2-2023

Flexible job-shop scheduling via graph neural network and deep reinforcement learning

Wen SONG

Xinyang CHEN

Qiqiang LI

Zhiguang CAO
Singapore Management University, zgcao@smu.edu.sg

Citation

SONG, Wen; CHEN, Xinyang; LI, Qiqiang; and CAO, Zhiguang. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. (2023). *IEEE Transactions on Industrial Informatics*. 19, (2), 1600-1610.

Available at: https://ink.library.smu.edu.sg/sis_research/8197

Flexible Job Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning

Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao

***Abstract*—Recently, Deep Reinforcement Learning (DRL) has been applied to learn priority dispatching rules (PDRs) for solving complex scheduling problems. However, existing works face challenges in dealing with flexibility, which allows an operation to be scheduled on one out of multiple machines and is often required in practice. Such one-to-many relationship brings additional complexity in both decision-making and state representation. This paper considers the well-known Flexible Job-shop Scheduling Problem (FJSP), and addresses these issues by proposing a novel DRL method to learn high-quality PDRs end-to-end. The operation selection and machine assignment are combined as a composite decision. Moreover, based on a novel heterogeneous graph representation of scheduling states, a Heterogeneous Graph Neural Network based architecture is proposed to capture complex relationships among operations and machines. Experiments show that the proposed method outperforms traditional PDRs and is computationally efficient, even on instances of larger scales and different properties unseen in training.**

***Index Terms*—Flexible Job-shop Scheduling, Graph Neural Network, Deep Reinforcement Learning.**

I. Introduction

CLOUD manufacturing, as an emerging paradigm of next generation manufacturing systems in the age of Industry 4.0, has received much attention in the past decade [1]. It virtualizes and integrates distributed manufacturing resources into a common cloud platform, so as to provide flexible, high-quality and on-demand manufacturing services, supported by advanced technologies such as Cyber-Physical Systems (CPS), Logistic Internet-of-Things (LIoT) [2], and Artificial Intelligence (AI). To unleash its full potential, effective resource scheduling is a critical factor for cloud manufacturing systems [3]. Its main task is to schedule the manufacturing demands (jobs) onto the manufacturing resources (machines), so as to achieve the optimal system performance. However, different from traditional production, resource scheduling for cloud manufacturing is much more complicated. Its cloud-based open environment makes manufacturing scheduling much more diverse, flexible and dynamic. Moreover, scale of the scheduling problem required to be solved is often large, due to the large volume of involved demands and resources. Therefore, scheduling algorithms and systems for cloud manufacturing should be fast and adaptive in responding to the scheduling requests, and must be able to cope with manufacturing flexibility and large-scale problems, which is challenging to design and develop.

This paper focuses on the Flexible Job-shop Scheduling Problem (FJSP), a well-known generalization of the Job-shop Scheduling Problem (JSP) with wide applications in cloud manufacturing [4], [5]. Different from JSP, FJSP allows operations to be processed on any machine from a set of alternative ones, hence is more suitable in handling the flexibility and diversity of the task-resource relations in new manufacturing paradigms such as cloud manufacturing [6]. Due to its important application value, FJSP receives much attention in the literature [6], [7].

FJSP is well-known to be NP-hard since it is harder than JSP, which is already strongly NP-hard, due to the requirement of machine assignment decisions [6]–[8]. Therefore, exact methods such as mathematical programming and constraint programming often suffer from prohibitively long computational time, especially in solving large-sized problems. In practice, heuristics methods are often employed which sacrifice optimality for efficiency. Priority dispatching rule (PDR) is a well-known and practical heuristic scheduling method. It solves the scheduling problem in a constructive fashion, which iteratively dispatches jobs to machines based on some priority rules (e.g. First In First Out, FIFO) [9]. Meta-heuristics, such as evolutionary algorithms, form another popular paradigm of heuristics, which have also received much attention. They employ complicated search procedures to explore the solution space to find high-quality solutions [6]. Compared to meta-heuristics, PDRs are intuitive, easy to implementation and very fast in computation, making it more preferable in dealing with problems in cloud manufacturing, which are often large-scale and even dynamic [10].

While PDRs have been widely applied in practice, their scheduling quality is still quite far from optimality. This could result from the following reasons. First, the construction process is greedy based on the priority measure, which could be myopic. Second, the decisions are based mostly on information from the eligible jobs and machines at each step, while the global information is largely ignored. Finally, current PDRs are mainly designed based on human experience, which usually have no guarantee on the optimality and lack the ability of adapting to specific problems and situations. In the age of Industry 4.0, however, the development of new

This work is supported by the National Natural Science Foundation of China under Grant 62102228, and the Shandong Provincial Natural Science Foundation under Grant ZR2021QF063. *(Corresponding author: Qiqiang Li.)*

Wen Song and Xinyang Chen contributed equally.

Wen Song is with the Institute of Marince Science and Technology, Shandong University, Qingdao 266237, China (e-mail: wensong@email.sdu.edu.cn).

Xinyang Chen and Qiqiang Li are with the School of Control Science and Engineering, Shandong University, Jinan 250061, China (e-mail: chenxy19@mail.sdu.edu.cn, qqli@sdu.edu.cn).

Zhiguang Cao is with Singapore Institute of Manufacturing Technology (SIMTech), Singapore 138634 (email: zhiguangcao@outlook.com).

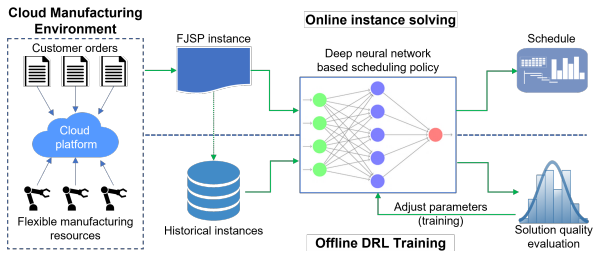


Fig. 1: Conceptual architecture of the DRL based method

technologies provides new impetus to overcome these limitations. In particular, large amounts of historical/simulation data about the frequently solved scheduling problem is much easier to obtain, from which advanced AI algorithms such as Deep Reinforcement Learning (DRL) can be utilized to discover better patterns for solving scheduling problem (conceptual architecture is shown in Fig. 1). In this direction, a number of recent works attempt to automatically generate PDRs for scheduling problems using DRL in an end-to-end fashion [11]–[16]. By viewing the decision making of PDRs as an Markov Decision Process (MDP), they train scheduling policies that take the future into account, so as to alleviate the myopic nature of PDRs. They employ deep neural networks to capture the global scheduling status instead of only local information. Furthermore, the reinforcement training process is guided by the cumulative reward towards the direction of optimizing performance, which is fully automatic without the need of human intervention [17].

Despite the promising results, most existing DRL methods only focus on non-flexible problems such as JSP. The existence of flexibility imposes two major challenges on the design of effective learning mechanism. First, decisions in FJSP are more complicated with not only operation selection but also machine assignment. Second, the scheduling status could be much harder to encode using neural network, due to the complex one-to-many relationships between operations and machines. Consequently, more complex decision-making framework and more informative representation techniques are necessary. The key research questions are: 1) how to formulate the scheduling process so as to incorporate machine assignment, and 2) how to design the representation scheme and neural architecture to extract useful information from raw scheduling states.

To address the above challenges, this paper proposes a novel end-to-end DRL method to learn high-quality PDRs for solving FJSP. For research question 1), this paper proposes an MDP formulation for PDR based FJSP scheduling, where an action is to select an eligible operation-machine pair, such that both the operation selection and machine assignment decisions can be made at the same time. For research question 2), by extending disjunctive graph of FJSP with machine nodes, this paper proposes a novel heterogeneous graph structure to represent the MDP states, such that the complicated relationships among operations and machines can be captured. Moreover, a two-stage Graph Neural Network (GNN) is proposed to obtain feature embeddings of the nodes in the heterogeneous graph, based on which a policy network is designed and trained using Proximal Policy Optimization (PPO). Different from the GNNs used in existing DRL based scheduling methods (e.g. [13], [14], [18]), the GNN proposed in this paper works on heterogeneous graphs specialized for FJSP, which captures the status for not only operations, but also machines and operation-machine relationships. Extensive experiments are conducted on synthetic instances and public benchmarks. Results show that while maintaining high computational efficiency, the proposed method can outperform traditional hand-crafted PDRs, and effectively generalize to larger-sized problems and public benchmarks unseen in training.

Besides the methodological novelty, the proposed method also has good practical value. Its neural architecture is size-agnostic, hence the trained policy can be applied to solve instances of varying sizes, not only the training size. More importantly, the trained policy can rapidly solve large-scale FJSP instances, and deliver reasonably good schedules better than traditional PDRs, making it a good choice for the management staff to optimize the production resource usage.

To summarize, this paper makes the following contributions:

- An end-to-end DRL method to solve FJSP, which can train size-agnostic policies that outperform traditional hand-crafted PDRs while maintaining high efficiency.
- An MDP formulation with an integrated decision-making approach, which combines the operation selection and machine assignment decisions as one decision.
- A heterogeneous graph structure for state representation, which effectively integrates operation and machine information with relatively low graph density.
- A Heterogeneous GNN based neural architecture, which extracts rich information from the heterogeneous state graph for high-quality scheduling decision making.

The rest content is organized as follows. Section II reviews recent DRL based scheduling methods. Section III describes FJSP and its graph representation. Section IV introduces the proposed method in detail. Section V provides experimental results and analysis. Section VI concludes the paper.

II. Related Works

In this section, we briefly review conventional FJSP methods, and the recent DRL based scheduling methods.

A. Conventional FJSP Methods

Conventional methods for solving FJSP can be classified into exact methods, heuristics and meta-heuristics [7]. Typical exact methods are mixed integer linear programming [19] and constraint programming [20]. They have theoretical guarantee of finding the optimal schedule, but requires exponentially long computational time. Heuristics are developed based on expert knowledge, which do not possess optimality guarantee but are considerably faster than exact methods. Typical FJSP heuristics include PDR [21], A* [22], local search [23], etc. Meta-heuristics can be further classified into single solution-based (e.g. simulated annealing [24], tabu search [25]) and population-based (e.g. genetic algorithm [26], [27]) methods, which works on a single solution or a pool of solutions, respectively. Complete reviews of FJSP methods can be found

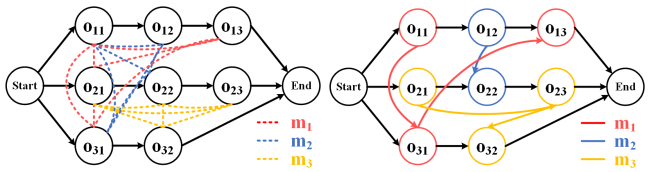
in [6], [7]. This paper focuses on PDRs, which are widely used in practical production systems due to its easy implementation and fast computation. The aim is to discover high-quality PDRs using DRL, and the relevant works are discussed below.

B. DRL based Scheduling Methods

Recently, quite a few works employ DRL to solve complex scheduling problems. A key issue in DRL based scheduling is state representation. For JSP, some researchers use vectors or matrices to represent states, and use Multilayer Perceptron (MLP) [10], [12], [28], [29] or Convolutional Neural Network (CNN) [11] to extract state features. A major limitation of such representation is that it is hard bounded by fixed matrix dimensions, and cannot solve problems of different sizes. A recent work [15] partially resolves this issue for Hybrid Flow-shop Scheduling Problem (HFSP), by viewing matrices as relationship between two item groups (jobs and machines), and use self-attention [30], a size-agnostic structure, to process each item. However, this model is still limited to fixed number of HFSP stages. For Permutation Flow-Shop Scheduling Problem (PFSP), a Recurrent Neural Network (RNN) based architecture is designed in [31] to handle varying number of jobs and machines. However, it is specific to PFSP and not applicable to FJSP.

An arguably better representation technique for scheduling problems is GNN [32] which can process graphs of varying sizes, thus overcomes the limitation of matrix representation. Zhang et al. [13] combine DRL and GNN to learn high-quality PDRs for JSP. They model states as disjunctive graphs with different connections, and use GNN to encode the state graphs. Park et al. [14] adopt a similar scheme using richer node (i.e. operation) features, and at the same time, considering the different relationships between nodes in the GNN messages passing process. Ni. et al [18] employ DRL to learn a local search heuristic for solving HFSP. They represent the solution at each search step as a multi-graph structure, and overcome the scale limitation in [15] by an attention based pooling across different stages.

Till now, most related works focus on solving JSP using DRL, and cannot be applied to FJSP due its additional complexity in decision-making and representation. For FJSP, the decision framework should be able to handle two types of decisions, i.e. operation selection and machine assignment, and the representation scheme should be able to extract more informative state features especially for the flexible machines. While HFSP is flexible due to the parallel machines at each stage, its problem structure is significantly different from FJSP. The application of DRL in FJSP is rather sparse. Luo et al. [33] employ deep Q-network to solve dynamic FJSP. However, the action is to select from a pool of hand-crafted PDRs, which heavily relies on human experience. Han and Yang [34] propose an end-to-end DRL method for FJSP based on a 3D disjunctive graph. However, the attention based policy network they designed only processes raw features without considering the graph structure, hence is limited in extracting useful information for high-quality decision making.



(a) An instance (b) A solution

Fig. 2: Disjunctive graph of FJSP. Dotted line means processable, while solid line means scheduled.

III. Preliminaries

An FJSP instance of size $n \times m$ includes n jobs and m machines, forming two sets $\mathcal{J}$ and $\mathcal{M}$. Each job $J_i \in \mathcal{J}$ has an operation set $\mathcal{O}_i$, which contains n_i operations O_{ij} that must be processed in a specific order (i.e. precedence constraints). Each operation O_{ij} can be processed on any machine M_k from its compatible set $\mathcal{M}_{ij} \subseteq \mathcal{M}$ for a processing time p_{ijk} without preemption. Each machine can only process one operation at a time. To solve FJSP, one needs to assign each operation O_{ij} to a compatible machine and determine its start time S_{ij}, such that the makespan $C_{\max} = \max_{i,j}\{C_{ij}\}$ is minimized where C_{ij} is the completion time of O_{ij}.

A disjunctive graph for FJSP [35] can be written as $\mathcal{G} = (\mathcal{O}, \mathcal{C}, \mathcal{D})$. Specifically, $\mathcal{O} = \{O_{ij} | \forall i, j\} \cup \{Start, End\}$ is the node set, which includes all operations and two dummy ones (with zero processing time) representing the start and end of production. $\mathcal{C}$ is the set of conjunctive arcs, which are directed arcs that form n paths from $Start$ to End representing the respective processing sequence of J_i. $\mathcal{D} = \bigcup_k \mathcal{D}_k$ is the set of disjunctive arcs which are undirected, and $\mathcal{D}_k$ is a clique that connects the operations that can be processed on machine M_k. Note that unlike JSP, an operation in FJSP could be connected to multiple disjunctive arcs due to the flexibility. Solving FJSP is equivalent to selecting for each node a disjunctive arc, and fixing its direction. An illustration of disjunctive graph for FJSP is given in Fig. 2.

IV. Method

This section introduces the proposed method in detail. In this paper, solving FJSP is considered as a sequential decision-making process, which iteratively takes a scheduling action to assign an operation to a compatible machine at each state, until all operations are scheduled. The workflow of the proposed method is shown in Fig. 3. In each iteration, the scheduling state is first transformed into a heterogeneous graph structure (Section IV-B). Then, a Heterogeneous Graph Neural Network with a two-stage embedding process is applied to the heterogeneous graph to extract feature embeddings of the operations and machines (Section IV-C), which are consumed by the decision-making network to generate the action probability distribution, from which a scheduling action is sampled (Section IV-D). In the following subsection, the MDP formulation of the above process is first presented.

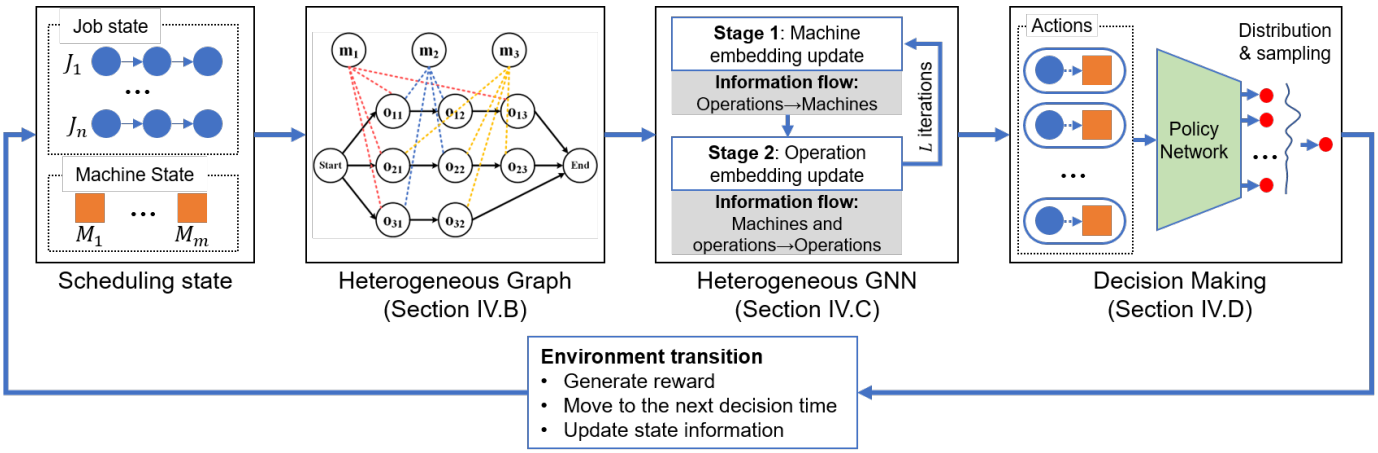


Fig. 3: The workflow of the proposed method.

A. MDP Formulation

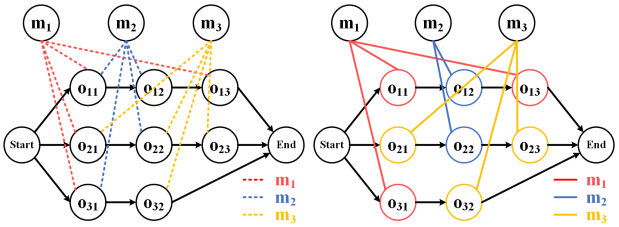
The scheduling process considered here works as follows. At each decision step t (time 0 or when an operation is completed), the agent observes the current system state s_t and makes a decision a_t, which is to allocate an unscheduled operation to an idle machine and start it from the current time, denoted as $T(t)$. Then the environment transits to the next decision step $t+1$. The process iterates until all the operations are scheduled. The corresponding MDP is defined below.

State. The conditions of all operations and machines at step t constitute state s_t. The initial state s_0 is an FJSP instance drawn from a distribution. Note that for each s_t, a partial schedule $S(t)$ is maintained, which is computed as follows. If O_{ij} is scheduled, its start time $S_{ij}(t)$ is the actual value S_{ij}. Otherwise, $S_{ij}(t)$ is an estimate computed with only precedence constraints. If its immediate predecessor O_{il} is scheduled on machine M_k, then $S_{ij}(t) = S_{il} + p_{ilk}$; otherwise $S_{ij}(t) = S_{il}(t) + \bar{p}_{il}$, where $\bar{p}_{ij} = \sum_{M_k \in \mathcal{M}_{ij}} p_{ijk}/|\mathcal{M}_{ij}|$ is an estimated processing time of O_{ij}. $S_{ij}(t)$ can be computed recursively for all the unscheduled operations in the direction from $Start$ to End.

Action. This paper uses an integrated approach to solve FJSP, which combines the operation selection and machine assignment as a composite decision. Specifically, an action $a_t \in A_t$ is defined as a feasible operation-machine pair (O_{ij}, M_k) at step t, where O_{ij} is eligible (i.e. its immediate predecessor is completed) and $M_k \in \mathcal{M}_{ij}$ is idle. O_{ij} starts on M_k immediately, i.e. $S_{ij} = T(t)$. The action set A_t is step dependent, which collects all the feasible pairs. Since each job can have at most one operation ready at a time and $|\mathcal{M}_{ij}| \leq m$, therefore $|A_t| \leq n \times m$.

Transition. Based on s_t and a_t, the environment deterministically transits to a new state s_{t+1}, which is the time when an operation is completed. In this paper, two difference states are distinguished by the topology and features of the corresponding heterogeneous graph structure.

Reward. The reward is defined as the difference between the makespan of the partial schedule at s_t and s_{t+1}, i.e. $r(s_t, a_t, s_{t+1}) = C_{\max}(s_t) - C_{\max}(s_{t+1})$. When the discount factor $\gamma = 1$, the cumulative reward in an solving episode is $G = \sum_{t=0}^{|\mathcal{O}|} r(s_t, a_t, s_{t+1}) = C_{\max}(s_0) - C_{\max}$. For a specific problem instance, $C_{\max}(s_0)$ is a constant, which means that minimizing $C_{\max}$ and maximizing G are equivalent.



(a) An instance (b) A solution

Fig. 4: Heterogeneous graph of FJSP. Dotted line means processable, while solid line means scheduled.

Policy. A policy $\pi(a_t|s_t)$ defines for each state s_t a probability distribution over the action set A_t. Later in this section, an DRL algorithm will be designed which parameterizes π as a neural network and optimizes it towards the direction of maximizing the expected cumulative reward.

B. Heterogeneous Graph

Previous works have employed disjunctive graph to represent JSP scheduling states and achieved good results. However, the disjunctive graph of FJSP is more complicated. First, the disjunctive arc set $\mathcal{D}$ could be much larger since operations can be processed by multiple machines. Such dense graph is hard to be efficiently processed [13]. Second, the processing times of an operation on different compatible machines are different, making it difficult to represent. To resolve the above issue, this paper defines a novel heterogeneous graph structure $\mathcal{H} = (\mathcal{O}, \mathcal{M}, \mathcal{C}, \mathcal{E})$ by modifying disjunctive graph. As shown in Fig. 4, the operation node set $\mathcal{O}$ and conjunctive arcs set $\mathcal{C}$ are kept, and a set of machine nodes $\mathcal{M}$ is added, each for one machine M_k. The disjunctive arc set $\mathcal{D}$ is replaced with an operation-machine (O-M) arc set $\mathcal{E}$, where each element $E_{ijk} \in \mathcal{E}$ is an undirected arc connects an operation node O_{ij} with a compatible machine node M_k.

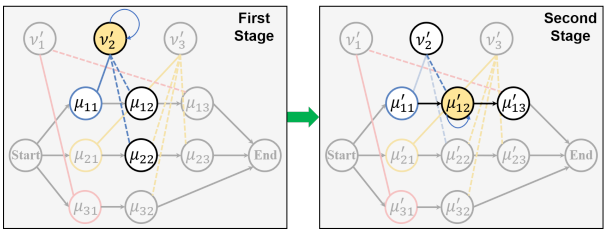


Fig. 5: The two-stage embedding scheme. Update of machine embedding ν'_2 and operation embedding μ'_{12} are highlighted for illustration.

The above heterogeneous graph representation has several advantages over the original FJSP disjunctive graph. First, the graph density is significantly reduced. Suppose for each machine M_k there are n_k processable operations, then $|\mathcal{D}| = \sum_k \binom{n_k}{2}$, while $|\mathcal{E}| = \sum_k n_k$. It is easy to prove that when $n_k > 3$ holds for all M_k, $|\mathcal{D}| > |\mathcal{E}|$, and the difference between $|\mathcal{D}|$ and $|\mathcal{E}|$ grows quadratically with the increase of n_k. For large-sized problems, the heterogeneous graph $\mathcal{H}$ requires much less O-M arcs than the disjunctive arcs in $\mathcal{G}$. Second, the machine nodes in $\mathcal{H}$ provide a convenient way to inject machine information and extract useful features to distinguish different machines in a state. Such information is very important to solve FJSP since operations need to be allocated to a suitable machine, and is difficult to obtain in $\mathcal{G}$. Finally, the processing time p_{ijk} is easily represented by simply attaching as a feature to the O-M arc E_{ijk}.

Based on the above definition of $\mathcal{H}$, this paper represents each state s_t as a heterogeneous graph $\mathcal{H}_t = (\mathcal{O}, \mathcal{M}, \mathcal{C}, \mathcal{E}_t)$, where $\mathcal{E}_t$ dynamically changes during a solving episode. Specifically, after an action (O_{ij}, M_k) is taken at step t, Only E_{ijk} is kept and other O-M arcs of O_{ij} are removed to obtain $\mathcal{H}_{t+1}$. Hence, the neighboring relationship among nodes also dynamically changes. For each step t, let $\mathcal{N}_t(O_{ij})$ be the neighboring machines of operation O_{ij}, and $\mathcal{N}_t(M_k)$ be the neighboring operations of machine M_k. For each operation, machine and O-M arc, raw feature vector $\mu_{ij} \in \mathbb{R}^6$, $\nu_{ij} \in \mathbb{R}^3$, $\lambda_{ij} \in \mathbb{R}$ are defined to reflect their states at step t. Detailed definition can be found in the appendix.

C. Heterogeneous Graph Neural Network

As typical in combinatorial problems, FJSP instances have varying sizes. To learn practical scheduling policy using DRL, the neural architecture must be able to operate on state graphs of different sizes. Previous works [13], [14], [18] have shown that GNN can achieve this size-agnostic property. However, they all deal with homogeneous graphs and hence are not applicable here. In the general GNN literature, research on heterogeneous graph is rather sparse [32]. While some Heterogeneous Graph Neural Networks (HGNNs) are proposed recently [36]–[38], they do not consider the unique properties of the FJSP heterogeneous graph $\mathcal{H}_t$. First, different node types in $\mathcal{H}_t$ have strong connection patterns. Neighbors of any machine can only be operations connected by undirected arcs, while an operation could be connected to both operations and machines by directed or undirected arcs. Second, features on O-M arcs (i.e. processing times) are of great importance for solving FJSP. However, existing HGNNs usually focus on node features only and do not consider arc features.

To exploit the properties and advantages of the heterogeneous graph structure, this paper proposes a novel HGNN architecture customized for FJSP to effectively encode $\mathcal{H}_t$. As shown in Fig. 5, the proposed method is featured with a two-stage embedding process, so as to take graph topological and numerical information (raw features) into account, and map the nodes in $\mathcal{H}_t$ into d-dimensional embeddings. In the first stage, machine embeddings $\nu'_k \in \mathbb{R}^d$ are updated by aggregating relevant information, while operation embeddings $\mu'_{ij} \in \mathbb{R}^d$ are updated in the second stage. Details are given as follows.

1) Machine node embedding: In $\mathcal{H}_t$, neighbors of a machine M_k is a set of operations $\mathcal{N}_t(M_k)$ which may have different meanings to M_k. For example, operations that are expected to start sooner might be more important than those who start later. This motivates us to consider Graph Attention Networks (GAT) [39], which automatically learns the importance of different nodes by applying the attention mechanism [30]. For a homogeneous graph, given a node i with feature x_i, GAT first computes the attention coefficient e_{ij} (a scalar) between i and each j in its first-order neighborhood $\mathcal{N}(i)$ (including i itself) as:

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{W}x_i || \mathbf{W}x_j\right]\right). \tag{1}$$

In other words, x_i and x_j are processed by a shared linear transformation $\mathbf{W}$ first, and then concatenated ($||$) and fed into a single-layer feedforward neural network with weights $\mathbf{a}$ and LeakyReLU activation. Then the coefficients are normalized across the neighborhood using softmax function:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{q\in\mathcal{N}(i)} \exp(e_{iq})}, \forall j \in \mathcal{N}(i). \tag{2}$$

Finally, GAT aggregates (linearly transformed) features over $\mathcal{N}(i)$ and applys a nonlinearity σ to get the embedding of i:

$$x'_i = \sigma\left(\sum\nolimits_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{W}x_j\right). \tag{3}$$

However, the original GAT is for homogeneous graphs only and does not consider arc features. Therefore, it is modified here to satisfy the need of this paper, which is to calculate the importance of a neighboring operation to a machine. First, it can be observed that for each machine M_k, there is only one O-M arc connects it with a neighboring operation. Therefore, the raw feature vector of each $O_{ij} \in \mathcal{N}_t(M_k)$ is extended by concatenating its original raw features with that of the corresponding O-M arc as $\mu_{ijk} = [\mu_{ij}||\lambda_{ijk}] \in \mathbb{R}^7$. Next, instead of using a shared one, here two linear transformations $\mathbf{W}^M \in \mathbb{R}^{d\times 3}$ and $\mathbf{W}^O \in \mathbb{R}^{d\times 7}$ are used for the machine nodes and operation nodes, respectively. Then for a machine M_k, the attention coefficients e_{ijk}, i.e. the importance of each neighboring operation $O_{ij} \in \mathcal{N}_t(M_k)$ can be calculated as:

$$e_{ijk} = \text{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{W}^M\nu_k || \mathbf{W}^O\mu_{ijk}\right]\right), \tag{4}$$

where $\mathbf{a} \in \mathbb{R}^{2d}$. In this way, information from heterogeneous nodes and O-M arcs can be effectively incorporated in the attention computation.

One thing not considered in Eq. (4) is the attention coefficient of machine M_k to itself, which is involved in the original GAT (see Eq. (1)). Here e_{kk} is computed using the machine specific weights $\mathbf{W}^M$ as follows:

$$e_{kk} = \text{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{W}^M \nu_k || \mathbf{W}^M \nu_k\right]\right). \quad (5)$$

Then all e_{ijk}, $\forall O_{ij} \in \mathcal{N}_t(M_k)$ are normalized together with e_{kk} using softmax function to obtain the normalized attention coefficients α_{ijk} and α_{kk}.

Finally, the machine embedding ν'_k is computed by fusing features from neighboring operations and itself. Due to the importance of processing time, the extended raw feature vector μ_{ijk} is used for each neighboring O_{ij}. The aggregation function for calculating ν'_k is as follows:

$$\nu'_k = \sigma\left(\alpha_{kk}\mathbf{W}^M \nu_k + \sum\nolimits_{O_{ij}\in\mathcal{N}_t(M_k)} \alpha_{ijk}\mathbf{W}^O \mu_{ijk}\right). \quad (6)$$

2) Operation node embedding: Different from machines, neighbors of an operation O_{ij} in $\mathcal{H}_t$ is of several types, including an immediate predecessor $O_{i,j-1}$, an immediate successor $O_{i,j+1}$, and the machines in $\mathcal{N}_t(O_{ij})$. Moreover, $O_{i,j-1}$ and $O_{i,j+1}$ are connected to O_{ij} by directed arcs from opposite direction, while $M_k \in \mathcal{N}_t(O_{ij})$ are connected by undirected arcs. Due to the heterogeneity of information sources and arc types, it is not very helpful and convenient to apply attention based mechanisms. Instead, this paper directly uses multiple MLPs to process information from each source (including the features of O_{ij} itself), concatenates the results, and projects it back to the d-dimensional space as the embedding of O_{ij}.

Specifically, five MLPs (denoted as MLP_{θ_0}, ..., MLP_{θ_4}) are defined, each with d-dimensional output, two d_h-dimensional hidden layers, and ELU activation. They are responsible for the final projection, and processing information from $O_{i,j-1}$, $O_{i,j+1}$ $M_k \in \mathcal{N}_t(O_{ij})$, and O_{ij}, respectively. Since $\mathcal{N}_t(O_{ij})$ may have multiple machines, element-wise sum is applied to obtain $\bar{\nu}'_{ij} = \sum_{k\in\mathcal{N}_t(O_{ij})} \nu'_k$ as the input to MLP_{θ_3}. The computation of embedding for O_{ij} is as follows:

$$\begin{aligned}\mu'_{ij} = \text{MLP}_{\theta_0}(\text{ELU}[&\text{MLP}_{\theta_1}(\mu_{i,j-1})||\text{MLP}_{\theta_2}(\mu_{i,j+1}) \\ &||\text{MLP}_{\theta_3}(\bar{\nu}'_{ij})||\text{MLP}_{\theta_4}(\mu_{ij})]).\end{aligned} \quad (7)$$

Note that there is no need to compute embeddings of the two dummy operations $Start$ and End.

3) Stacking and pooling: The above embedding process can be considered as a HGNN layer, which transforms raw features μ_{ij} and ν_k of each operation and machine into embeddings μ'_{ij} and ν'_k. To enhance feature extraction ability, here L HGNN layers with identical structure but independent trainable parameters are stacked to obtain the final embeddings $\mu'^{(L)}_{ij}$ and $\nu'^{(L)}_k$. Raw features μ_{ij} and ν_k of operation and machine nodes are only used in the first layer, while raw features λ_{ijk} of O-M arcs are used in all the L layers.

After the L layers of HGNN, mean pooling is applied to the obtained operation embedding set and machine embedding set separately. The resulting two d-dimensional vectors are then concatenated as the embedding $h_t \in \mathbb{R}^{2d}$ of the heterogeneous graph state $\mathcal{H}_t$ as follows:

$$h_t = \left[\frac{1}{|\mathcal{O}|}\sum\nolimits_{O_{ij}\in\mathcal{O}} \mu'^{(L)}_{ij} \parallel \frac{1}{|\mathcal{M}|}\sum\nolimits_{M_k\in\mathcal{M}} \nu'^{(L)}_k\right]. \quad (8)$$

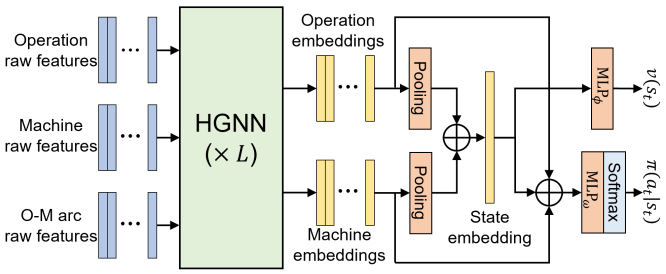


Fig. 6: The network architecture.

Through the above process, a varying-size heterogeneous graph can be transformed into a fixed-dimensional embedding. Let θ be the collection of all the HGNN parameters.

D. Decision Making

As the last part of the proposed neural architecture, below a policy network is designed. Remind that an action is a feasible operation-machine pair. Thanks to the above heterogeneous graph structure and HGNN, the policy $\pi(a_t|s_t)$ is easy and convenient to represent using the extracted embeddings. Specifically, for each feasible action $a_t = (O_{ij}, M_k) \in A_t$ at step t, the corresponding operation, machine and state embedding are concatenated, and fed into an MLP to get its priority index of being selected at state s_t as follows:

$$P(a_t, s_t) = \text{MLP}_\omega\left[\mu'^{(L)}_{ij} || \nu'^{(L)}_k || h_t\right], \quad (9)$$

where MLP_ω has two d_π-dimensional hidden layers and tanh activation. Then the probability of selecting each a_t is calculated by applying softmax over all $P(a_t, s_t)$:

$$\pi_\omega(a_t|s_t) = \frac{\exp\left(P(a_t, s_t)\right)}{\sum_{a'_t\in A_t}\exp\left(P(a'_t, s_t)\right)}, \forall a_t \in A_t. \quad (10)$$

During training, actions are sampled according to the policy π_ω, so as to enable exploration. For testing, there could be two strategies of utilizing a trained policy π_ω to solve a given instance, including 1) greedily picking actions with the maximum probability, and 2) sampling actions following π_ω at each state, same as in training. Different from the greedy one, the solutions delivered by the sampling strategy could be different each run due to the stochasticity. This provides additional possibilities of finding better solutions since in general, current DRL algorithms can only converge to sub-optimal policies. In particular, the sampling strategy solves N_s copies of a given instance to obtain N_s solutions, from which it picks the best one as in [40]. Note that for neural policies, the additional overhead of sampling is often small since GPU is able to sample solutions in parallel.

E. Training

This paper uses PPO [41] for training which employs an actor-critic structure. Actor is the policy network π_ω and critic v_ϕ is another network that predicts the value $v(s_t)$ of a state s_t. Here the critic is designed as an MLP, which takes input the state embedding h_t computed by the HGNN to get $v_\phi(s_t)$. MLP_ϕ has the same structure as MLP_π, i.e.

two hidden layers with d_ϕ dimension and tanh activation, but with different parameters ϕ and different input dimension ($2d$ instead of $4d$). The overall network architecture is shown in Fig. 6. As shown in Algorithm 1, the training is performed for $\mathcal{I}$ iterations, during which a batch of $\mathcal{B}$ instances (replaced every 20 iterations) are solved by the DRL agent in parallel (Line 3-10). During training, the policy is validated on a set of independent validation instances every 10 iterations.

Algorithm 1: Training procedure with PPO

Input: HGNN network, policy network and critic network with trainable parameters θ, ω and ϕ

1 Sample a batch of $\mathcal{B}$ FJSP instances;
2 **for** $iter = 1, 2, ..., \mathcal{I}$ **do**
3 **for** $b = 1, 2, ..., \mathcal{B}$ **do** `// In parallel`
4 Initialize s_t based on instance b;
5 **while** s_t *is not terminal* **do**
6 Extract embeddings using HGNN;
7 Sample $a_t \sim \pi_\omega(\cdot|s_t)$;
8 Receive reward r_t and next state s_{t+1};
9 $s_t \leftarrow s_{t+1}$;
10 Compute the generalised advantage estimates $\hat{A}_t$ for each step;
11 Compute the PPO loss $\mathcal{L}$, and optimize the parameters θ, ω and ϕ for $\mathcal{R}$ epochs;
12 Update network parameters;
13 **if** $iter \mod 10 = 0$ **then**
14 Validate the policy;
15 **if** $iter \mod 20 = 0$ **then**
16 Sample a new batch of $\mathcal{B}$ FJSP instances;
17 **return**;

V. Experiments

This section shows experimental results on synthetic and public FJSP instances to validate the proposed method.

A. Experimental Settings

1) Evaluation Instances: As in most related works, this paper generates synthetic FJSP instances for training and testing. The generation method is similar to the well-known procedure in [35]. As listed in Table I, six problem sizes are considered. For each size, an instance is sampled by drawing from the corresponding uniform distribution in Table I (p_{ijk} is sampled from $\mathrm{U}(0.8\bar{p}_{ij}, 1.2\bar{p}_{ij})$). This paper performs training on the four smaller sizes, and uses the largest two (30×10 and 40×10) to test the generalization capability of the trained policies. The training instances are generated on-the-fly (with 100 validation instances), while for testing 100 instances are sampled for each size. Besides the synthetic instances, two well-known FJSP benchmarks are used for generalization analysis, including the 10 mk instances (mk01-mk10) in [35] and the three groups of la instances (rdata, edata and vdata, each with 40 instances) in [42]. These instances are of various sizes (ranging from 10×6 to 30×10), drawing from distributions significantly different from those in Table I. Hence, testing on these benchmarks can further verify the proposed method in generalizing to out-of-distribution instances. More details about these instances can be found in [43].

TABLE I: Instance generation distributions

Size ($n \times m$)	n_i[1]	$\lvert\mathcal{M}_{ij}\rvert$[2]	$\bar{p}_{ij}$[3]
10×5	U(4, 6)	U(1, 5)	U(1, 20)
20×5	U(4, 6)	U(1, 5)	U(1, 20)
15×10	U(8, 12)	U(1, 10)	U(1, 20)
20×10	U(8, 12)	U(1, 10)	U(1, 20)
30×10	U(8, 12)	U(1, 10)	U(1, 20)
40×10	U(8, 12)	U(1, 10)	U(1, 20)

[1] Number of operations in Job J_i;
[2] Number of compatible machines for operation O_{ij};
[3] Average processing time of operation O_{ij}.

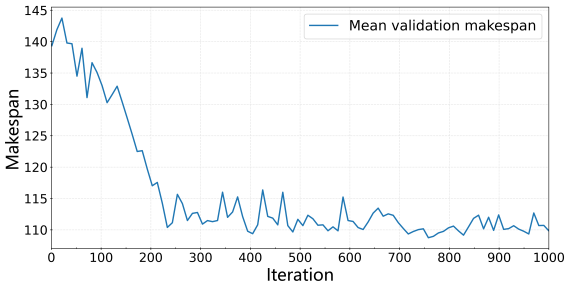


Fig. 7: Training curve on 10×5 instances.

TABLE II: Results on synthetic instances of training size

Size		OR-Tools[1]	DRL-G	DRL-S	MOR	SPT	FIFO	MWKR
10×5	$C_{\max}$	96.59 (15%)	111.67	**105.61**	116.69	129.06	119.62	115.29
	Gap	-	15.71%	**9.38%**	20.89%	33.57%	23.94%	19.44%
20×5	$C_{\max}$	188.45 (0%)	211.22	**207.50**	217.17	229.89	216.13	216.98
	Gap	-	12.10%	**10.13%**	15.29%	22.08%	14.73%	15.17%
15×10	$C_{\max}$	145.42 (5%)	166.92	**160.36**	173.40	198.20	185.08	169.18
	Gap	-	14.80%	**10.31%**	19.29%	36.30%	27.27%	16.34%
20×10	$C_{\max}$	197.24 (0%)	215.78	**214.87**	221.86	254.59	234.21	220.85
	Gap	-	9.43%	**8.97%**	12.53%	29.09%	18.82%	11.99%

[1] (.%): percentage of instances solved optimally within 1800 seconds.

2) Configuration: This paper sets the number of HGNN iterations as $L=2$, and dimensions of machine and operation embeddings as $d=8$. For the MLPs, the hidden dimensions are set as $d_h=128$, and $d_\pi=d_\phi=64$. For training, the number of training iterations and instance batch size is set to $\mathcal{I}=1000$ and $\mathcal{B}=20$. For the PPO loss function, the coefficients of the policy loss (with 0.2 clip ratio), value loss and entropy term are set as 1, 0.5 and 0.01, respectively. The PPO optimization epochs is set to $\mathcal{R}=3$, and discount factor is set to $\gamma=1.0$. The network is updated using the Adam optimizer, with learning rate $lr=2\times10^{-4}$. For testing, the trained policies are tested using both the greedy and sampling strategy (with $N_s=100$ solutions) mentioned in Section IV-D, named DRL-G and DRL-S, respectively. These hyperparameters are empirically tuned on the smallest problem size 10×5, and fixed on the remaining sizes. The hardware is a machine with Intel Xeon Gold 6152 CPU, one Nvidia Titan V GPU and Ubuntu 16.04 64-bit OS. The code in Pytorch is publicly available[1].

3) Baselines: The learned FJSP scheduling policies are compared with four well-known PDRs that work well in practice, including FIFO (Firt In First Out), MOR (Most

[1]https://github.com/songwenas12/fjsp-drl

Operations Remaining), SPT (Shortest Processing Time), and MWKR (Most Work Remaining) [35], [44]. For MWKR, average processing time $\bar{p}_{ij}$ is used to compute the priority, just as [35] did. For fair comparison, the baseline PDRs are implemented in the same environment as the proposed method, so as to verify that a learned policy is indeed better than a hand-crafted one. At each decision point, after an operation is selected by a PDR, it is assigned to the corresponding idle machine immediately. This paper also compares with Google OR-Tools[2], a powerful constraint programming solver showing strong performance in solving industrial scheduling problems [45]. Since OR-Tools is an exact solver, a time limit of 1800 seconds is set, and the obtained optimal or best solutions are used to evaluate solution quality of the learned or hand-crafted PDRs. For the public benchmarks, this paper also compares with results of the DRL method [34] and two recent Genetic Algorithms (GA) in [26] and [27], as well as the best known solutions collected in [43]. For each solution with makespan $C_{\max}$, its relative gap to the makespan $C^{BS}_{\max}$ of the best solution (not necessarily optimal) is calculated as follows:

$$\epsilon = \left(C_{\max}/C^{BS}_{\max} - 1\right) \times 100\%. \tag{11}$$

B. Performance on Synthetic Instances

The training process of the proposed DRL method is fairly stable, and converged on all the four training sizes. Here for brevity, the training curve on 10×5 is in Fig. 7, which is plotted base on the average makespan on the 100 validation instances. It can be observed that without human intervention, the DRL agent can indeed learn high-quality scheduling policy from scratch, based on its own solving experiences. Next, the trained policies will be evaluated on the synthetic instances of the same size as in training, and larger sizes unseen in training. Run time analysis will also be provided.

1) Evaluation on instances of training sizes: For each of the training size, Table II reports the average makespan and gap to the OR-Tools solutions on the 100 testing instances drawn from the same distribution as in training. As can be seen, even for small-sized problems, OR-Tools can only solve a small portion of instances optimally within the time limit, showing the complexity of FJSP. For the PDR based methods, the proposed method (in both strategies) consistently outperforms all baseline PDRs in the four training sizes. The average gap of the proposed method to the OR-Tools solutions ranges from 9% to 16% using the greedy strategy, which is similar to the results of [13] and [14] in JSP study. Meanwhile, the sampling strategy can further boost the performance and reduce the gap to be within 11%. To have a more detailed comparison, the proposed method is used as reference to compute the gap of each baseline PDRs, and the boxplots are shown in Fig. 8. It can be observed that the proposed method performs better than baseline PDRs on more than 75% of the instances, except on 15×10 where it manages to surpass MWKR on nearly 75% of the instances. Improvement of the proposed method against baseline PDRs on 20×5 is relatively small than on other sizes. This may be caused by the graph structure of the 20×5 instances. For this distribution, each job tend to have fewer operations and fewer compatible machines, which may affect the effectiveness of HGNN.

[2]https://developers.google.com/optimization

TABLE III: Results on the large-sized synthetic instances

Size		OR-Tools[1]	DRL-G	DRL-S	MOR	SPT	FIFO	MWKR
30×10	$C_{\max}$	294.10 (0%)	313.04	**312.20**	320.18	347.40	328.50	319.89
	Gap	-	6.46%	**6.18%**	8.90%	18.12%	11.74%	8.79%
40×10	$C_{\max}$	397.36 (0%)	416.18	**415.14**	425.19	443.30	427.22	425.70
	Gap	-	4.75%	**4.49%**	7.02%	11.57%	7.53%	7.15%

[1] (.%): percentage of instances solved optimally within 1800 seconds.

TABLE IV: Run time (in seconds) comparison

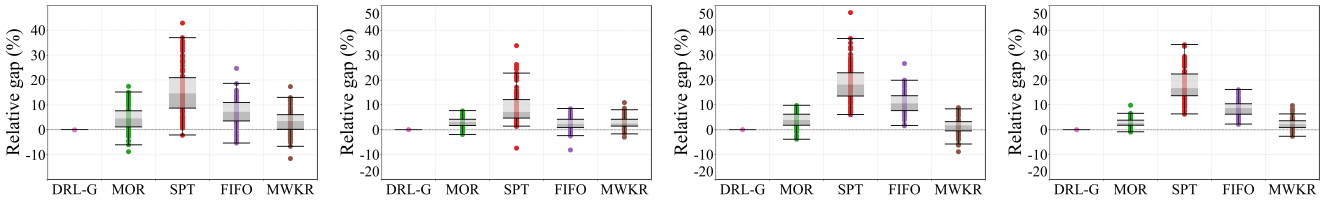
Size	OR-Tools	DRL-G	DRL-S	MOR	SPT	FIFO	MWKR
10×5	1597.88	0.32	0.82	0.15	0.15	0.15	0.16
20×5	1800.00	0.63	1.71	0.29	0.31	0.29	0.31
15×10	1724.00	0.99	2.99	0.42	0.44	0.42	0.46
20×10	1800.00	1.27	4.52	0.58	0.60	0.58	0.61
30×10	1800.00	1.90	8.60	0.87	0.90	0.87	0.93
40×10	1800.00	2.53	15.37	1.17	1.20	1.16	1.25

2) Generalization performance on large-sized instances: This paper further examines the capability of the proposed size-agnostic policy in generalizing to unseen large-sized instances. To this end, the policy trained on 20×10 instances is directly run on 30×10 and 40×10 instances, and the results are summarized in Table III. It can be observed that the advantage of the proposed method still maintains on these large instances, showing that the patterns learned on small and medium-sized instances are still effective in solving large-sized ones.

3) Run time analysis: Table IV lists the average run time of the proposed method and PDR baselines. It can be seen that that the proposed method maintains the high efficiency of PDR based methods, and the run time increases mildly with the increase of problem size. For a given size, the run time of the proposed method is longer than other PDRs. This is because neural network inference is more costly than the simple rules employed by hand-crafted PDRs, which is consistent with other works (e.g. [13], [14]) and is reasonable considering the performance boost. For training, the proposed method takes about 0.36h, 0.69h, 1.18h and 1.64h on the four training sizes 10×5, 20×5, 15×10 and 20×10, respectively. It can be concluded that the training time increases relatively mildly with the increase of problem size. Considering that training is offline, such time efficiency is acceptable.

C. Performance on Public Benchmarks

This subsection further evaluates the generalization performance of the trained policies on the two public benchmarks that are often used in traditional research, by directly running the four policies trained in Section V-B (named DRL $n \times m$) on the benchmark instances. Results are summarized in Table V (gaps are computed with respect to the best solutions in [43]), which also includes results from the recent DRL [34] and GA [26], [27] baselines. In particular, [26] proposes a self-learning GA (SLGA) that uses reinforcement learning to adjust the GA parameters during solving FJSP instances. [27]



(a) 10×05 (b) 20×05 (c) 15×10 (d) 20×10

Fig. 8: Relative gaps of the hand-crafted PDRs to the DRL policy

TABLE V: Results on the public benchmarks

Method	mk			la (rdata)			la (edata)			la (vdata)		
	$C_{\max}$	Gap	Time	$C_{\max}$	Gap	Time	$C_{\max}$	Gap	Time	$C_{\max}$	Gap	Time
OR-Tools	174.90	1.42%	905.45s	938.38	0.28%	1203.89s	1026.70	-0.19%	226.53s	924.40	0.38%	1367.77s
SLGA [26]	181.30	6.21%	283.28s	-	-	-	-	-	-	-	-	-
RegGA [27]	183.00	8.39%	280.10s	-	-	-	-	-	-	836.13[1]	3.20%	191.40s
2SGA [27]	175.20	3.17%	57.60s	-	-	-	-	-	-	812.20[1]	0.39%	51.43s
DRL[2]10×5	201.00	27.83%	0.91s	1030.83	11.15%	1.03s	1187.48	15.53%	1.00s	955.90	4.25%	1.00s
	190.40[3]	**19.02%**	3.11s	**986.10**	**5.75%**	3.56s	**1116.33**	**8.18%**	3.58s	932.00	1.44%	3.47s
DRL 20×5	215.60	39.00%	0.92s	1080.30	17.19%	0.99s	1198.55	16.62%	0.98s	967.78	6.19%	0.98s
	197.40	24.68%	3.10s	1016.90	9.03%	3.54s	1135.38	9.91%	3.61s	943.53	2.79%	3.51s
DRL 15×10	197.70	25.39%	0.90s	1031.33	11.14%	0.97s	1182.08	15.00%	1.04s	954.33	4.02%	0.96s
	191.30	19.32%	3.21s	987.08	5.81%	3.50s	1118.35	8.57%	3.56s	**931.45**	**1.36%**	3.54s
DRL 20×10	200.30	30.05%	0.90s	1044.03	12.85%	0.99s	1204.45	17.24%	1.00s	950.75	3.96%	0.99s
	193.50	21.12%	3.13s	1004.33	7.50%	3.55s	1135.78	9.93%	3.60s	934.83	1.77%	3.46s
Han [34]	216.90	34.04%	-	-	-	-	-	-	-	-	-	-
MOR	202.31	29.59%	0.40s	1064.04	14.71%	0.45s	1211.23	17.91%	0.45s	969.25	6.09%	0.45s
SPT	238.80	45.73%	0.41s	1184.80	27.70%	0.46s	1305.35	26.19%	0.46s	1085.46	18.20%	0.46s
FIFO	206.09	30.72%	0.39s	1082.08	16.63%	0.44s	1255.46	22.07%	0.45s	980.69	7.50%	0.44s
MWKR	200.17	28.19%	0.45s	1046.20	12.53%	0.51s	1179.90	14.92%	0.50s	962.01	5.11%	0.50s

[1] The gap of RegGA and 2SGA are computed on 30 out of 40 la vdata instances (la01-la30), on which [27] reported results.
[2] For the two rows of each DRL policy, the above and below row is result of the greedy and sampling strategy, respectively.
[3] **Bold** means the best performance among the PDR based methods.

proposes a two-stage GA (2SGA) for FJSP, which improves the Regular GA (RegGA) by a first stage that generates high-quality initial population. Results of both 2SGA and RegGA in [27] are included here.

The upper part of Table V shows that OR-Tools obtains very good solutions, though it only solves half instances optimally. The GA methods also find high-quality solutions. However, run time of these search-based methods is much longer than the PDR based ones in the lower part of Table V. It can be seen that the proposed method with greedy strategy generally performs better than other baseline PDRs on both benchmarks, and the sampling strategy further reduces the gaps. This shows that the learned policies generalize well to these out-of-distribution instances. The four learned policies have almost the same run time, since they have the same neural structure with only different parameters. Compared to the recent method [34] which reported results on the mk benchmark, the proposed method significantly outperforms it by a large margin (except the policy trained on 20×5 instances), showing the advantage of the HGNN network in extracting rich state information for better decision making. An interesting observation is that among the four of our policies, the one trained on the smallest size 10×5 performs the best in general. With sampling, it even achieves the optimal solutions on two mk instances (mk03 and mk08), and finds solutions within 5% gap to the best solutions in [43] for more than 50% (69 out of 120) of the la instances. This might be because for the simpler learning task, the training of DRL agent could be more sufficient so that it can discover better patterns. We will investigate this in the future. As another interesting direction, the fast and high-quality solving performance of the proposed method provides additional possibilities of being integrated with the search-based methods, for example generating initial populations in 2SGA.

VI. Conclusions and Future Work

Solving flexible scheduling problems efficiently is of great importance to the next generation manufacturing paradigms such as cloud manufacturing. This paper proposes a novel end-to-end DRL method to learn high-quality PDRs for FJSP, which is widely used in practise but rarely studied by existing DRL based methods. The underlying MDP is formulated using an integrated approach, which combines operation selection and machine assignment as one decision. Then a heterogeneous graph structure is proposed to represent scheduling states, which is processed by a novel HGNN architecture so

as to convert the numerical and topological information in the graph into feature embeddings. Based on the HGNN, an actor-critic architecture is designed and trained with PPO. Results show that the proposed method outperforms baseline PDRs with reasonable efficiency, and generalizes well to unseen instances of larger sizes and from public benchmarks. For future work, the method will be extended to handle more challenging factors in practical production, such as batching, due dates, and uncertainties. In addition, the multi-optima property [46] of FJSP (i.e., an instance could have multiple optimal solutions) will be exploited to enhance the training performance. The possibilities of combining with advanced search mechanisms such as GA will also be investigated.

APPENDIX
DEFINITION OF RAW FEATURE VECTORS

For each state S_t, the raw features of each operation, machine and O-M arcs are defined as follows (dependence on t is ignored for clearance):

Raw features of operation nodes. For each $O_{ij} \in \mathcal{O} \setminus \{Start, End\}$, raw feature vector $\mu_{ij} \in \mathbb{R}^6$ has 6 elements:

- Status: a binary value indicates whether O_{ij} has been scheduled (1) or not (0) till step t.
- Number of neighboring machines: $|\mathcal{N}_t(O_{ij})|$.
- Processing time: p_{ijk} if O_{ij} is scheduled, otherwise $\bar{p}_{ij}$.
- Start time: the estimated or actual start time of O_{ij} in the corresponding partial schedule $S(t)$.
- Number of unscheduled operations in the job: the number of operations in $\mathcal{O}_i$ that have not been scheduled.
- Job completion time: C_i in the partial schedule $S(t)$.

Note that for the two dummay operations $Start$ and End, zero vectors are used when necessary.

Raw features of machine nodes. For each $M_k \in \mathcal{M}$, 3 features are used to form its raw feature vector $\nu_k \in \mathbb{R}^3$:

- Available time: the time when M_k completes all its assigned operations and can process new operations.
- Number of neighboring operations: $|\mathcal{N}_t(M_k)|$.
- Utilization: ratio of the non-idle time to the total production time of M_k till $T(t)$, and is within the range [0,1].

Raw feature of O-M arcs. For each $E_{ijk} \in \mathcal{E}$, its raw feature vector $\lambda_{ijk} \in \mathbb{R}$ contains only one element, i.e. the corresponding processing time p_{ijk}.

REFERENCES

[1] L. Thames and D. Schaefer, "Software-defined cloud manufacturing for industry 4.0," *Procedia CIRP*, vol. 52, pp. 12–17, 2016.

[2] H. Golpîra, S. A. R. Khan, and S. Safaeipour, "A review of logistics internet-of-things: Current trends and scope for future research," *Journal of Industrial Information Integration*, vol. 22, p. 100194, 2021.

[3] Y. Liu, L. Wang, X. V. Wang, X. Xu, and L. Zhang, "Scheduling in cloud manufacturing: state-of-the-art and research challenges," *International Journal of Production Research*, vol. 57, no. 15-16, pp. 4854–4879, 2019.

[4] Y. Fang, C. Peng, P. Lou, Z. Zhou, J. Hu, and J. Yan, "Digital-twin-based job shop scheduling toward smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 12, pp. 6425–6435, 2019.

[5] Y. Zhang, J. Wang, S. Liu, and C. Qian, "Game theory based real-time shop floor scheduling strategy and method for cloud manufacturing," *International Journal of Intelligent Systems*, vol. 32, no. 4, pp. 437–463, 2017.

[6] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 904–916, 2019.

[7] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, "Review on flexible job shop scheduling," *IET Collaborative Intelligent Manufacturing*, vol. 1, no. 3, pp. 67–77, 2019.

[8] 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," *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1809–1830, 2019.

[9] B. Chen and T. I. Matis, "A flexible dispatching rule for minimizing tardiness in job shop scheduling," *International Journal of Production Economics*, vol. 141, no. 1, pp. 360–365, 2013.

[10] 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 Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4276–4284, 2019.

[11] 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. 71 752–71 762, 2020.

[12] L. Wang, X. Hu, Y. Wang, S. Xu, S. Ma, K. Yang, Z. Liu, and W. Wang, "Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning," *Computer Networks*, vol. 190, 2021.

[13] 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," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[14] J. Park, J. Chun, S. Kim, Y. Kim, and J. Park, "Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning," *International Journal of Production Research*, vol. 59, no. 11, pp. 3360–3377, 2021.

[15] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," in *Advances in Neural Information Processing Systems*, 2021.

[16] Z. Wang, C. Liu, and M. Gombolay, "Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints," *Autonomous Robots*, 2021.

[17] L. Xin, W. Song, Z. Cao, and J. Zhang, "Step-wise deep learning models for solving routing problems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4861–4871, 2020.

[18] F. Ni, J. Hao, J. Lu, X. Tong, M. Yuan, J. Duan, Y. Ma, and K. He, "A multi-graph attributed reinforcement learning based optimization algorithm for large-scale hybrid flow shop scheduling problem," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 3441–3451.

[19] C. Özgüven, L. Özbakır, and Y. Yavuz, "Mathematical models for job-shop scheduling problems with routing and process plan flexibility," *Applied Mathematical Modelling*, vol. 34, no. 6, pp. 1539–1548, 2010.

[20] D. Müller, M. G. Müller, D. Kress, and E. Pesch, "An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning," *European Journal of Operational Research*, 2022.

[21] M. A. Ortíz, L. E. Betancourt, K. P. Negrete, F. De Felice, and A. Petrillo, "Dispatching algorithm for production programming of flexible job-shop systems in the smart factory industry," *Annals of Operations Research*, vol. 264, no. 1, pp. 409–433, 2018.

[22] B. Huang, Y. Sun, and Y. Sun, "Scheduling of flexible manufacturing systems based on petri nets and hybrid heuristic search," *International Journal of Production Research*, vol. 46, no. 16, pp. 4553–4565, 2008.

[23] O. Sobeyko and L. Mönch, "Heuristic approaches for scheduling jobs in large-scale flexible job shops," *Computers & Operations Research*, vol. 68, pp. 97–109, 2016.

[24] V. Kaplanoğlu, "An object-oriented approach for multi-objective flexible job-shop scheduling problem," *Expert Systems with Applications*, vol. 45, pp. 71–84, 2016.

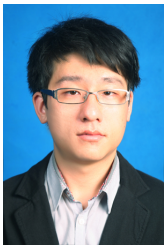
[25] S. Jia and Z.-H. Hu, "Path-relinking tabu search for the multi-objective flexible job shop scheduling problem," *Computers & Operations Research*, vol. 47, pp. 11–26, 2014.

[26] R. Chen, B. Yang, S. Li, and S. Wang, "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 149, p. 106778, 2020.

[27] D. Rooyani and F. M. Defersha, "An efficient two-stage genetic algorithm for flexible job-shop scheduling," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 2519–2524, 2019.

[28] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Deep reinforcement learning for semiconductor production scheduling," in *2018 29th Annual SEMI Advanced*

Semiconductor Manufacturing Conference (ASMC). IEEE, 2018, pp. 301–306.
[29] D. Shi, W. Fan, Y. Xiao, T. Lin, and C. Xing, "Intelligent scheduling of discrete automated production line via deep reinforcement learning," *International Journal of Production Research*, vol. 58, no. 11, pp. 3362–3380, 2020.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.
[31] Z. Pan, L. Wang, J. Wang, and J. Lu, "Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.
[32] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.
[33] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Computers & Industrial Engineering*, vol. 159, p. 107489, 2021.
[34] B. Han and J. Yang, "A deep reinforcement learning based solution for flexible job shop scheduling problem," *International Journal of Simulation Modelling*, vol. 20, no. 2, pp. 375–386, 2021.
[35] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, no. 3, pp. 157–183, 1993.
[36] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in Neural Information Processing Systems*, vol. 32, pp. 11 983–11 993, 2019.
[37] X. Fu, J. Zhang, Z. Meng, and I. King, "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding," in *Proceedings of The Web Conference 2020*, 2020, pp. 2331–2341.
[38] D. Jin, C. Huo, C. Liang, and L. Yang, "Heterogeneous graph neural network via attribute completion," in *Proceedings of the Web Conference 2021*, 2021, pp. 391–400.
[39] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.
[40] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations (ICLR)*, 2019.
[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.
[42] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *OR Spektrum*, vol. 15, no. 4, pp. 205–215, 1994.
[43] D. Behnke and M. J. Geiger, "Test instances for the flexible job shop scheduling problem with work centers," Tech. Rep., 2012.
[44] M. Montazeri and L. Van Wassenhove, "Analysis of scheduling rules for an fms," *The International Journal of Production Research*, vol. 28, no. 4, pp. 785–802, 1990.
[45] G. Da Col and E. C. Teppan, "Industrial size job shop scheduling tackled by present day cp solvers," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2019, pp. 144–160.
[46] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 188–21 198, 2020.



Wen Song received the B.S. degree in automation and the M.S. degree in control science and engineering from Shandong University, Jinan, China, in 2011 and 2014, respectively, and the Ph.D. degree in computer science from the Nanyang Technological University, Singapore, in 2018. He was a Research Fellow with the Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU). He is currently an Associate Research Fellow with the Institute of Marine Science and Technology, Shandong University. His current research interests include artificial intelligence, deep reinforcement learning, planning and scheduling, and operations research.

Xinyang Chen received the B.S. degree in automation from Shandong University, Jinan, China, in 2019 and the M.S. degree in control science and engineering from the same university in 2022. His main research interests include deep reinforcement learning, cyber-physical system, and production scheduling.

Qiqiang Li received the Ph.D. degree in industrial automation from the Institute of Industrial Process Control, Zhejiang University in 1998. He is a Professor with the School of Control Science and Engineering and the Institute of Marine Science and Technology, Shandong University. His research area focuses on modeling, optimization, and simulation of complex systems. His current research interests are concerned with economic operation optimization of photovoltaic systems, energy efficiency of process industry and commercial buildings.

Zhiguang Cao received the B.Eng. degree in automation from the Guangdong University of Technology, Guangzhou, China, in 2009, the M.Sc. degree in signal processing from Nanyang Technological University (NTU), Singapore, in 2012, and the Ph.D. degree from the Interdisciplinary Graduate School, NTU, in 2017. He was a Research Assistant Professor with the Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore, and a Research Fellow with the Future Mobility Research Lab, NTU. He is currently a Scientist with the Singapore Institute of Manufacturing Technology (SIMTech), Agency for Science Technology and Research (A*STAR), Singapore. His research interests currently focus on neural combinatorial optimization.