Graph-based Reinforcement Learning for Flexible Job Shop Scheduling with Transportation Constraints

Sihoon Moon, Sanghoon Lee, and Kyung-Joon Park[†], *Senior Member, IEEE*
Department of Electrical Engineering and Computer Science
DGIST
Daegu 42988, South Korea
e-mail: {msh0576, leesh2913, kjp}@dgist.ac.kr

IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society | 979-8-3503-3182-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/IECON51785.2023.10312647

***Abstract*—Recently, deep reinforcement learning (DRL) has been employed in flexible job-shop scheduling problems (FJSP) to minimize makespan within flexible manufacturing systems (FMS). In practice, numerous modern enterprises are incorporating automated guided vehicles (AGV) into their FMS implementations. However, existing DRL-based FJSP solutions do not account for transportation constraints. To tackle this practical issue, we propose a novel graph-based DRL method, called Heterogeneous Job Scheduler (HJS), which interprets the environment status using the graph structure and then training the DRL model based on graph embeddings. Our findings indicate that the proposed approach surpasses conventional dispatching rules and existing DRL-based methods in terms of makespan, running time, and generalization performance.**

***Index Terms*—Flexible job-shop scheduling with transportation constraints, Reinforcement learning, Smart manufacturing systems**

I. Introduction

Flexible manufacturing systems (FMS) play an essential role in the modern manufacturing industry for personalized production [1]–[4]. Recently, in practical manufacturing systems, numerous enterprises have adopted transportation resources, such as automated guided vehicles (AGV), to enhance flexibility and diversity in FMS [5]. However, incorporating transportation resources into FMS introduces a significant challenge due to the increased complexity of production scheduling problems. These problems not only require the consideration of operation allocation to compatible machines but also AGV allocation for transporting intermediate products.

Such FMS with transportation resources can be mathematically formulated as a flexible job-shop scheduling problem with transportation constraints (FJSP-T) [6]. Numerous methods have been studied to resolve FJSP-T, including meta-heuristics such as genetic algorithms (GA) and ant colony optimization (ACO) [6]–[8]. Despite meta-heuristics can find relatively near-optimal solutions at the cost of computation time efficiency compared with the exact solvers (e.g. constraint programming), they still suffer from long computation time in the large-scale problem, due to the complicated search procedures to explore the solution space to find high-quality solution [9]. Additionally, their scheduling quality is still far from the optimality. To overcome these limitations, deep reinforcement learning (DRL)-based scheduler is a promising approach that can find near-optimal solutions with low computation time [9], [10].

Most existing DRL methods for manufacturing process optimization attempt to solve FJSP ignoring transportation constraints [9], [11]. However, the learning mechanism design taking transportation into account brings more challenging: (i) complex one-to-many relation between operations, machines and vehicles makes it difficult to represent schedule status of FJSP-T. Decision-making in DRL based simply on raw feature states of FJSP-T without considering the relation induces limited learning performance on large-scale problems [12]. (ii) decisions in FJSP-T is more complex as they require not only operation and machine selection but also vehicle selection. Consequently, novel decision-making framework and informative representation techniques for operation-machine-vehicle pair are necessary.

To address these challenges, we propose an end-to-end DRL module for solving FJSP-T, called Heterogeneous Job Scheduler (HJS) [1], which is a graph neural network (GNN)-based RL algorithm. For challenge (i), we design a novel heterogeneous graph structure capable of expressing FJSP-T. To extract rich information for operation-machine-vehicle relation, we develop a GNN encoder based on the heterogeneous graph structure. The proposed encoder uses an attention mechanism that learns the importance between heterogeneous graph nodes. For challenge (ii), we propose a three-stage decoder model that schedules operation-machine-vehicle pairs. It can learn to generate near-optimal schedule from the graph embedding for minimizing the makespan.

To summarize, our contributions are as follows:

This work was supported by the DGIST R&D Program of the Ministry of Science and ICT (23-DPIC-16).

†K.-J. Park is the corresponding author.

[1]The source code is available on : https://github.com/msh0576/FJSPT-Scheduler

- We design a novel heterogeneous disjunctive graph for FJSP-T. This graph represents features of operations, machines, vehicles, and their relation with low graph density.
- We develop an encoder based on GNN to represent the proposed heterogeneous graph. The proposed encoder enables message passing between different types of nodes, such as operations, machines, and vehicles, by allowing each node to aggregate its neighboring nodes' messages.
- By combining the encoder, and RL framework, we design HJS module, which has size-agnostic properties. Thus the trained module can work on not only the trained instances but also instances of varying graph size. The proposed method outperform traditional dispatching rules and existing DRL-based methods. Especially, it can generalize to unseen large-scale problems.

II. Problem Description and Notations

Flexible job-shop scheduling problem with transportation constraints (FJSP-T) can be defined as follows. There exists a set of n jobs $\mathcal{J} = \{J_1, ..., J_n\}$, a set of m machines $\mathcal{M} = \{M_1, ..., M_m\}$ and a set of v vehicles $\mathcal{V} = \{V_1, ..., V_v\}$. Each job J_i consists of n_i consecutive operations $\mathcal{O}_i = \{O_{i1}, ..., O_{in_i}\}$ with precedence constraints. An operation of job J_i denoted by O_{ij} can be processed on a subset of eligible machines $\mathcal{M}_{ij} \subset \mathcal{M}$. This means that operation O_{ij} takes different processing time T^p_{ijk} for each machine $M_k \in \mathcal{M}_{ij}$. To allocate operation O_{ij} on machine M_k, vehicle V_u transports the intermediate products of O_{ij} to the machine. Transportation time T^t_{ijuk} of V_u is a sum of off-load transportation time T^t_{iju} that V_u being off-load status approaches to the product location of O_{ij} to load it and on-load transportation time T^t_{uk} that V_u being on-load status transports the product to machine M_k. In FJSP-T, the optimization goal is to assign operations to a compatible machine transporting with vehicles and determine the sequence of operations on a machine/vehicle for minimizing the makespan,

$$C_{max} = \max C_{in_i}, \forall i \in \{1, ..., n\}, \tag{1}$$

where C_{in_i} is a completion time of final operation O_{in_i} of job J_i. Overall notations are presented in Table I.

Constants	
m	total number of machines
n	total number of jobs
n_i	total number of operations of job i
v	total number of vehicles
d_e	dimension of embedding
d_k	dimension of query, key and value
Indexes	
i	job index, $i = 1, ..., n$
j	operation index of $J_i, j = 1, ..., n_i$
k	machine index, $k = 1, ..., m$
u	vehicle index, $u = 1, ..., v$
Sets	
$\mathcal{J}$	job node set, $\mathcal{J} = \{J_1, ..., J_n\}$
$\mathcal{O}_i$	operation node set for job J_i, $\mathcal{O}_i = \{O_{i1}, ..., O_{in_i}\}$
$\mathcal{M}$	machine node set, $\mathcal{M} = \{M_1, ..., M_m\}$
$\mathcal{M}_{ij}$	available machine node set for operation O_{ij}, $\mathcal{M}_{ij} \subseteq \mathcal{M}$
$\mathcal{V}$	vehicle node set, $\mathcal{V} = \{V_1, ..., V_v\}$
$\mathcal{N}_m(O_{ij})$	neighboring machine nodes for O_{ij}
$\mathcal{N}_v(O_{ij})$	neighboring vehicle nodes for O_{ij}
$\mathcal{N}(M_k)$	neighboring operation nodes for M_k
$\mathcal{N}(V_u)$	neighboring operation nodes for V_u
Variables	
T^p_{ijk}	processing time of operation O_{ij} on machine M_k
T^t_u	transportation time of vehicle V_u
T^t_{ijuk}	time that V_u transports products corresponding to O_{ij} to M_k
T^s_{ij}	start time of O_{ij}
C_i	completion time of job J_i

TABLE I
NOTATION

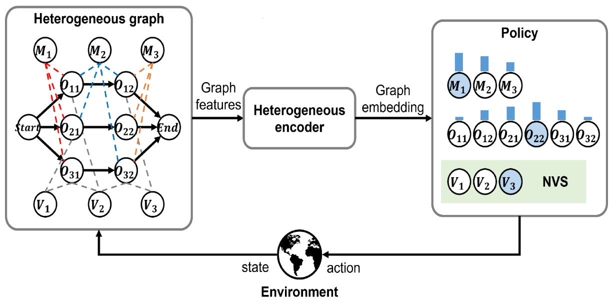


Fig. 1. Workflow of the heterogeneous job scheduler.

III. Heterogeneous Job Scheduler (HJS)

To resolve FJSP-T, we propose HJS module consisting of three main components: a heterogeneous graph, encoder and policy. The workflow of HJS, as illustrated in Fig. 1, consists of the following steps: 1) receiving raw feature states from the manufacturing process, 2) constructing the heterogeneous graph based on the features, 3) encoding the graph through the encoder, and 4) making action decisions using the policy. We first design the heterogeneous graph for FJSP-T that can express relation of operations, machines and vehicles with low graph density. The proposed encoder represents the heterogeneous graph with three different attributes of node sets, operation $\mathcal{O}$, machine $\mathcal{M}$, and vehicle $\mathcal{V}$. The policy generates a composite action of the operation-machine-vehicle (O-M-V) pair based on the graph representation. Finally, based on end-to-end RL algorithm, we train the HJS module to minimize makespan.

A. Heterogeneous Graph for FJSP-T

A disjunctive graph for FJSP [13] is defined as $\mathcal{G} = (\mathcal{O}, \mathcal{C}, \mathcal{D})$. $\mathcal{O}$ is an operation node set, and $\mathcal{C}$ is a set of conjunctive arcs. There are n directed arc flows, each representing the processing sequence of job J_i. $\mathcal{D} = \cup_k \mathcal{D}_k$ is a set of (undirected) disjunctive arcs, $\mathcal{D}_k$ is a clique that connects operations capable of processing on machine M_k. Since an operation can be processed on multiple machines, it can be connected to multiple disjunctive arcs.

By modifying the disjunctive graph, we propose a novel heterogeneous graph to represent FJSP-T. The graph is defined as $\mathcal{H} = (\mathcal{O} \cup \mathcal{M} \cup \mathcal{V}, \mathcal{C}, \mathcal{E}_m \cup \mathcal{E}_v)$. Compared with the traditional graph, machine node set $\mathcal{M}$, vehicle node set $\mathcal{V}$, compatible machine arc set $\mathcal{E}_m$ and compatible vehicle arc set $\mathcal{E}_v$ are added. Machine node $M_k \in \mathcal{M}$ and vehicle node $V_u \in \mathcal{V}$ represent machine and vehicle features, respectively. The disjunctive arc set $\mathcal{D}$ is replaced with $\mathcal{E}_m \cup \mathcal{E}_v$. Element of the compatible machine arc set $E^m_{ijk} \in \mathcal{E}_m$ denotes processing time that operation O_{ij} is processed on compatible machine $M_k \in \mathcal{M}_{ij}$. Vehicle arc $E^v_{ijuk} \in \mathcal{E}_v$ represents transportation time that vehicle V_u transports products corresponding to O_{ij} to machine M_k. The vehicle arc $E^v_{ijuk} = \{E^v_{iju}, E^v_{ijk}\}$ consists of off-load transportation time E^v_{iju} and on-load transportation time E^v_{ijk}, because the transportation time is a sum of the off-load and on-load transportation time.

B. Markov Decision Process

We define a Markove decision process (MDP) model for FJSP-T. At each time step t, the agent observes the system state s_t, and makes a decision a_t. The action a_t allows an unscheduled operation to be processed on an idle machine by transporting it with an idle vehicle. After taking the action, the environment transits to the next state s_{t+1}, and receives reward r_{t+1}. This process repeats until all the operations are scheduled. The MDP model is specifically defined below.

1) State: At decision step t, state s_t is a heterogeneous graph $\mathcal{H}_t(\mathcal{O} \cup \mathcal{M} \cup \mathcal{V}, \mathcal{C}, \mathcal{E}_{mt} \cup \mathcal{E}_{vt})$. In this graph, we define raw features of nodes and edges. Raw feature vector $\mu_{ij} \in \mathbb{R}^7$ of operation O_{ij} consists of 7 elements; status, number of neighboring machines, number of neighboring vehicles, processing time, number of unscheduled operations in job J_i, job completion time, and start time. Raw feature vector $\mu_k \in \mathbb{R}^4$ of machine M_k consists of 4 elements; status, number of neighboring operations, available time, and utilization. Raw feature vector $\mu_v \in \mathbb{R}^4$ of vehicle V_u consists of 4 elements; status, number of neighboring operations, available time, and current location. Raw feature vector $\nu^m_{ijk} \in \mathbb{R}$ of compatible machine arc $E^m_{ijk} \in \mathcal{E}_m$ and vector $\nu^v_{ijuk} \in \mathbb{R}$ of compatible vehicle arc $E^v_{ijuk} \in \mathcal{E}_v$ contain one element, processing time T^p_{ijk} for O_{ij}-M_k pair and transportation time T^t_{ijuk} for O_{ij}-M_k-V_u pair, respectively. We refer to the raw feature vectors of nodes and edges in [9]

2) Action: To resolve FJSP-T, we define a composite action, $a_t = (O_{ij}, M_k, V_u)$, which constitutes operation selection, machine assignment and vehicle usage. This means the operation O_{ij} is allocated to the machine M_k and V_u transports it. Specifically, action $a_t \in A_t$ is to select a feasible operation-machine-vehicle pair. Feasible operation O_{ij} implies its immediate predecessor $O_{i(j-1)}$ is completed. A feasible machine is an idle one among the compatible machines, $M_k \in \mathcal{M}_{ij}$. A feasible vehicle is an idle one among all vehicles at time t, $V_u \in \mathcal{V}_t$.

3) State Transition: Once the action is taken, the environment deterministically transits to the next state s_{t+1}. We first define the feasible next decision step $t+1$. This step is determined by operation events, that is, the earliest release time of the new operation among the remaining feasible operations after time t. At step $t+1$, the graph structure and node features change by the action $a_t = (O_{ij}, M_k, V_u)$, such as node O_{ij} has only one O-M-V pair in the graph, and other compatible pairs are removed, and specific features of nodes change as described in Section III-B1.

4) Reward: The goal is to learn to schedule operations such that the makespan is minimized. We design the reward function as the difference between the makespan corresponding to s_t and s_{t+1}, $r(s_t, a_t, s_{t+1}) = C_{max}(s_t) - C_{max}(s_{t+1})$. Here, we define $C_{LB}(O, s_t)$ as the lower bound of the estimated completion time of operation O at state s_t. We compute this lower bound recursively, $C_{LB}(O_{ij}, s_t) = C_{LB}(O_{i(j-1)}, s_t) + \bar{T}^p_{ij}$, where O_{ij} is an unscheduled operation in job J_i. If $O_{i(j-1)}$ is scheduled in s_t, $C_{LB}(O_{i(j-1)}, s_t)$ is replaced with the actual completion time $C_{i(j-1)}$. We define the makespan in s_t as the maximum lower bound of unscheduled operations for all jobs, $C_{max}(s_t) = \max_{i,j}\{C_{LB}(O_{ij}, s_t)\}$. The makespan in the terminal state $s_{|O|}$ has the actual process makespan, $C_{max}(s_{|O|}) = C_{max}$, since all operations are scheduled. When the discount factor $\gamma = 1$, the cumulative reward is $G = \sum_{t=0}^{|O|} r(s_t, a_t, s_{t+1}) = C_{max}(s_0) - C_{max}$, where $C_{max}(s_0)$ is a constant for a specific instance. Therefore, maximizing G is equivalent to minimizing the makespan.

C. Heterogeneous Encoder

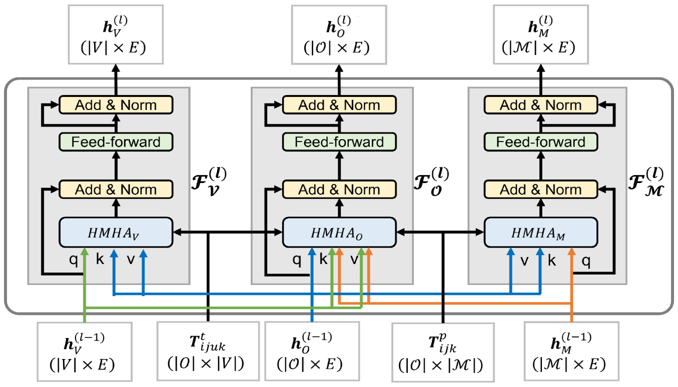
The proposed encoder, shown in Fig. 2, theoretically follows Graph attention networks (GANs) [14], which is constructed by stacking multiple (L) graph attention layers. Let $h^{(l)}_{ij}$, $h^{(l)}_k$ and $h^{(l)}_u$ be an operation machine and vehicle node embedding vector, respectively, through layer $l \in \{1, ..., L\}$. Function $\mathcal{F}^{(l)}_X$ updates embedding vector $h^{(l)}_x$ of node $x \in X$ by aggregating knowledge of the self node, its neighboring nodes, and their relation as follows:

$$\begin{aligned} h^{(l)}_{ij} &= \mathcal{F}^{(l)}_O(h^{(l-1)}_{ij}, \{h^{(l-1)}_k, \nu_{ijk} | M_k \in \mathcal{N}_m(O_{ij})\}, \\ &\qquad \{h^{(l-1)}_u, \nu_{iju} | V_u \in \mathcal{N}_v(O_{ij})\}) \\ h^{(l)}_k &= \mathcal{F}^{(l)}_M(h^{(l-1)}_k, \{h^{(l-1)}_{ij}, \nu_{ijk} | O_{ij} \in \mathcal{N}(M_k)\}) \\ h^{(l)}_u &= \mathcal{F}^{(l)}_V(h^{(l-1)}_u, \{h^{(l-1)}_{ij}, \nu_{iju} | O_{ij} \in \mathcal{N}(V_u)\}) \end{aligned} \tag{2}$$

where $h^{(l-1)}$ is a node embedding at layer $l-1$. The learnable update function $\mathcal{F}^{(l)}_X$ is composed of heterogeneous multi-head attention (HMHA), add & normalization (AN), and feed-forward layer (FF). The encoder has three independent update functions to deal with three different classes of nodes; $\mathcal{F}^{(l)}_O$ for operation node, $\mathcal{F}^{(l)}_M$ for machine node, and $\mathcal{F}^{(l)}_V$ for vehicle node.

1) Heterogeneous Multi-Head Attention Block: We design HMHA block that nodes embed messages of different classes of nodes and their relation, using an attention model (AM) [15]. AM learns attention scores between nodes of how much they relate to each other. For example, an operation and the corresponding compatible machine with a low processing time may have a high attention score because this O-M pair contributes to the low makespan.

For operation node embedding $\mathbf{h}^{(l)}_O$, HMHA$_O$ block for node O_{ij} embeds messages of neighboring machine nodes $M_k \in \mathcal{M}_{ij}$ and vehicle nodes $V_u \in \mathcal{V}$ into an operation node O_{ij}. Unlike machine and vehicle nodes, only the operation node has two different classes of neighboring nodes, $\mathcal{M}_{ij}$ and $\mathcal{V}$. Thus, we construct two HMHA blocks that independently aggregate



$h_V^{(l)}$ $(|\mathcal{V}| \times E)$
$h_O^{(l)}$ $(|\mathcal{O}| \times E)$
$h_M^{(l)}$ $(|\mathcal{M}| \times E)$
Add & Norm
Feed-forward
Add & Norm
$HMHA_V$
$HMHA_O$
$HMHA_M$
q k v
$\mathcal{F}_{\mathcal{V}}^{(l)}$
$\mathcal{F}_{\mathcal{O}}^{(l)}$
$\mathcal{F}_{\mathcal{M}}^{(l)}$
$h_V^{(l-1)}$ $(|\mathcal{V}| \times E)$
T_{ijuk}^t $(|\mathcal{O}| \times |\mathcal{V}|)$
$h_O^{(l-1)}$ $(|\mathcal{O}| \times E)$
T_{ijk}^p $(|\mathcal{O}| \times |\mathcal{M}|)$
$h_M^{(l-1)}$ $(|\mathcal{M}| \times E)$

Fig. 2. Heterogeneous encoder architecture.

messages for each neighboring node, HMHA_O^M on machine nodes and HMHA_O^V on vehicle nodes.

For HMHA_O^M of message passing from $M_k \in \mathcal{M}_{ij}$ to O_{ij}, it requires query q_{ij} for operation node O_{ij}, and key k_k/value v_k for machine node M_k, respectively:

$$\text{q}_O = W^{\text{q}_O}\mathbf{h}_O^{(l-1)}, \text{k}_M = W^{\text{k}_M}\mathbf{h}_M^{(l-1)}, \text{v}_M = W^{\text{v}_M}\mathbf{h}_M^{(l-1)}, \tag{3}$$

where W^{q_O}, W^{k_M}, and W^{v_M} are trainable parameter matrices, their dimension is described in Section VI. $\mathbf{h}_O^{(l-1)} \in \mathbb{R}^{|\mathcal{O}| \times d_e}$, and $\mathbf{h}_M^{(l-1)} \in \mathbb{R}^{|\mathcal{M}| \times d_e}$ are operation, and machine embedding matrices at layer $l-1$. Following the attention mechanism [15], we can calculate *compatibility* σ_{OM} by equation (12), *attention weight* $\bar{\sigma}_{OM}$ by equation (13), and *single-head operation node embedding matrix* $\mathbf{h}_{O,z}^{M,(l)} \in \mathbb{R}^{|\mathcal{O}| \times d_v}$ by equation (14). Finally, by equation (15), HMHA_O^M adds up the heads $\mathbf{h}_{O,z}^{M,(l)}$, Z times in parallel, and outputs *multi-head operation node embedding matrix* $\mathbf{h}_O^{M,(l)}$:

$$\begin{aligned}\mathbf{h}_O^{M,(l)} =& \text{HMHA}_O^M(\mathbf{h}_M^{(l-1)}) \\ =& \sum_{z \in Z} W_{O,z}^M \mathbf{h}_{O,z}^{M,(l)},\end{aligned} \tag{4}$$

where $W_{O,z}^M \in \mathbb{R}^{d_v \times d_e}$ is a trainable parameter matrix.

As a difference from the existing attention mechanism that aggregates only node messages, we need to include their relation (e.g., processing time) in the embedding $\mathbf{h}_O^{M,(l)}$. We propose an *augmented compatibility* $\tilde{\sigma}_{OM}$, which uses two-step linear transformation on the concatenation of compatibility σ_{OM} and processing time matrix $\mathbf{T}^p \in \mathbb{R}^{|\mathcal{O}| \times |\mathcal{M}|}$:

$$\tilde{\sigma}_{OM} = \text{ReLU}(W^2(\text{ReLU}(W^1[\sigma_{OM}||\mathbf{T}^p]))), \tag{5}$$

where $W^1 \in \mathbb{R}^{2 \times d_e}$ $W^2 \in \mathbb{R}^{d_e \times 1}$ are trainable parameter matrices, $[\cdot||\cdot]$ denotes a concatenation function, and ReLU is an activation function. Replacing compatibility σ_{OM} with $\tilde{\sigma}_{OM}$ and following the multi-head attention computation procedure of equation (4), we can obtain multi-head operation node embedding that includes processing time knowledge, $\mathbf{h}_O^{M,(l)} = \text{HMHA}_O^M(\mathbf{h}_M^{(l-1)}, \mathbf{T}^p)$.

For HMHA_O^V of message passing from $V_u \in \mathcal{V}$ to O_{ij}, we can compute multi-head operation node embedding matrix for vehicle nodes with transportation matrix $\mathbf{T}^t$, as the same computation procedure of $\mathbf{h}_O^{M,(l)}$, $\mathbf{h}_O^{V,(l)} = \text{HMHA}_O^V(\mathbf{h}_V^{(l-1)}, \mathbf{T}^t)$. In this computation, the used query, key, and value are defined as follows,

$$\text{q}_O = W^{\text{q}_O}\mathbf{h}_O^{(l-1)}, \text{k}_V = W^{\text{k}_V}\mathbf{h}_V^{(l-1)}, \text{v}_V = W^{\text{v}_V}\mathbf{h}_V^{(l-1)}. \tag{6}$$

However, we cannot directly use $\mathbf{T}^t$ as the input of HMHA_O^V block, because since transportation time matrix $\mathbf{T}^t$ includes O-V arcs (off-load transportation time) as well as O-M arcs (on-load transportation time) in the heterogeneous graph, but the block can only perform message-passing of O-V arcs. Therefore, we use an *on-off-separated node embedding approach*. HMHA_O^V block generates operation node embedding $\mathbf{h}_O^{V,(l)}$ considering only off-load transportation time $\mathbf{T}_{\text{off}}^t \in \mathbb{R}^{|\mathcal{O}| \times |\mathcal{V}|}$, $\mathbf{h}_O^{V,(l)} = \text{HMHA}_O^V(\mathbf{h}_V^{(l-1)}, \mathbf{T}_{\text{off}}^t)$. On the other hand, on-load transportation time $\mathbf{T}_{\text{on}}^t \in \mathbb{R}^{|\mathcal{O}| \times |\mathcal{M}|}$ is added in the computation of HMHA_O^M by re-designing the augmented compatibility $\tilde{\sigma}_{OM}$ of equation (5):

$$\tilde{\sigma}_{OM} = \text{ReLU}(W^2(\text{ReLU}(W^1[\sigma_{OM}||(\mathbf{T}^p + \mathbf{T}_{\text{on}}^t)]))). \tag{7}$$

Finally, HMHA_O block outputs operation embedding matrix $\mathbf{h}_O^{(l)}$ for entire neighboring nodes as a linear transformation on the concatenation of $\mathbf{h}_O^{M,(l)}$ and $\mathbf{h}_O^{V,(l)}$:

$$\begin{aligned}\mathbf{h}_O^{(l)} =& \text{HMHA}_O(\mathbf{h}_M^{(l-1)}, \mathbf{h}_V^{(l-1)}, \mathbf{T}^p, \mathbf{T}^t) \\ =& W_O[\mathbf{h}_O^{M,(l)}||\mathbf{h}_O^{V,(l)}],\end{aligned} \tag{8}$$

where $W_O \in \mathbb{R}^{2d_e \times d_e}$ is a trainable parameter matrix.

HMHA_M block for machine node embedding $\mathbf{h}_M^{(l)}$ and HMHA_V block for vehicle node embedding $\mathbf{h}_V^{(l)}$ are structurally equivalent to the computation procedure of HMHA_O^M block and HMHA_O^V block, respectively. Thus, $\mathbf{h}_M^{(l)} = \text{HMHA}_M(\mathbf{h}_O^{(l-1)}, [\mathbf{T}^p]^{\text{T}})$, and $\mathbf{h}_V^{(l)} = \text{HMHA}_V(\mathbf{h}_O^{(l-1)}, [\mathbf{T}_{\text{off}}^t]^{\text{T}})$. Feed-forward (FF) block and add & normalization (AN) block in Fig. 2 are referred to in [10].

D. RL algorithm

We define policy $\pi_\theta(a_t|s_t)$, which generates O-M-V pair at a decision step,

$$\pi_\theta(a_t|s_t) = \pi_{\theta_o}(O_{ij}|s_t)\pi_{\theta_m}(M_k|O_{ij}, s_t), \tag{9}$$

where state s_t is a graph embedding that is computed as the mean of all node embeddings at the final layer, $\bar{h}^{(L)} = \frac{1}{|\mathcal{O}|+|\mathcal{M}|+|\mathcal{V}|}(\sum_{i=1}^{n}\sum_{j=1}^{n_i} h_{ij}^{(L)} + \sum_{k=1}^{m} h_k^{(L)} + \sum_{u=1}^{v} h_u^{(L)})$. We define two sub-policy models, π_{θ_o} and π_{θ_m}, as multi-layer perception (MLP) [16] that outputs probability distribution of the corresponding operation and machine nodes. In addition, for vehicle node selection, we use a heuristic approach, nearest vehicle selection (NVS) method. This is expressed as follows:

$$\begin{aligned}V_u =& \text{NVS}(O_{ij}, M_k) \\ =& \arg\min_{V_u \in \mathcal{V}_t} T_{iju}^t + T_{uk}^t,\end{aligned} \tag{10}$$

where it finds the nearest vehicle among current eligible vehicles $V_u \in \mathcal{V}_t$ given the current selected operation and machine nodes.

Size ($n \times m \times v$)	n_i	$\lvert\mathcal{M}_{ij}\rvert$	T^p_{ij}	$T^t_{kk'}$
10×5×5	U(4,6)	U(1,5)	U(1,20)	U(1,10)
20×10×10	U(8,12)	U(1,10)	U(1,20)	U(1,10)
30×15×15	U(12,18)	U(1,15)	U(1,20)	U(1,10)
40×20×20	U(16,24)	U(1,20)	U(1,20)	U(1,10)
50×25×25	U(20,30)	U(1,25)	U(1,20)	U(1,10)

TABLE II
SYNTHETIC INSTANCE PARAMETER DISTRIBUTION

In the training process, we update the policy $\pi_\theta(a_t|s_t)$ and encoder by using a policy-gradient method, which is known to be effective for end-to-end learning [17]. Here, for readability, all training parameters used in HJS are denoted as θ. In an episode, given policy π_θ, we record the stream of state, action, and reward samples, $\tau = (s_0, a_0, r_0, ..., s_T, a_T, r_T)$. We can calculate the total return $G(\tau) = \sum_{t=0}^{T} r_t$ of the trajectory. Our objective is to maximize the objective $J(\theta) = \mathbb{E}_{\pi_\theta}[G(\tau)]$. We use REINFORCE algorithm [18] to train the policy π_θ.

IV. PERFORMANCE EVALUATION

In this section, we evaluate the effectiveness of our proposed method (HJS) from three perspectives: makespan, generalization and running time. To this end, we generate various synthetic instances and compare the performance of HJS with that of several baseline dispatching rules, including FIFO (first in first out), SPT (shortest processing time first), and LPT (longest processing time first). In addition, we also compare the performance of HJS with two DRL-based baseline algorithms, namely MatNet [10] and Heterogeneous Graph Neural Network (HGNN) [9]. It is worth noting that DRL algorithms for solving FJSP-T entirely have not yet been developed, and existing DRL algorithms aim only to solve FJSP. In light of this, we augment these algorithms with a simple vehicle selection mechanism, i.e., the NVS method, to make them applicable to FJSP-T.

1) Experimental Settings: Likewise most related studies [6], [9], [13], we generate synthetic FJSP-T instances for training and testing. As shown in Table II, in order to generate random instances, we sample an instance from the uniform distribution, where the number of operation for each job is sampled in proportion to the number of machine, $n_i \sim \mathrm{U}(0.8|\mathcal{M}|, 1.2|\mathcal{M}|)$, and the number of compatible machine for each operation is sampled from the distribution, $|\mathcal{M}_{ij}| \sim \mathrm{U}(1, |\mathcal{M}|)$. Processing time T^p_{ijk} for O_{ij}-M_k pair and transportation time $T^t_{kk'}$ between M_k and $M_{k'}$ are sampled from $\mathrm{U}(0.8\bar{T}^p_{ij}, 1.2\bar{T}^p_{ij})$ and $\mathrm{U}(0.8\bar{T}^t_{kk'}, 1.2\bar{T}^t_{kk'})$, respectively. We train our model on two small-scale instances (10×5×5, 20×10×10), and test it on the large-scale instances (30×15×15, 40×20×20, 50×25×25) to verify generalization capability. When we evaluate the capabilities of the methods, we generate 100 different instances for each size and average the obtained results.

2) Performance on makespan: To verify the effectiveness of the proposed method in finding near-optimal solutions, we evaluate the performance in terms of makespan. Table III shows the results of all methods against four instances, where DRL methods are sufficiently trained on the same size instances. The bold character denotes the best result for each instance.

	Size			
	10×5×5	10×5×10	10×10×5	20×10×10
SPT	178.01 (1.58)	166.43 (1.57)	277.13 (3.41)	373.90 (6.27)
LPT	181.42 (1.59)	167.92 (1.60)	293.40 (3.31)	388.74 (6.15)
FIFO	176.85 (1.62)	166.16 (1.62)	276.30 (3.41)	376.10 (6.69)
HGNN	176.95 (1.75)	167.51 (1.76)	280.98 (3.96)	- -
MatNet	152.17 (1.83)	146.01 (1.84)	265.69 (3.91)	313.81 (7.33)
HJS (Ours)	**144.48 (1.40)**	**136.24 (1.39)**	**244.55 (2.97)**	**294.35 (5.40)**

TABLE III
MAKESPANCE PERFORMANCES ON SMALL-SCALE INSTANCES. EACH ELEMENT DENOTES MAKESPAN (RUNNING TIME (SEC))

	Size		
	30×15×15	40×20×20	50×25×25
SPT	567.21 (13.96)	746.95 **(25.15)**	924.04 (24.07)
LPT	602.15 (13.92)	797.52 (25.19)	1026.78 **(24.04)**
FIFO	569.89 (14.17)	751.41 (25.66)	934.88 (24.98)
HGNN	571.15 (29.45)	759.27 (98.72)	957.58 (168.66)
MatNet	468.59 (16.71)	614.53 (30.95)	775.5 (31.0)
HJS (Ours)	**435.82 (12.76)**	**573.54** (25.59)	**726.22** (28.54)

TABLE IV
GENERALIZATION PERFORMANCES ON LARGE-SCALE INSTANCES.

As can be seen, HJS outperforms both dispatching rules and existing DRL approaches. Specifically, to compare with the dispatching rule (FIFO), as the reference for the proposed method, it is observed that 22% makespan improvement on instance 10×5×5 and 27% improvement on instance 20×10×10 can be achieved. For the DRL method (MatNet), the proposed method can improve makespan by 5% and 6% on instance 10×5×5 and 20×10×10, respectively. This improvement can be attributed to our use of the graph embedding approach that fully encoding operations-machines-vehicles relation.

3) Performance on generalization: We evaluate the generalization capability by testing the model trained on small-scale instance (20×10×10) on unseen large-scale instances.

In Table IV, we can know that the proposed method outperforms all of dispatching rules and DRL methods. Notably, on the largest instance (50×25×25), our proposed method improves the makespan by 28% compared to FIFO and 6% compared to MatNet. It is worth noting that the proposed method can provide superior solutions than the dispatching rules even on unseen and large-scale instances. This results indicate that patterns learned on small-scale instances can effectively solve large-scale ones.

4) Run time analysis: In Table III and IV, the symbol $(\cdot)$ denotes running time (second) required for solving the corresponding instance. As shown, the proposed method yields the shortest running times for small-scale instances, but not for large-scale ones. This finding is consistent with previous studies [9], [16], which have reported that neural network inference is computationally more demanding than simple dispatching rules. However, it should be noted that the proposed method still achieves the shortest running time for the 30×15×15 instance and performs only slightly worse (with a gap of 1% to the SPT result) than the rule-based SPT approach for the 40×20×20 instance. Consequently, the proposed method can generate the best makespan solutions while maintaining the same level of computational efficiency as the rule-based solutions for any size instances.

V. Conclusions

In this paper, we have proposed a novel graph-based DRL method, HJS, to address FJSP-T. We have demonstrated that the proposed method offers superior solutions in terms of makespan and running time compared to the existing dispatching rules and DRL methods in small-scale instances. Furthermore, the graph-based method, capable of learning complex relation between operation-machine-vehicle nodes, contributes to the generalization performance. This is evidenced by the best makespan and relatively short running time results obtained, even in the unseen large-scale instances.

VI. Appendix

A. Attention mechanism

We first describe the attention mechanism, where there is a single type of node. Given node set X, let $h_x^{(l-1)}$ be an embedding vector of node $x \in X$ at layer $l-1$. The mechanism requires three vectors of *query* q, *key* k, and *value* v to construct the node embedding. These are expressed as follows,

$$\mathrm{q}_x = W^{\mathrm{q}} h_x^{(l-1)}, \mathrm{k}_y = W^{\mathrm{k}} h_y^{(l-1)}, \mathrm{v}_y = W^{\mathrm{v}} h_y^{(l-1)},\ x, y \in X \tag{11}$$

where $W^{\mathrm{q}}, W^{\mathrm{k}} \in \mathbb{R}^{d_e \times d_{\mathrm{k}}}$ and $W^{\mathrm{v}} \in \mathbb{R}^{d_e \times d_{\mathrm{v}}}$ are the trainable parameter matrices. $d_{\mathrm{k}} = \frac{d_e}{Z}$ is the query/key dimensionality, and d_{v} is the value dimensionality. Here, if $y = x$, then this is called the self-attention mechanism. With the query q_x of node x and the key k_y of node y, we calculate *compatibility* σ_{xy} as the scaled dot-product:

$$\sigma_{xy} = \begin{cases} \frac{\mathrm{q}_x^T \mathrm{k}_y}{\sqrt{d_k}} & \text{if } y \text{ is a neighbor of } x \\ -\infty & \text{otherwise} \end{cases} \tag{12}$$

where $-\infty$ prevents message passing between non-adjacent nodes. From the compatibility, we compute *attention weights* $\bar{\sigma}_{xy} \in [0, 1]$ using a softmax:

$$\bar{\sigma}_{xy} = \frac{e^{\sigma_{xy}}}{\sum_{y' \in X} e^{\sigma_{xy'}}}. \tag{13}$$

This measures the importance between x and y; higher attention weights $\bar{\sigma}_{xy}$ means that node x depends more on node y. Finally, we compute the attention-based single-head node embedding $h_{x,z}^{(l)}$ of node x at layer l as a weighted sum of messages v_y:

$$h_{x,z}^{(l)} = \sum_{y \in X} \bar{\sigma}_{xy} \mathrm{v}_y, \tag{14}$$

where $z \in \{1, ..., Z\}$ is a head index.

Multi-head attention (MHA) allows a node to receive neighboring messages from different types of attentions, Z times in parallel ($Z = 8$). The final multi-head attention value for node x is computed by adding up the heads from all types of attentions. This is a function of node embedding of all nodes as follows:

$$\begin{aligned} h_x^{(l)} &= \mathrm{MHA}_x(\{h_y | y \in X\}) \\ &= \sum_{z \in Z} W_z^{\mathrm{o}} h_{x,z}^{(l)}, \end{aligned} \tag{15}$$

where $W_z^{\mathrm{o}} \in \mathbb{R}^{d_{\mathrm{v}} \times d_e}$ is a trainable parameter matrix.

References

[1] Z. Qin and Y. Lu, "Self-organizing manufacturing network: A paradigm towards smart manufacturing in mass personalization," *Journal of Manufacturing Systems*, vol. 60, pp. 35–47, 2021.

[2] S. Kim, Y. Won, K.-J. Park, and Y. Eun, "A Data-Driven Indirect Estimation of Machine Parameters for Smart Production Systems," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 6537–6546, 2022.

[3] Y. Won, S. Kim, K.-J. Park, and Y. Eun, "Continuous productivity improvement using IoE data for fault monitoring: An automotive parts production line case study," *Sensors*, vol. 21, no. 21, p. 7366, 2021.

[4] S. Kim, Y. Won, K.-J. Park, and Y. Eun, "An Indirect Estimation of Machine Parameters for Serial Production Lines with Bernoulli Reliability Model," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 5540–5545.

[5] Y. Zhang, H. Zhu, D. Tang, T. Zhou, and Y. Gui, "Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems," *Robotics and Computer-Integrated Manufacturing*, vol. 78, p. 102412, 2022.

[6] W. Ren, Y. Yan, Y. Hu, and Y. Guan, "Joint optimisation for dynamic flexible job-shop scheduling problem with transportation time and resource constraints," *International Journal of Production Research*, vol. 60, no. 18, pp. 5675–5696, 2022.

[7] S. Zhang, X. Li, B. Zhang, and S. Wang, "Multi-objective optimisation in flexible assembly job shop scheduling using a distributed ant colony system," *European Journal of Operational Research*, vol. 283, no. 2, pp. 441–460, 2020.

[8] J. Yan, Z. Liu, C. Zhang, T. Zhang, Y. Zhang, and C. Yang, "Research on flexible job shop scheduling under finite transportation conditions for digital twin workshop," *Robotics and Computer-Integrated Manufacturing*, vol. 72, p. 102198, 2021.

[9] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible Job-Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1600–1610, 2022.

[10] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5138–5149, 2021.

[11] 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.

[12] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *arXiv preprint arXiv:2102.09544*, 2021.

[13] 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.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[15] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *arXiv preprint arXiv:1803.08475*, 2018.

[16] 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, pp. 1621–1632, 2020.

[17] R. S. Sutton, A. G. Barto *et al.*, "Introduction to reinforcement learning," 1998.

[18] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.