of the original transformer architecture, but the positional encoding is abandoned in favor of graph embedding features that reflect the interrelationships among different operations (see Section III-A). The reason for abandoning positional encoding is that the elements in the input sequence (i.e., operation features) have no temporal/spatial relationships, and the position of the elements does not need to be considered.

The encoding process is mainly parallel matrix computation, which accelerates the running of the DGERD transformer. The input matrix X ($X \in \mathbb{R}^{n \times d_{\text{model}}}$, where n is the length of the input sequence and d_{model} is the hyperparameter that controls the model size) is updated using a stack of N_{en} identical encoder layers. Each layer has two sublayers: a multihead attention layer and a fully connected feed-forward network (FNN). Each sublayer is followed by a residual connection and a layer normalization. For the first layer, the Q, K, and V matrices all come from the input matrix X; for the subsequent layers, they come from the output of the previous layer. The output of the last encoder layer is the encoded matrix Z, where each row z_i represents the encoded features of an operation. The encoded matrix can be seen as a context that represents the job-shop environment. Finally, the mean of all operation features is used as an aggregated embedding $\bar{Z}$

$$\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} z_i. \tag{11}$$

D. Recurrent Decoding Strategy

During decoding, the agent selects an action (feasible operation) and causes the state transition at each unit time step. For a seq2seq framework, the states generally contain the output at previous time step and the context. The proposed decoder is based on dot-product attention mechanism; so the state can be represented by (Q, K, V, mask). The decoding happens recurrently (as shown in Fig. 2, right). At each time step t, the decoder outputs an operation index π_t based on the previous output π_{t-1}, the encoded matrix Z, and the additional information fetched from the environment.

First, a node embedding $z_{\pi_{t-1}}$ representing the previous output π_{t-1} is selected from matrix Z, and then the aggregated embedding $\bar{Z}$ and $z_{\pi_{t-1}}$ ($z_{\pi_{t-1}} = z_{\text{start}}$ at the first step) are horizontally concatenated to form a context embedding $\mathcal{C}_t$ [as shown in (12)]. A context embedding reflects both the overall characteristics ($\bar{Z}$) of the scheduling problem and the current decoding state ($z_{\pi_{t-1}}$). It will be used as the query in the attention function (9) to yield the output of this time step

$$\mathcal{C}_t = \begin{cases} \left[\bar{Z}, z_{\text{start}}\right], & \text{if } t = 1 \\ \left[\bar{Z}, z_{\pi_{t-1}}\right], & \text{if } t > 1 \end{cases}. \tag{12}$$

With the progress of scheduling, the scheduling environment is changing all the time. It is laborious for the model to make decisions only according to the static information obtained at the very beginning of the scheduling procedure. Therefore, some additional information is added to the matrix Z to support the model's decision.

Define O_C as a candidate set. When an operation is ready to be processed, it is appended to O_C. The normalized earliest start time ES_{ih}^t and finish time EF_{ih}^t for operation o_{jh} are used as additional information (if $o_{jh} \notin O_C$, $\text{ES}_{jh}^t = \text{EF}_{jh}^t = 0$). Add these two additional pieces of information into matrix Z as shown in (13), and the modified matrix $Z' = \{z_1', \ldots, z_n'\}$ can be regarded as the key K and value V in the attention function (9)

$$z_i' = z_{jh}' = \left[z_{jh}, \text{ES}_{jh}^t, \text{EF}_{jh}^t\right]. \tag{13}$$

The decoder is also composed of a stack of N_{de} identical layers. Its structure is similar to that of the encoder. During the decoding, Z' and $\mathcal{C}_t$ are projected to $d_k = \frac{d_{\text{model}}}{\mathcal{M}}$ ($\mathcal{M}$ is the number of attention-heads) dimensions first by matrices W^Q, W^K, and W^V so that they can take the dot product.

To prevent the model from focusing attention on the completed operations that do not affect the subsequent scheduling steps, we use a look-back mask to mask these operations. When the first operations of all jobs are completed, the start node is also masked. The mask is a matrix whose masked cells are set

to a large negative number and otherwise 0. It is added to the matrix QK^T before Softmax. Then the masked elements in the result of Softmax are approximately equal to 0, and the rest are uninfluenced.

For a sequence-to-sequence JSSP model, the output dictionary size depends on the length of the input sequence. Because the number of operations in the factory varies from time to time, it is impractical to use a fixed-length vocabulary for the output sequence. Therefore, we employ a pointer layer at the end of the decoder and use it to output variable-length solutions. The pointer network layer is also based on a self-attention mechanism, but it only computes the compatibility $u_i^{\mathcal{C}}$ of the query with partial keys. Following Bello *et al.* [15] and Kool *et al.* [16], we scale the result within $[-A, A]$ using tanh

$$u_i^{\mathcal{C}} = \begin{cases} A \cdot \left(\frac{\mathcal{C}_t^T k_i}{\sqrt{d_k}} \right), & \text{if } o_i \in O_C \\ -\infty, & \text{else} \end{cases}. \tag{14}$$

Here, k_i is the feature vector of an operation that is obtained from the last decoder layer. In the pointer layer, a pointer mask is used to guarantee the feasibility of solutions. Only the operations in the candidate set O_C are exposed, and the others are masked by setting their compatibility value to $-\infty$. The $u_i^{\mathcal{C}}$ can be seen as unnormalized log-probabilities (logits) and can be used as pointers to the input elements. A Softmax is calculated to estimate the final output probability

$$p_i = p_\theta \left(\pi_t = o_i | \pi_{1:t-1}, X; \theta \right) = \frac{e^{u_i^{\mathcal{C}}}}{\sum_{j=1}^{n} e^{u_j^{\mathcal{C}}}}. \tag{15}$$

In the offline learning phase, the agent selects actions (feasible operations) by sampling according to probability p_i, thus ensuring the diversity of experiences. In the online optimization phase, the action with the highest p_i is selected and the corresponding operation is added to the solution sequence, thus ensuring the quality of the solution.

E. Deep Reinforcement Learning Training Strategy

Herein, the DGERD transformer is trained by the policy gradient approach. The agent is expected to generate solutions with smaller $C_{\max}$; so the reward is set to $-C_{\max}$ and the loss function is defined as $\mathcal{L}(\theta|X) = -\text{Reward} = C_{\max}$. The model is optimized using stochastic gradient descent

$$\nabla \mathcal{L} \left(\theta | X \right) = \left(C_{\max} - b_e \right) \nabla \log p_\theta \left(\boldsymbol{\pi} | X \right). \tag{16}$$

Considering that different JSSP instances have different levels of difficulty, we use a baseline b_e to correct this deviation. This ensures that our model gets a reasonable punishment rather than a small reward when $C_{\max}$ is not ideal. b_e is initialized to $C_{\max}$ at the first epoch and updated by moving average with decay β in subsequent epochs

$$b_e = \begin{cases} C_{\max}, & \text{if } e = 1 \\ \beta b_{e-1} + (1-\beta) C_{\max}, & \text{if } e > 1 \end{cases}. \tag{17}$$

The complete process of training the DGERD transformer by reinforcement learning is shown in Algorithm 1.

Algorithm 1: Training DGERD Transformer by DRL.

Input: Number of epochs E, number of operations T, decay β
Output: θ

Init θ
$s \leftarrow$ GenerateInstance
$G \leftarrow$ DisjunctiveGraph(s)
for $e = 1 \rightarrow E$ **do**
  Init $\boldsymbol{\pi} \leftarrow$ Empty
  Init $\pi_0 \leftarrow$ <Start>
  $X \leftarrow$ GraphEmbedding(G, s)
  $Z \leftarrow$ Encode(X)
  $\bar{Z} \leftarrow$ mean(Z)
  for $t = 1 \rightarrow T$ **do**
    $z_t \leftarrow$ SelectNode(π_{t-1})
    $\mathcal{C}_t \leftarrow$ ContextEmbedding($\bar{Z}, z_t$)
    $Z_t' \leftarrow$ AdditionalInformation(Z, s)
    $\pi_t \leftarrow$ Decode($\mathcal{C}_t, Z_t'$)
    Append π_t to $\boldsymbol{\pi}$
  end for
  if $e = 1$ **then**
    $b_e \leftarrow C_{\max}$
  else
    $b_e \leftarrow \beta b_{e-1} + (1-\beta) C_{\max}$
  end if
  $\nabla \mathcal{L} \leftarrow (C_{\max} - b_e) \nabla \log p_\theta(\boldsymbol{\pi}|X)$
  $\theta \leftarrow$ Adam($\theta, \nabla \mathcal{L}$)
end for

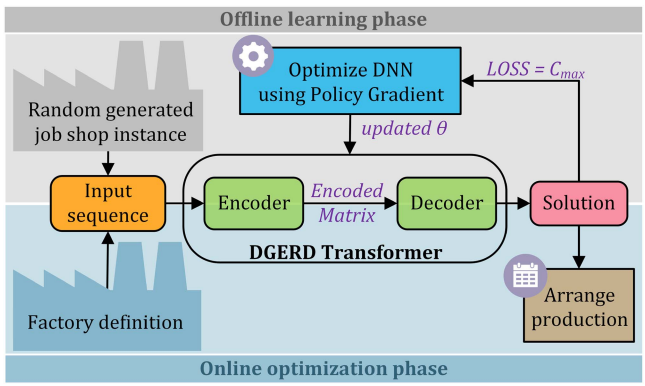

Fig. 5. Training and scheduling process of the DGERD transformer.

The training and scheduling process of the DGERD transformer framework is shown in Fig. 5. In the offline learning phase, a job-shop instance is generated randomly and then transformed into an input sequence. After encoding and decoding by the DGERD transformer once, the input sequence is mapped to a solution sequence. Finally, the parameters of the DNN are optimized using the policy gradient method by setting $C_{\max}$ as the loss. In the online optimization phase, the actual production information of the factory is taken as the input, and the solution sequence is also obtained after encoding and decoding. Production managers can arrange the production plan directly according to this solution sequence.

IV. Experimental Study

In the real production environment, customer orders, available processing equipment, and other factors that affect production are changing constantly. Therefore, the performance of the scheduling algorithm is not only measured by the quality of the solution but also by the speed of generating the solution. Scheduling methods based on deep learning can provide acceptable solutions in a short time, but training DNNs is time consuming. It is impractical to train and reserve a model for each problem structure, which requires that the model has good generalizability and can solve as many problems as possible after the completion of training. Therefore, the generalizability of the algorithm and the ability to generate high-quality solutions quickly are of the utmost importance and were our guidelines when designing the DGERD transformer algorithm.

The test instances herein are all from the international standard library OR-Library, covering instance sizes from 15×15 to 100×20 (jobs $\times$ machines). The test instances are not added to the training set or validation set. In each experiment, we trained our model only once (including multiple episodes) and then evaluated it in instances of various sizes to test the model's generalizability and potential for practical application in production. The experimental conditions were as follows: Tensorflow2 platform, Linux operating system, Intel i7-8750H CPU.

A. Model Training and Hyperparameter Determination

The training sets were generated randomly and contained JSSP instances of different sizes. In practice, to improve the training efficiency and optimize the model along a more accurate gradient, we packaged multiple problem instances, whose sizes were the same, into a training batch and made them interact with the model together. After training on a batch of instances for several epochs, the model continued training on another one with different instances size until the convergence of $C_{\max}$ was observed in an independent validation instance.

Determining appropriate hyperparameters is very important for DRL models. Considering that the search space of hyperparameters is huge and it is very challenging to find the optimal hyperparameters, we used a random search to optimize the hyperparameters. We perform a series of hyperparameter tuning tests and observe the performances of these hyperparameters on instance ta41. The top 5 results are shown in Table I. The hyperparameters in group No. 1 were used for subsequent experiments.

Among the hyperparameters, p and q, which guide the graph embedding, affect the performance the most. We tested the combinations $(p, q) \in \{(0.5, 1), (1, 0.5), (1, 1), (2, 1), (1, 2), (1, 4), (2, 4)\}$, and all except (2, 4) made the model converge, but the solution quality ($C_{\max}$, the smaller the better) differed greatly, as shown in Fig. 6(a). According to the experimental results, the combinations (1, 0.5), (0.5, 1), and (1, 4) do not lead to disjunctive embeddings of reliable quality and make the scheduling performance unstable. The combination (2, 1) yields the best results and is the most stable. A possible explanation for this result is that a reasonable strategy that encourages moderate exploration (return parameter $p = 2$) and attaches importance to both the breadth and depth of searching (in–out parameter $q = 1$) obtains high-quality embeddings steadily when parsing disjunctive graphs.

Other significant hyperparameters are those that pertain to the DNN structure, especially the numbers of encoder and decoder layers. Six combinations (encoder, decoder)$\in\{(3, 4), (4, 3), (4, 4), (5, 4), (4, 5), (4, 6)\}$ were compared in the experiments, and the results are shown in Fig. 6(b). According to the experimental results, a medium-size structure performs best, whereas a network structure that is either too deep or too shallow causes performance degradation.

TABLE I
Top Five, Results in Hyperparameter Tuning

No.	1	2	3	4	5
Return parameter p	2	2	2	2	2
In-out parameter q	1	1	1	1	1
Number of walks	10	15	10	20	15
Walk length	80	90	70	60	70
Number of encoder layers	4	4	4	3	4
Number of decoder layers	4	3	4	4	4
Number of attention-heads	4	6	6	2	4
d_{model}	16	32	32	16	16
Number of FNN units	64	256	64	128	256
Dropout rate	0.1	0.1	0.5	0.2	0.1
Learning rate	0.001	0.001	0.001	0.001	0.001
Decay β	0.9	0.95	0.9	0.95	0.9
$C_{\max}$	2382.63	2395.50	2420.76	2446.05	2476.61

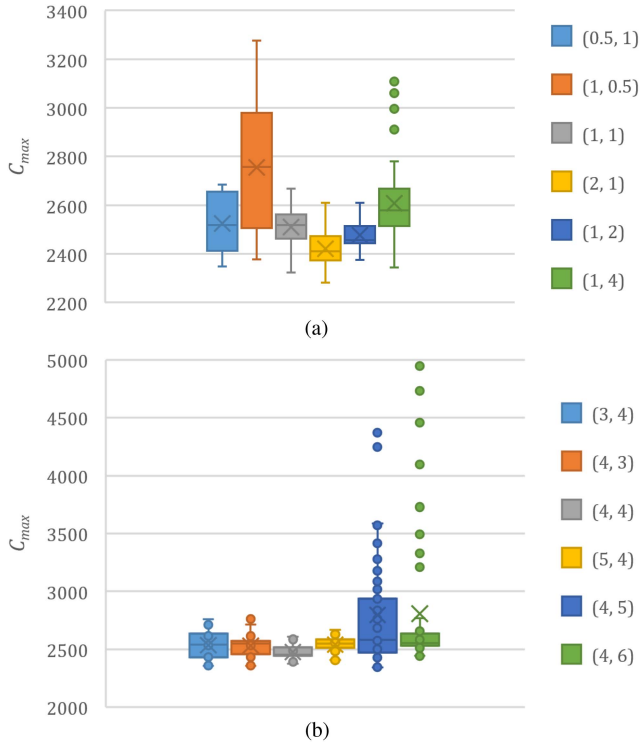

Fig. 6. Scheduling results of the DGERD transformer under different hyperparameters. (a) Return parameter *p* and in–out parameter *q*. (b) Network structure (the number of coder layer and decoder layer).

B. Experimental Results and Analysis

Facing the complexity of real-world production, a strategy that is popular with many dispatchers is to follow one of several

TABLE II
HEURISTIC DISPATCHING RULES

Rule	Content
SPT	Select the job with the shortest processing time
LPT	Select the job with the longest processing time
SRM	Select the job with the shortest remaining machining time
SRPT	Select the job with the shortest remaining processing time
SSO	Select the job with the shortest processing time of subsequent operation
LSO	Select the job with the longest processing time of subsequent operation
TWK	Select the job with the minimum total working time
TWKR	Select the job with the minimum total working time remaining

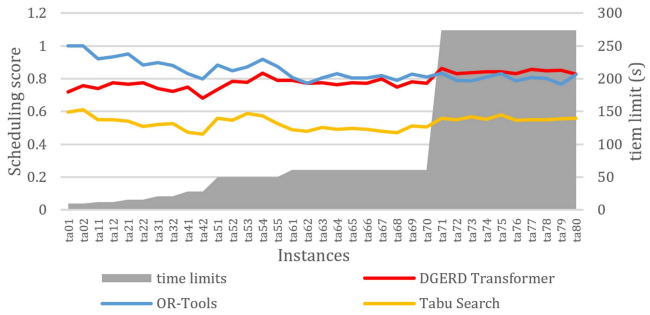

Fig. 7. Comparison of the experimental results using the DGERD transformer, tabu search, and OR-Tools under time limits.

heuristic dispatching rules (as given in Table II) to determine the scheduling order. Therefore, we begin by comparing the DGERD transformer with 16 heuristic dispatching rules (including combination rules [17]). The indicator of the comparison is $C_{\max}$, and the experimental results for the DGERD transformer are averaged over 30 repeated experiments. The results are given in Table III (where the best results are given in bold).

The results show that, in most of the instances, the DGERD transformer is superior to all of the heuristic dispatching rules (except in the smallest-scale instances ta01 and ta02). Comparing with the best rule in each instance, the average performance improvement of the DGERD transformer is 2.69%.

To test the performance of the DGERD transformer further, we compared it with a meta-heuristic algorithm [tabu search (TS) [32]] and Google OR-Tools. TS has been used widely in the JSSP, but it takes many iterations to converge when solving a large-scale scheduling problem. OR-Tools is a fast and powerful software for combinatorial optimization. It uses state-of-the-art algorithms to narrow the search set and can find an optimal (or suboptimal) solution. Considering that often, in practice, the scheduling requirement is not finding the optimal solution but obtaining a high-quality solution in the shortest time possible, we conducted comparative experiments within the execution time limit (shown in Fig. 7). The limit time is 1.1 times the run time of the DGERD transformer, and the results are presented as an average scheduling score (scheduling score $= C_{\text{LB}}/C_{\text{alg}}$, where C_{LB} is the lower bound of an instance and C_{alg} is the minimum makespan obtained by an algorithm).

In Fig. 7, the experimental results show that OR-Tools is a very competitive solver. It obtained outstanding solutions in most instances. For the small-scale cases (ta01 and ta02), it can

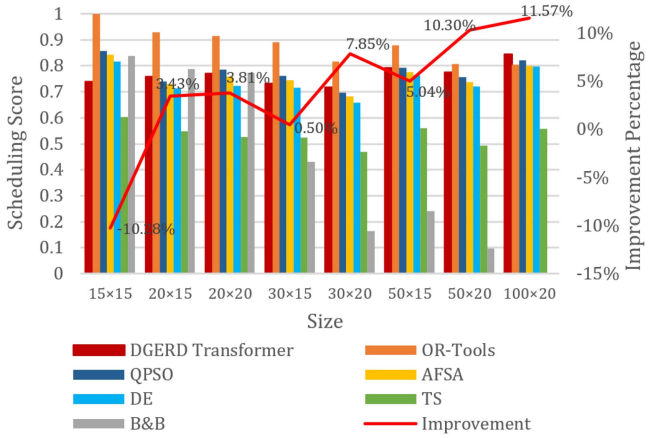

Fig. 8. Comparative experimental results of the DGERD transformer and other well-known algorithms on eight groups of instances with different sizes.

obtain optimal solutions stably within the time limit (scheduling score $= 1$). However, as the size of the problem increases, the scheduling score of OR-Tools decreases gradually. The scheduling score of TS has no significantly downward trend, but the solution quality obtained by TS is not ideal in the limited time, making it inferior to the other two algorithms. The performance of the DGERD transformer is not as good as that of OR-Tools when solving small-scale problems, but the gap between them narrows with increasing problem scale. When the problem size exceeds 50×20 (from ta71 to ta80), the DGERD transformer outperforms OR-Tools.

Three meta-heuristic algorithms and an exact algorithm are further added to test the performance of the DGERD transformer on eight sets of instances. The former includes quantum particle swarm optimization (QPSO) algorithm [33], artificial fish swarm algorithm (AFSA) [34], and differential evolution (DE) algorithm [35], which perform well in JSSP. As one of the best-known exact algorithms, a branch and bound (B&B) algorithm [3] developed for JSSP is also used for comparison. It optimizes the solution of JSSP with the help of a disjunctive graph. Same as the previous experiments, we conducted comparative experiments within the execution time limit. The running times needed for the DGERD transformer to solve these eight groups of experiments are 9.8, 12.1, 16.4, 20.4, 27.5, 50.0, 64.3, and 273.1 s, respectively. The limit time is set to 1.1 times the run time of the DGERD transformer. The results (shown in Fig. 8) are presented as an average scheduling score.

From Fig. 8, we can see that the performances of the DGERD transformer and meta-heuristic algorithms are stable in different instances. Their standard deviations are close to 0.04, lower than 0.06 of OR-Tools and 0.28 of the branch and bound algorithm. This is because although the DGERD transformer used the same parameters in all instances, the model had been trained on instances of different sizes; so it is stable across different test sets. Such stability depends on the strong generalizability of the DGERD transformer algorithm. Compared with TS, other three meta-heuristic algorithms can guarantee the quality of the solutions in a short computing time. The DGERD transformer achieved better results than meta-heuristic algorithms when the

TABLE III
COMPARISON OF THE EXPERIMENTAL RESULTS USING THE DGERD TRANSFORMER AND HEURISTIC DISPATCHING RULES

Instance	Size	DGERD transformer	SPT	LPT	SRM	SRPT	SSO	LSO	LPT+ LSO	LPT* TWK	LPT/ TWK	LPT* TWKR	LPT/ TWKR	SPT+ SSO	SPT* TWK	SPT* TWK	SPT* TWKR	SPT/ TWKR
ta01	15×15	1711.07	1872	1812	2163	2148	2148	1957	1892	1762	1973	1860	1984	2199	1926	**1647**	2204	1664
ta02	15×15	1639.30	1913	1562	1814	2114	1905	1759	1950	1598	1761	1757	1764	2104	1933	1778	1957	**1538**
ta11	20×15	**1833.00**	2273	2117	2353	2442	2343	2216	2471	2073	2163	1930	2237	2045	2358	2037	2184	1886
ta12	20×15	**1765.07**	2527	2213	2459	2160	2253	2187	2174	2142	2179	1908	2207	2176	2406	2160	2556	1969
ta21	20×20	**2145.63**	2488	2691	3071	2955	2610	2647	2443	2648	2770	2802	2732	2898	3019	2338	2789	2206
ta22	20×20	**2015.89**	2510	2515	2796	2726	2636	2522	2500	2613	2422	2348	2545	2678	2666	2788	2822	2111
ta31	30×15	**2382.63**	2993	2589	3101	3156	2916	2478	2670	2622	2565	2491	2636	2976	3001	2619	3154	2435
ta32	30×15	**2458.52**	3050	2624	3166	3272	2890	2634	2773	2681	2509	2562	2649	2847	3271	2663	2981	2512
ta41	30×20	**2541.22**	3105	3155	3482	3232	3058	2873	3129	3158	3222	3014	3315	3148	3362	3266	3269	2898
ta42	30×20	**2762.26**	3772	3356	3641	3624	3528	3096	3199	3348	3194	2954	3043	3358	3738	3361	3965	2813
ta51	50×15	**3762.60**	4456	3881	4174	4443	4418	3844	3873	3902	3859	3833	3956	4099	4460	4058	4590	3768
ta52	50×15	**3511.20**	4179	3891	4588	4371	4059	3715	3878	3900	4023	3678	3976	4187	4556	3724	4694	3588
ta61	50×20	**3633.48**	4500	4467	5024	5041	4520	4188	4432	4409	4258	3943	4283	4523	4813	4174	4933	3752
ta62	50×20	**3712.30**	4933	4416	4764	4821	4757	4217	4551	4365	4485	4000	4494	4699	4863	4323	5310	3925
ta71	100×20	**6321.22**	7830	6949	7916	8118	7594	6754	7065	6874	7067	6819	7264	7638	8186	7445	7912	6705
ta72	100×20	**6232.22**	7611	6675	7607	7639	7077	6674	6689	6758	7199	6471	7011	7601	7640	6910	7643	6351

instance size is larger than 30 × 15. The branch and bound algorithm achieved high scheduling scores in the first three groups of instances, but in the following groups, its scores declined significantly with the increase of problem size. This is due to the fact that the large-scale problems have too many branching possibilities. In addition, the branch and bound algorithm is extremely memory intensive. It could not solve the instances with size greater than 50 × 20 because of the serious memory overflow.

The average improvements of the DGERD transformer compared to other algorithms are calculated for each group of instances (shown in Fig. 8, red broken line). It is observed that although the DGERD transformer does not perform well for the first group of instances, its advantages generally grow as the scale of problems increases. It achieved remarkable results in large-scale problems. For the last group of instances, it outperformed the other algorithms by 11.57%.

The above experimental results indicate that the DGERD transformer is more suitable for dealing with large-scale scheduling problems, and its performance at solving large-scale problems in limited time surpasses that of some state-of-the-art algorithms or solvers.

There are two possible reasons for the good performance of the DGERD transformer in large-scale scheduling.

1) Attention mechanism can help agents to learn long-range dependencies better. Even when dealing with complex large-scale scheduling problems, models can still focus attention on important information. On the contrary, the baseline methods are difficult to find high-quality solutions in a large solution space with time limit.

2) The fast computing speed further makes the DGERD transformer more competitive. Its encoding process is mainly parallel matrix computation, which accelerates the running of the model. In our experiments, the running time of the encoder is less than 0.31 s even for the instances of size 100 × 20. But the baseline algorithms may take more time to converge.

V. CONCLUSION

Herein, we proposed a DRL framework known as the DGERD transformer for solving the JSSP. It integrates cutting-edge techniques such as disjunctive graph embedding, an attention mechanism, and DRL. We innovatively used graph embedding techniques to extract disjunctive graph features that are used to support the scheduling decisions. The proposed DGERD transformer takes the place of traditional models such as DQN and DDPG. It is based on a multihead attention mechanism and has a parallel encoder and a recurrent decoder. Benefiting from the self-attention mechanism, the DGERD transformer can better learn the long-range dependencies in the sequence, making it more advantageous for solving large-scale problems. Compared with the existing DRL-based JSSP algorithms, the DGERD transformer has better generalizability. It can solve problems of different sizes after one training, thereby giving it more potential in practical production. The experimental results showed that in the vast majority of the test instances, the DGERD transformer framework significantly outperformed some of the mainstream heuristic dispatching rules. In a reasonable execution time, the solutions obtained by the DGERD transformer were of better quality than those from the TS algorithm in all test instances. As the instance size exceeds 30 × 15, the DGERD transformer produces better results than other methods (except Google OR-Tools). When the problem size exceeds 50 × 20, the DGERD transformer outperforms OR-Tools. With good generalizability and the ability to solve large-scale problems, the DGERD transformer framework makes it possible to solve rolling scheduling or even real-time scheduling effectively in complex factories with dynamic and changeable production environments.

This study only showed the application potential of graph embedding and an attention mechanism in large-scale JSSPs. Exploring new disjunctive graph embedding strategies and trying different DNN structures to improve the performance of the model and reduce the response time further is a worthy direction for further research and exploration.

REFERENCES

[1] Y. Fang, C. Peng, P. Lou, Z. Zhou, J. Hu, and J. Yan, "Digital-twin-based job shop scheduling toward smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 12, pp. 6425–6435, Dec. 2019.

[2] H. M. Wagner, "An integer linear-programming model for machine scheduling," *Nav. Res. Logistics Quart.*, vol. 6, no. 2, pp. 131–140, 1959.
[3] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete Appl. Math.*, vol. 49, no. 1, pp. 107–127, 1994.
[4] E. Aarts, E. H. Aarts, and J. K. Lenstra, *Local Search in Combinatorial Optimization*. Princeton, NJ, USA: Princeton Univ. Press, 2003.
[5] H. Gao, S. Kwong, B. Fan, and R. Wang, "A hybrid particle-swarm tabu search algorithm for solving job shop scheduling problems," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2044–2054, Nov. 2014.
[6] M. H. Zahmani, B. Atmani, A. Bekrar, and N. Aissani, "Multiple priority dispatching rules for the job shop scheduling problem," in *Proc. 3rd Int. Conf. Control, Eng. Inf. Technol.*, 2015, pp. 1–6.
[7] A. Xanthopoulos and D. E. Koulouriotis, "Cluster analysis and neural network-based metamodeling of priority rules for dynamic sequencing," *J. Intell. Manuf.*, vol. 29, no. 1, pp. 69–91, 2018.
[8] K. Tamssaouet, S. Dauzère-Pérès, and C. Yugma, "Metaheuristics for the job-shop scheduling problem with machine availability constraints," *Comput. Ind. Eng.*, vol. 125, pp. 1–8, 2018.
[9] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Manage. Sci.*, vol. 34, no. 3, pp. 391–401, 1988.
[10] 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, Jan. 1996.
[11] 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.
[12] 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, 2019.
[13] Y. P. S. Foo and Y. Takefuji, "Stochastic neural networks for solving job-shop scheduling. I. Problem representation," in *Proc. IEEE Int. Conf. Neural Netw.*, 1988, pp. 275–282.
[14] M. Adibi, M. Zandieh, and M. Amiri, "Multi-objective scheduling of dynamic job shop using variable neighborhood search," *Expert Syst. Appl.*, vol. 37, no. 1, pp. 282–287, Jan. 2010.
[15] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–16.
[16] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–25.
[17] B. A. Han and J. J. Yang, "Research on adaptive job shop scheduling problems based on dueling double," *IEEE Access*, vol. 8, pp. 186474–186495, Oct. 2020.
[18] C. C. Lin, D. J. Deng, Y. L. Chih, and H. T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4276–4284, Jul. 2019.
[19] J. A. Palombarini and E. C. Martínez, "Automatic generation of rescheduling knowledge in socio-technical manufacturing systems using deep reinforcement learning," in *Proc. IEEE Biennial Congr. Argentina*, 2018, pp. 1–8.
[20] B. Waschneck *et al.*, "Deep reinforcement learning for semiconductor production scheduling," in *Proc. 29th Annu. SEMI Adv. Semicond. Manuf. Conf.*, 2018, pp. 301–306.
[21] 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.
[22] 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, Apr. 2020.
[23] B. Waschneck *et al.*, "Optimization of global production scheduling with deep reinforcement learning," *Procedia Cirp*, vol. 72, pp. 1264–1269, Jan. 2018.
[24] I. B. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Trans. Automat. Sci. Eng.*, vol. 17, no. 3, pp. 1420–1431, Jul. 2019.
[25] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proc. 22th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 855–864.
[26] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for e-commerce recommendation in Alibaba," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 839–848.
[27] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Adv. Neural Inf. Process. Syst.*, vol. 28, pp. 2692–2700, 2015.
[28] A. Vaswani *et al.*, "Attention is all you need," in *Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[29] B. Roy and B. Sussmann, "Les problemes d'ordonnancement avec contraintes disjonctives," *Note ds*, no. 9, 1964.
[30] E. Balas, "Machine sequencing via disjunctive graphs: An implicit enumeration algorithm," *Oper. Res.*, vol. 17, no. 6, pp. 941–957, 1969.
[31] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. IEEE Workshop Spoken Lang. Technol.*, 2013, pp. 414–419.
[32] S. Ponnambalam, P. Aravindan, and S. Rajesh, "A tabu search algorithm for job shop scheduling," *Int. J. Adv. Manuf. Technol.*, vol. 16, no. 10, pp. 765–771, 2000.
[33] M. R. Singh and S. S. Mahapatra, "A quantum behaved particle swarm optimization for flexible job shop scheduling," *Comput. Ind. Eng.*, vol. 93, pp. 36–44, 2016.
[34] D. Pythaloka, A. T. Wibowo, and M. D. Sulistiyo, "Artificial fish swarm algorithm for job shop scheduling problem," in *Proc. 3rd Int. Conf. Inf. Commun. Technol.*, 2015, pp. 437–443.
[35] W. Wisittipanich and V. Kachitvichyanukul, "Differential evolution algorithm for job shop scheduling problem," *Ind. Eng. Manage. Syst.*, vol. 10, no. 3, pp. 203–208, 2011.

Ruiqi Chen received the B.Eng. degree in industrial engineering from Soochow University, Suzhou, China, in 2021.

He is currently a Research Intern with Advanced Planning and Scheduling (APS) Laboratory, Soochow University, and is responsible for the design of deep reinforcement learning algorithm for APS systems. He is a leader of an intelligent manufacturing system project. His research interests include production scheduling, intelligent manufacturing, deep reinforcement learning, and combinatorial optimization problem.

Mr. Chen's project won the Challenge Cup, and the "Internet+" Innovation and Entrepreneurship Competition in Soochow University. He was a winner of the Chinese College Students Mechanical Engineering Innovation and Creativity Competition (Intelligent Manufacturing Competition) in 2019.

Wenxin Li received the B.Eng. degree in industrial engineering from Soochow University, Suzhou, China, in 2021. She is currently working toward the M.Sc. degree in management science and engineering with Shanghai University, Shanghai, China.

Her main research interests include production scheduling, logistics and supply chain management, operation research, and deep reinforcement learning.

Hongbing Yang received the Ph.D. degree in control theory and application from Southeast University, Nanjing, China, in 2009.

From 2014 to 2015, he was a Visiting Scholar with the University of Toronto, Toronto, ON, Canada. He is currently an Associate Professor with the School of Mechanical and Electrical Engineering, Soochow University, Suzhou, China. His research interests include deep reinforcement learning, intelligent manufacturing, modeling, and simulation of production systems.