DOI: 10.2195/lj_proc_en_li_202410_01

A scalable deep reinforcement learning approach for minimizing the total tardiness of the parallel machine scheduling problem

Funing Li, Ruben Noortwyck, Robert Schulz

Institute of Mechanical Handling and Logistics
Faculty 07
University of Stuttgart

Various problems in the logistics field can be modeled as parallel machine scheduling problem (PMSP), which involves the optimized assignment of a set of jobs to a collection of parallel machines. Deep reinforcement learning (DRL) has demonstrated promising capability in solving similar problems. To this motivation, we propose a practical reinforcement learning-based framework to tackle a PMSP with new job arrivals and family setup constraints. We design a variable-length state matrix containing information of all jobs and employ a Recurrent Neural Network (RNN) model to represent the DRL agent. In the numerical experiment, we first train the agent on a small PMSP instance with 3 machines and 30 jobs. Then we implement this trained agent to solve a set of instances with significant larger instance. Its performance are also compared with two dispatching rules. The extensive experimental results demonstrate the scalability of our approach and its effectiveness across a variety of scheduling scenarios.

[Logistics scheduling, Deep reinforcement learning, Dynamic parallel machine scheduling problem, Recurrent neural network]

1 Introduction

Logistics plays a crucial role in many areas, whether it's in supplying stores, fulfilling online orders, or in production. Although logistics is very important, there is reluctance to pay for it, as it is non-value-adding. As a result, logistics must always be cost-effective and continuously optimized. Optimization opportunities include the implementation of efficient supply chains, the use of modern technologies such as automation or artificial intelligence, and the improvement of route planning or warehouse management. In addition to cost optimization, punctuality in logistics is essential. Delivery delays can have significant financial consequences, such as contractual penalties or production stoppages, which can impact the entire value chain. Therefore, it is important that logistics processes are designed to be efficient and regularly optimized.

Against this background, several researchers have optimized the logistics processes by modeling the related problems as a parallel machine scheduling problem (PMSP). The PMSP involves assigning a sequence of jobs to a set of machines that operate simultaneously, which offers a practical mathematical framework for optimization and development. For example, [1] describe the problem of scheduling of electrical vehicle to charging station as a PMSP and then minimize the total charging time on all vehicles. [2] formulate a distribution problem of semiconductors on processing machines also as a PMSP model. Maintenance strategies are then developed based on this model.

In this work, we focus on a particular instance of the PMSP characterized by constraints related to new job arrivals and family setups. This model effectively abstracts the scheduling process pertinent to the modern manufacturing environment described previously. The objective function is to minimize the total tardiness. A detailed description is provided in Section 2.

As a widely-used mathematical model, the methods for solving the PMSP have been extensively studied over the past few decades. The majority of conventional approaches for tackling these problems can be broadly classified into two categories: rule-based methods and metaheuristic algorithms [3]. However, rule-based methods often fail to deliver high-quality results, since they rely on predefined rules that do not account for the unique characteristics of each specific scheduling problem. Meanwhile, metaheuristic algorithms require numerous iterations for yielding an appropriate solution for one instance, which is computationally expensive.

Furthermore, both conventional approaches struggle to respond to dynamic environments.

To obtain a proper solution in a time-efficient manner, several research studies have employed reinforcement learning (RL), including deep reinforcement learning (DRL) based approach to solve the PMSP [4]. RL is a promising branch in the field of machine learning and has achieved remarkable development in many challenging decision-making tasks, such as controlling the tokamak plasma for nuclear fusion [5] and discovering faster matrix multiplication algorithms [6]. In the RL-based approach, an agent is employed to learn an optimal policy through interaction with the environment. This mechanism aligns well with the process of most scheduling problems, making RL a potential promising alternative solution for the PMSP.

However, even though RL-based approaches show superiority over rule-based methods and metaheuristic algorithms in solving PMSPs, they still have limitations. First, most of them are not tackle the scheduling problem in an end-to-end manner, which means they cannot select job directly based on the raw information of the manufacturing environment. On the contrary, they rely on hand-crafted state features and pre-defined dispatching rules [7, 8, 9], which require extensive domain knowledge and lead to tedious work. Moreover, with the integration of hand-crafted state features and pre-defined rules, human bias might be introduced, and the potential of the data-driven method, i.e., RL, could not be fully leveraged. Second, the majority of the existing RL-based approaches need to be re-designed and re-trained when being applied to larger instances [10, 11], which is highly time-consuming and hence restrict their practical applicability in diverse and evolving manufacturing environments.

With the motivations above, we propose a highly adaptable and scalable DRL approach for solving the PMSPs. First, we formulate the instance of PMSP with variable-length representation of states and actions to enable the DRL agent to solve instances of any scales. Meanwhile, we employ a specific deep neural network called recurrent neural network (RNN) to approximate the policy of the agent, enabling the agent to process the variable-length matrix. Finally, we compare the proposed DRL-based approach with several conventional methods on various instances, demonstrating its efficiency and scalability.

The remainder of this paper is organized as follows. Section 2 presents the background of RL and offers a detailed description of the PMSP that to be solved in this work. The details of the proposed approach is established in Section 3. Section 4 provides the training process of the agent and its performance comparisons with two comparative methods. Finally, conclusions are drawn in Section 5.

2 Background

2.1 Parallel machine scheduling problem

PMSP involves assigning a sequence of jobs to a set of machines that operate simultaneously, where each job has its individual processing time and due dates. Job completed after its due date results in tardiness. The objective of this work is to optimize the allocation of jobs to minimize the sum of tardiness incurred on all jobs, which is referred as total tardiness.

The notations used for the problem description is given in the upper part of the Table 1. With this notations, the objective function can be formulated as followed:

$$TT = \sum_{j=1}^{n} \max(0, C_j - d_j) \tag{1}$$

Moreover, two constraints are taken into consideration for a more realistic representation of modern manufacturing environments. The relevant notations is given at the lower part of the Table 1.

- The family setups. In real manufacturing environments, jobs are usually categorized into several families according to their characteristics, where sequential processing two jobs from different families requires an additional setup time in between [12].
- The new job arrivals constraints. In modern dynamic manufacturing environments, the arrival of unforeseen new jobs makes real-time adjustments to the schedule necessary. Under this constraint, besides the initial jobs, new batches of jobs will be continuously introduced into the system.

2.2 Reinforcement learning

RL approaches aim to teach an agent a policy of executing actions in order to maximize the cumulative reward that the agent receives from the interaction with its environment [13]. This environment is generally modeled as a Markov Decision Process (MDP), which is a mathematical framework to describe decision-making problems. An MDP can be represented by a 5-tuple representation (S, A, p, R, γ), where S is a set of all possible states, A is a set of all actions that the agent can take, p is the state-transition function which provides the probability of a transition between every pair of states under each action, R is a reward function which generates a real value to each state-action pair, $\gamma \in (0, 1)$ is the discount factor that balances immediate versus long-term rewards.

At each decision time point t, the agent observes the current state of the environment $s_t \in S$ and then conducts the action $a_t \in A$ according to the policy π, which is a mapping from states to actions. In response to the action, the

DOI: 10.2195/lj_proc_en_li_202410_01

Table 1: Notations for PMSP

Notations for basic PMSP	
t	current time
n_t	number of available jobs at time point t
n_{init}	total number of initial jobs
m	total number of machines
j, g	index of jobs
i, k	index of machines
J_j	the jth job
M_i	the ith machine
p_j	processing time of J_j
d_j	due date of J_j
C_j	completion time of J_j
Constraint-related notations	
N_{family}	number of families
f_{J_j}	family of J_j
f_{M_i}	setup status, family of the job being processed by M_i
N_{batch}	number of new job batches
n_{new}	number new jobs per one batch
r_j	arrival time of J_j, is 0 if J_j is initial available

environment changes to the next state s_{t+1} with transition probability $p(s_{t+1} \mid s_t, a_t)$ and receives an immediate reward calculated by $R(s_t, a_t)$. To a long-term process, the return, G_t is defined as the total accumulated reward from time step t onwards, discounted by γ. The definition is given as follows:

$$G_t = R(s_t, a_t) + \gamma R(s_{t+1}, a_{t+1}) + \gamma^2 R(s_{t+2}, a_{t+2}) + \dots \tag{2}$$

The goal of RL is to find an optimal policy π that maximizes the discounted cumulative reward. The objective function $J(\pi)$ can be described by the following expression:

$$J(\pi) = \max_{\pi} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{3}$$

3 Proposed method

To implement DRL-based approach, we first carefully formulated the considered PMSP as MDP as described in Section 2. Then we employ a specific deep neural network, namely RNN, to represent the policy of the agent. The RNN is updated by RL algorithm, depending on the rewards returned to the agent by the environment.

Figure 1 demonstrates the interaction mechanism between the agent and the scheduling environment. According to this mechanism, the state is discrete and the decision point for the agent is defined as every time a machine becomes idle. State transition occurs when a job is selected for this current idle machine.

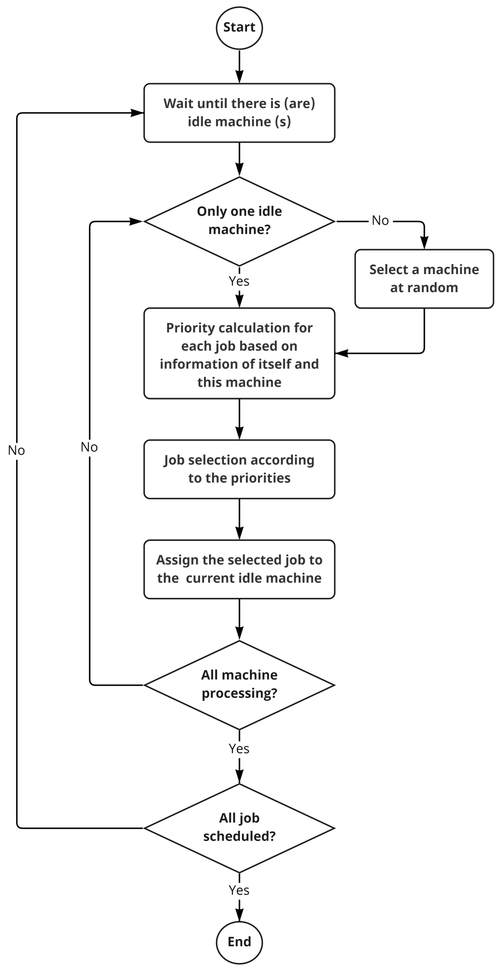
Start
Wait until there is (are) idle machine (s)
Only one idle machine?
No
Select a machine at random
Yes
Priority calculation for each job based on information of itself and this machine
Job selection according to the priorities
Assign the selected job to the current idle machine
All machine processing?
No
Yes
All job scheduled?
No
Yes
End

Figure 1: Overall Scheduling Framework

3.1 State representation

Based on the interaction mechanism shown in Figure 1, we adapt the concept of state representation from our previous work [14]. The state matrix will be the input of the agent at each decision point, based on which the agent takes corresponding action.

Figure 2 demonstrates the state matrix at time point t, while Table 2 provides the details of the job and machine information. Each state matrix contains the the processing time, the due date and the family of all jobs, while also presents the current time and the setup status of the current idle machine. Therefore, the agent can calculate a priority for each job with a global perspective, in which the information of all jobs and the the information of the current idle machine are taken into consideration.

Moreover, this state matrix is characterized with a variable length. The number of rows is equal to the number of available jobs at the given time point, where each row provides the information of the corresponding job and the current idle machine. The number of rows decreases with the selection of jobs and increases with the arrival of new jobs. With this flexible state representation, instances with arbitrary scale can be formulated and then processed by the agent, yielding a high flexibility and scalability.

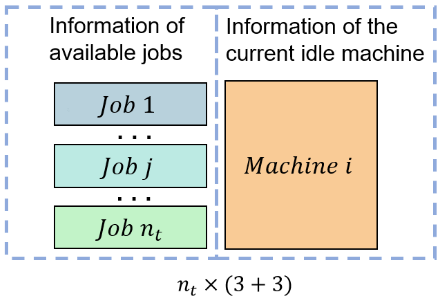
Information of available jobs
Information of the current idle machine
Job 1
Job j
Job n_t
Machine i
$n_t \times (3+3)$

Figure 2: Variable-Length State Matrix with only Jobs Information

Table 2: Job and machine information

State features in the state matrix	
Job features	processing time p_j due date d_j family f_{J_j}
Machine features	current time t family of the previous job f_{M_i}

3.2 Action representation

To fully leverage the potential of the RL as a data-driven method, the action of the agent is designed to the index of jobs. Therefore, the agent can select job directly without relying on any pre-defined rules. In particular, the agent first calculate the priorities of all available jobs based on the state matrix. The vector containing these priorities serves as the action space. Since the number of available jobs varies over the process, the action space also has a variable dimensions.

The agent will then select the most proper job for the current idle machine. To maintain explorative of the agent during training, we utilize the Softmax function to convert the priorities into a probability distribution. The Softmax function is defined as follows:

$$y_j = \frac{e^{x_j}}{\sum_{k=0}^{n_t} e^{x_k}} \tag{4}$$

, where x_j and x_k are the priorities of the jth job and kth job respectively, while y_j is the probability that the jth job will be selected. It can be seen that the job with higher priority corresponds to a higher probability of being selected. This function ensures a wider range of selection can be explored in the early training stage, in which the calculated priorities of the network are less accurate.

3.3 Neural network structures

Without loss of generality, the DRL-agents are usually represented by fully-connected neural networks, which is also known as the multi-layer Perceptron (MLP). However MLP is not suitable for processing our proposed states and actions with variable length, since it requires fixed-size input and can only output fixed-size vectors. When new jobs arrive and the total job number exceeds this pre-defined size, the entire network needs to be reconstructed and retrained, which is extremely time-consuming and restricts the applications.

To handle the state- and action representation with variable length, we apply the RNN to represent the agent. RNNs are dominant in sequence transduction problem such as natural language translation [15] and powering conversational robot [16]. They process matrix in a row-independent manner and thus highly appropriate for handling matrix with variable length.

Figure 3 depicts the RNN processing an PMSP instance with only two jobs, where the machine information is omitted for a clearer presentation. It can be seen that at the beginning of the scheduling process, the memory cell of the RNN is in the initial state. Then the RNN computes the information of the first job into its priority and simultaneously summarizes it into the memory cell. When calculating the priority of the second job, the summarized information of the first job will be read from the memory cell by the RNN and taken into account in the priority calculation. Then the information in the memory cell will be updated for the subsequent jobs, which could be considered as the summarized information of the first and second jobs.

3.4 Reward function design

Given that the objective is to minimize total tardiness, equating to the cumulative sum of all delays incurred in the scheduling process, it becomes intuitive in our MDP framework to define the reward as the negative of tardiness. Meanwhile, when the total tardiness remains 0, imply an optimal schedule, a large positive reward R_{opt} is granted.

Moreover, to provide more guidance to the agent, we add an additional reward function related to family setups. This additional reward function rooted in the observation that frequent changes in setup state consequently lead to increased processing time. Hence, a schedule of high quality intuitively features fewer setup changes, as each additional setup can heighten the probability of incurring tardiness for the subsequent jobs. In particular, the agent receives a positive additional reward R_{family} when it selects

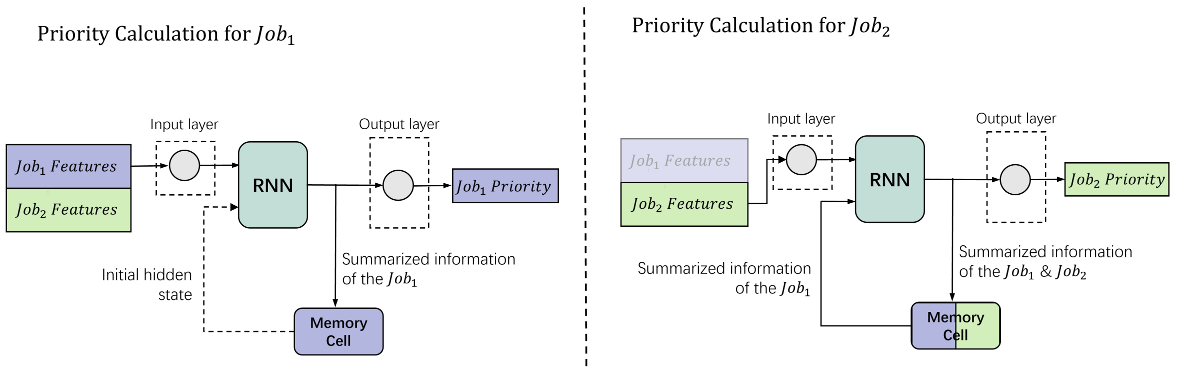

Figure 3: Working mechanism of the RNN

a job that identical to the setup state among jobs from different families, thereby avoiding additional setup time; Conversely, the agent is penalized with a negative additional reward $-R_{family}$ if it chooses a job from a different family, despite the availability of same-family jobs.

The procedure to calculate the reward is given in the Algorithm 1. In this work, we set R_{family} and R_{opt} to constants with values of 1 and 200, respectively.

3.5 Neural network updating algorithm

Based on the reward function, we utilize the Proximal Policy Optimization (PPO) [17] to update the DRL agent, i.e., the RNN. This algorithm is realized by actor-critic architecture. The actor network takes the current state as input, then executes the previously mentioned task of calculating the priority for each available job. The critic network also takes the current state as input but estimates the value of the states based on the current policy, which is a scalar and utilized for updating the actor network. Both networks are represented by RNN. Figure 4 illustrates the interaction mechanism between the actor network and the critic network.

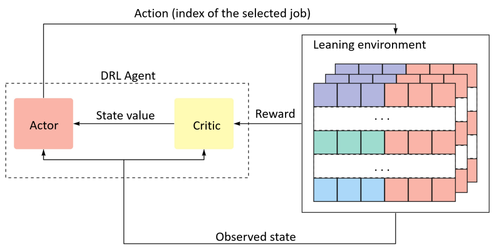

Figure 4: Cooperation mechanism between the actor network and the critic network

Algorithm 1: Reward Function Design

```
Initialization: $s_t \leftarrow s_0,\ n_t \leftarrow n,\ reward \leftarrow 0,\ Tardiness \leftarrow 0,\ TT \leftarrow 0$
while $n_t \neq 0$ do
    $a_t \leftarrow \pi_\theta(s_t)$
    $i \leftarrow a_t$
    if $f_i = f_M$ then
        for $j \leftarrow 0$ to $n_t$ do
            if $f_j \neq f_M$ then
                $reward \leftarrow reward + R_{family}$
                break
            end if
        end for
    else
        for $j \leftarrow 0$ to $n_t$ do
            if $f_j = f_M$ then
                $reward \leftarrow reward - R_{family}$
                break
            end if
        end for
    end if
    if $t + p_i/v_M > d_i$ then
        $Tardiness \leftarrow (t + p_i/v_M - d_i)$
    else
        $Tardiness \leftarrow 0$
    end if
    $reward \leftarrow reward + Tardiness$
    $TT \leftarrow TT + Tardiness$
    $s_t \leftarrow s_{t+1}, n_t \leftarrow n_t - 1$
end while
```

DOI: 10.2195/lj_proc_en_li_202410_01

4 Numerical experiments

In this section, we will first train the proposed agent on one training PMSP instance with relative small scale. Then the well-trained agent is then employed to solve various testing PMSP instances with larger scale. Its performance is then compared with the results of two dispatching rules. The comparison demonstrates the flexibility and the effectiveness of our approach.

4.1 Training process of the agent

We develop our DRL agent in PyTorch. The proposed agent is trained in an RL environment that simulates a PMSP. We construct the environment based on OpenAI Gym [18]. The training process and the comparison process that follows are conducted on a PC with Intel Core i911900KF@3.50GHz CPU, 16GB RAM, and a single Nvidia RTX 3080 GPU.

Table 3 gives the details of the training instance. This instance describes a production environment, which contains 3 machines and 20 initial jobs from 6 families. New jobs enter the production environment in batches of 5 at a constant time interval of 10 time units after the 20th time unit. A total of 2 such batches are added to simulate dynamic and fluctuating production demands that commonly occur in the real world.

In addition, the parameters r and R in this table are employed to generate the due date of the jobs according to the following uniform distribution [14, 19]:

$$U(MP(1-r-\frac{R}{2}), MP(1-r+\frac{R}{2})) \tag{5}$$

, where the indicator MP refers to the modified cumulative processing time of all jobs and is calculated as:

$$MP = \frac{\sum_{j=1}^{n} p_j + \left(\frac{n+N_F}{2} \cdot S\right)}{m} \tag{6}$$

, where n is the total number of jobs, which is calculated as $n = n_{init} + N_B * n_{new}$.

The due dates of the new jobs are generated with a modified distribution [20], ensuring the due date of a job would not earlier than its arrival time:

$$U(\max((r_j+p_{max}+S), MP(1-r-\frac{R}{2})), MP(1-r+\frac{R}{2})) \tag{7}$$

Figure 5 illustrates the training process of the proposed agent, in which the abscissa is the number of episodes, and the ordinate is the average total tardiness that the agent obtained in the previous 20 episodes. The total tardiness yielded by the agent is marked with a blue line, while a red line represents the optimal solution, namely zero total tardiness.

It is evident that, as the training process proceeds, the curve of total tardiness initially remains at a relatively high level due to the exploration phase, during which the agent is learning about the production environment. As the agent begins to understand the environment better, the total tardiness starts to decrease steadily. Eventually, the curve converges almost to zero. This stable curve and the significant low level of tardiness indicate that the agent has learned how to select the most appropriate job based on information about the production environment as well as the job itself.

Table 3: Parameter settings of the training instance

Parameter	Value
total number of machines m	3
number of initial jobs n_{init}	20
number of batches N_B	2
number of new jobs per batch n_{new}	5
total number of families of jobs N_F	6
processing time of a job p_j	Unif [5, 15]
average tardiness factor r	0.1
relative range of due dates R	0.2
setup time S	10

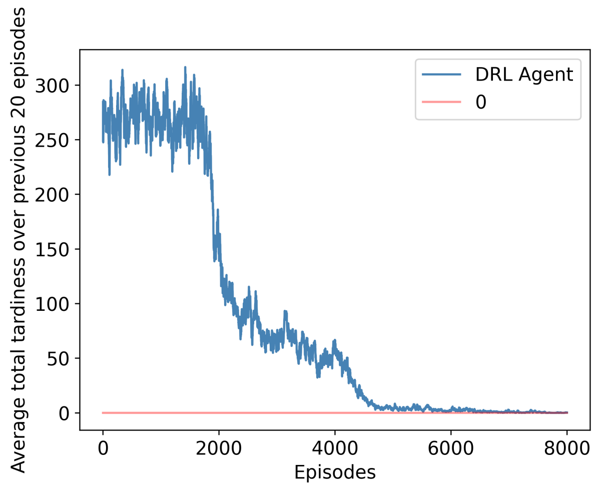

Figure 5: Average total tardiness over previous 20 episodes obtained by the DRL agent in the whole training process, the optimal solution (0 total tardiness) is marked with a red line

4.2 Generalization capability of the trained agent

To evaluate the generalization of the proposed approach, the agent trained in the previous section is then employed on much larger instances without re-training. The parameters defining the scales these larger instances are detailed in Table 4.

Moreover, to demonstrate the superiority of the proposed approach, we take two classic dispatching rules into the comparison, namely the First In First Out (FIFO) and

Table 4: Scales of PMSP Instances for Testing

Parameter	Value
total number of machines m	{8, 10}
number of initial jobs n_{init}	{200, 300, 400}
number of batches N_B	{2, 3}
number of new jobs per batch n_{new}	{5, 10}

Earliest Due Date (EDD). FIFO rule ignores the features of the jobs and processes them in the order they enter the system, while EDD selects the job with the earliest due date.

The total tardiness of all large instances obtained by the proposed DRL-based approach and the two comparative dispatching rules is given in Table 5.

Table 5: Total Tardiness Obtained by Proposed DRL approach, FIFO, and EDD

m	N_B	n_{new}	n_{init}	DRL	FIFO	EDD
8	2	5	100	147.14	988.25	225.87
			110	31.25	1015.36	358.31
			120	74.66	1447.61	205.25
		10	100	196.20	1582.08	226.37
			110	95.02	1932.57	453.86
			120	412.76	2355.34	268.70
	3	5	100	98.92	1282.62	228.66
			110	143.09	1411.92	429.72
			120	111.42	1731.55	215.87
		10	100	298.05	2472.53	428.93
			110	420.47	2788.94	314.02
			120	414.35	3492.54	465.69
10	2	5	100	111.95	719.09	172.95
			110	98.68	731.16	174.39
			120	169.47	1075.69	109.39
		10	100	182.23	1151.08	266.72
			110	198.68	1432.16	343.38
			120	217.79	1779.10	177.45
	3	5	100	29.85	921.66	161.41
			110	53.13	1015.65	193.44
			120	200.60	1276.47	172.69
		10	100	234.90	1817.32	211.78
			110	266.88	2082.17	269.77
			120	310.18	2651.27	431.90

5 Conclusion

In this work, we develop an RNN-based DRL approach to address a PMSP, characterizing by new job arrivals and family setups constraints. The main contributions of our approach are its high scalability and the end-to-end scheduling manner. To be more specific, our approach can directly select the most appropriate job based on the raw information from the scheduling environment of arbitrary scales, reducing laborious manual work and thus increasing the applicability in various scenarios.

To achieve this, we first propose a variable-length state matrix, allowing it to adapt to scheduling environment of any scale. To process this variable-length state matrix, we then utilize an RNN model to represent the DRL agent. This neural network processes inputs in row-wise manner that enables it to dynamically handle matrix of any size.

The experimental results validate the scalability and effectiveness of our approach. We first train the proposed DRL agent on a instance with 20 initial jobs, 10 new jobs in 2 batches, and 3 machines. The well-trained agent is then applied on a set of large-scale instances, where scales ranging from 110 jobs with 8 machines to 150 jobs with 10 machines. The performance is also compared with the results of two classic dispatching rules to demonstrate its superiority. Across numerous diverse and large-sized instances, our method exhibits remarkable scalability and effectiveness.

Looking ahead, there are several promising avenues for future research building on the foundation of this work. Given that the PMSP has applications in various fields, our approach has great potential for providing high-quality solutions. Additionally, incorporating more dynamic constraints into the model, such as machine breakdowns and maintenance schedules, could further enhance its applicability in real-world manufacturing environments.

REFERENCES

[1] C. Zhang, Y. Liu, F. Wu, B. Tang, and W. Fan, "Effective charging planning based on deep reinforcement learning for electric vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 1, pp. 542–554, 2020.

[2] Y.-D. Kim, B.-J. Joo, and S.-Y. Choi, "Scheduling wafer lots on diffusion machines in a semiconductor wafer fabrication facility," *IEEE Transactions on Semiconductor Manufacturing*, vol. 23, no. 2, pp. 246–254, 2010.

[3] M. Đurasević and D. Jakobović, "Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: A survey," *Artificial Intelligence Review*, vol. 56, no. 4, pp. 3181–3289, 2023.

[4] B. M. Kayhan and G. Yildiz, "Reinforcement learning applications to machine scheduling problems: A comprehensive literature review," *Journal of Intelligent Manufacturing*, vol. 34, no. 3, pp. 905–929, 2023.

[5] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J.-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[6] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[7] L. Guo, Z. Zhuang, Z. Huang, and W. Qin, "Optimization of dynamic multi-objective non-identical parallel machine scheduling with multi-stage reinforcement learning," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 1215–1219.

[8] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Applied Soft Computing*, vol. 91, p. 106208, 2020.

[9] S. Luo, L. Zhang, and Y. Fan, "Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3020–3038, 2022.

[10] S. Lang, F. Behrendt, N. Lanzerath, T. Reggelin, and M. Müller, "Integration of deep reinforcement learning and discrete-event simulation for real-time scheduling of a flexible job shop production," in *2020 Winter Simulation Conference (WSC)*, 2020, pp. 3057–3068.

[11] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, vol. 8, pp. 71 752–71 762, 2020.

[12] M. M. Liaee and H. Emmons, "Scheduling families of jobs with setup times," *International Journal of Production Economics*, vol. 51, no. 3, pp. 165–176, 1997.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] F. Li, S. Lang, B. Hong, and T. Reggelin, "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*, vol. 35, no. 3, pp. 1107–1140, 2024.

[15] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain, "Machine translation using deep learning: An overview," in *2017 international conference on computer, communications and electronics (comptelix)*. IEEE, 2017, pp. 162–167.

[16] M. Dhyani and R. Kumar, "An intelligent chatbot using deep learning with bidirectional rnn and attention model," *Materials today: proceedings*, vol. 34, pp. 817–824, 2021.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[19] C. N. Potts and L. N. Van Wassenhove, "A branch and bound algorithm for the total weighted tardiness problem," *Operations Research*, vol. 33, no. 2, pp. 363–377, 1985.

[20] F. Li, S. Lang, Y. Tian, B. Hong, B. Rolf, R. Noortwyck, R. Schulz, and T. Reggelin, "A transformer-based deep reinforcement learning approach for dynamic parallel machine scheduling problem with family setups," *Journal of Intelligent Manufacturing*, pp. 1–34, 2024.

Funing Li, M.Sc., Research Assistant at the Institute of Mechanical Handling and Logistics, University of Stuttgart.
Phone: +49 711 685 83698
E-Mail: funing.li@ift.uni-stuttgart.de

Ruben Noortwyck, M.Sc., Research Assistant at the Institute of Mechanical Handling and Logistics, University of Stuttgart. Phone: +49 711 685 83475

DOI: 10.2195/lj_proc_en_li_202410_01

E-Mail: ruben.noortwyck@ift.uni-stuttgart.de

Univ.-Prof. Dr.-Ing. Robert Schulz, Director of the Institute of Mechanical Handling and Logistics, University of Stuttgart. Phone: +49 711 685 83771
E-Mail: robert.schulz@ift.uni-stuttgart.de

Address: Institute of Mechanical Handling and Logistics, University of Stuttgart, Holzgartenstraße 15B, 70174 Stuttgart, Germany,