Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

IMITATION LEARNING FOR REAL-TIME JOB SHOP SCHEDULING USING GRAPH-BASED REPRESENTATION

Je-Hun Lee
Hyun-Jung Kim

Department of Industrial and Systems Engineering
Korea Advanced Institute of Science and Technology
291 Daehak Street
Daejeon, 34141, REPUBLIC OF KOREA

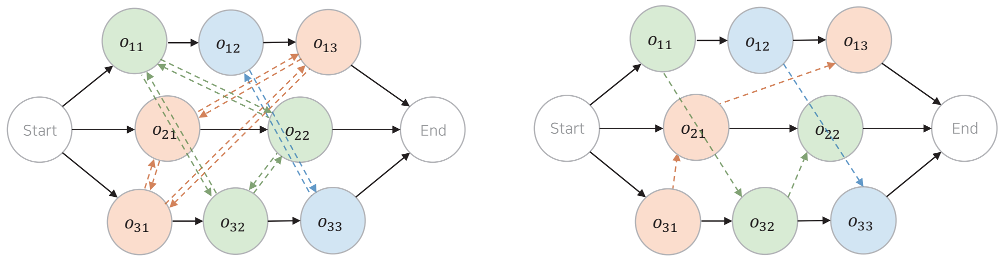
ABSTRACT

Scheduling of manufacturing systems in practice is challenging due to dynamic production environments, such as random job arrivals and machine breakdowns. Dispatching rules are often used because they can be easily applied even in such dynamic manufacturing environments. However, dispatching rules often fail to provide a satisfactory production schedule because they cannot consider overall system states when assigning jobs. Therefore, we develop a real-time scheduling method using imitation learning, especially behavior cloning, to solve job shop scheduling problems. We define a set of available actions, a target optimal policy, and a dynamic graph-based state representation method for imitation learning. The proposed method is size-agnostic, which then can be applied to unseen larger problems. The experimental results show that the proposed method performs better to minimize makespan than other dispatching rules in dynamic job shops.

1 INTRODUCTION

A job shop scheduling problem (JSSP) is known to be NP-hard (Garey et al. 1976). In the JSSP, a job has multiple operations, each of which is conducted in a specific machine sequentially. It is required to determine the operation sequence on each machine while minimizing the makespan. The JSSP can be found in many manufacturing areas, especially for semiconductors, liquid crystal displays (LCDs), and automotive industries (Tanev et al. 2004; Fayad and Petrovic 2005; Liu et al. 2014; Lee et al. 2021). Some studies have proposed exact optimization methods for the JSSP (Manne 1960; Lomnicki 1965), but numerous studies have applied problem-specific heuristics (Mason et al. 2005; Pfund et al. 2008), meta-heuristics (Gonçalves et al. 2005; Elmi et al. 2011), or dispatching rules (Dominic et al. 2004; Chen and Matis 2013) due to the high complexity of the problem. Among them, dispatching rule-based scheduling methods are widely used in practice because they can make decisions on assigning jobs instantly for dynamic production environments. Several companies have tried to use some exact or heuristic optimization approaches beyond dispatching rules (Klemmt et al. 2017), but many of manufacturing lines in Korea have still been using dispatching rule-based real-time scheduling programs (Ko et al. 2013; Lee et al. 2018) due to the complexity and dynamics of scheduling environments. The real-time dispatcher is especially useful for manufacturing systems with various job types which arrive randomly (Hu 2013; Wang et al. 2017).

In general, the simple dispatching rules do not provide satisfactory schedules in the JSSP. Therefore, many studies have proposed machine learning-based approaches to provide more sophisticated dispatching rules. Lee et al. (2020) proposed a supervised learning-based method with sample schedules obtained from sequential simulation to provide weights used for dispatching rules. Aydin and Öztemel (2000) and Chen et al. (2010) proposed Q-learning algorithms to choose an appropriate dispatching rule and



(a) (b)

Figure 1: Disjunctive graphs: (a) initial state, (b) complete state.

a combination of rules, respectively, in each state. The states are defined only with a certain range of processing times on machines and WIP (work in process) in Aydin and Öztemel (2000) and Chen et al. (2010), respectively. Note that if dispatching rules are used as the action set instead of individual operations, the solution space is restricted because they choose only an operation with the extreme value of a certain criteria, especially processing time for the shortest processing time(SPT) rule, and better schedules cannot be obtained compared to combinations of dispatching rules. Lin et al. (2019) and Yang et al. (2021) used deep reinforcement learning (DRL) for selecting a certain dispatching rule in each state. They have used statistical values as state features, such as the average processing times and maximum processing time of jobs. Hence, it is not possible to differentiate the production states of different instances as long as their statistical values are the same.

Zhang et al. (2020) and Park et al. (2021) proposed reinforcement learning (RL) methods using disjunctive graphs. They represented the state of the scheduling environment by using the disjunctive graph. A disjunctive graph has conjunctive arcs and disjunctive arcs which represent the precedence between operations of each job and in each machine, respectively. Figure 1 shows two examples of disjunctive graphs. Node $o_{i,j}$ represents the j^{th} operation of job i, and the nodes (or operations) requiring the same machine have the same color. After determining the sequence of operations in each machine, the direction of disjunctive arcs is determined. Their proposed approaches are size-agnostic in that the number of operations can be changed because the relationship learned between nodes by using a graph neural network (GNN) model can be applied to new nodes. In the methods of Zhang et al. (2020) and Park et al. (2021), selecting an action usually means choosing an available node, which represents an operation, and states include features each node has, such as the processing time of the corresponding operation. Zhang et al. (2020) considered all the first unscheduled operations of each job as a set of available actions regardless of whether the operation is waiting to be processed or not. Hence, selecting an action (or an operation) can change the start time of operations already scheduled, which makes it impossible to be used for real-time scheduling. Park et al. (2021) used real-time features in a state but only considered the operations, which can start immediately or soon, as an available action set. This action set can make only partial solution space, so we propose a novel definition of a set of available actions which can make wider solution space. The definition is described in Section 2.1.

Imitation learning is one of the methods used to learn expert decisions without explicit reward (Hussein et al. 2017; Attia and Dayan 2018). There are two approaches in imitation learning, behavioral cloning (BC) and apprenticeship learning (AL); AL proceeds a learning process by generating a reward function from the expert demonstrations, but BC uses supervised learning to predict actions based on states. RL algorithms search for many trajectories including actions not giving good schedules in exploration. Therefore, it can be very beneficial to use the information of optimal solutions while searching for nodes as in imitation learning.

Therefore, we propose an imitation learning approach based on a graph for real-time job shop scheduling problems. The method first builds optimal policies from optimal schedules of small-sized JSSPs and then train them by using BC with a dynamic disjunctive graph. Zhang et al. (2020) proposed an adding-arc strategy to reduce the number of disjunctive arcs in the graph, but it cannot have the machine information on which the unscheduled nodes should be assigned. As an alternative, we propose a dynamic disjunctive graph to make the original one sparse and efficient. The optimal policy generation method is described in Section 2. We then train an agent by imitation learning to learn the optimal action selections using a complex GNN model described in Section 3. We test the method in dynamic environments where jobs arrive randomly as well as deterministic ones. The experimental results in Section 4 show better performance of the proposed method compared to dispatching rules.

2 OPTIMAL POLICY DATA SET FOR JSSP

We first derive a data set containing optimal policies in order to learn a scheduling agent which dynamically assigns operations to idle machines. Optimal schedules are obtained from constraint programming (CP) (Zhou 1996). From the beginning, at each decision-making point (state) where there are one or more idle machines that have multiple available actions (nodes), an operation assigned in an optimal schedule, the set of available actions, and node features that show the states of remaining operations and machines, are stored in the data set. We do not consider the situations where there is only one available operation node for an idle machine as a state because the machine can just start the operation. We explain the definition of available actions and node features, and procedure of policy generation in this section.

2.1 Action Set

Each time a machine becomes idle, choosing one of the operations waiting can lead to a sub-optimal solution. For example, Figure 2 shows three schedules that can be derived according to different definitions of available action set for a JSSP with four jobs and four machines. The boxes with the same color indicate the operations from the same job, and the value in the box represents the processing time of each operation. Assume that the current time is 1 ($t = 1$) where machines 3 and 4 have completed operations and machine 1 and 2 are processing the blue and green job, respectively. For machine 3, only the second operation of the purple job is waiting to be processed, and choosing the operation leads to the complete schedule as illustrated in Figure 2 (c). Many dispatching rules consider the only job at $t = 1$, however, a better solution in terms of makespan can be obtained if machine 3 keeps idle for one more second. Therefore, it is important to consider not only operations waiting but also operations that can start soon. Park et al. (2021) consider the green job too because it may soon enter machine 3 because its first operation is being processed on machine 2. Hence, they can reserve the green one at $t = 1$ on machine 3, and the machine maintains idle status until $t = 2$. The definition of an available action set of Park et al. (2021) can lead to the complete schedule illustrated in Figure 2 (b).

Furthermore, we can generate a larger number of schedules than Park et al. (2021) when we allow reserving more operations. Figure 2 (a) shows better makespan than Figure 2 (b) because Figure 2 (a) also considers the blue job at $t = 1$ on machine 3. We propose the novel definition of an available action set which can generate the case of Figure 2 (a). The following shows our proposed definition of the set of available actions at a state.

Definition 1 *Set of available actions: The set of available operations $\mathscr{A}$ is defined as $\{o_{ij} \in W_k \cup R_k \mid k \in K_{idle}, |W_k \cup R_k| > 1\}$, where W_k is a set of operations waiting for machine k, and R_k is a set of reservable operations for machine k, K_{idle} is a set of idle machines. R_k is defined as $\{o_{i'j'} \mid r_{i'j'} < p_{ij}, \forall o_{ij} \in W_k, o_{i'j'} \in U_k \setminus W_k\}$, where U_k is a set of operations that have to be processed on machine k but not been assigned yet, p_{ij} is the processing time of o_{ij}, and r_{ij} is the ready time of o_{ij}.*

The ready time of an operation indicates the earliest possible arrival time to the corresponding machine. It is computed by $r_{ij} = \max\{C_{in_i} + \sum_{l=n_i+1}^{j-1} p_{il}, C_{in_i+1} + \sum_{l=n_i+2}^{j-1} p_{il}, \cdots, C_{ij-1}\}$ where C_{ij} is the earliest completion time of o_{ij}, and n_i is the index of the operation of job i that is being processed or waiting to be processed. For n_i, o_{in_i} is being processed or waiting to process. C_{ij} is the remaining processing time of o_{ij}

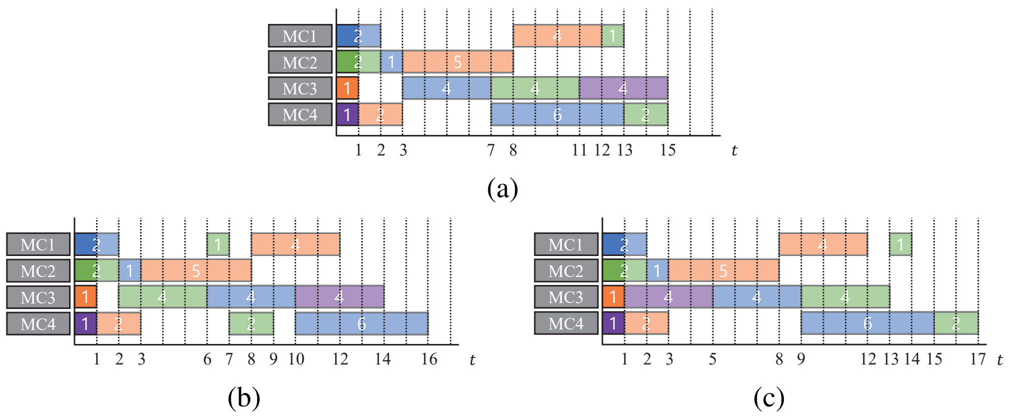


Figure 2: Three schedules that can be derived according to different definitions of available action set: (a) proposed definition, (b) Park et al. (2021), (c) only waiting job.

if o_{ij} is being processed and also considers jobs that have not arrived yet but are reserved for processing in the machine. In Figure 2 (a), selecting the blue job for machine 3 as an action at $t = 1$ is the operation reservation. The blue job will be processed between $t = 3$ and $t = 7$ in machine 3. After reserving the blue one for machine 3 at $t = 1$, the ready time of the green operation for machine 3 becomes 7.

The definition allows considering all the unscheduled operations as potential candidates with a proportion, while Park et al. (2021) consider only unscheduled operations that have all their predecessors within the job route have already been scheduled. From the above definition, the set of available actions includes not only operations waiting but also operations that have not arrived yet and have less ready times than processing times of all operations waiting. We further provide Proposition 1 to reduce the number of available actions. Actions satisfying the condition in the proposition are eliminated from the action set. This has been applied to Definition 1.

Proposition 1 *For $o_{ij} \in W_k$ and $o_{i'j'} \in U_k \setminus W_k$, selecting $o_{i'j'}$ is dominated by selecting o_{ij} if $r_{i'j'} \geq p_{ij}$.*

Proof. If $r_{i'j'} \geq p_{ij}$, $o_{i'j'}$ arrives after completing o_{ij}, and therefore, there is no need to select $o_{i'j'}$ instead of o_{ij}. Note that, selecting o_{ij} can make an active schedule but selecting $o_{i'j'}$ makes an inactive schedule. $\square$

2.2 Optimal Policy Generation

An optimal policy is to select an optimal action at each decision-making point. An agent is going to learn the optimal policy as a target in our proposed imitation learning method. However, when multiple machines become idle at the same time, multiple actions should be assigned simultaneously. Figure 3 shows intermediate states and different transitions which can be appeared from S_0 to state S_7. States S_0, S_1, $\cdots$, S_6 are detected at the same time. If S_7 is found in an optimal schedule, we have to generate target optimal policies of the intermediate states to learn a scheduling agent. Each state S_4, S_5, and S_6 has one optimal action. For instance, the optimal action of state S_4 is O_{21}. However, states S_0, S_1, S_2, S_3 have multiple optimal actions. For S_0, the three actions (selecting O_{12}, O_{32}, and O_{21}) are optimal actions, and the scheduling agent should be learned to select one of them with the same probability. Hence, the target optimal policy of S_0 has the value $1/3$ for each of the optimal actions. In the same way, the optimal policy of state S_1 is to have the probability of $1/2$ for selecting O_{32} and O_{21}, respectively. The generated target optimal policies are used in the learning step.

2.3 Graph-based State Representation

We propose a dynamic graph to represent the state of a decision-making point. In the graph for the JSSP, disjunctive arcs are changed to conjunctive arcs each time an operation is assigned. Once an operation is

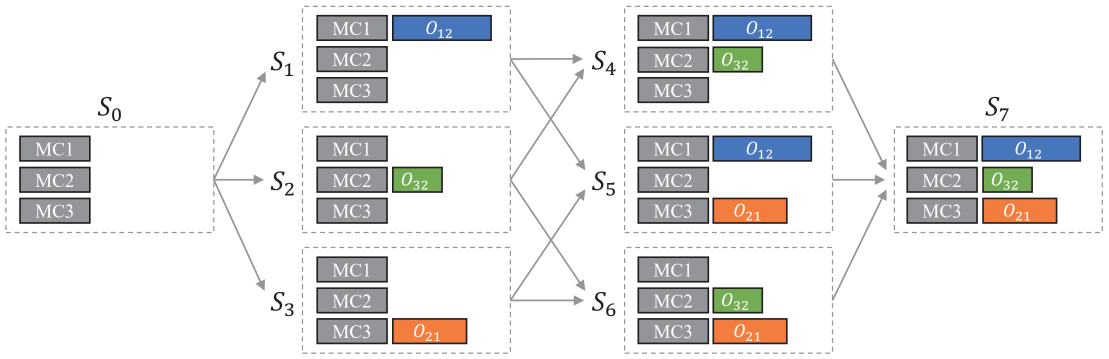


Figure 3: Intermediate states and transitions at a same time.

completed, choosing one of the remaining operations is required. Therefore, completed operations have little effect on future decisions. We hence propose a dynamic graph that has no finished nodes as illustrated in Figure 4(b). The proposed dynamic disjunctive graph in Figure 4(b) does not contain completed nodes compared to the original disjunctive graph in Figure 4(a). The reservation nodes, which are not yet ready to be processed but are selected by the agent, do not affect disjunctive arcs because the other operations can be processed before the reserved operations.

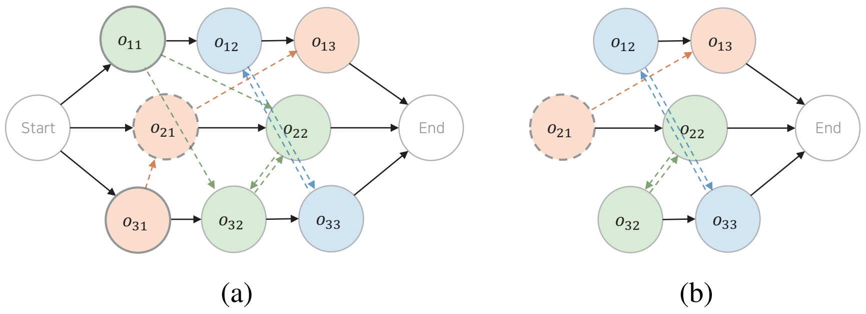


Figure 4: The graph representation: (a) disjunctive graph, (b) dynamic disjunctive graph.

We define the node features to embed the current schedule by considering dynamic situations as follows:

- Operation status: [1, 0] and [0, 1] indicate that the operation has not been scheduled yet and has been scheduled, respectively. [0, 1] also considers the operation being processed or having been reserved.
- Processing time: The processing time of each operation is used. If the operation is being processed, the current remaining processing time is used.
- Tail processing time: It is the sum of processing times of succeeding operations that have to be performed to complete its job excluding itself.
- Relative processing time: The ratio of the processing time of the node to the total remaining processing time in its job. The total remaining processing time is equal to the sum of the tail processing time and its processing time. If the operation is the last one in its job, the value becomes 1.
- Number of succeeding operations: The number of succeeding operations from the operation excluding itself is used. If the operation is the last one in its job, the value becomes 0.

These features have no information about the operation nodes completed before. Note that the processing time-related features, processing time and tail processing time, are normalized with the maximum processing time. If a dynamic disjunctive graph is not used, an additional dimension for completed operations and the original processing times of the nodes are required.

3 LEARNING FOR OPTIMAL POLICY

A GNN model is used for node embedding from the graph-based state, and it computes selection probabilities, called a policy, for available actions. The parameters of the GNN model are updated in the learning step to minimize loss with optimal policies previously generated, and the agent selects an action in the execution step with the highest selection probability for every decision-making point. Since the GNN model learns the relation between nodes, it can be applied with no additional learning even if the graph size is changed after learning step, and the selection probabilities are also able to be computed even if the number of nodes is changed. In this section, we first describe how to compute the selection probabilities by GNN models we use and then describe the learning step.

3.1 GNN Model

A GNN model provides embedding vectors as the output by aggregating node features. We design two GNN models denoted as GNN1 and GNN2. They are modified from GNN models proposed by Park et al. (2021) and Park et al. (2020), respectively. We further consider an attention mechanism to learn with different weights for nodes (Vaswani et al. 2017), which means that a GNN model learns which nodes are important for the current state described by a graph. We also add a global embedding vector to reflect the structure information of the graph (Park et al. 2020).

For the first model, GNN1, the l^{th} embedding vector of a node v, h_v^l, is represented as follows:

$$h_v^l = f_1\Big(A_1\Big(h_u^{l-1}, u \in \mathcal{N}_p(v)\Big) \| A_2\Big(h_u^{l-1}, u \in \mathcal{N}_s(v)\Big) \| \\ A_3\Big(h_u^{l-1}, u \in \mathcal{N}_d(v)\Big) \| act\Big(\sum_{u \in \mathcal{V}} h_u^{l-1}\Big) \| h_v^{l-1} \| x_v\Big)$$

where $\mathcal{N}_p(v)$, $\mathcal{N}_s(v)$, and $\mathcal{N}_d(v)$ are the sets of neighbor nodes connected with node v by input and output conjunctive edges and a disjunctive edge, respectively, $\mathcal{V}$ is a set of all nodes, x_v is the node features of node v, $act(\cdot)$ is an activation function, $f_z(\cdot)(z \in \mathbb{N})$ is a learnable multi-layer network built with N_l layers and $act(\cdot)$, $A_z(\cdot)(z \in \mathbb{N})$ is a learnable attention network (Vaswani et al. 2017), and $\|$ is the vector concatenation operator. $\mathcal{N}_p(v)$ and $\mathcal{N}_s(v)$ are $\emptyset$ for the first and last operations of each job, respectively. For the other operations, the size of $\mathcal{N}_p(v)$ and $\mathcal{N}_s(v)$ becomes one in a JSSP. This GNN model aggregates the $(l-1)^{th}$ embedding vectors of neighbors and itself into its l^{th} embedding vector. The initial embedding vector of node v, h_v^0, becomes $e(x_v)$ where e is a non-linear transformation function built with N_l layers. The final embedding vector h_v becomes $h_v^{N_a}$ where N_a is the number of aggregation of the GNN model.

The attention network is described as follows:

$$A\Big(h_v, u \in \mathcal{N}(v)\Big) = f_2\Big(\|_{y=1}^{Y}\Big(\sum_{v \in \mathcal{N}} \alpha_v^y h_v\Big)\Big)$$

$$\alpha_v = \frac{exp\Big(f_3(h_v)\Big)}{\sum_{u \in \mathcal{N}(v)} exp\Big(f_3(h_u)\Big)}$$

where Y is the number of heads of attention network, α_v^y is normalized attention score of head y for node v.

The second model, GNN2, embeds the node features into each layer composed of the same type of edges, which are input conjunctive edges, output conjunctive edges, and disjunctive edges. There are three

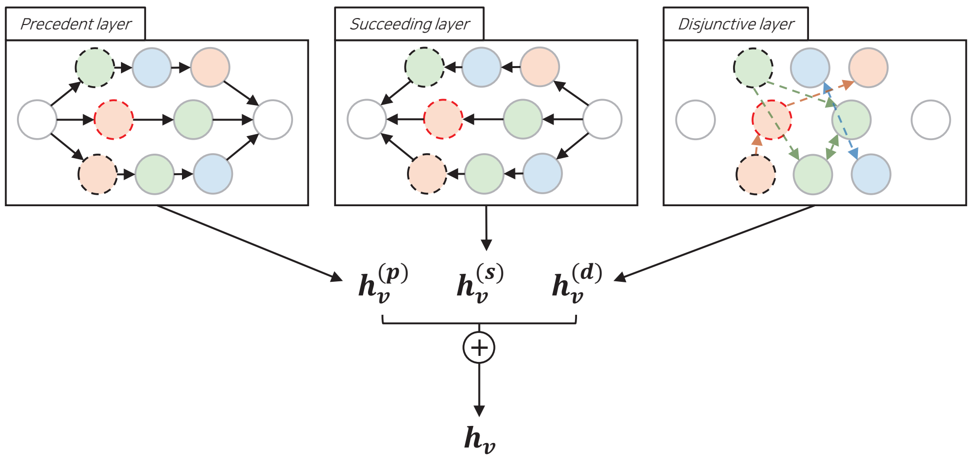


Figure 5: Structure of the proposed multiplex GNN model

layers (p: input conjunctive edges, s: output conjunctive edges, d: disjunctive edges) on the disjunctive graph from the three types of edges. The l^{th} embedding vectors for the layers are represented as follows:

$$h_v^{(p)l} = f_4\left(A_4\left(h_u^{(p)l-1} \| x_u, u \in \mathcal{N}_p(v)\right)\right)$$

$$h_v^{(s)l} = f_5\left(A_5\left(h_u^{(s)l-1} \| x_u, u \in \mathcal{N}_p(v)\right)\right)$$

$$h_v^{(d)l} = f_6\left(A_6\left(h_u^{(d)l-1} \| x_u, u \in \mathcal{N}_p(v)\right)\right)$$

The final embedding vector h_v becomes the average vector of $f_7(h_v^{(p)N_a} || h_v^{(s)N_a} || h_v^{(d)N_a})$ for $v \in \mathcal{V}$.

Finally, we consider the global embedding vector in order to get a policy for the available actions. The selection probability of node v, $\phi(v)$, is calculated by using the softmax function as follows:

$$\phi(v) = \frac{exp\left(f_8(h_v || g)\right)}{\sum_{u \in \mathcal{A}} exp\left(f_8(h_u || g)\right)}$$

where $\mathcal{A}$ is the set of available actions, and g is the global embedding vector that is the average vector of h_v for all operation nodes in the current graph. The network $f_8(\cdot)$ outputs a scalar score for each node in order to compute a probability.

3.2 Learning Process

The parameters of the GNN model are updated to minimize the loss, which is the mean squared error (MSE) between the target optimal policy and the predicted policy of the GNN model. During the learning process, we use a cyclical learning rate so that the best value of a learning rate can be obtained without many experiments (Smith 2017). After learning, the agent selects an available operation node with the highest selection probability at each decision-making point.

4 EXPERIMENTAL RESULTS

4.1 Optimal Decision Generation

We train the scheduling agent we propose by using the optimal policies derived from JSSP. We generate various JSSP instances in order to learn the characteristics of JSSP without being biased. Each one has one or more decision-making points (states).

An instance in the problem set titled 'hun *m*x*n*' has *m* jobs and *n* machines. There are 120 instances for each problem size, and the characteristics according to an instance index are shown in Table 1. The third column of Table 1 indicates the ratio of machines that are processed by jobs. If it is 1, all jobs have to visit all machines. Among the 120 instances, 20 instances are flow shop problems. We train the model by simultaneously using problem sets, hun 4x4, hun 6x6, hun 8x8, and hun 10x10, and choose the policy which has the best objective value for the largest problem set hun 10x10. The hyper-parameters used for training are provided in the Appendix A.

Table 1: Characteristics of JSSP instances 'hun *m*x*n*'.

Instance index	Range of processing times	Ratio of machines to go	Additional feature
1-20	[1,20]	1	-
21-40	[1,100]	1	-
41-60	[81,100]	1	-
61-80	[1,20]	0.8	-
81-100	[1,20]	0.6	-
101-120	[1,20]	1	Fixed machine flow

Since the proposed model has a GNN-based size-agnostic structure, we test the model for some of 'tai' benchmark instances for which the optimal solutions are known as well as the instances used for training with no additional learning (Taillard 1993). We also test the model to several dynamic job arrival environments extended from 'tai' benchmark instances that have 50 jobs. The operation sequences and processing times are the same as the original 'tai' benchmark instances. Among the 50 jobs, 20 jobs are ready at the beginning, and the remaining 30 jobs arrive at intervals of 400 in a batch of 10 jobs. The last batch of 10 jobs arrives at 1200.

4.2 Performance

We evaluate the scheduling performance of the proposed method by computing the optimality gap defined as follows:

$$\text{Optimality gap}(\%) = \frac{\text{Makespan from the proposed method} - \text{Optimal makespan}}{\text{Optimal makespan}} \times 100$$

The optimal schedules are obtained by using CP in Zhou (1996). The performance of the proposed method is compared with random selection and several dispatching rules listed as follows:

- Shortest processing time (SPT)
- Longest processing time (LPT)
- Shortest total processing time (STPT)
- Longest total processing time (LTPT)
- Shortest tail processing time (STT)
- Longest tail processing time (LTT)
- Least operation remaining (LOR)
- Most operation remaining (MOR)

Table 2: Optimality gap(%) for JSSP instances according to methods.

	Train problem sets				Test problem sets		
Method	hun 4x4	hun 6x6	hun 8x8	hun 10x10	tai 15x15	tai 50x15	tai 50x20
random	12.72	18.02	21.99	23.9	31.37	25.03	31.40
SPT	10.01	13.73	19.28	21.29	25.89	24.11	26.86
LPT	17.58	24.49	28.33	29.97	41.22	41.21	45.15
SRPT	17.45	23.94	31.39	34.07	47.68	38.47	48.12
LRPT	8.80	11.51	14.37	15.37	19.15	16.86	19.15
STT	20.38	26.4	33.73	34.95	44.6	41.10	47.24
LTT	5.44	9.72	12.23	12.64	19.49	14.28	16.36
LOR	16.05	23.45	30.57	31.96	40.93	35.56	45.00
MOR	7.46	11.97	14.59	15.33	20.53	17.37	18.88
IL_1	2.48	7.42	13.84	17.42	23.16	30.31	33.54
IL_2	3.37	7.14	16.45	22.92	38.82	38.54	49.44
IL_dyn_1	**2.25**	7.42	12.69	16.73	20.81	18.56	27.31
IL_dyn_2	2.82	6.48	13.84	19.57	39.99	35.69	49.22
IL_rsv_1	3.03	6.62	11.59	15.95	33.24	28.47	42.87
IL_rsv_2	5.39	7.3	9.84	**12.34**	19.35	16.75	23.6
IL_rsv_dyn_1	2.3	**5.73**	11.46	15.55	27.67	41.27	55.78
IL_rsv_dyn_2	4.92	7.41	**9.65**	12.81	**18.84**	**11.46**	**14.86**

The performance of the proposed method and rules are shown in Table 2. The values in the table are the average optimality gaps of the JSSP instances. In Table 2, the average gap from 20 simulations is used for the random selection, and operation reservation is not considered in the dispatching rules. The performance of the proposed method is evaluated according to different action maskings, graph types, and GNN models. 'IL' indicates the proposed imitation learning method, 'dyn' means to use a dynamic disjunctive graph, 'rsv' represents that the agent considers operation reservation (Definition 1), and the number indicates the GNN model type the method used.

Overall, the proposed imitation learning shows better performance than dispatching rules. Among the dispatching rules, the LTT rule has the smallest gap, but IL_rsv_dyn_2 performs better than LTT for all instances except for hun 10x10. The average optimal gap of IL_rsv_dyn_2 is 11.42% where the LTT rule has a gap of 12.88%.

The performance with the operation reservation (Definition 1) is better than the performance by considering only jobs waiting now. IL_rsv_dyn_2 and IL_rsv_2 show better results than IL_dyn_2 and IL_2, respectively, for instances that have eight or more jobs. In terms of the GNN model, the proposed multiplex GNN model (GNN2) has a smaller gap when considering the operation reservation together. IL_rsv_dyn_2 and IL_rsv_2 perform better than IL_rsv_dyn_1 and IL_rsv_1 for instances that have eight or more jobs, respectively. Especially, the operation reservation and GNN2 are more effective for larger instances, which are in tai 15x15, tai 50x15, and tai 50x20, which are not used for the learning steps. Also, the model with a dynamic disjunctive graph shows better performance for the unseen larger instances. Comparing IL_rsv_dyn_2 and IL_rsv_2, the gap of IL_rsv_dyn_2 is larger than IL_rsv_2 by 0.02% for learning instances but smaller by 4.85% for test instances.

We additionally test the best model, IL_rsv_dyn_2, and rules for scheduling environments with dynamic job arrivals. Their results are shown in Table 3. The performance is compared with makespan because it is hard to obtain an optimal solution in this case. The LTT rule has the best value among the dispatching rules, and IL_rsv_dyn_2 shows a 1.44% improvement over the LTT rule on average.

Table 3: Makespan for dynamic JSSP instances according to methods.

	Test problem sets	
Method	dynamic tai 50x15	dynamic tai 50x20
random	3628.1	4237.1
SPT	3585.3	4096.7
LPT	4054.4	4546.2
SRPT	3993.5	4610.9
LRPT	3410.4	3713.4
STT	3999.5	4672.6
LTT	3357.0	3679.8
LOR	3889.7	4527.1
MOR	3389.4	3738.2
IL_rsv_dyn_2	**3291.1**	**3646.1**

5 CONCLUSION

We have proposed an imitation learning-based method to minimize makespan for a JSSP. We have defined a data set from optimal schedules, which consists of available actions, a target optimal policy, and dynamic graph-based state representations. The agent is then learned by using a GNN model for the dynamic disjunctive graph. The proposed method shows better performance than dispatching rules for unseen larger benchmark instances as well as the instances used for learning. Additionally, we may apply meta-learning methods to improve the performance of the unseen environment in the future.

ACKNOWLEDGMENTS

This work was supported by VMS solutions Co. Ltd. and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education [2022R1A2C400197811].

A HYPER-PARAMETERS

- Activation function: leaky ReLU (Maas et al. 2013)
- Learning optimizer: Adam (Kingma and Ba 2014)
- Number of learning steps: 1000
- Learning batch size: 512
- Base learning rate: 0.01
- Max learning rate: 0.0001
- Step size up: 5
- Step size down: 15
- Number of layers N_l: 2
- Number of aggregations N_a: 2
- Number of heads for attention Y: 3
- Dimension of embedding vector: 8

REFERENCES

Attia, A., and S. Dayan. 2018. "Global Overview of Imitation Learning". *arXiv preprint arXiv:1801.06503*. https://arxiv.org/abs/1801.06503, accessed 17th September 2022.

Aydin, M. E., and E. Öztemel. 2000. "Dynamic Job-shop Scheduling using Reinforcement Learning Agents". *Robotics and Autonomous Systems* 33(2-3):169–178.

Chen, B., and T. I. Matis. 2013. "A Flexible Dispatching Rule for Minimizing Tardiness in Job Shop Scheduling". *International Journal of Production Economics* 141(1):360–365.

Chen, X., X. Hao, H. W. Lin, and T. Murata. 2010. “Rule Driven Multi Objective Dynamic Scheduling by Data Envelopment Analysis and Reinforcement Learning”. In *2010 IEEE International Conference on Automation and Logistics*, 396–401. IEEE.

Dominic, P. D., S. Kaliyamoorthy, and M. S. Kumar. 2004. “Efficient Dispatching Rules for Dynamic Job Shop Scheduling”. *The International Journal of Advanced Manufacturing Technology* 24(1):70–75.

Elmi, A., M. Solimanpur, S. Topaloglu, and A. Elmi. 2011. “A Simulated Annealing Algorithm for the Job Shop Cell Scheduling Problem with Intercellular Moves and Reentrant Parts”. *Computers & Industrial Engineering* 61(1):171–178.

Fayad, C., and S. Petrovic. 2005. “A Fuzzy Genetic Algorithm for Real-world Job Shop Scheduling”. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 524–533. Springer.

Garey, M. R., D. S. Johnson, and R. Sethi. 1976. “The Complexity of Flowshop and Jobshop Scheduling”. *Mathematics of Operations Research* 1(2):117–129.

Gonçalves, J. F., J. J. de Magalhães Mendes, and M. G. Resende. 2005. “A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem”. *European Journal of Operational Research* 167(1):77–95.

Hu, S. J. 2013. “Evolving Paradigms of Manufacturing: From Mass Production to Mass Customization and Personalization”. *Procedia Cirp* 7:3–8.

Hussein, A., M. M. Gaber, E. Elyan, and C. Jayne. 2017. “Imitation Learning: A Survey of Learning Methods”. *ACM Computing Surveys* 50(2):1–35.

Kingma, D. P., and J. Ba. 2014. “Adam: A Method for Stochastic Pptimization”. *arXiv preprint arXiv:1412.6980*. https://arxiv.org/abs/1412.6980, accessed 17th September 2022.

Klemmt, A., J. Kutschke, and C. Schubert. 2017. “From Dispatching to Scheduling: Challenges in Integrating a Generic Optimization Platform into Semiconductor Shop Floor Execution”. In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D’Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 3691–3702. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Ko, K., B. H. Kim, and S. K. Yoo. 2013. “Simulation based Planning & Scheduling System: MozArt®”. In *Proceedings of the 2013 Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 4103–4104. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lee, J., Y. Kim, J. Kim, Y.-B. Kim, H.-J. Kim, B.-H. Kim, and G.-H. Chung. 2018. “A Framework for Performance Analysis of Dispatching Rules in Manufacturing Systems”. In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 3550–3560. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lee, J.-H., H.-J. Kim, Y. Kim, Y. B. Kim, B.-H. Kim, and G.-H. Chung. 2021. “Machine Learning-based Periodic Setup Changes for Semiconductor Manufacturing Machines”. In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, 1–10. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lee, J.-H., Y. Kim, Y. B. Kim, B.-H. Kim, G.-H. Jung, and H.-J. Kim. 2020. “A Sequential Search Method of Dispatching Rules for Scheduling of LCD Manufacturing Systems”. *IEEE Transactions on Semiconductor Manufacturing* 33(4):496–503.

Lin, C.-C., D.-J. Deng, Y.-L. Chih, and H.-T. Chiu. 2019. “Smart Manufacturing Scheduling with Edge Computing using Multiclass Deep Q Network”. *IEEE Transactions on Industrial Informatics* 15(7):4276–4284.

Liu, T.-K., Y.-P. Chen, and J.-H. Chou. 2014. “Solving Distributed and Flexible Job-shop Scheduling Problems for a Real-world Fastener Manufacturer”. *IEEE Access* 2:1598–1606.

Lomnicki, Z. 1965. “A “Branch-and-bound” Algorithm for the Exact Solution of the Three-machine Scheduling Problem”. *Journal of the Operational Research Society* 16(1):89–100.

Maas, A. L., A. Y. Hannun, and A. Y. Ng. 2013. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In *Proceedings of the International Conference on Machine Learning*, Volume 30.

Manne, A. S. 1960. “On the Job-shop Scheduling Problem”. *Operations Research* 8(2):219–223.

Mason, S., J. Fowler, W. Carlyle, and D. Montgomery. 2005. “Heuristics for Minimizing Total Weighted Tardiness in Complex Job Shops”. *International Journal of Production Research* 43(10):1943–1963.

Park, C., D. Kim, J. Han, and H. Yu. 2020. “Unsupervised Attributed Multiplex Network Embedding”. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 34, 5371–5378.

Park, J., J. Chun, S. H. Kim, Y. Kim, and J. Park. 2021. “Learning to Schedule Job-shop Problems: Representation and Policy Learning using Graph Neural Network and Reinforcement Learning”. *International Journal of Production Research* 59(11):3360–3377.

Pfund, M. E., H. Balasubramanian, J. W. Fowler, S. J. Mason, and O. Rose. 2008. “A Multi-criteria Approach for Scheduling Semiconductor Wafer Fabrication Facilities”. *Journal of Scheduling* 11(1):29–47.

Smith, L. N. 2017. “Cyclical Learning Rates for Training Neural Networks”. In *IEEE Winter Conference on Applications of Computer Vision*, 464–472. IEEE.

Taillard, E. 1993. “Benchmarks for Basic Scheduling Problems”. *European Journal of Operational Research* 64(2):278–285.

Tanev, I. T., T. Uozumi, and Y. Morotome. 2004. “Hybrid Evolutionary Algorithm-based Real-world Flexible Job Shop Scheduling Problem: Application Service Provider Approach”. *Applied Soft Computing* 5(1):87–100.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. “Attention is All You Need”. *Advances in Neural Information Processing Systems* 30.

Wang, Y., H.-S. Ma, J.-H. Yang, and K.-S. Wang. 2017. “Industry 4.0: A Way from Mass Customization to Mass Personalization Production”. *Advances in Manufacturing* 5(4):311–320.

Yang, S., Z. Xu, and J. Wang. 2021. “Intelligent Decision-making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning”. *Sensors* 21(3).

Zhang, C., W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi. 2020. “Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning”. *Advances in Neural Information Processing Systems* 33:1621–1632.

Zhou, J. 1996. “A Constraint Program for Solving the Job-shop Problem”. In *International Conference on Principles and Practice of Constraint Programming*, 510–524. Springer.

AUTHOR BIOGRAPHIES

JE-HUN LEE is a Ph.D. student in Department of Industrial & Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST). He received a MS in industrial engineering from Sungkyunkwan University. He is interested in scheduling methodologies and applications and operations management. His e-mail address is jehun.lee@kaist.ac.kr. His website is https://sites.google.com/view/jehun-lee/.

HYUN-JUNG KIM is an Associate Professor with the Department of Industrial & Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST). She received B.S., M.S., and Ph.D. in industrial and systems engineering from KAIST. Her research interests include discrete event systems modeling, scheduling, and control. Her email address is hyunjungkim@kaist.ac.kr. Her website is https://msslab.kaist.ac.kr.