Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

REINFORCEMENT LEARNING FOR UNRELATED PARALLEL MACHINE SCHEDULING WITH RELEASE DATES, SETUP TIMES, AND MACHINE ELIGIBILITY

Sang-Hyun Cho[1], Hyun-Jung Kim[1], and Lars Mönch[2]

[1]Dept. of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea
[2]Dept. of Mathematics and Computer Science, University of Hagen, Universitätsstraße 1, Hagen, 58097, GERMANY

ABSTRACT

This paper presents a novel approach for solving unrelated parallel machine scheduling problems through reinforcement learning. Notably, we consider three main constraints: release date, machine eligibility, and sequence- and machine-dependent setup time to minimize total weighted tardiness. Our work presents a new graph representation for solving the problem and utilizes graph neural networks combined with reinforcement learning. Experimental results show that our proposed method outperforms traditional dispatching rules and an apparent tardiness cost-based algorithm. Furthermore, since we represent and solve the problem using graphs, our method can be used regardless of the number of jobs or machines once trained.

1 INTRODUCTION

In a manufacturing environment, it is common for each process to be handled by multiple machines. The main task here involves assigning jobs to machines and deciding the sequence of jobs for each machine. In particular, we tackle an unrelated parallel machine scheduling problem (UPMSP) in which job processing times change depending on the machine. This problem is common in manufacturing environments, such as semiconductor and steel production, often due to differences in machine types or versions (Wang, Wang, and Chen 2013; Pan, Wang, Mao, Zhao, and Zhang 2012). For example, in semiconductor manufacturing, the ion implantation process can be modeled as UPMSP with sequence- and machine-dependent setup time (Mönch, Fowler, and Mason 2013). We are focusing on UPMSP in real manufacturing settings, where many different constraints are considered. This includes constraints, such as release dates that specify when the job can be processed, machine eligibility, which ensures that the job can only be processed on specific machines, and sequence- and machine-dependent setup time. The goal is to provide a high-quality scheduling solution that not only complies with all specified constraints but also does so within a short computational time, which is a critical factor for real-world applicability. Therefore, we employed a reinforcement learning algorithm to tackle these challenges.

There has been extensive research on UPMSP using heuristic algorithms (Đurasević and Jakobović 2023). In particular, many studies have been conducted to consider sequence- and machine-dependent setup time and to minimize tardiness. Lee *et al.* (2013) introduced a tabu search algorithm specifically designed for the UPMSP with sequence- and machine-dependent time. Their approach focuses on minimizing total tardiness through the incorporation of various neighborhood generation methods. Lin *et al.* (2014) developed an enhanced version of the apparent tardiness cost (ATC) algorithm, which takes into account specific factors, such as setup time and release dates. They derived an initial solution with an ATC-based approach and further improved it using an iterated hybrid metaheuristic method to achieve improved results. Zeidi *et al.* (2015) proposed a hybrid algorithm that combines a genetic algorithm (GA) with simulated annealing (SA) to focus on minimizing the total weighted sum of tardiness and earliness. In this approach,

SA is used to generate the initial solutions for the GA. Afzalirad *et al.* (2016) explored the integration of GA and ant colony optimization (ACO) in the context of the block erection scheduling problem in shipyards. They used meta-heuristic approaches considering setup time, machine eligibility and release precedence in complex scheduling environments. Diana *et al.* (2018) proposed a hybrid metaheuristic combining iterated local search (ILS) with variable neighborhood descent (VND), termed ILS-VND. This approach leverages the strengths of VND's local search heuristic within an ILS framework, employing multiple restarts to navigate the solution space effectively. Perez-Gonzalez *et al.* (2019) adopted a three-phase approach in their study for unrelated parallel machine scheduling with setup time and machine eligibility. Initially, they focused on constructing a solution where jobs were allocated to machines. In instances where any machine remained unassigned, they optionally proceeded to phase two. The final phase involved conducting single-machine scheduling for each machine individually. Notably, at each stage, various algorithms were applied, and experiments were conducted across a range of problem sizes and combinations, allowing for a comprehensive evaluation of their approach.

Recently, there has been a significant increase in research on neural combinatorial optimization, especially using reinforcement learning. This research area has made significant progress, especially in traditional logistics problems, such as the traveling salesman problem (TSP) and the vehicle routing problem (VRP). Kool et al. (2018) show the applicability of reinforcement learning to the VRP problem by using a transformer architecture for the VRP that uses an encoder to insert the coordinates of each node and sequentially selects the location using a decoder. Kwon *et al.* (2020) used the symmetric characteristics of TSP by incorporating it into the baseline of the REINFORCE algorithm and further improved performance through data augmentation. Bi *et al.* (2022) used knowledge distillation as a strategic method to train a more generalized model that performs well across a variety of data distributions. This approach aimed to address a common limitation in previous models: their performance was often restricted to the specific data distributions on which they were trained.

Significant progress has also been made in applying reinforcement learning in scheduling tasks, including the job shop scheduling problems (JSSP) and the parallel machine scheduling problems (PMSP). Zhang *et al.* (2020) introduced an approach to represent the problem as a disjunctive graph and then solve it through reinforcement learning with a graph neural network. Park *et al.* (2021) advanced this field further by defining various edge relationships in JSSP. Iklassov *et al.* (2023) introduced a new strategy for solving JSSP by segmenting problems according to their difficulty. In the training phase, they dynamically train the model on problems of appropriate difficulty, resulting in significant performance improvements. Li *et al.* (2023) proposed a reinforcement learning approach for tackling the PMSP with due dates and setup constraints. They used a gated recurrent unit within the proximal policy optimization (PPO) framework and incorporated a two-stage training strategy to achieve better scheduling outcomes. Liu *et al.* (2023) developed a novel deep reinforcement learning-based approach for a dynamic PMSP that effectively selects dispatching rules to handle unexpected events in a manufacturing environment, such as machine failure and requirement change. Kwon*et al.* (2021) extended their previously researched POMO framework (2020) to address the asymmetric traveling salesman problem (ATSP) and the flexible flow shop problem (FFSP). In this study, they developed a method that effectively computes representations when a matrix contains relational information between two items.

The remaining paper is organized as follows. We will discuss the scheduling problem at hand in the next section. Moreover, we will introduce an appropriate graph representation for the UPMSP. In Section 3, we will describe the proposed graph-based reinforcement learning framework. The design of experiments and related computational experiments will be discussed in Section 4. Finally, conclusions and future research directions will be provided in Section 5.

2 PROBLEM FORMULATION

2.1 Scheduling Problem and Overall Approach

We consider n jobs that have to be scheduled on m unrelated parallel machines. Each job j has a ready time $r_j \geq 0$, a due date d_j, and a weight w_j that represents the importance of the job. The machines are unrelated, i.e., the processing time of a job depends on the machine and the job. Moreover, we have machine- and job sequence-dependent setup times. The setup time for processing job j after job i on machine k is s_{ijk}. The initial setup time for job j on an empty machine k is denoted by s_{0jk}. Moreover, we have machine eligibility restrictions, i.e., it is possible that jobs cannot be processed on all machines. We are interested in minimizing the total weighted tardiness (TWT) of the jobs defined as

$$TWT = \sum_{j=1}^{n} w_j T_j, \tag{1}$$

where we have for the tardiness of job j $T_j := \max\{C_j - d_j, 0\}$. Here, C_j is the completion time of job j. Using the three-field notation from deterministic machine scheduling, the problem can be represented as

$$Rm|r_j, M_j, s_{ijk}|TWT, \tag{2}$$

where the m unrelated parallel machines are indicated by R_m. Moreover, M_j refers to the eligibility restrictions for the job. It is well known that the single-machine scheduling problem $1||TWT$, a special case of the problem at hand, is strongly NP-hard. Hence, problem (2) is also strongly NP-hard. Therefore, we have to look for efficient heuristic approaches to tackle large-sized problem instances of (2) using a reasonable amount of computing time.

In the present paper, we construct schedules through a constructive process, employing a neural network model that sequentially selects jobs for allocation to machines. To facilitate this, we represent the UPMSP as a graph and utilize a graph neural network combined with reinforcement learning techniques.

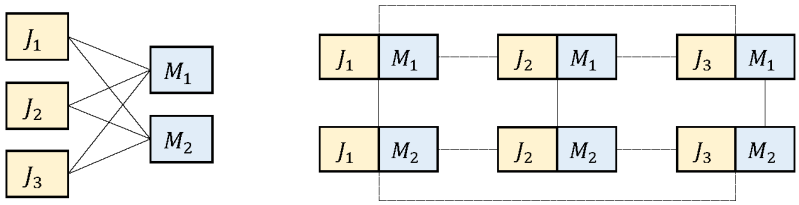
2.2 Graph Representation

We address the UPMSP while considering three main constraints, namely release dates, machine eligibility, and sequence- and machine-dependent setup times. We start by modeling the problem through a graphical representation. Instead of employing the commonly used bipartite graph, which connects machine nodes to job nodes, we had to develop a new technique to accurately represent sequence- and machine-dependent times. Therefore, we introduced the concept of a line graph, where nodes and edges are transformed (Cai, Li, Wang, and Ji 2021). In this transformation, edges in the original graph G are considered as nodes in the line graph $L(G)$. Two nodes in $L(G)$ are connected if and only if the two corresponding edges share a common node (see Figure 1a and Figure 1b).

2.3 State

We represent the state as a graph $G_t(V_t, E_t)$, where the graph is defined by nodes and edges representing the UPMSP environment at time step t. Each time step t corresponds to a decision point in the constructive scheduling process. The node set V_t is partitioned into two distinct subsets: V_t^a and V_t^b, representing machine-job pair nodes and machine nodes, respectively. Furthermore, the edge set E_t of our graph is segmented into four distinct subsets: E_t^a, E_t^b, E_t^c, and E_t^d, each describing different types of relationships within the scheduling environment:

- **Machine-Job Pair Nodes** (V_t^a): These nodes represent the pairs formed by a machine k and a job j at any specific simulation time, denoted as sim_t. Importantly, V_t^a includes all machine-job pairs, excluding those involving jobs that have been completed by time sim_t. The nodes, denoted



(a) (b)

Figure 1: (a) illustrates the parallel machine scheduling problem using a bipartite graph, where nodes represent machines (M_i) and jobs (J_i). (b) represents the line graph of (a), transforming each edge between a machine-job pair in the original graph into a node, and connecting nodes if their corresponding edges in the original graph share a common job or machine.

as v^a_{jk}, are characterized by a feature vector $[w_j, r_j, d_j, m_{jk}]$, where w_j is the weight of job j; r_j is the release date of job j, adjusted as $\max(r_j - sim_t, 0)$; d_j is the due date of job j, adjusted to $d_j - sim_t$ to reflect the time remaining until the job's due date from the current simulation time; and m_{jk} signifies whether job j can be assigned to machine k.

- **Machine Nodes** (V^b_t): These nodes represent individual machines at any given sim_t. The nodes, denoted as v^b_k within this set, encapsulate the current state of each machine k, characterized by a feature vector that typically includes the remaining processing time for the current job assigned to the machine.
- **Edges Connecting Same Job Nodes** (E^a_t): These edges are bidirectional edges that connect nodes that represent the same job across different machines. Specifically, within the set V^a_t, edges in E^a_t link nodes v^a_{jk} that share the same job identifier j. Figure 2a illustrates this relationship.
- **Edges Connecting Same Machine Nodes** (E^b_t): These directional edges connect nodes associated with the same machine but different jobs. Each edge features $s_{jj'k}$ to model the setup time from job j to job j' on the same machine k. Figure 2b illustrates this relationship.
- **Inverted Edges Connecting Same Machine Nodes** (E^c_t): These edges are the inverse of E^b_t, connecting nodes in reverse order while still relating to the same machine and possibly different jobs. The reversal is designed to aid learning processes by using $s_{j'jk}$ as an edge feature, reflecting the setup time from job j' to job j on machine k.
- **Edges Connecting Machine Nodes to Pair Nodes** (E^d_t): These edges are bidirectional and establish connections between machine nodes and machine-job pair nodes. This relationship incorporates a feature representing the setup time required for each job considering the machine's last job. Specifically, within the set V^a_t, edges in E^d_t connect nodes v^b_k (machine nodes) to v^a_{jk} (machine-job pair nodes) and vice versa. Figure 2c illustrates this relationship.

2.4 Action and State Transition

Action a_t is classified into two types. The first type of action involves assigning a job j to the selected machine k by selecting an unassigned job-machine pair (j,k). When the RL agent selects the node that represents the unassigned job-machine pair (j,k), the job j is assigned to machine k and begins at $t = \min(sim_t, r_j)$, where sim_t represents the earliest time at which at least one machine, including k, becomes available. This ensures that job j starts at the earliest possible moment, when both the machine k is available and it is after the job j's release time. The second type of action is the *wait* action. When the RL agent selects the *wait* action, this action is executed once the machine node is selected, effectively postponing the sim_t until the minimum time point when at least one currently processing machine finishes its job and becomes available for a new job assignment.

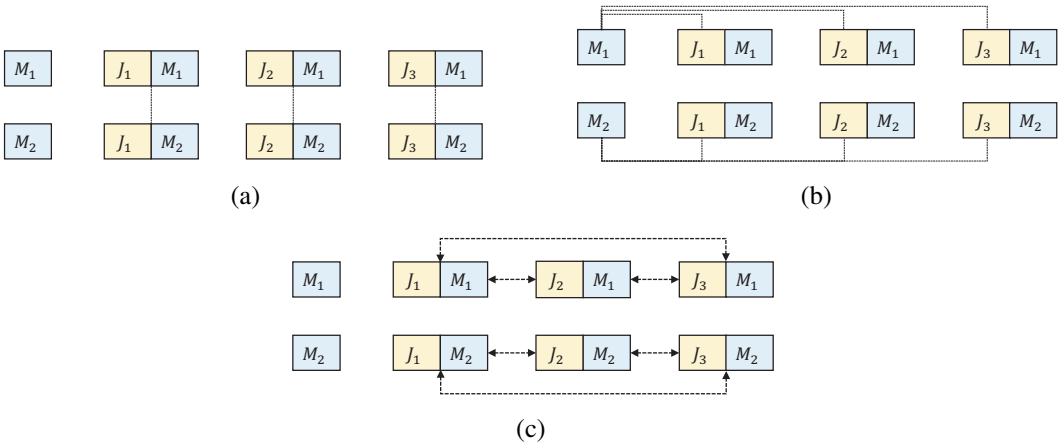


Figure 2: Graphical representations of states within a scheduling framework. Each subfigure represents distinct edge sets: (a) E_t^a for Edges Connecting Same Job Nodes, (b) E_t^d for Edges Connecting Machine Nodes to Pair Nodes, and (c) E_t^b and E_t^c for Edges Connecting Same Machine Nodes and their inverted counterparts, respectively.

2.5 Reward

The reward of the scheduling decisions is calculated when the entire scheduling process is completed, rather than being assigned for individual job-machine allocations. At this point, we use the TWT value of the schedule, the performance measure of problem (2), with a negative sign as a reward. Note that the reward is given as the negative value of the TWT to reflect the objective of minimizing the total weighted tardiness.

3 GRAPH-BASED REINFORCEMENT LEARNING FRAMEWORK

3.1 Overall Approach

Our policy network is divided into two primary components, namely an encoder and an action probability computation module. The encoder utilizes a graph neural network (GNN) to compress node information, effectively managing states that are represented as graphs. This process involves applying the GNN to subgraphs, each corresponding to one of four edge types: E^a, E^b, E^c, and E^d. These subgraphs are processed separately, and the resulting representations are concatenated to form a comprehensive state representation. The action probability computation module then converts this combined representation into probabilities. The model's parameters are optimized using the REINFORCE algorithm, a Monte Carlo policy gradient method. This method updates the model parameters using complete trajectories (Williams 1992).

3.2 Encoder

The encoder initiates by transforming each node in the state into hidden vectors of a specified size to standardize input dimensions across different nodes. This transformation is described by the following equation:

$$h_i(0) = \text{ReLU}(W_0 \cdot x_i), \tag{3}$$

where $h_i(0)$ denotes the initial hidden vector for any given node i,W_0 is a trainable matrix parameter, and ReLU is the rectified linear unit activation function. This step prepares the node features for further processing.

After this initial transformation, we apply a Graph Attention Network version 2 (GATv2) (Brody, Alon, and Yahav 2021) to subgraphs corresponding to the four edge types: E_t^a, E_t^b, E_t^c, and E_t^d. These subgraphs, composed of nodes V_t^a and V_t^b and their respective edges, are processed independently, and the node features are then concatenated to form a comprehensive state representation that captures various relationship dynamics within the graph. We obtain:

$$e_{ij}^k = a^k(l) \cdot \text{LeakyReLU}(W_1^k(l) \cdot [h_i(l), h_j(l), f_{ij}]), \tag{4}$$

$$\alpha_{ij}^k = \frac{\exp(e_{ij}^k)}{\sum_{j' \in N_i^k} \exp(e_{ij'}^k)}, \tag{5}$$

$$h_i^k(l) = W_3^k(l) \cdot \text{LeakyReLU}\left(\sum_{j \in N_i^k} \alpha_{ij}^k \cdot W_2^k(l) h_j(l) \right). \tag{6}$$

In these equations, f_{ij} represents the setup time, crucial for relationships within the edge types: Edges Connecting Same Machine Nodes (E_t^b), Inverted Edges Connecting Same Machine Nodes (E_t^c), and Edges Connecting Machine Nodes to Pair Nodes (E_t^d). The attention coefficients a^k, weights $W_1^k(l)$, $W_2^k(l)$, and $W_3^k(l)$ are all learnable parameters. Here, $l = 1, \ldots, L$ denotes the layer index, where L is the total number of layers in the network, and k ranges from 1 to 4, each corresponding to one of the four edge types E_t^a, E_t^b, E_t^c, and E_t^d. N_i^k represents the set of neighboring nodes connected to node i via edge type k in the graph. LeakyReLU, with a negative slope coefficient of 0.2, is an activation function used in the neural network.

Building upon this adaptive foundation, the final embedded vector for each node i is computed as follows:

$$h_i(l+1) = \text{ReLU}(h_i(l) + W_4 \cdot [h_i^1(l), h_i^2(l), h_i^3(l), h_i^4(l)]), \tag{7}$$

where $[\cdot, \cdot]$ symbolizes vector concatenation. The index l represents the current layer of processing in the GNN, with each layer designed to refine the node representations by integrating relational and feature information from the graph structure. The notation $h_i(l+1)$ refers to the updated node representation at the next layer, ensuring that each iteration builds upon the last to enhance the overall representation of node states. This aggregation method ensures a comprehensive representation of each node's state, effectively encapsulating the diverse graph-based interactions and characteristics it possesses. This process is repeated L times, with the hidden representations of the nodes at the final iteration denoted as $h_i(L)$.

3.3 Action Probability Computation

Following the encoder's process, the encoded hidden vectors are converted into probabilities for selecting action. To ensure compliance with constraints such as machine eligibility and task completion status, a masking process is applied. Specifically, mask_i assigns a value of 0 to nodes representing infeasible actions, while assigning a value of 1 to nodes representing feasible actions. This ensures that only feasible machine-job pair nodes are considered for job allocation. Here, the parameters W_5 and W_6 are learnable, and the softmax activation function is applied to convert the outputs into probabilities. We have:

$$y_i = W_6 \cdot \text{ReLU}(W_5 \cdot h_i(L)) + \log(\text{mask}_i), \tag{8}$$

$$p_i = \text{Softmax}_{(i)}(y_i). \tag{9}$$

3.4 Training Algorithm

We employed the REINFORCE algorithm for training. This approach involves solving each problem instance K times (sampling rollout) and using the average of these K outcomes as the baseline for variance

Algorithm 1 Training Procedure using REINFORCE

```
Initialize policy model π with random weights
for i = 1 to max_iter do
    for j = 1 to B do
        Solve each problem instance K times using model π
        Compute reward for each of the K solutions
        Calculate average reward of the K solutions as baseline
        Compute gradient ∇_θ J_j(θ) ← (1/K) Σ_{k=1}^{K} (R(τ_j^k) − b_j) ∇_θ log p_θ(τ_j^k)
    end for
    Perform gradient update θ ← θ + (α/B) Σ_{j=1}^{B} ∇_θ J_j(θ)
end for
```

reduction. Gradient updates are performed after every B problem instances to improve the policy model incrementally. Many studies solving combinatorial problems using reinforcement learning have utilized the REINFORCE algorithm (Kool et al. (2018), Kwon et al. (2020)). Following this trend, we also adopted the REINFORCE algorithm for our approach.

In Algorithm 1, $R(\tau_j^k)$ represents the cumulative reward of instance j from the k-th rollout, where each instance j is rolled out K times and b_j is the baseline calculated as the average cumulative reward from these K rollouts. Here, τ_j^k denotes the schedule obtained during the k-th rollout of instance j, θ refers to the policy parameters, $\nabla_\theta \log p_\theta(\tau_j^k)$ is the policy gradient of the k-th rollout of instance j. This gradient is used to update the policy parameters θ via gradient ascent with learning rate α, aiming to maximize the expected cumulative rewards.

4 COMPUTATIONAL EXPERIMENTS

4.1 Instance Generation

To evaluate the performance of the proposed algorithm, we generated problem instances with varying numbers of jobs (*12, 50*) and machines (*3, 6*) according to the instance generation scheme outlined in (Jaklinović, Đurasević, and Jakobović 2021).

Parameter Settings:

- **Job Processing Times:** Job processing times p_{jk} were generated from a discrete uniform distribution $DU(1,100)$.
- **Job Weights:** Job weights w_j were drawn from $U(0,1)$.
- **Setup Times:** Setup times s_{ijk} for jobs were generated from $DU(0,s_{max})$ where $s_{max} = 10$.

Release and Due Dates:

- **Release Times:** Release times for jobs r_j were determined based on half the average processing time:

$$\hat{p} = \frac{1}{m \cdot n} \sum_{k=1}^{m} \sum_{j=1}^{n} p_{jk}, \tag{10}$$

$$r_j \sim DU\left(0, \lfloor \hat{p}/2 \rfloor\right). \tag{11}$$

- **Due Dates:** Due dates were set using tightness and range parameters (T and R), which varied in increments of 0.2, starting from 0.2 up to a maximum of 1.0:

$$d_j - r_j \sim DU\left(\lfloor (\hat{p} - r_j) \cdot (1 - T - R/2) \rfloor, \lfloor (\hat{p} - r_j) \cdot (1 - T + R/2) \rfloor\right). \tag{12}$$

Machine Eligibility Constraints: Machine eligibility constraints ensure that only a specified proportion of jobs, fixed at 0.5, are suitable for each machine. Jobs are randomly selected with equal probability to meet this criterion until the specified proportion is achieved. If jobs remain unassigned to any machine, one machine is randomly selected for each of those jobs to ensure assignment to at least one machine.

4.2 Model and Training Parameters

Each of our GAT models is configured with 8 attention heads and a hidden dimensionality of 256, across three layers. We set the learning rate to 10^{-4} and use the Adam optimizer. For each problem instance, we sampled 16 schedules, using the mean of these results as the baseline in the REINFORCE algorithm. The model is then updated after every 10 problem instances have been processed. Specifically, our experiments focused on problem instances involving either 12 or 25 jobs and 3 machines, with the training process set at 1000 iterations (*max_iter = 1000*) to ensure thorough learning and convergence. Consequently, a total of 10,000 instances were used for training.

RL training and testing are conducted using an Intel Core i7-9700 CPU with 64.00 GB of RAM and an NVIDIA GeForce RTX 3070 Ti GPU with 8.00 GB of memory. The dispatching rules are also processed on the same Intel Core i7-9700 CPU. All implementation and testing of these models are carried out using Python.

4.3 Comparison Methods

To assess the effectiveness of our proposed algorithm, we compared it with a commonly used dispatch rule, earliest due date (EDD). In addition, we used ATCSR_Rm, an ATC-based algorithm that incorporates both release times and setup times, providing a more detailed evaluation of scheduling performance (Lin and Hsieh 2014).

To objectively evaluate the performance of our algorithm, we addressed small-sized problems by solving them optimally using mixed integer linear programming (MILP). We utilized the formulation presented in Lin *et al.* (2014) and extended it to include machine eligibility constraints, applying CPLEX for the computations. For all the instances with 12 jobs, we were able to find the optimal solutions.

- **Earliest Due Date (EDD):** This strategy prioritizes jobs according to their due dates, scheduling the job with the closest due date first. We also took into account machine eligibility constraints while assigning jobs using this method.
- **ATCSR_Rm (Lin and Hsieh 2014):** ATCSR_Rm assigns weights to remaining jobs by considering the factors weighted shortest processing time (WSPT), slack time, setup time, and ready time. Each of the factors slack time, setup time, and ready time is adjusted by a corresponding positive look-ahead parameter, denoted as k_1, k_2, and k_3, respectively. Rather than sticking to a fixed parameter setting, we utilized a parameter value from a grid as suggested in the relevant literature. For each instance, we experimented with all possible look-ahead parameter combinations to identify the configuration that yields the smallest TWT value. For pairs of jobs and machines that are not eligible under machine eligibility constraints, we assigned a prohibitively high processing time to effectively prevent their assignment. Next, we assume that a specific machine k^* becomes idle at time t^* after job l is processed on it. In this situation, the following ATCSR index value is computed for each unscheduled job j:

$$I_{ATCSR,j}(t^*) := \frac{w_j}{p_{jk^*}} \exp\left(-\frac{(d_j - p_{jk^*} - \max(r_j, t_{k^*} + s_{ljk^*}))^+}{k_1 \overline{p}}\right) \cdot \exp\left(-\frac{s_{ljk^*}}{k_2 \overline{s}}\right) \cdot \exp\left(-\frac{(r_j - t_{k^*} - s_{ljk^*})^+}{k_3 \overline{p}}\right), \tag{13}$$

where we abbreviate $x^+ := \max(x,0)$. Here, $\overline{p}$ is the average processing time of all unscheduled jobs, and $\overline{s}$ is the average setup time across all machines. The first factor on the right-hand side of (13) is the index of the WSPT dispatching rule, the second factor is a slack term, while the third and fourth factor are setup and ready terms, respectively. The grid is as follows:

$$
\begin{aligned}
k_1 &\in \{0.2, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.4, 2.8, 3.2, 3.6, 4.0, 4.4, 4.8, 5.2, 5.6, 6.0, 6.4, 6.8, 7.2\}, \\
k_2 &\in \{0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5, 1.7, 1.9, 2.1\}, \\
k_3 &\in \{0.001, 0.0025, 0.004, 0.005, 0.025, 0.04, 0.05, 0.25, 0.4, 0.6, 0.8, 1.0, 1.2\}.
\end{aligned}
$$

Overall, we consider 3146 different combinations of the look-ahead parameters to find a schedule with smallest TWT value.

4.4 Computational Results

4.4.1 Small-sized Instances

In evaluating our algorithm's performance on small-scale problems involving 12 jobs and 3 machines, we expanded our analysis by combining due tightness and due range parameters across five levels—0.2, 0.4, 0.6, 0.8, and 1.0—creating 25 unique sets. Each set was then used to solve 20 different problems. We utilized a model trained on instances with 12 jobs and 3 machines for this evaluation. Through this testing, we found that our algorithm outperformed the dispatching rules. This performance advantage is likely a result of our model's ability to effectively learn from various details of the problem, such as setup times, machine eligibility, and release dates, and incorporate this information to improve the scheduling decisions.

Table 1: Experimental results for small-sized instances.

n	m	T	R	MIP	EDD	ATCSR_Rm	RL
12	3	0.2	0.2	33.59	146.20	74.13	75.48
12	3	0.2	0.4	22.88	169.61	64.36	44.78
12	3	0.2	0.6	25.26	186.36	71.30	55.37
12	3	0.2	0.8	28.42	187.48	65.14	61.62
12	3	0.2	1.0	31.99	153.73	72.71	73.90
12	3	0.4	0.2	62.51	163.00	125.85	95.52
12	3	0.4	0.4	59.14	232.46	106.08	105.71
12	3	0.4	0.6	63.32	241.88	126.18	100.47
12	3	0.4	0.8	61.17	243.10	102.34	120.28
12	3	0.4	1.0	89.88	270.14	146.23	153.79
12	3	0.6	0.2	121.09	329.94	211.37	170.74
12	3	0.6	0.4	110.57	270.08	153.64	148.26
12	3	0.6	0.6	136.03	299.47	202.11	190.09
12	3	0.6	0.8	186.26	446.51	243.81	248.47
12	3	0.6	1.0	149.30	427.67	201.60	222.38
12	3	0.8	0.2	243.49	437.49	295.56	286.78
12	3	0.8	0.4	251.34	514.02	311.74	327.28
12	3	0.8	0.6	220.40	422.69	285.36	282.30
12	3	0.8	0.8	213.87	458.30	295.46	279.99
12	3	0.8	1.0	158.95	375.34	251.39	217.93
12	3	1.0	0.2	340.10	596.18	397.11	395.58
12	3	1.0	0.4	279.37	504.08	354.69	349.60
12	3	1.0	0.6	272.36	552.70	332.12	339.97
12	3	1.0	0.8	213.74	420.06	252.69	259.37
12	3	1.0	1.0	229.86	483.01	289.97	274.75
Average				144.20	341.34	201.32	195.22

4.4.2 Large-sized Instances

In evaluating our algorithm on larger-scale problems involving 50 jobs and 6 machines, we similarly expanded our analysis by combining due tightness and due range parameters across five levels—0.2, 0.4, 0.6, 0.8, and 1.0, thus creating 25 unique sets. Each set was utilized to solve 20 different problems. We utilized a model trained on instances with 25 jobs and 3 machines for this evaluation. The results confirmed that our algorithm consistently outperformed the dispatching rules in these settings as well. It is particularly noteworthy that our model was originally trained with 25 jobs and 3 machines. Even so, it performed well when tested on larger problems, proving that it is reliable and flexible for different sizes of tasks.

Table 2: Experimental results for large-sized instances.

n	m	T	R	EDD	ATCSR_Rm	RL
50	6	0.2	0.2	807.22	55.14	107.36
50	6	0.2	0.4	718.08	81.96	56.55
50	6	0.2	0.6	787.78	94.15	89.32
50	6	0.2	0.8	636.06	75.05	28.31
50	6	0.2	1.0	418.07	162.46	37.30
50	6	0.4	0.2	1185.76	229.98	182.66
50	6	0.4	0.4	1066.22	157.44	149.32
50	6	0.4	0.6	1240.91	306.27	179.69
50	6	0.4	0.8	1204.32	323.75	187.02
50	6	0.4	1.0	1172.80	412.01	183.32
50	6	0.6	0.2	1668.18	646.60	358.43
50	6	0.6	0.4	2042.09	595.01	446.75
50	6	0.6	0.6	2240.49	649.32	414.85
50	6	0.6	0.8	2230.45	796.23	583.86
50	6	0.6	1.0	1985.87	616.16	479.45
50	6	0.8	0.2	3056.44	1139.97	849.25
50	6	0.8	0.4	3171.92	1256.58	1118.12
50	6	0.8	0.6	2822.55	1052.38	1039.31
50	6	0.8	0.8	2936.82	1000.15	833.40
50	6	0.8	1.0	2665.25	749.99	624.85
50	6	1.0	0.2	3714.21	1786.38	1926.88
50	6	1.0	0.4	3498.91	1633.02	1592.88
50	6	1.0	0.6	3490.89	1372.09	1346.38
50	6	1.0	0.8	3186.04	1037.52	1014.10
50	6	1.0	1.0	2875.88	1128.98	896.31
Average				2032.93	863.61	570.94

5 CONCLUSION

In this study, we have presented a novel reinforcement learning-based approach to the UPMSP, considering key constraints such as unequal release dates of the jobs, machine eligibility, and sequence- and machine-dependent times to minimize the TWT value. Experimental results have demonstrated that the proposed algorithm performs well across various problem sizes compared to list scheduling based on a dispatching rule that is appropriate for the TWT measure.

For future work, we aim to apply this approach in dynamic and stochastic environments, leveraging the effectiveness of reinforcement learning compared to deterministic scheduling approaches applied in a rolling horizon setting. Moreover, we intend to conduct research based on meta-learning methodologies to enable our model to adapt effectively to different scenarios, including variations in release times and machine eligibility constraints.

REFERENCES

Afzalirad, M. and J. Rezaeian. 2016. "Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions". *Computers & Industrial Engineering* 98:40–52.

Bi, J., Y. Ma, J. Wang, Z. Cao, J. Chen, Y. Sun *et al*. 2022. "Learning generalizable models for vehicle routing problems via knowledge distillation". *Advances in Neural Information Processing Systems* 35:31226–31238.

Brody, S., U. Alon, and E. Yahav. 2021. "How attentive are graph attention networks?". *arXiv preprint arXiv:2105.14491*.

Cai, L., J. Li, J. Wang, and S. Ji. 2021. "Line graph neural networks for link prediction". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44(9):5103–5113.

Diana, R. O., S. R. de Souza, and F. Moacir Filho. 2018. "A Variable Neighborhood Descent as ILS local search to the minimization of the total weighted tardiness on unrelated parallel machines and sequence dependent setup times". *Electronic Notes in Discrete Mathematics* 66:191–198.

Iklassov, Z., D. Medvedev, R. S. O. De Retana, and M. Takac. 2023. "On the study of curriculum learning for inferring dispatching policies on the job shop scheduling". In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI*, 5350–5358.

Jaklinović, K., M. Đurasević, and D. Jakobović. 2021. "Designing dispatching rules with genetic programming for the unrelated machines environment with constraints". *Expert Systems with Applications* 172:114548.

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

Kwon, Y.-D., J. Choo, B. Kim, I. Yoon, Y. Gwon and S. Min. 2020. "Pomo: Policy optimization with multiple optima for reinforcement learning". *Advances in Neural Information Processing Systems* 33:21188–21198.

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

Lee, J.-H., J.-M. Yu, and D.-H. Lee. 2013. "A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness". *The International Journal of Advanced Manufacturing Technology* 69:2081–2089.

Li, F., S. Lang, B. Hong, and T. Reggelin. 2023. "A two-stage RNN-based deep reinforcement learning approach for solving the parallel machine scheduling problem with due dates and family setups". *Journal of Intelligent Manufacturing*:1–34.

Lin, Y.-K. and F.-Y. Hsieh. 2014. "Unrelated parallel machine scheduling with setup times and ready times". *International Journal of Production Research* 52(4):1200–1214.

Liu, C.-L., C.-J. Tseng, T.-H. Huang, and J.-W. Wang. 2023. "Dynamic Parallel Machine Scheduling With Deep Q-Network". *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Mönch, L., J. W. Fowler, and S. J. Mason. 2013. *Production planning and control for semiconductor wafer fabrication facilities: modeling, analysis, and systems*, Volume 52. Springer Science & Business Media.

Pan, Q.-K., L. Wang, K. Mao, J.-H. Zhao and M. Zhang. 2012. "An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process". *IEEE Transactions on Automation Science and Engineering* 10(2):307–322.

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.

Perez-Gonzalez, P., V. Fernandez-Viagas, M. Z. García, and J. M. Framinan. 2019. "Constructive heuristics for the unrelated parallel machines scheduling problem with machine eligibility and setup times". *Computers & Industrial Engineering* 131:131–145.

Wang, I.-L., Y.-C. Wang, and C.-W. Chen. 2013. "Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics". *Flexible Services and Manufacturing Journal* 25:343–366.

Williams, R. J. 1992. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". *Machine learning* 8:229–256.

Zeidi, J. R. and S. MohammadHosseini. 2015. "Scheduling unrelated parallel machines with sequence-dependent setup times". *The International Journal of Advanced Manufacturing Technology* 81:1487–1496.

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.

Đurasević, M. and D. Jakobović. 2023. "Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey". *Artificial Intelligence Review* 56(4):3181–3289.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2024-00334171). This work was initiated whilst the third author was visiting KAIST in October 2023. He would like to thank for the hospitality.

AUTHOR BIOGRAPHIES

SANG-HYUN CHO is a Ph.D. student in Department of Industrial & Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST). He received a M.S. in industrial and systems engineering from KAIST. He is interested in scheduling methodologies and applications. His e-mail address is ie02002@kaist.ac.kr.

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.

LARS MÖNCH is Professor in the Department of Mathematics and Computer Science at the University of Hagen, Germany. He received a master's degree in applied mathematics and a Ph.D. in the same subject from the University of Göttingen, Germany. His current research interests are in simulation-based production control of semiconductor wafer fabrication facilities, applied optimization and artificial intelligence applications in manufacturing, logistics, and service operations. His email address is lars.moench@fernuni-hagen.de.