Discrete Lot-Sizing Problem of Single Machine based on Reinforcement Learning Approach*

Tae Jong Park and Young Jae Jang

***Abstract*— We propose a scheduling method for the discrete lot-sizing problem (DLSP) of multiple products on a single machine. To obtain an exact solution to the DLSP, the time period should be set sufficiently small despite the exponential increase in computational time. We propose a framework for solving the DLSP using reinforcement learning. We formalize the scheduling process as a sequential decision-making problem with the Markov decision process. In solving the formalized problem, we propose the structure of a deep neural network for reinforcement learning. We select proper methodology reinforcement learning via numerical experiments. Additionally, we adopt an ensemble to guarantee stability and a limited-lookahead method to get an accurate solution. Numerical experiments show that the proposed method performs well compared with a previously reported mixed integer program model.**

I. INTRODUCTION

As customer demands become increasingly diverse, the setup of machines in factories needs to be changed more frequently and the volume per setup becomes increasingly small. A change in setup usually involves a changeover cost and/or time associated with changes in tools, the repositioning of materials, and trial runs. Furthermore, if planners reduce the number of setup changes to improve productivity, backorder and inventory holding costs increase. In this context, the dynamic lot-sizing and scheduling problem concerns the production planning of multiple products over multiple periods to meet the scheduled demand using a machine of limited capacity. With production planning, multiple products can be produced, and a deterministic, discrete demand volume is given for each product.

Depending on the characteristic of the planning process, the dynamic lot-sizing problem can be classified into several models. The main difference between models is bucket size. The Capacitated-Lot Sizing Problem(CLSP) is a typical example of big-bucket model where the length a period is long enough[1]. The CLSP considers only the quantities and timings of the production. The CLSP ignores the sequence of the products within a period despite of consideration of setup carryover. The basic formulation of the Discrete Lot-Sizing Problem(DLSP) is small-bucket model for the dynamic lot-sizing problem. The DLSP allows at most one product per period, and production must be for the full period or not at all[2]. Based on the all-or-nothing assumption, the sequence of products can be described in the DLSP.

Numerous studies have investigated the dynamic lot-sizing problem adopting various approaches. Buschkühl et al. [3] and Copil et al. [4] reviewed models and algorithms used for the dynamic lot-sizing problem. Mathematical-programming-based approaches have also been proposed to solve the CLSP ([1];[5];[2]), in which an exact solution is generated; however, that a high-quality solution can be achieved within a limited time is not guaranteed. To overcome this limitation, a meta-heuristic method has been suggested, whereby a high-quality schedule is generated within a specific time limit ([6]; [7]). In the real-world, schedules are revised frequently, and they need to be generated instantaneously following environment changes (e.g., failure of a machine or exhaustion of materials). Meta-heuristic algorithms can quickly find a proper solution, but their computational time is not at a level suitable for real-world application. In particular, finding a good solution takes much time when approaching the optimal schedule. To manage this computational-time-related problem, intuitive heuristic algorithms have been used ([8];[5])). However, although these algorithms have real-world practicality, they do not guarantee a high-quality schedule that considers backorders and capacity constraints. Recently, machine learning approaches have been explored to obtain a near-optimal schedule within a limited computational time in job-shop scheduling ([9]; [10]). However, we are unaware of any study that has applied machine learning to the dynamic lot-sizing problem. A supervised learning approach may not be applicable if obtaining high-quality schedules for training a model is difficult. In reinforcement learning (RL), an agent learns a policy that defines how the agent selects the next action from a specific state at each decision epoch. RL approaches can generate schedules superior to those generated using existing approaches.

We consider the DLSP model because we focus on making a schedule of a machine, which means the model not only solves lot-sizing but also describes the sequence of production in a small-bucket environment. Building upon the foregoing literature review, we propose a scheduling method for the DLSP using an RL approach based on a deep Q-network (DQN). We address the DLSP for a single machine with a sequence-independent setup cost. We are motivated by the following industry requirement. Products can be classified into several groups. If setup changes arise within the same group, they take a constant and short period of time. In contrast, setup changes across different groups take different and long period of times. Managers allocate

*The authors thank the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea for their financial support (NRF-2021R1A2C300817)

The authors are with the Korea Advanced Institute of Science and Technology (KAIST), Daejeon 350-701, South Korea parktaejong@kaist.ac.kr;yjjang@kaist.ac.kr

products to each machine so that the products of each group do not mix with those of other groups. After such allocation, the planner determines which product to produce and how much of the product to produce in each time period. We focus on the latter process.

The remainder of this paper is organized as follows. Section II defines assumptions and formalizes the DLSP using the Markov decision process (MDP). Section III discusses the algorithm, and Section IV discusses the experiments and algorithm performance. Section V presents the conclusions and discusses the limitations of the study.

II. PROBLEM FORMULATION

A. Problem Assumptions

The model is based on the following assumptions.

- The scheduled demand can exceed the capacity. The demand is the estimated production of the downstream machines. Line balancing cannot always be achieved. And setup time and cost makes it difficult to estimate the exact capacity in the phase of demand allocation. This demand—capacity imbalance appears frequently in the scheduling of a bottleneck machine. Backorders can thus be incurred despite sufficient inventory.
- Backorders are not covered later if the demand is not fulfilled within a predefined time period. The downstream machine does not consider backorders of the past period.
- No setup cost is incurred if the setup is maintained from one period to the next;No ordering cost is incurred if the setup is carried over. The decision of whether to carry over a setup is thus crucial.
- The setup cost is constant and is independent of the sequence.
- Machine failures do not occur.
- There is no idle operation.

B. Mixed Integer Programming Model

A previous study proposed a mixed-integer programming (MIP) model. We apply setup carryover constraints of [1] in the MIP model of the DLSP. We compare the performance of the MIP model with that of our RL model.

The DLSP model formulation :

$$Min \sum_{j=1}^{N}\sum_{t=1}^{T} sc \cdot y_{jt} + \sum_{j=1}^{N}\sum_{t=1}^{T} bo \cdot I_{jt}^{-} + \sum_{j=1}^{N}\sum_{t=1}^{T} h \cdot I_{jt}^{+} \quad (1)$$

$$y_{jt} \cdot P_j^s + w_{jt} \cdot P_j^c + I_{jt-1}^{+} - I_{jt}^{+} + -I_{jt}^{-} = d_{jt} \qquad ^{\forall} j, t \quad (2)$$

$$\sum_{j=1}^{N} w_{jt} + \sum_{j=1}^{N} y_{jt} = 1 \qquad ^{\forall} t \quad (3)$$

$$w_{jt} \leq y_{jt-1} + w_{jt-1} \qquad ^{\forall} j, t = 2, ..., T \quad (4)$$

$$w_{jt} + w_{jt+1} \leq 1 + v_t \qquad ^{\forall} j, t = 1, ..., T-1 \quad (5)$$

$$y_{jt} + v_t \leq 1 \qquad ^{\forall} j, t \quad (6)$$

$$y_{jt} \in \{0, 1\} \qquad ^{\forall} j, t \quad (7)$$

$$w_{jt} \in \{0, 1\} \qquad ^{\forall} j, t \quad (8)$$

$$v_t \geq 0 \qquad ^{\forall} t \quad (9)$$

$Indices\ and\ index\ set$:
j = Products or items, $j = 1, ..., N$
t = Periods, $t = 1, ..., T$

$Data$:
sc = setup cost
bo = backorder cost
h = inventory holding cost
P_j^s = Production quantity for item j at full capacity for each period when the setup changes arise
P_j^c = Production quantity for item j at full capacity for each period when the setup carryover arise
d_{jt} = gross demand for item j in period t

$Variables$:
I_{jt}^{+} = Inventory of item j at the end of period t
I_{jt}^{-} = Backorders of item j at the end of period t
y_{jt} = Binary setup change variable(=1, if the setup for item j is changed in period t, 0 otherwise)
w_{jt} = Binary setup carryover variable(=1, if the setup for item j is carried over from period $t-1$ to period t, 0 otherwise)
v_t = Dummy variable for setup carryover constraint

The objective function (1) minimizes the total sum of setup, backorder and inventory holding cost. Constraint (2) is the balance constraint, constraint (3) ensure that either setup change or setup carryover must be selected. Constraint (4)-(6) are setup carryover constraints proposed by [1]. Constraint (4) ensure that a setup can only be carried over into period t if either a setup change arised in period $t-1$ or a setup is carried over from period $t-2$ to period $t-1$. A setup can only be carried over two consecutive bucket boundaries, if v_t = 1 in (5). And consecutive setup carryover is only possible if there is no setup change in period t by (6).

C. Markov Decision Process Model

The DLSP can be formalized using an MDP. The problem can be formalized with an MDP. It involves a tuple (S, A, P, R, γ) called the transition or segment. The state space S denotes the set of states and the action space A denotes the set of actions. The transition probability function P represents the probability : $p(S_{t+1} = s' | S_t = s, a_t = a)$. The reward function R denotes the immediate reward obtained after a transition from s to s'. And γ is discount factor.

The representation of the state is designed to describe the status of the machine and products. The state $s_t \in S$ is constructed by concatenating the machine setup status and net demand quantity at t.

- Machine status (s^m): The current setup status is the main consideration in choosing the next action because the setup cost is a crucial factor. The status is encoded through one-hot encoding and represented by a N-dimensional vector. If the jth value of the vector takes a value of 1, the machine is currently producing the jth product.

- Net demand timetable (s^p_j): A timetable expresses the demand quantity of the jth product excluding the inventory.
- Product inventory (s^i) : The quantity of inventory for each item.
- Total demand (s^q) : The total quantity of required demand for each item. The decision maker must cover the required demand to avoid backorders.

Action $a_t \in A$ is the selection of the next product to produce at time t. The machine must select a product, and the action set does not include there being no operation. An action is executed between all adjacent decision epochs. If the same action as s^m is selected, the setup is carried over. If action different from s^m is selected, the setup is changed; Selecting action in an MDP and determining decision variables value in the MIP model is equivalent.

The transition is deterministic. If the machine selects an action, it executes that action; this relates to the assumption that no machine failures occur.

During the transition from t to $t+1$ by selecting the action a_t in the state s_t, we consider the immediate reward r_t in period t. It is consist of the number of backorders of product j in time period t, the number of setup changes in time period t and the number of inventory in time period t.

$$r_t(s_t, a_t) = sc \cdot \sum_{j=1}^{N} y_{jt} + bo \cdot \sum_{j=1}^{N} I^-_{jt} + h \cdot \sum_{j=1}^{N} I^+_{jt} \quad (10)$$

The sum of the immediate reward for every decision epoch is equivalent to the objective function of the MIP model.

The discount factor γ must be 1 to prevent the distortion. We define the DLSP as an episodic task.

III. SOLUTION APPROACH

In this section we propose scheduling method based on RL in detail.

A. Deep Q-Network

We use Deep-Q-Network(DQN) to select the next action between decision epoch. In this section we describe detailed method based on MDP formulation. The basic idea for the DQN is proposed in [11]. Hessel et al.[12] proposed several extensions to the DQN. The multi-step learning is a method to avoid a myopic view. It often leads DQN to faster learning. The immediate reward of n-step learning is :

$$r_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} \quad (11)$$

TD-error of n-step DQN is :

$$r_t^{(n)} + \gamma^n min_{a_{t+n}} Q(s_{t+n}, a_{t+n}) - Q(s_t, a_t) \quad (12)$$

Q is a state action value function. An experience replay buffer(ERB) improves data efficiency and breaks correlation between successive states. Each experience transition includes $r_t^{(n)}$ in an ERB. Variance of $r_t^{(n)}$ is high depending on action in the DLSP. In other words, we can not get the same $r_t^{(n)}$ though we select the same action in the same state because of variance of the immediate reward. The environment becomes non-stationary from the perspective of agents. It causes TD-target to become unstable. We use single step learning as a result of non-stationary problem. To overcome a myopic view of single step and get an accurate selection, we adopt a limited-lookahead method.

B. Structure of Deep Network

The dimension of state about the net demand time table(s^p_j) can be increased exponentially as the number of products is increased. If time period is 1 hour and planning horizon is 24 hour, 24 dimension increase per product. High dimension can cause difficulty in the learning. We can reduce the dimension of the state with encoder. The encoder for each product state has same parameters because the net demand time table of each product is homogeneous. The encoder reduces the dimensionality of each product state to 4. After encoding for each net demand time table, every components of state is concatenated.

$$s^p = \|_{j=1}^{N} f_\xi(s^p_j) \quad (13)$$

$$Q_(s, a) = f_\theta(s^m \| s^i \| s^q \| s^p) \quad (14)$$

where ξ and θ are parameters of the encoder f_ξ and Q-network f_θ, respectively. Fig. 1 shows overall structure of DQN.

C. Ensemble

For the stability and fast training, we have used Convlution Neural Network(CNN) structure in the encoder. The DQN with CNN structure is more stable than Fully Connect Network(FCN) structure but performance is not good compared to FCN. To overcome stability issue of FCN, we adopt ensemble mechanism. $\theta^{(i)}$, $\xi^(i)$ is the parameters of ith Q-network and encoder network. After training phase 10 DQNs are trained, we choose the best parameter in test phase :

$$\arg \min_{\theta^{(i)}, \xi^{(i)}} [\sum_{t=0}^{T} r_t(s_t, a_t; \theta^{(i)}, \xi^{(i)})] \quad (15)$$

D. Limited Lookahead Method

Mentioned above, the n-step method may interfere with convergence owing to an increase in variance. However, the one-step method may generate a myopic policy that selects a greedy action. After the training phase, we conduct an lookahead method to prevent a myopic policy and improve an accuracy of selection. But a lookahead for every action is an exhaustive trial in the aspect of computational time. An effective way to improve an computational efficiency required by lookahead is to truncate the time horizon and action space[13]. We adopt 2-step lookahead method and select only three actions($\bar{A}_t$) for which the Q value is larger

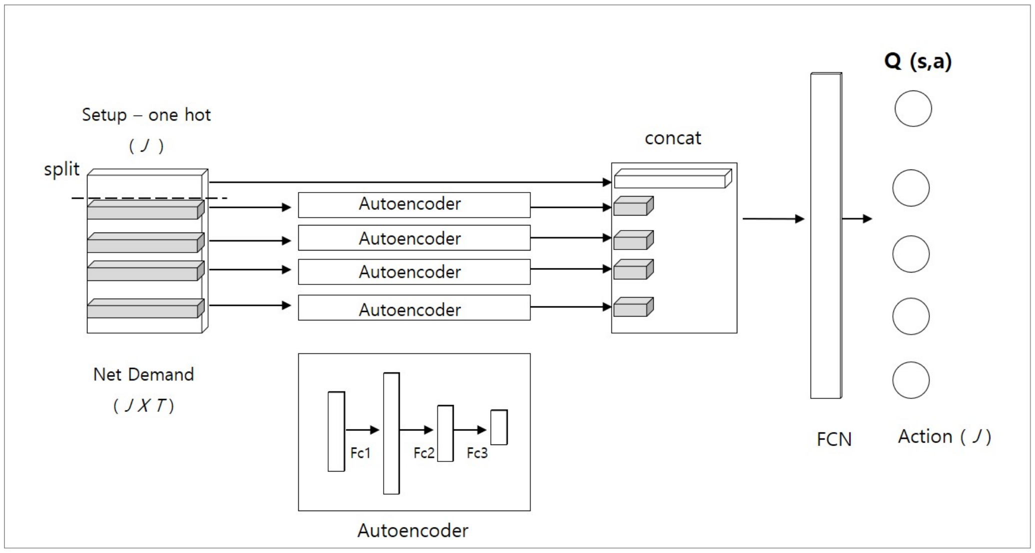

Fig. 1. Structure of DQN

at the state s in the period t. The computational time can be reduced because $\bar{A}_t$ is subset of A.

$$Q(s_t, a_t) = \min_{a_t \in \bar{A}_t} (r_t(s_t, a_t) + \min_{a_{t+1} \in \bar{A}_{t+1}} Q(s_{t+1}, a_{t+1})) \tag{16}$$

N-step Q-values is recursively calculated by equation (16). Q-values is estimated by a pre-trained Q-network. Simulations with a limited lookahead method are run for a predefined time. Each action is selected and executed until n-step epoch is finished in simulation environment.

IV. Experimental Results

A. Data generation

To the best of our knowledge, there are no benchmark data in the literature about the dynamic lot sizing problem[14]. The episode generator is based on data provided by a tire factory. These data have the following characteristic:

- Processing time for each product ranges from 110 seconds to 150 seconds.
- Inventory level is proportional to the quantity of demand per hour(d_j^t). It ranges from 1 to 3 times of (d_j^t).
- There are less than 25 types of products that can be produced in each machine per day. 25 numbers of output nodes are enough in Q-network.
- The machine to make a schedule produces goods to meet the demand of the post-processing machine. The number of post-processing machines ranges from 3 to 5.
- The processing time in the post-processing machine ranges from 400 seconds to 500 seconds.
- The initial setup is selected randomly between 25 products.

We made 10000 data sets for training and validation according to the above characteristics.

Algorithm 1 Training Procedure for the RL scheduler

```
1:  for episode i=1,2,...,$N_n$ do
2:      for time period t=0,1,...,$T$ do
3:          Observe $s_t$
4:          if $t \geq 1$ then
5:              Observe $r_{t-1}$
6:              Store post-transition $r_{t-1}$ and $s_t$
7:              $(s_{t-1}, a_{t-1}, r_{t-1}, s_t)$
8:          end if
9:          Sample a random number $p \in [0, 1]$
10:         if $p \leq \epsilon$ then
11:             Execute action $a_t$ according to $\epsilon$-greedy policy
12:         else
13:             $a_t = \arg\min_a Q(s_t, a)$
14:         end if
15:         Store pre-transition $s_t$ and $a_t$ $(s_t, a_t, ., .)$ in ERB
16:         Execute action $a_t$
17:     end for
18:     Store $r_T$ and $s_{T+1}$ in transition
19:     $(s_T, a_T, r_T, s_{t+1})$
20:     for iteration j=1,2,...,$N_I$ do
21:         Sample $N_B$ transition from Prioritized ERB
22:         Compute total $L(\theta)$ function
23:         Update Q-network parameters $\theta$, $\theta \leftarrow \theta + \eta\nabla_\theta L(\theta)$
24:     end for
25:     Update target-network parameters $\hat{\theta} \leftarrow \theta$ for every $N_U$
26: end for
```

B. Training Details

The Q-Network is trained using the Adam optimizer. We use the rectifier linear unit (RELU) as an activation function in the hidden layers. The determination of hyperparameters affects the performance of RL-based scheduling. We randomly search for the optimal values despite the large search space. We choose the hyperparameters as follows:

TABLE I
HYPERPARAMETERS USED FOR TRAINING THE DQN

Hyperparameter	Value
Number of episodes (N_n)	10000
Minibatch size (N_B)	128
Number of trainings per episode (N_I)	5
Target Q update frequency (N_U)	10 episodes
Interval of the decision epoch	0.5 hour
Replay buffer size	5000
Discount factor (γ)	1.0
Optimizer	Adam
Learning rate (η)	0.0005
Probability of selecting a random action (ϵ)	$16 * 10^{-2}$
Number of hidden layers in the encoder	2
Number of nodes for each layer in the encoder	256, 64
Number of hidden layers in the Q-Network	2
Number of nodes for each layer in the Q-Network	512, 64

C. Performance evaluation

The computation time of the MIP model is limited to 30 minutes, and the solution is obtained using CPLEX 20.1.0. The experiments are conducted on a server with an Intel i5-9600K 3.70-GHz central processing unit and 16 GB of memory. And we set the setup, backorder, inventory holding cost to 100, 10, 0.1 respectively.

In order to get the optimality gap between an MIP model and the proposed model, we generate 4 instances with 24, 32, 40, and 48 planning horizon respectively. Each instance has 30 episodes produced under the aforementioned assumptions. 4 instances can be solved to optimality in the limited time. The results of scheduling is shown in table II. We evaluate the scheduling performance by calculating the optimality gap. The optimality gap is :

$$\delta = (1 - \frac{\hat{TC}}{TC*}) * 100(\%) \tag{17}$$

where $\hat{TC}$ is the cost of RL and $TC*$ is the optimal solution of the MIP model. The average of the optimality gap is 2.4%.

In the real world, we set the planning horizon to 84 periods or longer than it to generate the schedule of 48 planning horizon. We abandon the residual schedule from the back to prevent distortion at the back end of scheduling.

TABLE II
COMPUTATIONAL RESULT FOR SHORT TERM PLANNING HORIZON

Planning Horizon		24	32	40	48
CPLEX	Avg. Total Cost	1474	1896	2287	2416
	Computational Time(sec)	3.85	20.52	147	923
RL	Avg. Total Cost	1498	1945	2352	2483
	Computational Time(sec)	4	5	7	9

TABLE III
COMPUTATIONAL RESULT FOR LONG TERM PLANNING HORIZON

Planning Horizon		84	96	108	120
CPLEX	Avg. Total Cost	4287	4920	5442	6055
	Computational Time(sec)	1800	1800	1800	1800
RL	Avg. Total Cost	4374	4972	5560	6155
	Computational Time(sec)	16	18	20	22

As shown in II, the computational time of the MIP model is dependent on the planning horizon. The MIP model could not quickly generate a high-quality schedule in the long-term planning horizon. To valid the proposed model in the long-term planning horizon, we generate 4 instances with 84, 96, 108, and 120 planning horizon respectively. Each instance has the same characteristic aforementioned. The results of scheduling is shown in table III. We also evaluate the scheduling performance by calculating the relative gap (17). We call (17) the relative gap because the MIP model can not find the optimal solution within within 30 minutes. The average of the relative gap is 1.7%. The proposed model finds a high-quality solution within 20 seconds while the MIP model takes 30 minutes to get a solution. The proposed model can be an alternative to the MIP model in dynamic environment where a schedule of many machines has to be revised frequently.

V. CONCLUSION

Although many studies have investigated the DLSP, none have addressed a dynamic environment owing to its high computational time and low accuracy. In this work, we present an RL-based practical scheduling method that can be applied in the real world. We design the DLSP of a single machine as an MDP. We then propose the structure of the DQN where transition occurs according to the proposed MDP. We use an ensemble mechanism to generate a stable schedule. After the DQN is trained, we conduct a limited lookahead method to prevent an agent from selecting a myopic action. The proposed method generates a high-quality schedule comparing to the MIP model taken from the literature in evaluating instances generated by making assumptions. Managers allocate products to different machines, but it is not guaranteed that the allocation is optimal. In terms of overall system efficiency, allocation can be more important than single-machine scheduling. Although the proposed method works well for the single-machine DLSP, the DLSP in parallel machines requires a different approach. Future work will therefore focus on the parallel-machine DLSP using a multi-agent reinforcement learning framework.

REFERENCES

[1] C. Suerie and H. Stadtler, “The capacitated lot-sizing problem with linked lot sizes,” *Management Science*, vol. 49, no. 8, pp. 1039–1054, 2003.

[2] B. Fleischmann, “The discrete lot-sizing and scheduling problem,” *European Journal of Operational Research*, vol. 44, no. 3, pp. 337–348, 1990.

[3] L. Buschkühl, F. Sahling, S. Helber, and H. Tempelmeier, "Dynamic capacitated lot-sizing problems: a classification and review of solution approaches," *Or Spectrum*, vol. 32, no. 2, pp. 231–261, 2010.

[4] K. Copil, M. Wörbelauer, H. Meyr, and H. Tempelmeier, "Simultaneous lotsizing and scheduling problems: a classification and review of models," *OR spectrum*, vol. 39, no. 1, pp. 1–64, 2017.

[5] D. Gupta and T. Magnusson, "The capacitated lot-sizing and scheduling problem with sequence-dependent setup costs and setup times," *Computers & Operations Research*, vol. 32, no. 4, pp. 727–747, 2005.

[6] Y.-F. Hung, C.-P. Chen, C.-C. Shih, and M.-H. Hung, "Using tabu search with ranking candidate list to solve production planning problems with setups," *Computers & Industrial Engineering*, vol. 45, no. 4, pp. 615–634, 2003.

[7] R. Berretta and L. F. Rodrigues, "A memetic algorithm for a multistage capacitated lot-sizing problem," *International Journal of Production Economics*, vol. 87, no. 1, pp. 67–81, 2004.

[8] P. S. Dixon and E. A. Silver, "A heuristic solution procedure for the multi-item, single-level, limited capacity, lot-sizing problem," *Journal of opérations management*, vol. 2, no. 1, pp. 23–39, 1981.

[9] I.-B. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1420–1431, 2019.

[10] J. Park, J. Chun, S. H. Kim, Y. Kim, and J. Park, "Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning," *International Journal of Production Research*, vol. 59, no. 11, pp. 3360–3377, 2021.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[12] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[13] D. Bertsekas, *Dynamic programming and optimal control: Volume I.* Athena scientific, 2012, vol. 1.

[14] M. A. Belo-Filho, F. M. Toledo, and B. Almada-Lobo, "Models for capacitated lot-sizing problem with backlogging, setup carryover and crossover," *Journal of the Operational Research Society*, vol. 65, no. 11, pp. 1735–1747, 2014.