GENERATING DISPATCHING RULES FOR THE INTERRUPTING SWAP-ALLOWED BLOCKING JOB SHOP PROBLEM USING GRAPH NEURAL NETWORK AND REINFORCEMENT LEARNING

Vivian Wen Hui Wong
Department of Civil and
Environmental Engineering
Stanford University
Stanford, CA, USA

Sang Hun Kim
Samsung Electronics Co., Ltd.
Republic of Korea

Junyoung Park
Department of Industrial and
Systems Engineering
Korea Advanced Institute of
Science and Technology (KAIST)
Republic of Korea

Jinkyoo Park
Department of Industrial and
Systems Engineering
Korea Advanced Institute of
Science and Technology (KAIST)
Republic of Korea

Kincho H. Law
Department of Civil and
Environmental Engineering
Stanford University
Stanford, CA, USA

ABSTRACT

The interrupting swap-allowed blocking job shop problem (ISBJSSP) is a complex scheduling problem that is able to model many manufacturing planning and logistics applications realistically by addressing both the lack of storage capacity and unforeseen production interruptions. Subjected to random disruptions due to machine malfunction or maintenance, industry production settings often choose to adopt dispatching rules to enable adaptive, real-time re-scheduling, rather than traditional methods that require costly re-computation on the new configuration every time the problem condition changes dynamically. To generate dispatching rules for the ISBJSSP problem, a method that uses graph neural networks and reinforcement learning is proposed. ISBJSSP is formulated as a Markov decision process. Using proximal policy optimization, an optimal scheduling policy is learnt from randomly generated instances. Employing a set of reported benchmark instances, we conduct a detailed experimental study on ISBJSSP instances with a range of machine shutdown probabilities to show that the scheduling policies generated can outperform or are at least as competitive as existing dispatching rules with predetermined priority. This study shows that the ISBJSSP, which requires real-time adaptive solutions, can be scheduled efficiently with the proposed machine learning method when production interruptions occur with random machine shutdowns.

Keywords: Smart Manufacturing, Job Shop Problems, Priority Dispatching Rule, Machine Learning, Reinforcement Learning, Graph Neural Networks

1. INTRODUCTION

Effective scheduling strategies to various production scheduling problems have been widely studied in academia and industry with the goal to streamline manufacturing systems and hence to improve production efficiency. For example, the classical job shop scheduling problem (JSSP), which aims to find optimal assignment of jobs composed of operations given some prescribed machine sharing and precedence constraints, is an NP-hard combinatorial optimization problem that finds many practical applications. Many manufacturing systems in the real settings, however, have more constraints to consider than the capacity of machines. For example, many components of vehicles and machines are often expensive items that are huge in size. It is therefore not desirable to have to invest in the storage of intermediate components and products [1]. The lack of storage capacity is therefore a constraint in this case. Furthermore, unforeseen interruptions to production, such as machine shutdowns, could occur that changes the list of available machines. To model modern manufacturing systems more realistically by considering both the *lack of storage capacities* and *production interruptions*, this work studies a new class of job scheduling problem, the interrupting swap-allowed blocking job shop scheduling problem (ISBJSSP). Many methods, such as mathematical optimization [2], branch-and-bound search [3] and meta-heuristic algorithms [4,5], have been developed to generate optimum or near-optimum solutions to the JSSP problems. However, these solutions are not adaptive, requiring a completely new execution when encountering a new scenario or a new configuration. These non-adaptive solutions are therefore not suitable for the ISBJSSP setting, where the problem condition constantly changes, for example, due to machine

interruptions. To cope with potential dynamic changes, priority dispatching rules (PDRs), which are simple rule-based heuristics, are the most common approach used in modern manufacturing systems, as they can be applied instantaneously to an unknown instance. PDRs, first-in-first-out (FIFO) as an example, simply loads jobs based on some predetermined priority [6]. Although PDRs are widely used in real world situations due to their simplicity and speed, their performance varies widely depending on the problem condition. For example, shortest processing time (SPT) is a common benchmarking PDR that performs well in heavily loaded or congested job shop problem instances, but fails with low load levels [7]. These simple rules, although they can deal with dynamic changes, have poor generalizability and need to be manually selected or combined based on the job shop condition. Furthermore, with the random interruptions in the ISBJSSP formulation, where problem conditions change often, it is not clear apriori whether any of the PDRs can be effective on an ISBJSSP problem.

To improve the generalizability of dispatching rules, researchers have started to leverage artificial intelligence (AI) methods to solve job shop scheduling problems. To consider both the ability to adapt and to generalize, methods that are based on reinforcement learning (RL) are receiving increasing attention in the research community on planning problems. Much like PDRs, these methods output sequential decisions according to a dispatching policy. The difference is that rather than using predetermined priority rules, RL's dispatching policy is learned by observing and accumulating experience from previous simulations.

RL has been used to learn policies in various planning and scheduling problems. Traditional RL algorithms, such as Q-learning and its variants, are often used to learn dispatching rules for small-scale job scheduling problems with discrete state space [8,9]. In contrast, large-scale job shop scheduling problems are relatively unexplored. For large-scale, continuous state space problems, it is necessary to consider deep RL methods that approximate the value function.

Sparked by increased availability in computational power in recent years, deep RL methods combining deep neural networks with RL have received much attention due to its powerful ability to generate solutions for continuous state space. Examples of deep RL applications to planning and scheduling problems include task scheduling in computing [10], robotic scheduling for manufacturing [11], and semiconductor manufacturing scheduling [12]. However, the problem formulation of deep RL for job shop problems varies widely, for even the classical JSSP. Liu et al. [13] used process time matrices to represent the state space and trained a deep RL model to select from a list of existing PDRs. Park et al. [14] and Zhang et al. [15] use disjunctive graph and graph representation learning to obtain a vectorized state space, and directly learned new dispatching rules. For ISBJSSP, there lacks a formal Markov Decision Process formulation that enables the study of deep RL approach for this new class of job scheduling problem.

This paper introduces a GNN-RL scheduler, combining graph neural network (GNN) and deep RL learning for ISBJSSP

that considers dynamic interruptions to machine availability in job shop scheduling. The formulation of the ISBJSSP as a dynamic disjunctive graph and a Markov Decision Process (MDP) is formally presented. To generate training data sets and experimental test scenarios, we implement an ISBJSSP simulator, building upon a python-based JSSP simulator (pyjssp) previously developed by Park et al. [14]. The simulator is designed to simulate ISBJSSP instances to include blocking constraints, swapping conditions, and machine shutdown interruptions to mimic realistic concerns in practical applications. Using the simulator, GNN-RL models are trained with randomly generated ISBJSSP instances. In this study, the performance of the trained models are evaluated on two sets of benchmark scenarios. The first test set is a benchmark with 10×10 instances that are commonly used in job shop scheduling studies [16]. To demonstrate the scalability and generalization of the GNN-RL models, the second test set includes job shop instances with varying sizes [17]. The experimental results show that GNN-RL schedulers can be used to schedule unknown ISBJSSP instances robustly and efficiently and can potentially be applied in a real manufacturing environment without shutting down the entire job shop when interruption occurs.

The paper is organized as follows: Section 2 introduces the ISBJSSP formulation and how the problem can be modeled as a disjunctive graph and a Markov Decision Process. Section 3 describes the methodology employed to learn ISBJSSP scheduling models. Section 4 describes the experimental results obtained with benchmark ISBJSSP instances of different sizes. Section 5 concludes this paper with a brief summary and discussion.

2. PROBLEM FORMULATION

This section briefly introduces the background for the job shop problem and the various constraints that exist. The modeling of ISBJSSP as a disjunctive graph and a Markov Decision Process (MDP) is then described.

2.1 Job Shop Scheduling

For the classical JSSP of size $m \times n$, there exists a set of n jobs, $O: \{O_1, O_2, ..., O_n\}$, to be optimally allocated and executed on a set of m machines, $M: \{M_1, M_2, ..., M_m\}$. Each job has a series of tasks or operations that must be processed according to the problem's precedence and machine-sharing constraints. In this study, without loss of generality, we assume each job, $O_j: \{o_{1j}, o_{2j}, ..., o_{pj}\}$, has the same number of p operations. Each operation o_{ij} has a pre-defined processing time for completion. The precedence constraint implies that for all consecutive operations o_{ij} and $o_{i+1,j}$ of job O_j, o_{ij} must be completed before starting $o_{i+1,j}$. Furthermore, the machine-sharing constraint indicates that each operation o_{ij} must be processed uninterrupted on a dedicated machine. Additionally, each operation of a job is assigned on a different machine, and each machine that is in process of an operation is only freed when that operation finishes. Therefore, given the above constraints,
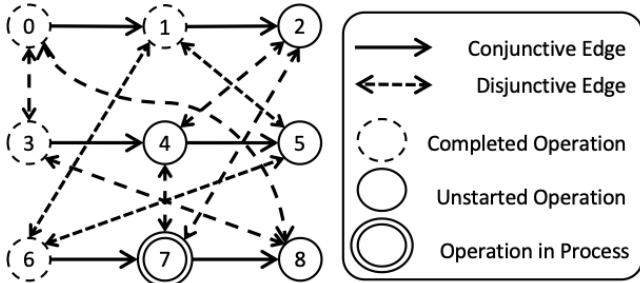
FIGURE 1: Example of a disjunctive graph for an instance containing three jobs, each with three operations. Directed conjunctive edges represent precedence constraints. Bidirectional disjunctive edges represent machine constraints, where the nodes in a cycle are operations that require to be processed on the same machine. Nodes with dashed perimeters indicate completed operations. Nodes with solid perimeters indicate operations that have not been started. The double-outlined node indicates the operation currently being processed.

the number of machines $m = p$, the number of operations for each job. The objective of the classical JSSP is to find a schedule that minimizes the makespan, the total time to finish all jobs [18].

The classical job shop problem assumes that there is sufficient storage or buffer space available to store each job in between consecutive operations. However, buffers are undesirable in practical applications. Therefore, many real manufacturing applications are better modeled as the Blocking JSSP (BJSSP) [19]. BJSSP introduces the blocking constraint in that no buffers are available for storing a job as the job moves between machines; the job must wait on the machine until it can be processed on the next machine. That is, for any job O_j, its operation o_{ij} is a blocking operation until o_{ij}'s succeeding operation, $o_{i+1,j}$, starts.

In practical job shops without buffers, parts that are blocking idling machines can be swapped to avoid a deadlock situation. A deadlock situation occurs when no unprocessed operations could be processed, because each unprocessed operation is waiting for a blocked machine to become "unblocked" and available. Thus, BJSSP often allows swapping to avoid deadlocks, referred to as the swap-allowed blocking job shop problem (SBJSSP) [16]. A swap can be done if there exists a set of blocking operations, each one waiting for a machine blocked by another operation in the set. A swap resolves the deadlock and ensures that the manufacturing process can proceed and that there exists at least one solution to a randomly generated SBJSSP instance.

2.2 ISBJSSP

Although the SBJSSP models can be applied to many manufacturing production lines, there is an additional factor that exists in a real production line but is often overlooked - the possibility of production interruption, for example, caused by machine failures. While solution methods such as mathematical programming, branch and bound search and meta-heuristic methods (such as tabu search, genetic algorithm, simulated annealing, etc.) can generate optimal solution to static (uninterrupted) job shop scenarios, the dynamic scenarios with real time machine interruptions would require recomputing a

new solution for each scenario change by the methods. The possibility of such interruption also results in priority dispatching rules [20] being generally favored in practice, as dispatching rules can easily adapt to dynamic changes in availability of machines in real time. Our work includes this additional constraint where machine availability can be interrupted in the formulation: At any given time step, an idling machine in M has a probability $P_{interrupt}$ that the machine is unavailable to process any job for a period of $T_{interrupt}$ time steps. If a job's next operation is waiting on an unavailable machine, the job will block the machine used by the precedent operation due to the lack of buffer. When the shutdown machine becomes available again after $T_{interrupt}$ time steps, the machine will then process one of the waiting jobs, determined by the job shop scheduler. We refer the job shop problem with the interruption constraint as the interrupting swap-allowed blocking job shop problem (ISBJSSP).

2.3 Dynamic Disjunctive Graph Formulation

The ISBJSSP can be represented by a dynamic disjunctive graph $G = (V, C \cup D)$ [21]. Here, V are the nodes, each corresponding to an operation o_{ij} of job O_j. C is the set of conjunctive edges, where each edge connects two consecutive operations o_{ij} and $o_{i+1,j}$ of job O_j. The conjunctive edges represent the set of processing order constraints. D denotes the set of disjunctive edges, which connect any two vertices if the two corresponding operations need to be processed on the same machine. The disjunctive edges represent the machine-sharing constraints. Nodes and edges of the disjunctive graph, however, are subjected to deletions and additions due to the machine availability constraint, making the disjunctive graph dynamic. More specifically, when a machine is shutdown, the nodes (i.e., the operations) that need to be processed on the machine and their connected edges are temporarily removed from the graph, indicating that the machine is no longer observed at that time instance. When the machine becomes available after $T_{interrupt}$ time steps, the previously removed nodes and edges are added back to the graph.

Figure 1 shows a disjunctive graph of a small example instance. The instance contains three machines, on which three jobs, each with three operations, are to be processed. As an example, the first job contains the operations labeled with node numbers 0, 1, 2, which must be processed in this specified order due to the existence of precedence constraints. The precedence constraint is shown as directed edges in the disjunctive graph. Similarly, the second job must be processed in the order of 3, 4, 5, and the third job in the order of 6, 7, 8. The bi-directional disjunctive edges specify machine constraints. In our example, operations 1, 5, 6 need to be processed on a dedicated machine. Similarly, operations 0, 3, 8 share a machine, and operations 2, 4, 7 share a machine. At the time where this disjunctive graph was plotted as shown in Figure 1, operations 0, 1, 3, 6 are completed, and operation 7 is being processed. Furthermore, there is swap-allowed blocking to consider. For example, even when operation 7 is completed, it will block its machine,
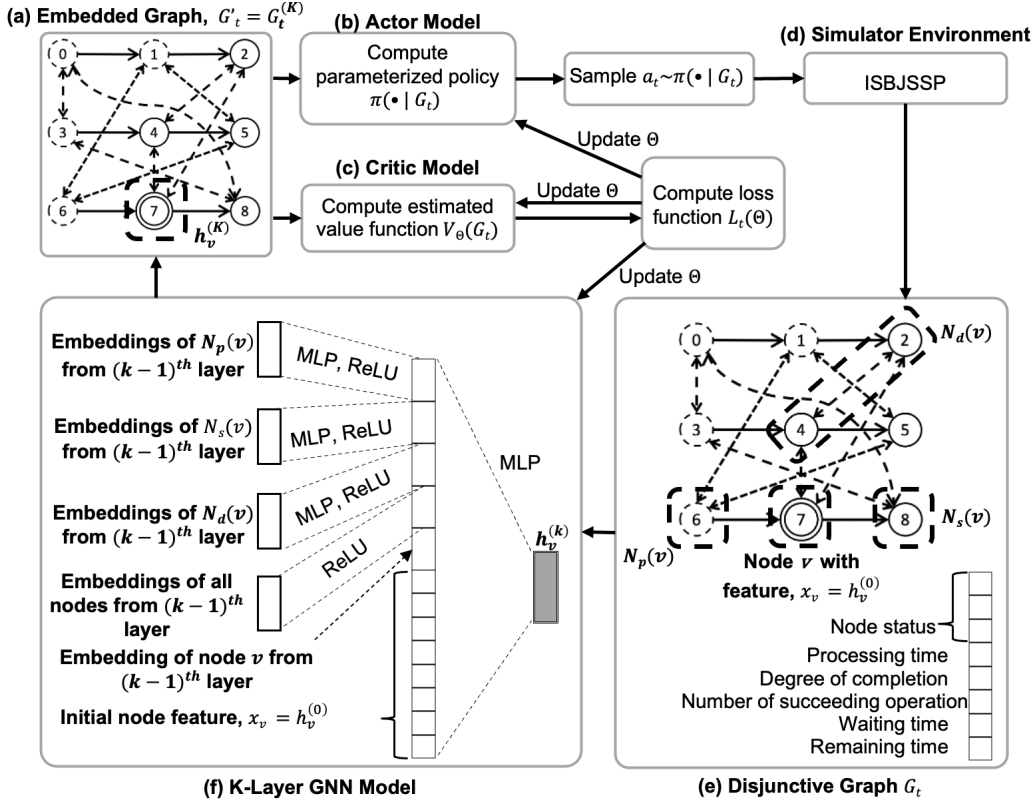
FIGURE 2: Proposed GNN-RL framework

preventing operations 2 and 4 to be processed, until the part moves to the next machine to commence operation 8. At the current time step, all three machines are either blocked by an unstarted operation or busy processing an operation, and are therefore not idle. As defined earlier, the machine shutdown interruption of probability $P_{interrupt}$ only occurs to idling machines.

To incorporate the time-dependent job shop information into the disjunctive graph, we assign a node feature vector x_v to each node $v = o_{ij} \in V$. The node features are stacked vectors with the following components:

- Node status: a one-hot index vector of size 3, indicating whether the operation v is not yet started, being processed, or is completed.
- Processing time: the total time required to finish operation v.
- Degree of completion: the ratio of the accumulated processing time of v's job to the total processing time of v's job (i.e., the job that contains the operation).
- Number of succeeding operations: the number of operations including both v and the operations after v in v's job.
- Waiting time: the time for which v must wait for processing after it is ready to be processed.
- Remaining time: the remaining processing time needed to complete the operation v once it has started.

Since node features x_v are time-dependent, the resulting graph is now dynamic and will be denoted as G_t hereon to represent the disjunctive graph of an ISBJSSP instance at time t.

2.4 Markov Decision Process Formulation

The scheduling process of an ISBJSSP instance can be viewed as a sequential decision-making process. Specifically, ISBJSSP can be formulated as an MDP, denoted as a (S, A, P, R, γ) tuple, whose elements represent the set of Markov states (S), set of actions (A), transition model (P), reward function (R) and discount factor (γ), respectively.

- State: A disjunctive graph G_t representing a snapshot of state $s_t \in S$ of the ISBJSSP instance at time t.
- Action: A scheduling action $a_t \in A$ of loading an operation to an available machine at time t.
- Transition model: The transition between states, which, in this study, is handled and generated by the job shop simulator.
- Reward function: A function defined to stipulate the behavior of an action. The reward function used in this study mimics the utilization of a machine and is defined as

$$r_t = -n_{w_t} \tag{1}$$

where n_{w_t} is the number of jobs waiting at time t.
- Discount factor: The discount factor for "caring" of future reward by an action.

3. METHODOLOGY

This section describes a machine learning approach for deriving a policy to solve the ISBJSSP. The method consists of two parts, graph neural network (GNN) and reinforcement learning (RL). Figure 2 depicts the overarching framework of the proposed GNN-RL approach.

A disjunctive graph G_t (Figure 2(e)) is observed from the ISBJSSP simulator environment (Figure 2(d)) and is used as the input to a GNN model (Figure 2(f)) for representation learning. The learned embedded graph G_t' (Figure 2(a)) is then used as the input to the RL algorithm, learning a parameterized policy, or a probability distribution of feasible actions, using an actor-critic model with proximal policy optimization (Figure 2(b,c)). Finally, an action to process a specific operation is sampled from the parameterized policy $\pi(\cdot|G_t)$ and executed via the ISBJSSP simulator (Figure 2(d)).

3.1 Representation Learning with GNN

The process of obtaining an embedded graph using GNN can be thought of as learning an embedding vector for each node v that represents the necessary neighborhood structure and node feature information around the node. The embedded graph in Figure 2(a) can be learned from a GNN model. A GNN is a neural network that consists of layers of differentiable functions with learnable parameters and computes an embedding vector for each node in the graph. A GNN layer needs to be designed such that for each target node, the embedding of the target node is updated not only using the previous layer's target node embedding, but also node embedding aggregated from the neighboring nodes in order to represent the structure information of the disjunctive graph, G_t, in the learned embeddings.

As shown in Figure 2(f), the computation process of a GNN layer implemented in this study can be separated into three steps. Firstly, a different multi-layer perceptron (MLP) network [22] with ReLU activation [23] is applied to each of the following three sets of nodes neighboring the target node v: the set of all precedent nodes $N_p(v)$ connected through the conjunctive (precedence constraint) edges, succeeding nodes $N_s(v)$ connected also through the conjunctive edges and disjunctive nodes $N_d(v)$ connected through the (bidirectional) disjunctive (machine-sharing constraint) edges. Secondly, the vector outputs of the three MLP networks, an aggregated representation of the overall graph, the node embedding updated in the previous layer, and the initial node feature of the target node are stacked in a vector, which, as the last step, is passed through another MLP network without activation. Mathematically, the operations of the kth layer of a GNN can be written as:

$$
\begin{aligned}
h_v^{(k)} = f_n^{(k)}(\, &\text{ReLU}(f_p^{(k)}(\sum_{i \in N_p(v)} h_i^{(k-1)})) \, || \\
&\text{ReLU}(f_s^{(k)}(\sum_{i \in N_s(v)} h_i^{(k-1)})) \, || \\
&\text{ReLU}(f_d^{(k)}(\sum_{i \in N_d(v)} h_i^{(k-1)})) \, || \qquad (2) \\
&\text{ReLU}(\sum_{i \in V} h_i^{(k-1)}) \, || \\
&h_v^{(k-1)} \, || \\
&h_v^{(0)})
\end{aligned}
$$

where $\text{ReLU}(\cdot) = \max(0,\cdot)$ is a non-linear activation function. The previously mentioned MLP networks are denoted by $f_p, f_s,$ and f_d, each computes a vector from node embeddings in the neighborhood of v (i.e., $N_p(v), N_s(v),$ and $N_d(v)$, respectively). V is the set of all nodes of the graph $G_t = (V, C \cup D)$. $h_v^{(0)}$ is the feature vector x_v of node v. $||$ is the vector concatenation operator.

After K GNN layers, we have computed an embedded graph $G_t^{(K)}$, as shown in Figure 2(a), whose node features are now the updated embedding vectors $h_v^{(K)} \, \forall \, v \in V$, from the input disjunctive graph G_t in Figure 2(e) with initial node features $h_v^{(0)}$.

3.2 Dispatching Policy Learning with RL

The graph embedding $G_t^{(K)}$ outputted from GNN described in the previous section is used as the input for the RL algorithm. More specifically, an actor-critic method is used [24]. As the name suggests, there are two neural networks in the RL process: an actor and a critic. As shown in Figure 2(b), the actor $\pi(a_t^v|G_t^{(K)})$ maps the embedded graph $G_t^{(K)}$ to the probability distribution over the set of all available actions, or the set of processible nodes. The actor model, used to compute the parameterized policy, is structured like a softmax function computing the probability of performing action a_t^v for the current state $G_t^{(K)}$ as follows:

$$
\pi(a_t^v|G_t^{(K)}) = \frac{\exp(f_l(h_v^{(K)}))}{\sum_{u \in A_{G_t}} \exp(f_l(h_u^{(K)}))} \qquad (3)
$$

where a_t^v denotes the action of selecting operation (node) v to process and v is a processible node in the disjunctive graph's node set V. Following the same notation in the previous section, $h_v^{(K)}$ is the embedded vector of node v, and $f_l \, (l = p, s, d)$ denotes a MLP network. A_{G_t} represents the set of available actions for the disjunctive graph G_t.

The critic model, as depicted in Figure 2(c), is another network that learns the value function to reliably optimize the policy. The current study approximates the critic function as

$$
V^\pi(G_t^{(K)}) \approx f_v(\sum_{i \in V} h_i^{(K)}) \qquad (4)
$$

where f_v is an MLP network, and $\sum_{i \in V} h_i^{(K)}$ returns a summation of all node embeddings.

It can be observed that the policy $\pi(a_t^v|G_t^{(K)})$ learned by the actor is now parameterized by a set of parameters $\Theta = \{\theta_p, \theta_s, \theta_d, \theta_n, \theta_l, \theta_v\}$, corresponding to the MLP networks f_p, f_s, f_d, f_n, f_l and f_v. The parameters can be iteratively updated via gradient ascent. In each training iteration, we use the policy $\pi_{\Theta_{old}}$ with the current "old" parameters Θ_{old} to interact with the job shop simulator (Figure 2(d)) and collect transition samples. The parameters Θ are updated to optimize the policy:

$$
\Theta = \Theta_{old} + \eta \nabla_\Theta L(\Theta) \qquad (5)
$$

where η is the learning rate and $L(\Theta)$ denotes an objective function to be optimized for an optimal policy.

In this work, proximal policy optimization (PPO) is employed to optimize the policy. To prevent unstable training due to substantial policy changes and encourage exploration

during training, Schulman et.al. [24] proposes an objective function $L(\Theta)$ to be optimized at each time step t as follows:

$$L_t(\Theta) = \mathbb{E}[L_t^{CLIP}(\Theta) - \alpha L_t^{VF}(\Theta) + \beta E_t(\pi_\Theta)] \quad (6)$$

where α and β are parameters for the objective function. $L_t^{CLIP}(\Theta), L_t^{VF}(\Theta)$, and $E_t(\pi_\Theta)$ are, respectively, a clipped-surrogate function, square-error value function loss, and an entropy bonus given as follows [20].

1. The clipped-surrogate function is defined as

$$L_t^{CLIP}(\Theta) = \mathbb{E}\left[\min(\rho_t \mathcal{A}_t, clip(\rho_t, 1-\epsilon, 1+\epsilon)\mathcal{A}_t)\right] \quad (7)$$

where ρ_t denotes a probability ratio of the current and old policies as

$$\rho_t = \frac{\pi_\Theta\left(a_t \middle| G_t^{(K)}\right)}{\pi_{\Theta_{old}}\left(a_t \middle| G_t^{(K)}\right)} \quad (8)$$

and the estimator of the advantage function $\mathcal{A}_t$ at time step t is computed as

$$\mathcal{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \quad (9)$$

$$\text{and } \delta_t = \rho_t + \gamma V_\Theta\left(G_{t+1}^{(K)}\right) - V_\Theta\left(G_t^{(K)}\right) \quad (10)$$

The coefficients γ and λ are, respectively, the discount factor and the parameter for the advantage function estimator. The clip operation ensures that ρ_t does not move outside the interval $[1-\epsilon, 1+\epsilon]$, thereby preventing substantial changes in policy.

2. The square-error value function loss is given as:

$$L_t^{VF}(\Theta) = \left(V_\Theta\left(G_t^{(K)}\right) - V_t^{target}\right)^2 \quad (11)$$

where $V_t^{target} = \sum_{i=t}^{T} r_i$ denotes the sum of rewards.

3. The entropy bonus term for the current policy $\pi_\Theta(a)$ is introduced to ensure sufficient exploration and is defined as

$$E_t(\pi_\Theta) = -\sum_a \log\left(\pi_\Theta(a)\right)\pi_\Theta(a) \quad (12)$$

where a is an action in the current embedded graph $G_t^{(K)}$. The PPO procedure maximizes the objective function $L(\Theta)$ by updating the parameters Θ following the gradient direction $\nabla_\Theta L(\Theta)$. Further discussion of the PPO algorithms can be found in References [24] and [14].

4. EXPERIMENTAL RESULTS

This section describes the results of the experiments on a number of benchmark instances to evaluate the schedulers trained using GNN-RL. We will first describe the benchmark instances and details of the schedulers used in the experiments. We then report the experimental results, including: (1) a performance comparison of the GNN-RL method with other dispatching rules for the standard SBJSSP (a special case of the ISBJSSP with $P_{interrupt} = 0$); (2) a performance comparison demonstrating the practicability of the above methods for the ISBJSSP, which is subjected to random machine interruptions with probability $P_{interrupt} > 0$; and (3) a demonstration of the GNN-RL method's ability to generalize a model trained with instances of a specific size to handle ISBJSSP instances of different sizes.

4.1 The Baseline Benchmark Problem Instances

As a baseline to evaluate and compare the GNN-RL methodology with the PDR methods, a set of 18 job shop scheduling problem instances, each being 10×10 in size (consisting of 10 machines and 10 jobs), are employed. Each job involves 10 operations and, depending on the benchmark instance, the operations may have different machine processing time. The 18 instances are commonly used as benchmarks for job shop scheduling [16]. Even though this study focuses on the ISBJSSP where machine interruptions may occur, the benchmark instances serve as a fair metric for evaluating scheduling efficacy between the GNN-RL method and the PDR methods.

4.2 Scheduler Models and Configurations

Priority Dispatching Rules (PDRs). As mentioned before, PDRs [20] are the most common approaches employed in practice for generating immediate solutions for scheduling job shops with unseen instances. We therefore compare the makespans obtained by the GNN-RL schedulers with those obtained using the following PDRs for prioritizing the preference for job execution:

- Most Total Work Remaining (MTWR): the job that has the greatest number of remaining operations
- Least Total Work Remaining (LTWR): the job that has the fewest number of remaining operations
- Shortest Processing Time (SPT): the job whose next operation has the shortest processing time
- Longest Processing Time (LPT): the job whose next operation has the longest processing time
- First In First Out (FIFO): the first job that arrives
- Last In First Out (LIFO): the last job that arrives
- Shortest Queue Next Operation (SQNO): the job whose next operation requires a machine that has the fewest number of jobs waiting
- Longest Queue Next Operation (LQNO): the job whose next operation requires a machine that has the most number of jobs waiting
- Shortest Total Processing Time (STPT): the job with the shortest total processing time
- Longest Total Processing Time (LTPT): the job with the longest total processing time
- Random: the job that is randomly selected from the set of all doable jobs

The PDRs can be applied irrespective of whether machine interruptions occur during the job operations.

GNN-RL Schedulers. Initially targeted for the baseline benchmark instances of 10×10 in size, a random ISBJSSP instance of size $m \sim \mathcal{U}(5,9) \times n \sim \mathcal{U}(m,9)$ and operation processing times from $\mathcal{U}(1,99)$, where $\mathcal{U}$ denotes a uniform distribution, is generated using the ISBJSSP simulator for training the GNN-RL models. The order of machines that each job visits is randomly permuted. After 20 episodes of training on the instance, a new ISBJSSP instance, once again with size $m \sim \mathcal{U}(5,9) \times n \sim \mathcal{U}(m,9)$, processing times from $\mathcal{U}(1,99)$ and randomly permuted machine order, is generated every 100 iterations. Note that, as discussed in a latter section, even though training is conducted on small-size instances to limit computational time and demand, the scheduling strategy that the

Algorithm 1 Training procedure for GNN-RL scheduler

Generate a random ISBJSSP instance as starting state G_0;
Initialize parameters Θ and the parameterized policy π_Θ;
Initialize iteration = 0;
repeat
  iteration += 1;
  for episode=1,2,…, T **do**
    Observe and collect transition sample $(G_{t-1}, a_{t-1}, r_{t-1}, G_t)$;
    Execute action $a_t \sim \pi_\Theta(\cdot \,|\, G_t^{(K)})$ to assign operations to available machines;
  end for
  Update parameters Θ with gradient ascent to maximize $L_t(\Theta)$, calculated from the collected transition samples;
  if iteration = 100 **then**
    Generate a new random ISBJSSP as starting state G_0;
    Reset iteration = 0;
  end if
until Validation performance has converged.

---

model learns can be transferred to solve instances of other sizes effectively. Algorithm 1 outlines the procedure to train a GNN-RL scheduler.

An Adam optimizer [25] with a learning rate (η) of 2.5×10^{-4} is used. Two different discount factors (γ) are used in training, which are 0.9 and 1.0 (no discount) respectively. We use a GNN with $K = 3$ layers to obtain the graph embeddings. The MLP networks, namely f_p, f_s, f_d, f_n, f_l and f_v, each consists of two hidden layers with 256 ReLU activation units. f_p, f_s and f_d have 8-dimensional inputs and outputs. f_n has 48-dimensional input and 8-dimensional output. f_l and f_v have 8-dimensional inputs and scalar outputs. For the PPO hyperparameters, we set $\lambda = 0.95$, $\epsilon = 0.2$, $\alpha = 0.5$, and $\beta = 0.01$, which are the same as proposed in [14]. For comparison purpose, models are trained without interruptions and with the possibility of machine interruptions. When trained on SBJSSP (without interruptions), we set the probability of interruption $P_{interrupt} = 0$. The trained GNN_RL models are used for a baseline comparison with the PDR methods. For models that are trained on ISBJSSP instances, we train a different model for each $P_{interrupt}$ value, ranging from 1% to 20%. The trained GNN-RL models for the ISBJSSP are then used to compare with the SPT (shortest process time) priority rule (which achieves the shortest makespan among the PDRs for the non-interrupting SBJSSP) for the cases with machine interruptions. All experiments are conducted on a machine equipped with an Intel Core i7-7820X processor.

4.3 Results on Non-interrupting SBJSSP

The goal of the job shop problem is to minimize the makespan, which is employed here as the evaluation criterion for comparing the performances of the different SBJSSP schedulers. Two GNN-RL schedulers, namely GNN-RL (1) with $\gamma = 0.9$ and GNN-RL (2) with $\gamma = 1.0$, are trained. Figure 3 shows the results of the two GNN-RL schedulers and the makespans obtained using the PDR schedulers for the 18 benchmark
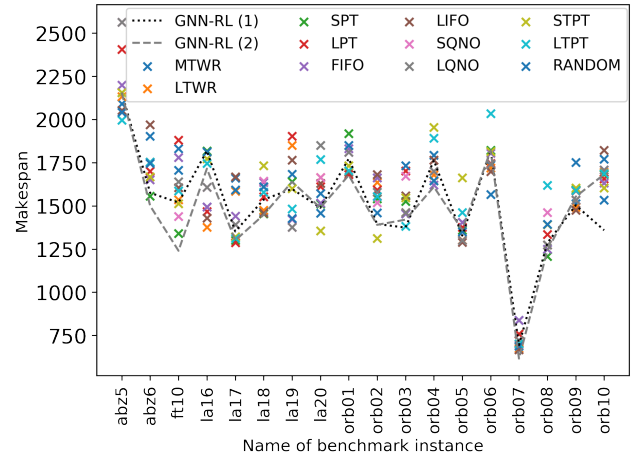

FIGURE 3: Makespans obtained using the two trained GNN-RL schedulers and the PDRs on the 18 benchmark instances without machine interruptions.

TABLE 1: Total makespan computed by the GNN-RL and PDR schedulers for the 18 benchmark instances, without machine interruptions

Scheduler name	Discount factor	Total makespan
GNN-RL (1)	0.9	27337
GNN-RL (2)	1.0	**26856**
MTWR	-	28687
LTWR	-	27996
SPT	-	27827
LPT	-	29009
FIFO	-	28169
LIFO	-	28497
SQNO	-	28563
LQNO	-	28818
STPT	-	28439
LTPT	-	28845
RANDOM	-	28988

instances. Among the PDRs, the best scheduler appears to be problem dependent. As shown in Figure 3, on most of the benchmark instances, at least one of the GNN-RL schedulers is able to outperform or as competitive as the PDR schedulers, assuming no machine interruptions occur.

Table 1 reports the sum of the makespans of all problem instances. As shown, on average over all the benchmark instances, the GNN-RL schedulers produce shorter makespans than those by the PDRs schedulers. The GNN-RL(2) scheduler with $\gamma = 1.0$ has the best average performance. Among the PDR schedulers, the SPT strategy, which prioritizes the jobs according to the next operation having the shortest processing time, appears to perform the best on the average.

4.4 Real-time Adaptive Scheduling of the ISBJSSP for the Baseline Benchmark

In practice, unforeseen interruptions could occur during production. For example, machines in a production line can misbehave unexpectedly at times that require a shutdown. To
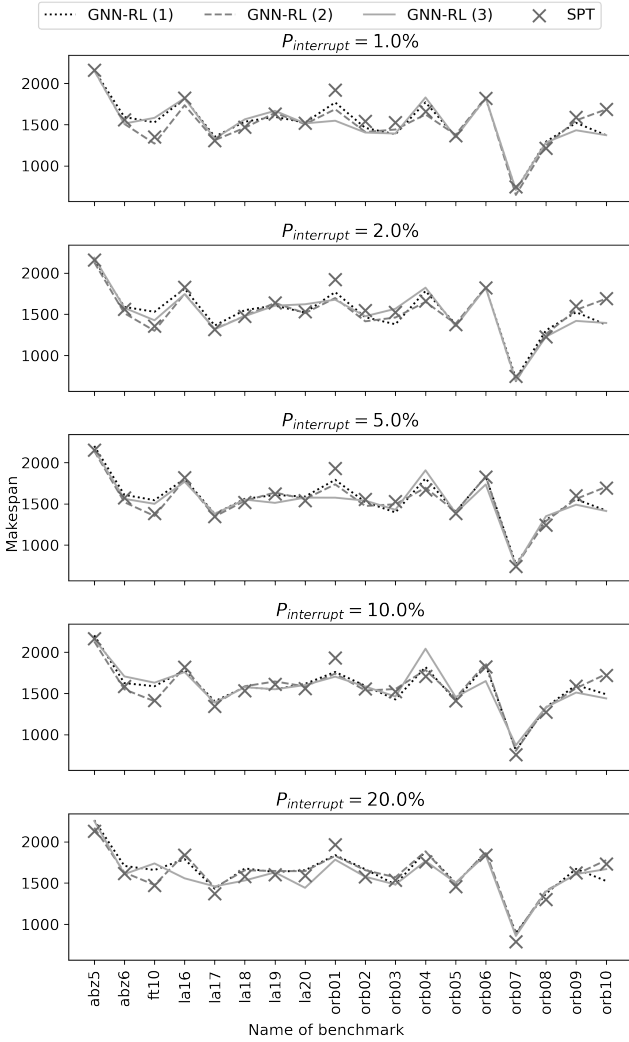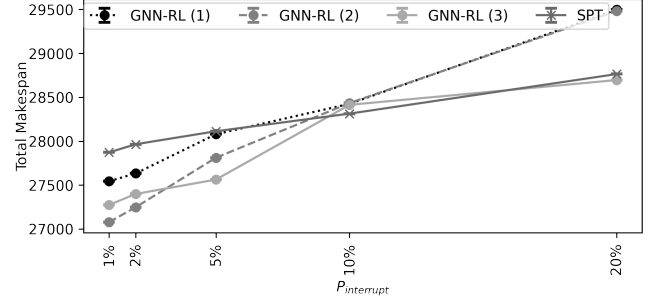
FIGURE 4: Mean of makespans obtained using the GNN-RL and the SPT schedulers on the 18 baseline benchmark instances.

$P_{interrupt}$		Total Makespan				
		1%	2%	5%	10%	20%
GNN-RL (1)	Mean	27543	27633	28079	28425	29496
	Std	199.2	312.9	415.8	451.8	561.2
GNN-RL (2)	Mean	**27077**	**27248**	27811	28433	29481
	Std	188.5	247.7	357.8	486.2	500.6
GNN-RL (3)	Mean	27274	27398	**27562**	28414	28696
	Std	174.5	242.4	363.4	493.5	509.7
SPT	Mean	27872	27965	28113	**28313**	28764
	Std	**142.4**	**177.5**	**239.6**	339.9	**332.4**

FIGURE 5: Total makespans of the GNN-RL and the SPT schedulers for different probabilities of interruption.

assess whether the GNN-RL method can cope with real-time changes, we simulate the scenarios where at any given time step, each idling machine, excluding those in the middle of processing a job, has a certain probability of failing or shutdown, denoted as $P_{interrupt}$, for a duration of 50 time steps. During a simulated machine failure, no further job can be assigned to the machine. More specifically, a machine being shutdown is equivalent to removing the nodes utilizing that machine from the disjunctive graph representation for 50 time steps according to our problem formulation. The schedulers have no prior knowledge on the probability and the down time of the machines.

Three GNN-RL models are trained. They include the same two models, namely GNN-RL (1) and GNN-RL (2) trained without machine interruptions, as described in Section 4.3. The third model, GNN-RL (3), has the same hyperparameters as GNN-RL (2), but is trained with the same probability of interruption, $P_{interrupt}$, as assigned to the simulation scenario.

We perform 50 simulations for each of the benchmark instances for a number of $P_{interrupt}$ values, ranging from 1% to 20%. Among the PDR schedulers, SPT shows the best average performance for almost all the cases and is therefore employed here to compare with the GNN-RL schedulers. Figure 4 shows a comparison of the average results between the GNN-RL and the SPT schedulers. It can be seen that for most instances, the GNN-RL schedulers outperform or are as competitive as the SPT scheduler for $P_{interrupt} < 10\%$. Furthermore, the GNN-RL (2) model and the GNN-RL (3) model trained with interruptions perform consistently better than the GNN-RL (1) model. Moreover, Figure 5 plots the performance, averaged over the 18 benchmark instances, of each scheduler with respect to $P_{interrupt}$. Also shown in Figure 5 are the means and standard deviations (Std) of the scheduling results for 50 randomly generated instances. Based on the results shown in Figures 4 and 5, the following can be observed:

1. As expected, when the probability of interruption for the machines increases, the makespans produced by the schedulers for completing all the jobs increase.
2. All GNN-RL models produce more efficient makespans than the SPT scheduler when the probability of machine interruption are lower than 5%. It can be seen from Figure 5 that, with interpolation, the GNN-RL models can potentially be effective up to 8-10% probability of machine interruptions. Beyond $P_{interrupt} = 10\%$, the SPT scheduler produces more efficient makespans in this case study.
3. It is interesting to observe that, for the set of benchmark instances tested, the GNN-RL (3) model trained with the same probability of interruption assigned to the simulator performs quite competitively for almost all cases.
4. As can be seen in Figure 5, when the probability of interruption becomes high ($P_{interrupt} > 10\%$), the standard
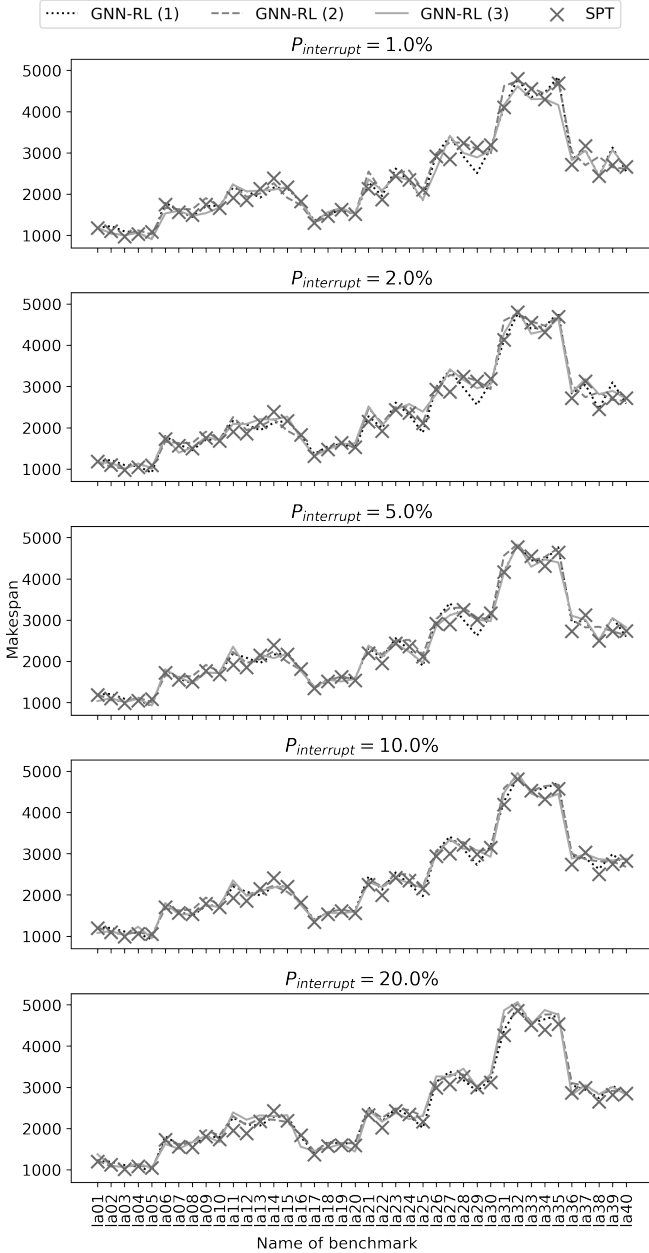
FIGURE 6: Mean of the makespans obtained on the LA benchmarks.

RL models trained previously with job shop instances of size $m \sim \mathcal{U}(5,9) \times n \sim \mathcal{U}(m,9)$ to the 40 LA benchmark instances, with a range of sizes from 10×5 to 30×10 and 15×15 [17]. Makespans are obtained for 50 simulations on each of benchmark instances. Figure 6 shows the makespans computed with the SPT scheduler, the GNN-RL (1) and (2) models trained without interruptions and the GNN-RL (3) models trained with interruptions. In general, especially for cases with $P_{interrupt} < 10\%$, on average, the GNN-RL schedulers perform better or at a similar level comparable to the SPT scheduler.

In summary, this experimental study with benchmark instances of different sizes shows that the GNN-RL methods remain robust for production scenarios with different job shop sizes even though the models are trained originally with the baseline benchmark of different size.

5. SUMMARY AND DISCUSSION

The ability to assign jobs to machines under possible changes of operational conditions is important in practice. This study shows that GNN and RL can be a viable approach for solving the ISBJSSP, a complex and computational demanding problem subjected to unforeseen changes to the problem condition. We implemented a simulator to generate ISBJSSP instances for training and to validate the GNN-RL models for real-time scheduling of the ISBJSSP. For the simulations with no machine interruptions, the dispatching rule generated by the best trained GNN-RL scheduler achieves the best overall makespan exceeding that of the PDRs. (It should be noted that under perfect job shop conditions, mathematical optimization can produce more superior schedules [16].) As the key objective of this study, we simulate scenarios where machines in the job shop can possibly be interrupted and shut down temporarily. The results show that the GNN-RL trained schedulers are robust under interruptions and outperform the PDRs approaches when the probability of machine interruption is low (less than 10% in the examples). In practice, it is very unlikely that the job shop would remain operational when the machines are deemed to have a high probability for being shut down. Furthermore, with an emphasis on robustness and practicality, our experimental study shows that the GNN-RL method is able to generalize to different job shop sizes subjected to a range of interruption probabilities. Given the speed of outputting actions and its decent performance, the GNN-RL method represents a viable approach applicable to real manufacturing problems that can be closely modeled as a ISBJSSP. While our research utilizes random simulations, domain-specific knowledge should be strategically incorporated in the real production environment to, for example, build specialized reward functions and fine-tune hyperparameters.

Future work could focus on developing methodologies for fine-tuning the parameters of the GNN and RL models in order to further improve the scheduling results. Empirical studies of other learning, adaptive algorithms could be explored. Additional experiments could be conducted to demonstrate on machine interruptions in real world job shop environments. Finally, further investigation on the GNN-RL method may

deviations for the GNN-RL schedulers are higher than the SPT scheduler. The higher standard deviation is probably due to the increase in uncertainties on machine interruptions that affect the predictability of the trained GNN-RL models. In summary, based on the experimentation on the 18 benchmark instances, the GNN-RL schedulers are shown to be robust for the scenarios where the probability of interruptions for each machine is less than 10%, even when the GNN-RL model is trained based on the scenarios with no machine interruptions.

4.5 Scheduling ISBJSSP Instances of Difference Sizes

To assess the scalability and generalization of GNN-RL models to instances of different sizes, we apply the same GNN-

include other domain-specific constraints, such as limited buffer capacity, queue time constraints and multi-line scheduling that are commonly encountered in semiconductor manufacturing.

ACKNOWLEDGEMENTS

REFERENCES

[1] Lange, J., 2019, "Solution Techniques for the Blocking Job Shop Scheduling Problem with Total Tardiness Minimization," PhD Thesis, Otto-von-Guericke-Universität Magdeburg, Fakultät für Mathematik.

[2] Manne, A. S., 1960, "On the Job-Shop Scheduling Problem," Operations Research, **8**(2), pp. 159-294.

[3] Lomnicki, Z. A., 1965, "A 'Branch-and-Bound' Algorithm for the Exact Solution of the Three-Machine Scheduling Problem," Operation Research Quarterly, **16**(1), pp. 89–100.

[4] Gröflin, H., and Klinkert, A., 2009, "A New Neighborhood and Tabu Search for the Blocking Job Shop," Discrete Applied Mathematics, **157**(17), pp. 3643–3655.

[5] Ahmadian, M. M., Salehipour, A., and Cheng, T. C. E., 2021, "A Meta-Heuristic to Solve the Just-in-Time Job-Shop Scheduling Problem," European Journal of Operational Research, **288**(1), pp. 14–29.

[6] Holthaus, O., and Rajendran, C., 1997, "Efficient Dispatching Rules for Scheduling in a Job Shop," International Journal of Production Economics, **48**(1), pp. 87–105.

[7] Subramaniam, V., Lee, G. K., Hong, G. S., Wong, Y. S., and Ramesh, T., 2000, "Dynamic Selection of Dispatching Rules for Job Shop Scheduling," Production Planning & Control, **11**(1), pp. 73–81.

[8] Aydin, M. E., and Öztemel, E., 2000, "Dynamic Job-Shop Scheduling Using Reinforcement Learning Agents," Robotics and Autonomous Systems, **33**(2–3), pp. 169–178.

[9] Gabel, T., and Riedmiller, M., 2008, "Adaptive Reactive Job-Shop Scheduling with Reinforcement Learning Agents," International Journal of Information Technology and Intelligent Computing, **24**(4), pp. 14–18.

[10] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., and Alizadeh, M., 2019, "Learning Scheduling Algorithms for Data Processing Clusters," SIGCOMM.

[11] Wang, Z., and Gombolay, M. C., 2020, "Learning Scheduling Policies for Multi-Robot Coordination With Graph Attention Networks," IEEE Robotics and Automation Letters, **5**(3), pp. 4509-4516.

[12] Waschneck, B., Reichstaller, A., Belzner, L., Belzner, L., Altenmuller, T., Bauernhansl, T., Knapp, A., and Kyek, A., 2018, "Deep Reinforcement Learning for Semiconductor Production Scheduling," 29th Annual SEMI Advanced Semiconductor Manufacturing Conference.

[13] Liu, C.-L., Chang, C. C., and Tseng, C. J., 2020, "Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems," IEEE Access, **8**, pp. 71752-71762.

[14] Park, J., Chun, J., Park, J., Kim, S. H., and Kim, Y., 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), pp. 3360–3377.

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

[16] Mascis, A., and Pacciarelli, D., 2002, "Job-Shop Scheduling with Blocking and No-Wait Constraints," European Journal of Operational Research, **143**(3), pp. 498–517.

[17] Lawrence, S., 1984, "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)," Graduate School of Industrial Administration, Carnegie-Mellon University.

[18] Brizuela, C., Zhao, Y., and Sannomiya, N., 2001, "No-Wait and Blocking Job-Shops: Challenging Problems for GA's," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, **4**, pp. 2349–2354.

[19] Oddi, A., Rasconi, R., Cesta, A., and Smith, S., 2012, "Iterative Improvement Algorithms for the Blocking Job Shop," ICAPS'12 - Proceedings of the 22nd International Conf. on Automated Planning and Scheduling, pp. 199–206.

[20] Blackstone, J. H., Phillips, D. T., and Hogg, G. L., 1982, "A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations," The International Journal of Production Research, **20**(1), pp. 27–45.

[21] Garey, M. R., Johnson, D. S., and Sethi, R., 1976, "The Complexity of Flowshop and Jobshop Scheduling," Mathematics of operations research, **1**(2), pp. 117–129.

[22] Haykin, S., 1994, *Neural Networks: A Comprehensive Foundation*, Prentice Hall PTR.

[23] Glorot, X., Bordes, A., and Bengio, Y., 2011, "Deep Sparse Rectifier Neural Networks.," *AISTATS*, G.J. Gordon, D.B. Dunson, and M. Dudík, eds., JMLR.org, pp. 315–323.

[24] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., 2017, "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347.

[25] Kingma, D. P., and Ba, J., 2015, "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, San Diego, CA, USA, May 7-9.*