공학박사학위논문

Deep Reinforcement Learning Based Scheduler for Flexible Job Shops with Sequence Dependent Setups

순서 의존적 셋업이 있는 유연 잡 샵을 위한 심층 강화학습 기반 스케줄러

2020 년 2 월

서울대학교 대학원

산업공학과

박 인 범

Deep Reinforcement Learning Based Scheduler for Flexible Job Shops with Sequence Dependent Setups

순서 의존적 셋업이 있는 유연 잡 샵을 위한 심층 강화학습 기반 스케줄러

지도교수 박 종 헌

이 논문을 공학박사 학위논문으로 제출함

2019 년 12 월

서울대학교 대학원

산업공학과

박 인 범

박인범의 공학박사 학위논문을 인준함

2019 년 12 월

위 원 장	조 성 준	(인)
부위원장	박 종 헌	(인)
위 원	이 경 식	(인)
위 원	정 재 윤	(인)
위 원	김 관 호	(인)

Abstract

Deep Reinforcement Learning Based Scheduler for Flexible Job Shops with Sequence Dependent Setups

In-Beom Park
Department of Industrial Engineering
The Graduate School
Seoul National University

This thesis studies a flexible job shop scheduling problem (FJSP) with sequence dependent setups. FJSP becomes significantly complicated when there are several constraints related to re-entrant flows in manufacturing systems. At the same time, the scheduling problems need to be solved frequently to effectively manage the variabilities in production requirements, available machines, and initial setup status. Accordingly, scheduling decisions are usually required to be made on an hourly basis, making it challenging to obtain high quality schedules within the time limit for large-scale manufacturing systems.

To minimize the makespan of FJSP, three scheduling methods using deep reinforcement learning (DRL) are presented in this thesis. First, we suggest a decentralized scheduler (DS) in which each agent determines setup decisions in a decentralized manner and learns a policy by sharing a neural network among the agents to deal with the changes in the number of machines. Furthermore, novel definitions of state,

action, and reward are proposed to address the variabilities in production requirements and initial setup status.

Second, we introduce a centralized scheduling approach in which an agent that determines actions given observations for jobs and machines. To reduce the complexity of state space inherent to the centralized learning, a novel definition of the state is developed by abstracting observations from an environment. Based on the centralized approach, we proposed two schedulers that select a rule-based method (CS-R) and a job-machine pair (CS-P) as an action, respectively. Specifically, CS-R employs the Q-network to learn a centralized policy and CS-P utilizes the actor-critic deep reinforcement learning method and Wolpertinger policy to approximate the continuous features of job-machine pairs.

To verify the robustness of the proposed method, neural networks (NNs) trained on small-scale scheduling problems are used to solve large-scale scheduling problems. Through extensive experiments on solving scheduling problems from real-world semiconductor packaging lines, we demonstrate that the proposed approach outperforms rule-based, meta-heuristic, and other RL methods in terms of the makespan while incurring shorter computation time than the meta-heuristics considered. Furthermore, the trained NN performs well in solving unseen real-world scale problems even under stochastic processing time, suggesting the viability of the proposed method for real-world manufacturing lines.

Keywords: Flexible job shop scheduling, Semiconductor packaging lines, Multi-chip product, Deep reinforcement learning, Neural networks

Student Number: 2012-21056

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and motivation

Production scheduling is defined as a sequential decision problem which aims to allocate each job to a machine. As the competition in real-world manufacturing systems has intensified, minimizing makespan of a schedule becomes significantly important. Recently, a flexible job shop problem (FJSP) has been extensively investigated due to its wide applicability for various manufacturing systems [1].

FJSP that allows an operation of a job to be processed by one of alternative machines is known to be NP-hard [2]. The scheduling problem becomes significantly complicated when there are several constraints related to sequence dependent setups and re-entrant flows in manufacturing systems [3, 4, 5]. Furthermore, the scheduler should deal with the variabilities in production requirements, available machines, and initial setup status. Accordingly, scheduling decisions are usually required to be made on an hourly basis [6], making it challenging to obtain high quality schedules within the time limit for large-scale manufacturing systems.

We focus on FJSP with sequence dependent setups in this thesis. Several metaheuristics have been proposed to solve FJSP with sequence dependent setup time [1, 7, 8]. However, it is intractable to solve real-world scheduling problems based on

those methods within a specific time limit since they need a lot of computations to find a near-optimal schedule [6]. On the other hand, rule-based methods have been still widely used by manufacturers to obtain a schedule in short computation time [9, 10, 11, 12]. Yet, one of their limitations is that high quality solution is not guaranteed since no single rule outperforms all the others for different shop configurations [13].

Various studies that addressed scheduling problems based on machine learning have been carried out to overcome the drawback of rule-based methods [14]. For scheduling real-world manufacturing systems, supervised learning approaches were employed by using a large number of schedules as training datasets to quickly solve unseen scheduling problems. Specifically, an adaptive rule was presented to dynamically determine parameters of the rule by training neural networks (NNs) [15, 16]. NN-based schedulers were developed by learning the proper rules [17] or the assignment preferences between jobs and machines [18]. Lim *et al.* [6] proposed a case-based reasoning approach to find the best decision for the state expressed in a Petri net. Yet, the performance achieved by these approaches may not be satisfactory in real-world scheduling systems since it is difficult to obtain high quality schedules which are required as training datasets.

By learning a global optimal policy from random explorations, reinforcement learning (RL) has also been actively employed to solve the scheduling problem. For example, several studies such as single machine scheduling [19, 20], parallel machine scheduling [21, 22], flow shop scheduling [23], scheduling of flexible manufacturing systems [24], and scheduling of semiconductor final testing [25] have been conducted by using Q-learning which is a representative technique for model-free RL [26]. These

methods train a single agent for an entire manufacturing system. Unfortunately, the single agent approach requires a great deal of training time as the size of the state and action space grows exponentially [27]. From the view of the RL-based scheduling, the size blows up rapidly when the numbers of jobs and machines increase.

Motivated by the above remarks, we propose scheduling methods for FJSP with sequence dependent setups using deep reinforcement learning (DRL). To minimize makespan and accommodate the variabilities of scheduling problems, novel definitions of state, action, and reward are proposed for schedulers. As a result, a new scheduling problem can be solved by using the trained neural networks even when the production requirements, the number of machines, and the initial setup status for a problem are changed from those of the scheduling problems used to train the networks.

Once a training process of each scheduler is completed, the computation time required to solve a scheduling problem with the proposed methods is comparable to that of a rule-based method. Through extensive experiments on datasets from real-world semiconductor packaging lines, we demonstrate that the schedulers trained from small-scale scheduling problems performs well in solving scheduling problems with unseen real-world scale problems, and the proposed method outperforms rule-based, meta-heuristic, and other RL methods in terms of the makespan. Furthermore, the robustness of the proposed methods is analyzed by considering the scheduling problems with stochastic processing time.

1.2 Objectives

The main objective of this thesis is to propose setup change scheduling methods for minimizing makespan in FJSP with sequence dependent setups. Due to the variabilities in production requirements, the number of available machines, and initial setup status, the scheduler should manage the variabilities and while producing high quality schedules within a specific time limit.

The thesis consists of decentralized and centralized approaches to reach its goals. The decentralized approach focuses on implementing multi-agent reinforcement learning in which each agent determines setup decisions in a decentralized manner and learns a centralized policy by sharing a neural network among the agents to deal with the changes in the number of machines. The centralized approaches are categorized into two parts according to the definition of an action. The basic concepts and purposes of the studies are summarized as follows.

First, a decentralized scheduler (DS) based on the Q-learning is proposed in Chapter 3. DS is designed to deal with the complexity of state and action spaces. The state and action representations are designed to accommodate the variabilities in the production requirements and the initial setup status. Moreover, the technique [28] that enables an NN to be shared between agents is employed to deal with the changes in the number of machines.

In Chapter 4, we propose two centralized schedulers in which an agent determines all actions given a global state. To reduce the complexity of state space in the centralized methods, a novel definition of the state is developed by abstracting observations from an environment. The schedulers select a rule-based method and a job-machine pair given a state, respectively. The rule selection scheduler employs

the Q-learning to learn a centralized policy. Meanwhile, the job-machine pair selection scheduler utilizes the actor-critic deep reinforcement learning method [29] and Wolpertinger policy [30] to approximate the continuous features of job-machine pairs such as the processing and setup time.

In Chapter 5, the effectiveness and robustness of the proposed methods are demonstrated through extensive experiments on datasets from real-world semiconductor packaging lines. The performances of the proposed schedulers are compared with the rule-based, meta-heuristic, and other RL methods in terms of makespan. Additionally, the results obtained by solving scheduling problems with stochastic processing time are presented to examine the robustness of the proposed schedulers.

1.3 Thesis outline

The thesis is organized as follows. In Chapter 2, the definition of the scheduling problem considered are introduced with mathematical formulations and notations. Then, previous research on the scheduling methods for flexible job shop with sequence dependent setups and semiconductor manufacturing facilities are investigated. Furthermore, the considered DRL models are introduced, and the RL-based scheduling methods are examined. The proposed scheduling methods, consisting of one decentralized method and two centralized methods. are introduced in Chapters 3 and 4, respectively. Chapter 5 presents comparison results between the proposed methods and considered alternatives on datasets from real-world semiconductor manufacturing lines. Finally, conclusions and future works of the thesis are drawn in Chapter 6.

Chapter 2

Background

2.1 Flexible job shop scheduling with sequence dependent setups

FJSP has been extensively investigated [31, 32, 33, 34, 35, 36]. The scheduling consists of two subproblems: the routing problem that assigns each operation to one of the machines and the sequencing problem that determines a sequence of operations on all the machines [33]. Different from a general FJSP, setup time is incurred when two different types of operations are successively processed when solving FJSP with sequence dependent setups (FJSP-SDST).

The considered scheduling problem for flexible job shops with sequence dependent setups is described as follows. There are jobs that belong to one of N_J job types. Jobs are processed by N_M machines of which the l^{th} machine is denoted as M_l. It is assumed that all jobs are waiting for being processed and all machines are idle at the start time of scheduling. We denote the j^{th} job type as J_j. Let $P(J_j)$ be the total number of jobs of J_j to be scheduled, indicating the production requirement of J_j. A job of J_j consists of $N(J_j)$ operations which need to be processed in the pre-determined order, $O_{j,1}, ..., O_{j,N(J_j)}$.

The k^{th} operation type of J_j is represented as $O_{j,k}$. The number of operation

types is denoted as N_O. Operations must be performed by one machine at a time without interruption once started. In addition, the operation sequences of two jobs are the same when their job types are identical. $O_{j,k}$ is said to be ready when there is at least one waiting operation whose type is $O_{j,k}$. We assume that the moving time for each operation is zero. $O_{j,k}$ can be processed on any machine among its alternatives, and the set of machines that can process $O_{j,k}$ is defined as $\mathcal{M}_{j,k}$. The processing time of $O_{j,k}$ is denoted as $p_{j,k}$. Note that $p_{j,k}$ is the same regardless of the machine where $O_{j,k}$ is processed.

An operation can be processed only by a machine that has been set up for its operation type. In detail, each machine has initial setup status at the beginning of a schedule. When a machine has been set up for processing an operation of $O_{j,k}$, its setup type is denoted as $O_{j,k}$. If an operation of $O_{j',k'}$ is assigned to the machine whose setup type is $O_{j,k}$, the operation of $O_{j',k'}$ can be processed at the machine only after setup change time, denoted as $\sigma_{j,k,j',k'}$, has passed. $\sigma_{j,k,j',k'}$ is 0 if $O_{j,k} = O_{j',k'}$, and it is positive, otherwise. The objective function is to minimize makespan, $C_{\max}$, which is the completion time of the last finished operation.

For the clarity of problem definition, mathematical formulations are presented in this section. The formulations are revised from the one presented in [7, 8] to accommodate the objective function and initial setup status. Table 2.1 shows the additional indices, parameters, and variables for mathematical formulations.

Table 2.1: Notations for mathematical formulations

Categories	**Notations**	**Descriptions**
Indices	$o_{j,k}$	the k^{th} operation of j^{th} job
	M_l	the l^{th} machine
Parameters	o_{max}	Maximum number of operations that can be assigned a machine
	$p(o_{j,k})$	Processing time of $o_{j,k}$
	$\sigma(o_{j,k}, o_{j',k'})$	Incurred setup time when $o_{j',k'}$ is processed on the machine whose setup type is $o_{j,k}$
	$\mathcal{M}_{j,k}$	Set of alternative machines that can process $o_{j,k}$
	$IS(M_l)$	Initial setup status of machine l
	H	A huge positive number
Integer variables	$c_{j,k,l}$	Completion time of $o_{j,k}$ on M_l
	$h_{r,l}$	Completion time of the r^{th} operation on M_l
Binary variables	$x_{r,l,j,k}$	Binary variable that has 1 if the r^{th} operation on machine l is $o_{j,k}$, 0 otherwise
	$z_{r,l}$	Binary variable that is equal to 1 if there is the r^{th} operation on machine l, 0 otherwise

$$\text{Minimize } C_{\max} \tag{2.1}$$

Subject to

$$C_{\max} \geq c_{j,k,l}; \forall\ (j,k), l \tag{2.2}$$

$$h_{r,l} \geq c_{j,k,l} + H(x_{r,l,j,k} - 1); \forall\ (j,k), l \tag{2.3}$$

$$h_{r,l} \leq c_{j,k,l} - H(x_{r,l,j,k} - 1); \forall\ (j,k), l \tag{2.4}$$

$$h_{1,l} - p(o_{j,k}) - \sigma(IS(M_l), o_{j,k}) - H(x_{1,l,j,k} - 1) \geq 0; \forall\ (j,k), l \tag{2.5}$$

$$\begin{aligned} &h_{r+1,l} - p(o_{j,k}) - \sigma(o_{j',k'}, o_{j,k}) - H(x_{r+1,l,j,k} + x_{r,l,j',k'} - 2) \geq h_{r,l}; \\ &\forall\ (j,k), (j',k'), r, l \text{ s.t. } (j,k) \neq (j',k') \end{aligned} \tag{2.6}$$

$$\begin{aligned} &h_{1,l} - p(o_{j,k+1}) - \sigma(IS(M_l), o_{j,k+1}) - H(x_{1,l,j,k+1} + x_{r',l',j,k} - 2) \geq h_{r',k'}; \\ &\forall\ (j,k), l, l', r' \text{ s.t. } (1,l) \neq (r',l'), r < o_{max} \end{aligned} \tag{2.7}$$

$$\begin{aligned} &h_{r+1,l} - p(o_{j,k+1}) - \sigma(o_{j',k'}, o_{j,k+1}) - H(x_{r+1,l,j,k+1} + x_{r',l',j,k} - 3) \geq h_{r',l'}; \\ &\forall\ (j,k), (j',k'), r, r', l, l' \text{ s.t. } (j,k) \neq (j',k'), (r,l) \neq (r',l'), r < o_{max} \end{aligned} \tag{2.8}$$

$$x_{r,l,j,k} = 0; \forall\ (j,k), r, l \text{ s.t. } M_l \notin \mathcal{M}_{j,k} \tag{2.9}$$

$$\sum_{l=1}^{m} \sum_{r=1}^{o_{max}} x_{r,l,j,k} = 1; \forall\ (j,k) \tag{2.10}$$

$$\sum_{j=1}^{n} \sum_{k=1}^{N_j} x_{r,l,j,k} = z_{r,l}; \forall\ (r,l) \tag{2.11}$$

$$z_{r+1,k} \leq z_{r,k}; \forall\ (r,l) \text{ s.t. } r < o_{max} \tag{2.12}$$

Equations (2.1) to (2.10) represent the mixed integer linear programming (MILP) model for the scheduling problem considered in this thesis. The objective function (2.1) is to minimize the makespan, C_{max}. Constraint (2.2) aims to enforce $_{\max}$ to be the maximum completion time of a schedule. Constraints (2.3) and (2.4) indicate that $h_{r,l}$ is equal to $c_{j,k,l}$ if $o_{j,k}$ is assigned to the r^{th} operation of M_l. Constraint (2.5) states that if the first assigned operation on M_l is $o_{j,k}$, the completion time of $o_{j,k}$ is equal to or greater than the sum of its processing time and setup time incurred from $IS(M_l)$. The setup from $o_{j',k'}$ to $o_{j,k}$ on M_l cannot be performed until $o_{j',k'}$ is finished, as enforced by constraint (2.6).

Constraints (2.7) and (2.8) indicate that $o_{j,k+1}$ cannot be performed on M_l before $o_{j,k}$ is processed on $M_{l'}$, for any pair of (l, l'). The difference between the constraints is whether $o_{j,k+1}$ is the first processed operation on M_l or not. The presence of alternative machines are considered in constraint (2.9). Constraint (2.10) indicates that $o_{j,k}$ can be performed on only one of the machines and $z_{r,l}$ is equal to 1 if there is the r^{th} operation on M_l. Finally, the precedence constraint on same machine is represented in constraint (2.10).

Since FJSP-SDST is known to be strongly NP-hard, most studies adopted meta-heuristic methods to deal with its complexity [1, 2, 7, 8, 37, 38, 39, 40, 41, 42, 43]. In particular, FJSP-SDST where an operation can be processed on one of two parallel machines was investigated in [2]. In this study, a tabu serach algorithm was proposed, and its performance was compared with a branch and bound algorithm. [39] proposed a genetic algorithm (GA) to address multicriteria including the makespan, machine workloads, and setup time. The heuristic that generates an initial solution for GA was also developed. In [8], several constraints such as attached/detached setup types,

machine release date, and batch size were considered. They proposed GA that can be implemented in parallel to deal with large-scale scheduling problems.

A neighborhood search function was developed in [42] by assuming a symmetric definition of setup time and small setup time compared to processing time. Shen *et. al.* developed a tabu search algorithm with specific neighborhood functions and a diversification structure after studying structure properties of FJSP-SDST using a disjunctive graph model. Extensive experimentation results show that their performance outperformed the methods of [2] and [42].

Although FJSP-SDST was successfully solved by the meta-heuristics, it is intractable to solve real-world scheduling problems based on those methods within a specific time limit since they need a lot of computations to find a near-optimal schedule [6].

2.2 Scheduling methods for semiconductor manufacturing facilities

Since we will demonstrate the performance of the proposed methods for solving semiconductor manufacturing scheduling problems, previous scheduling methods for semiconductor manufacturing systems have been investigated. Previous studies are classified according to their approaches, as presented in Table 2.2.

Simulation-based methods for scheduling semiconductor manufacturing lines are found in [4, 44, 45, 46, 47]. Most studies focused on investigating characteristics of the considered semiconductor manufacturing system. On the other hand, to improve the objective function of the scheduling problem, [4] proposed a simulation optimization approach for a real-world semiconductor back-end assembly facility through various scenario analyses with managerial implications. Since the performance of the method is determined by the optimization method, it is important to select an appropriate optimization method for a semiconductor manufacturing system.

Another studies aim to perform scheduling decisions by utilizing meta-heuristics [7, 25, 48]. [7] solves the setup change scheduling problem for semiconductor packaging facilities by utilizing a genetic algorithm (GA) to maximize machine utilization while minimizing the total setup change time of all machines. [48] proposed a hybrid GA for solving due date quoting and production scheduling problems in the semiconductor back-end production process. On the other hand, [25] utilizes cuckoo search algorithm for scheduling semiconductor test facilities. To reduce the evaluation time of an obtained schedule, the surrogate modeling is proposed to approximate the fitness of each schedule. However, it is intractable to solve real-world scheduling problems based on those methods within a specific time limit since they need a lot

of computations to find a near-optimal schedule [6].

Rule-based methods have been still actively used by semiconductor manufacturers to obtain a schedule since they have low implementation complexity and high computational efficiency [5, 9, 10, 11]. For example, [5] developed a reactive greedy randomised adaptive search procedure (GRASP) to hierarchically determine the setup status of a machine. Yet, one of their limitations is that high quality solution is not guaranteed since no single rule outperforms all the others for different shop configurations [13].

In recent decades, various studies that addressed scheduling problems based on machine learning have been carried out to overcome the drawback of rule-based methods [14]. For scheduling semiconductor manufacturing systems, supervised learning approaches were employed by using a large number of schedules as training datasets to quickly solve unseen scheduling problems. Specifically, an adaptive rule was presented to dynamically determine parameters of the rule by training neural networks (NNs) [15, 16]. [17], [49], and [50] developed an NN-based scheduler that learns the best rule for each decision. Lim *et al.* [6] proposed a case-based reasoning approach to find the best decision for the state expressed in a Petri net. The cosine similarity was calucated between the current state and stored schedules which were obatined by solving GA proposed in [7]. However, the performance achieved by these approaches may not be satisfactory in real-world scheduling systems since it is difficult to obtain high quality schedules which are required as training datasets. To overcome this limitation, [18] attempted to learn the assignment preferences between jobs and machines by training randomly generated dispatching decisions. Although these methods do not need high quality schedules as training datasets, it is difficult

Table 2.2: Summary of literature on scheduling methods for semiconductor manufacturing facilities

Approaches	**References**	**Methods**	**Performance metrics**
Simulation	[44]	-	Demand fulfillment
	[4]	-	Flow time
	[45], [47]	-	Cycle time, tardiness and utilization
Rule-based methods	[5]	GRASP	Setup time
	[9]	Compound priority dispatching	Cycle time throughput, and WIP
	[10]	Response surface methodology	Cycle time and WIP
	[11]	GRASP	Shortage and completion rates
Meta-heuristics	[7]	GA	Utilization and setup time
	[48]	Hybrid GA	Tardiness
	[25]	Cuckoo search with surrogate modeling	Makespan
Supervised learning	[15], [16]	NN	Utilization and WIP
	[17]	NN	Flow time
	[6]	Case-based reasoning	Utilization
	[18], [49]	NN	Utilization and flow time

to apply this method since the setup time was not considered.

Finally, We summarize the state-of-the-art methods for solving FJSP-SDST or scheduling problems of die attach and wire bonding stages in semiconductor packaging facilities, as presented in Table 2.3. Although these methods performed well in solving their scheduling problems, none of these methods addressed the variabilities in production requirements, the number of machines, and the initial setup status.

Table 2.3: Previous methods for solving scheduling problems considered

Ref.	**Descriptions**	**Comments**
[48]	Mathematical optimization of master problem and solving subproblems with GA	Minimizing tardiness
[8]	Parallel GA with consideration of attached and detached setups	Not considering initial setup status
[7]	GA with a genetic operation recommender	Maximizing utilization and minimizing setups
[1]	Tabu search by considering structure properties with a disjunctive graph	Assuming a symmetric definition between setups
[6]	Case-based reasoning by constructing the casebase using GA from [7]	Dependent on the quality of casebase
[18]	NN that allocates an appropriate job to a machine	Not considering setup time
[51]	DRL based scheduler that determines an appropriate job for each decision	Minimizing due date deviations and assuming the fixed numbers of jobs and machines

2.3 Deep reinforcement learning models

2.3.1 Deep Q-learning

Reinforcement learning (RL), which is one of the machine learning approaches, can be employed to solve sequential decision-making problem that can be formulated as a Markov decision process [52]. In RL, an agent executes an action given a state from the environment. After executing an actions, the agent receives a reward from the environment, and the goal of RL is to maximize cumulative rewards. In particular, Q-learning is a representative technique for model-free RL [26].

Without loss of generality, let s_t, a_t, r_t be the state, action, and reward at timestep t, respectively. The Q-learning aims to train an an action-value function, called a Q-function, which approximates the future value of each action given a state. The Q-functions is defined as follows.

$$Q(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + ...|s = s_t, a = a_t, \pi] \tag{2.13}$$

where γ is a discount factor and π indicates a policy $P(a|s)$. To train the Q-function, the following loss L should be minimized.

$$L = Q(s_t, a_t) - (r_t + \gamma \max_{a'} Q(s_{t+1}, a')) \tag{2.14}$$

where $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$ is a target-value. Since NNs have ability to capture non-linear and continuous state space, they are adopted as a Q-function. Unfortunately, the NN-based Q-learning is known to be unstable and even to diverge. This is due to the facts that small updates of Q may significantly change the policy π and the

correlations are presented in successive state and action pairs.

To overcome the the instability of an NN, deep Q-learning [53] was proposed. Its two major contributions are a target Q-network and the experience replay. First, the target Q-network is used to train the original Q-network. The target network is periodically updated, which ensures stability of learning by reducing correlations. Therefore, the loss from (2.14) is changed as follows.

$$L' = Q(s_t, a_t; \theta) - (r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \hat{\theta})) \tag{2.15}$$

where θ and $\hat{\theta}$ indicate the weights of original and target Q-networks, respectively. Second, a transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer and transitions are randomly sampled during the training of the Q-network. This technique alleviates correlations between the state, action and reward pairs.

2.3.2 Deep deterministic policy gradient

Although the deep Q-learning is used to successfully solve sequential decision making tasks with discrete action spaces, it is still intractable to solve the tasks with continuous action spaces since the policy optimization through Q-learning is too slow to be practical with large and continuous action spaces. To address this problem, deep deterministic policy gradient (DDPG) algorithm [29] was suggested. This method has been successfully utilized for solving real-world decision making problems [54, 55, 56, 57, 58]. The agent of DDPG consists of two NNs called the actor and critic networks. Specifically, the actor network $\mu(s; \theta_\mu)$ contributes to determine an action given a state s and weights θ_μ. The critic network $Q(s, a; \theta_Q)$, which is responsible for making the actor more intelligent, outputs the action-value given

the state s, action a, and the weights θ_Q. As in the Q-learning, the transitions are stored in the replay buffer to train actor and critic neural networks. In addition, target networks are used to train actor and critic networks.

The training process of the actor and critic is described in Fig. 2.1. Before starting the training process, transitions are randomly sampled from the replay buffer. Next, the critic loss L_C, which is computed by subtracting a Q-value and target value, is given by

$$L_C = Q(s_t, a_t; \theta_Q) - (r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}; \theta_\mu); \theta_Q)) \tag{2.16}$$

where a Q-value indicates action-value and a target value is calculated by using target networks that greatly improve the stability of training original networks [29]. After the critic is optimized by minimizing the sum of loss in (2.16), the weights of actor network is updated by using gradients of the critic with respect to action [59].

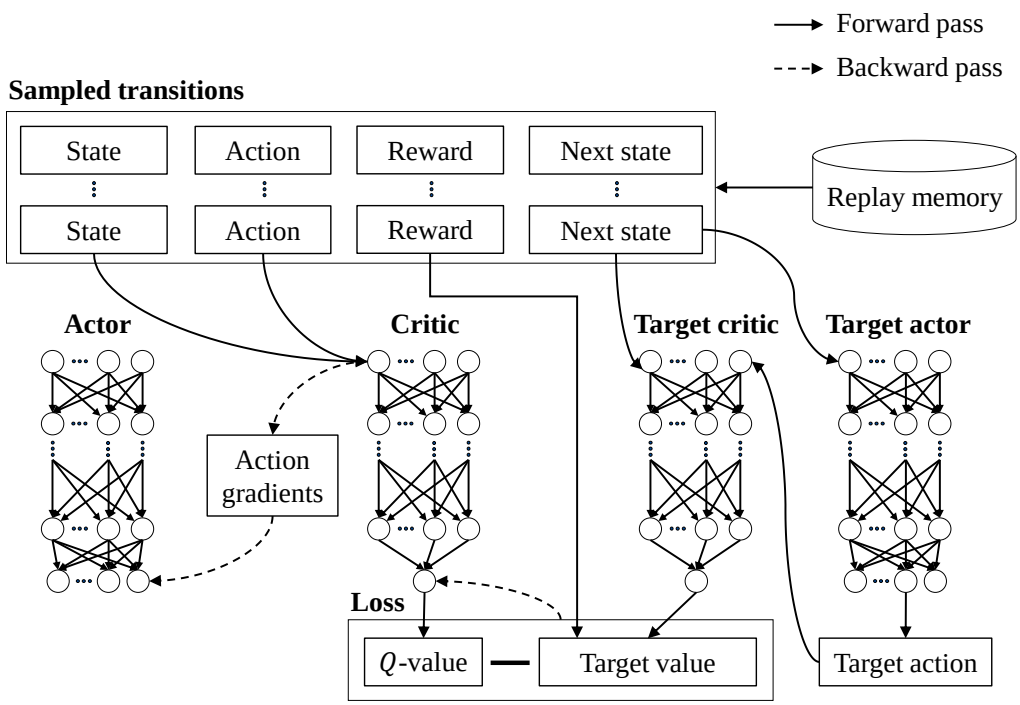


Figure 2.1: Training procedure of DDPG

2.4 Scheduling methods using reinforcement learning-based approaches

RL has been actively employed to solve various decision-making processes that can be formulated as a Markov decision process [52]. The goal of RL is to learn a global optimal policy that maximizes the expectation of cumulative rewards from random explorations. Recently, RL methods have been successfully utilized for solving real-world decision making problems [60, 61, 62, 63, 64, 65, 66, 67, 68].

Table 2.4 presents the summary of previous research for RL-based scheduling methods. First of all, single machine scheduling [19, 20] and job routing [69] have been conducted by using Q-learning which is a representative technique for model-free RL [26]. They utilized tabular Q-learning by discretizing the state and action spaces. However, tabular Q-learning has a limitation when solving problems with large continuous state spaces.

To overcome the drawback of the tabular Q-learning, parallel machine scheduling [21, 22] and flow shop scheduling [23] have been carried out by training a linear function which is responsible for approximating Q-values. They trained a single agent for an entire manufacturing system. Unfortunately, the single agent approach requires a great deal of training time as the size of the state and action space grows exponentially [27]. From the view of the RL-based scheduling, the size blows up rapidly when the numbers of jobs and machines increase.

To reduce the learning complexity inherent to the centralized learning, a multi-agent Q-learning-based method with an NN-based function approximation [70] was developed for minimizing the makespan of a job shop scheduling problem (JSP) [71] when the shop configurations are subject to change. The authors of [70] also

proposed a multi-agent policy gradient method [72] to solve JSP by considering stochastic processing time and intentional delays to a machine. However, there are three limitations when applying these methods to solve the scheduling problem considered in this paper. First, it is difficult for [70] to compute the state feature such as the estimated makespan of a specific job due to the presence of re-entrant flows considered. Second, the agents in [72] need to learn a new policy whenever the configurations such as the numbers of jobs and machines are changed. Finally, since the setup time was not considered in [70] and [72], the variability in the initial setup status cannot be accommodated.

In recent years, deep Q-learning are successfully adopted in various scheduling problems. For instance, [74] and [75] effectively utilizes a deep Q-learning for optimizing dual-objectives in parallel machine scheduling domains. Unfortunately, this methods cannot be applicable to our scheduling problems since each job in the parallel machine shop is finished by processing only one operation. To solve JSP, a multi-class deep Q-network based on the centralized approach [78] was introduced. [79] proposed actor-critic policy gradient method which utilizes a deep neural network as an agent. However, the variability in the initial setup status cannot be addressed by using these methods since they did not consider the setup time and re-entrant flows.

On the other hand, [51] suggested a deep Q-network that considered re-entrant production flows and sequence dependent setups for semiconductor production scheduling. This study took a multi-agent approach where each agent corresponds to a production stage and allocates jobs to one of the machines in the stage. Furthermore, they proposed a two-phase training method to improve the stability of the

Table 2.4: Summary of literature on reinforcement learning-based scheduling methods

Subjects	**Methods**	**References**
Single machine shop	Tabular Q-learning	[19], [20]
Parallel machine shop	Tabular Q-learning	[73]
	Q-learning with a linear function approximation	[21], [22]
	Deep Q-learning	[74], [75]
Flow Shop	Tabular Q-learning	[76]
	Q-learning with a linear function approximation	[23]
Job shop without setup	Tabular Q-learning	[69]
	Multi-agent Q-learning with an NN-based function approximation	[70]
	Multi-agent policy gradient method	[72], [77]
	Deep Q-learning	[78]
	Actor-critic policy gradient	[79]
Job shop with setup	Multi-agent Q-learning	[51]

network. Although the variability in the initial setup status can be addressed, it is still intractable in [51] to deal with the changes in the production requirements and the number of machines since the network is required to be re-trained when such changes occur.

Chapter 3

Deep Reinforcement Learning Based Scheduler: Decentralized Approach

3.1 Scheduling framework

The scheduling method based on a decentralized approach [80] is presented in this section. It consists of two phases: training and test, as illustrated in Fig. 3.1. Discrete event simulation is carried out in both phases to imitate the environment of the scheduling problem introduced in Chapter 3. The training and test problems indicate the scheduling problems to be solved in training and test phases, respectively. In the training phase, the training problem is repeatedly solved by using the simulator. Whenever a setup decision for a machine needs to be made, the simulator provides a state to the Q-network and receives an action. To make the Q-network intelligent, the learning algorithm for the Q-network proposed in [53] is utilized. Specifically, we adopt the replay buffer that contains the set of transitions each of which consists of state, action, reward, and next state. In order to alleviate correlations between successive transitions, transitions are randomly extracted from the replay buffer and used to train the Q-network by the learning algorithm. The weights of the Q-network are periodically replicated to the target Q-network, which improves the convergence stability of the Q-network [53]. After the learning process for the Q-

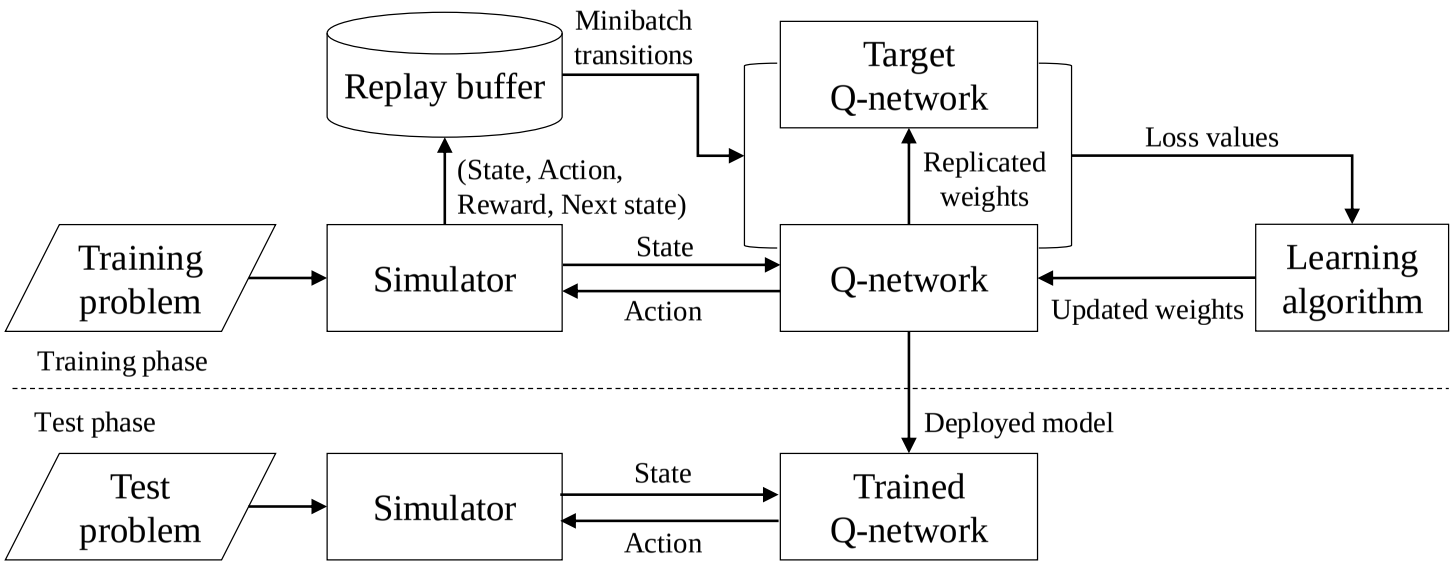


Figure 3.1: Scheduling framework of the decentralized scheduler

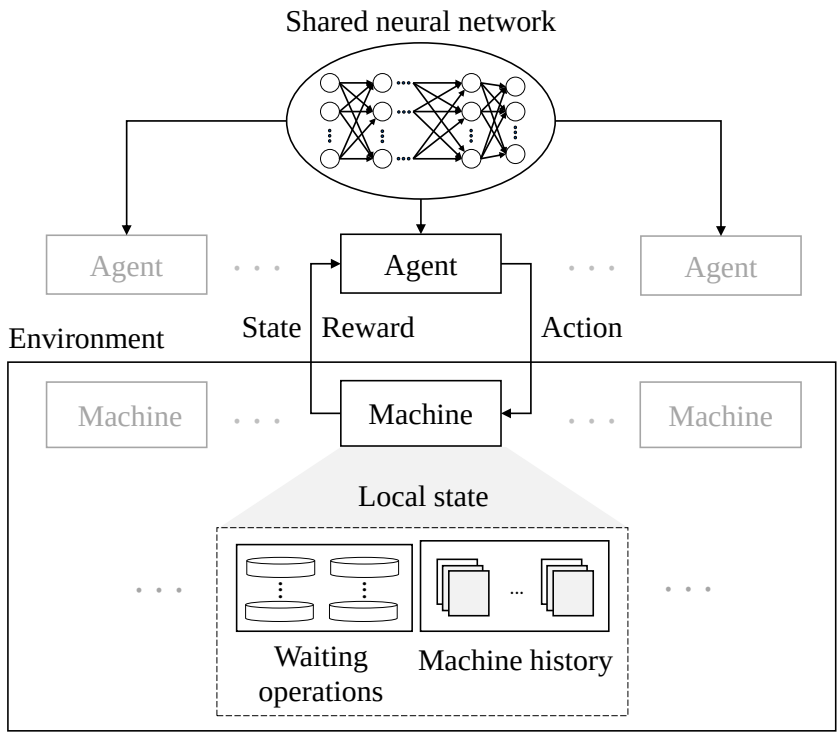


Figure 3.2: Agent-oriented view of the proposed decentralized scheduling approach

network is finished, a test scheduling problem is solved by the trained Q-network in the test phase.

The interactions between the agents and the environment are depicted in Fig. 3.2. The environment indicates a manufacturing system with the constraints addressed in this thesis, and an agent corresponds to one of N_M machines. An agent executes an action when the machine corresponding to the agent is idle and there is one or more waiting operations which can be processed by the machine. We highlight an agent which is capable of executing an action and its corresponding machine in Fig. 3.2. An agent executes an action based on a state and receives a reward from the environment. The state represents a local view of a machine, which includes

the operations waiting to be processed and the machine's historic information. An action represents a next setup type of the machine. To cope with the changes in the number of machines and the initial setup status, an NN is shared and used by all agents. Hence, the proposed method takes advantage of centralized learning by sharing an NN among agents while each agent executes an action in a decentralized manner, as proposed in [28]. This is more scalable than the decentralized learning that aims to train each agent independently [81]. As a result, the proposed method is able to solve new scheduling problems without re-training the Q-network even when the number of machines varies.

3.2 State, action, and reward

Without loss of generality, we propose the representation of state, action, reward, and state transition from the perspective of an agent. The i^{th} state, action, and reward of the agent are denoted as s_i, a_i, and r_i, respectively. Note that subscript i indicates that $(i-1)$ actions have been already executed by the agent. Additionally, $\tau(s_i)$ indicates the time when s_i is observed. The details of s_i, a_i, and r_i are presented in the following.

The state and action representations are designed to accommodate the variabilities in the production requirements and the initial setup status. We first define a feasible action set and a_i.

Action

Let $A(s_i)$ be the feasible action set at s_i, and a_i be one of the elements in $A(s_i)$. If $O_{j,k}$ becomes ready and the machine that can process $O_{j,k}$ is idle at s_i, $O_{j,k}$ is added to $A(s_i)$. If $A(s_i)$ does not contain any operation type, the machine is forced to be idle until there is an operation which can be processed by the machine. To record this situation as a transition, we denote a do-nothing action as δ_0. Since a non-delay schedule [82] is assumed, $\delta_0 \notin A(s_i)$ if there is at least one operation type in $A(s_i)$.

State

s_i is constructed by concatenating four local features of a machine, as shown in Table 3.1. First, it includes the number of waiting operations of $O_{j,k}$ which can be processed by the machine with its current setup status or after a setup change. Since the dimensionality of this feature is independent of the production requirements,

such changes can be accommodated without re-training the Q-network. Next, we represent the setup status as an N_O-dimensional vector using one-hot encoding [83] since the number of setup types is N_O. Through observing this feature, an agent is capable of capturing the current setup status of a machine. Finally, the action and utilization histories of the machine are included as state features. In detail, each attribute of the action history represents the number of times its corresponding action has been executed on the machine. The dimension of this feature is $N_O + 1$ which is equal to the number of all possible actions, including δ_0. The utilization history indicates the respective amounts of processing, setup, and idle time until $\tau(s_i)$.

All values in s_i are normalized to the values between 0 and 1, which reduces the impact caused by the use of different scales across state features. To achieve this, the elements of s_i are divided by their normalizing factors. The number of waiting operations of $O_{j,k}$ is divided by $P(J_j)$. For example, there are two job types, J_1 and J_2, and two operation types for each job type, $(O_{1,1}, O_{1,2})$ and $(O_{2,1}, O_{2,2})$, respectively. Additionally, let $(P(J_1), P(J_2))$ be set to (20, 10). If the current numbers of waiting operations of $O_{1,1}, O_{1,2}, O_{2,1}$, and $O_{2,2}$ are 4, 5, 6, and 7, respectively, the normalized values are $\frac{4}{20}$, $\frac{5}{20}$, $\frac{6}{10}$, and $\frac{7}{10}$. The normalizing factors of action and utilization histories are $(i-1)$ and $\tau(s_i)$, respectively. We remark that s_i can not be revisited since the action history is accumulated. When the first state of a machine is observed, the elements of the action and utilization histories are initialized to zero. The setup status of a machine is not normalized since the value of this feature is 0 or 1.

Table 3.1: Components of a local state

Features	Descriptions	Dimension
Waiting operations	The number of waiting operations of $O_{j,k}$ which can be processed by the machine	N_O
Setup status	Setup type of the machine represented as one-hot encoding	N_O
Action history	The number of performed actions on the machine	$N_O + 1$
Utilization history	The amounts of processing, setup, and idle time of the machine	3

State transition

The state transition from s_i to s_{i+1} is categorized into two classes depending on whether a_i is δ_0 or not. If a_i is δ_0, $\tau(s_{i+1})$ is defined as the earliest time when there is at least one operation whose operation type belongs to $A(s_{i+1})$. As an exception, we allow the state transition to be performed at $t = C_{\max}$ to mark the final do-nothing action of a machine. Otherwise, the state is changed from s_i to s_{i+1} after an operation is finished. If the previous setup type is $O_{j',k'}$ and a_i is $O_{j,k}$, $\tau(s_{i+1})$ is calculated as the sum of $\tau(s_i)$, $p_{j,k}$, and $\sigma_{j',k',j,k}$.

Reward

As a state transition takes place from s_i to s_{i+1}, r_i is observed. In RL, it is an ideal condition that the sum of all rewards corresponds to the objective function [23]. Therefore, r_i is designed to guarantee that maximizing the sum of rewards coincides with minimizing the objective function $C_{\max}$. To satisfy this condition, r_i is defined

as follows.

$$r_i = \begin{cases} -\left(\tau(s_{i+1}) - \tau(s_i) - p_{j,k}\right) & a_i = O_{j,k} \\ -\left(\tau(s_{i+1}) - \tau(s_i)\right) & a_i = \delta_0 \end{cases} \tag{3.1}$$

It follows from (3.1) that r_i indicates the negative value of setup or idle time between s_i and s_{i+1}. Sum of all rewards, called R, is equivalent to

$$R = -\left(N_M C_{\max} - \sum_{j=1}^{N_J} \sum_{k=1}^{N(J_j)} p_{j,k} \times P(J_j) \right) \tag{3.2}$$

Since N_M and the sum of all $p_{j,k} \times P(J_j)$ are constants regardless of the scheduling result, it is clear from (4.1) that maximizing R is equivalent to minimizing $C_{\max}$.

3.3 Example

An example is presented in Table 3.2 in order to clarify the state, action, and reward descriptions. Fig. 3.3 shows a schedule that can be obtained from the example. We describe the scheduling process in terms of the state, action, and reward of each machine. At the beginning of the scheduling process, two jobs and machines are initialized as shown in Table 3.2. In this example, the dimension of the state is 16, and the number of all possible actions is 5. First, we describe the case of M_1. When the ordering of operations types is given as $\langle O_{1,1}, O_{1,2}, O_{1,3}, O_{2,1} \rangle$, s_1 is constructed by concatenating the state components of waiting operations $(1, 0, 0, 1)$, setup status $(1, 0, 0, 0)$, action history $(0, 0, 0, 0, 0)$, and utilization history $(0, 0, 0)$. δ_0 can not be executed at time 0 since $A(s_1)$ is $\{O_{1,1}, O_{2,1}\}$. If a_1 is $O_{1,1}$, r_1 is 0 because both setup and idle time do not occur. When $O_{1,1}$ is finished, a_2 must be $O_{2,1}$ because only $O_{2,1}$ belongs to $A(s_2)$. After a_2 is executed, r_2 is $-(t_4 - t_1 - p_{2,1})$. In case of M_2, $A(s_1)$ is $\{\delta_0\}$. Hence, a_1 must be δ_0, and M_2 remains idle until $O_{1,2}$ becomes ready. Note that a_2, r_2, a_3, and r_3 are $O_{1,2}, 0, O_{1,3}$, and $-(t_3 - t_2 - p_{1,3})$, respectively. The $C_{\max}$ of scheduling results presented in Fig. 3.3 is t_4. Finally, we record the last transition (s_4, a_4, r_4, s_5) to indicate the final do-nothing action of M_2.

Table 3.2: Configuration for the example

Job types	Operations	Alternative machines	Initial setup status	$P(J_j)$
J_1	$O_{1,1}$	M_1	$O_{1,1}$	1
	$O_{1,2}$	M_2	-	
	$O_{1,3}$	M_1, M_2	$O_{1,2}$	
J_2	$O_{2,1}$	M_1	-	1

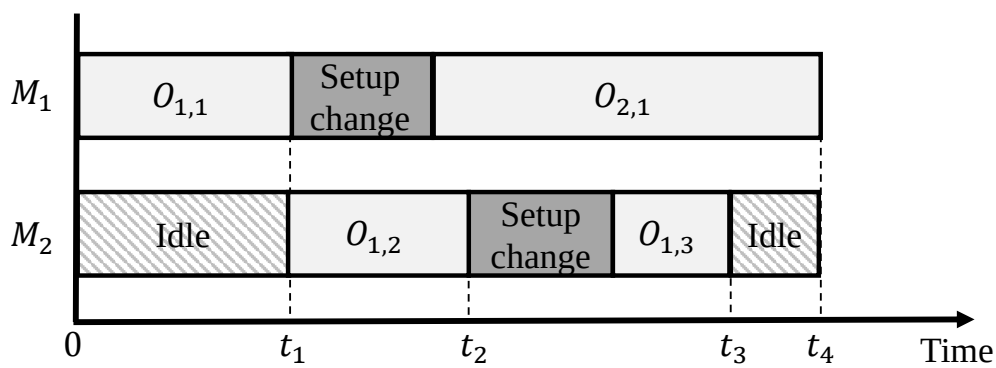


Figure 3.3: Schedule obtained from the example

3.4 Training and test

In the training phase, we employ a fully-connected NN architecture for the Q-network. Specifically, the input and output of the Q-network are the state of a machine and the predicted action-values of the individual actions, respectively. For example, $Q(s_i, a_i)$ indicates the predicted action-value, called Q-value, when a_i is executed at s_i. Since all agents share the same Q-network, we set the dimension of the output as $N_O + 1$ which is equal to the number of all possible actions.

Algorithm 3 describes the training phase of the proposed method. Given a scheduling problem for training the Q-network, lines 3–32 present a scheduling process which is terminated when all operations are finished. We call the completion of one scheduling process as an episode, and e indicates the index of the episode currently being performed. Lines 3–32 continue until e reaches the number of training episodes, denoted as N_E. At the starting time of an episode, each of N_M machines with initial setup status and each of jobs are initialized (Line 3), and time t is set to 0 (Line 4).

At $t = 0$, each agent corresponding to one of N_M machines observes the first state s_1 to execute an action (Lines 6). In this phase, an action is taken according to the ε-greedy policy [52], which has been widely adopted in the RL-based scheduling methods [51, 78]. This enables the Q-network to sufficiently explore the state space [53]. To indicate the decentralized execution, we denote the recent state and action pair of M_l as $H(M_l)$. In line 8, $H(M_l)$ is set to be the pair of s_1 and a_1.

After each agent executes its first action, lines 11–25 are repeated until t reaches $C_{\max}$. Let E_t be the set of machines where an action can be executed at time t. If there are more than one machine in E_t (Line 11), one of the machines needs to

Algorithm 1 Training process of decentralized scheduler

Input: Scheduling problem

Output: Q-network

Initialization : Set network Q with random weight θ, target network $\hat{Q}$ with $\hat{\theta} = \theta$, and replay buffer B to size N_B.
for $e = 1, 2, \ldots, N_E$ **do**
  Initialize jobs and N_M machines
  $t \leftarrow 0$
  for $l = 1, 2, \ldots, N_M$ **do**
    Observe s_1 as the first state of M_l
    Execute action a_1 according to ε-greedy policy
    $H(M_l) \leftarrow (s_1, a_1)$
  end for
  while $t < C_{\max}$ **do**
    Build E_t
    repeat
      Get s, a, and M_l using Algorithm 2
      $(s_i, a_i) \leftarrow H(M_l)$
      Observe r_i and set $(s_{i+1}, a_{i+1}) = (s, a)$
      $H(M_l) \leftarrow (s_{i+1}, a_{i+1})$
      Store transition (s_i, a_i, r_i, s_{i+1}) in B
      Sample N_{TR} transitions $(s_u, a_u, r_u, s_{u+1}) \in B$
      Set $q_u = Q(s_u, a_u; \theta)$
      Set $y_u = r_u + \gamma \mathbb{1}_F(s_{u+1}) \max_{a'} \hat{Q}(s_{u+1}, a'; \hat{\theta})$
      Calculate loss L from (3.3)-(3.4)
      Perform a gradient descent step on L w.r.t. θ
      $E_t \leftarrow E_t \setminus \{M_l\}$
    until $E_t = \emptyset$
    Set t to the earliest time when there is a finished operation
  end while
  for $l = 1, 2, \ldots, N_M$ **do**
    $(s_i, a_i) \leftarrow H(M_l)$
    Observe s_{i+1} and r_i
    Store (s_i, a_i, r_i, s_{i+1}) in B
  end for
  Replace $\hat{\theta} = \theta$ every N_U episodes
end for
return Q-network

Algorithm 2 Machine selection with ε-greedy policy

Input: E_t

Output: State s, action a, and M_l on which a is executed

1: Sample a random number $X \in [0, 1]$
2: **if** $X < \varepsilon$ **then**
3: Select a machine M_l randomly from E_t
4: Observe s of M_l
5: Select a randomly from $A(s)$
6: **else**
7: **for** $M_l \in E_t$ **do**
8: Observe s^l of M_l
9: $(q^l, a^l) = \left(\max_{a \in A(s^l)} Q(s^l, a; \theta), \operatorname{argmax}_{a \in A(s^l)} Q(s^l, a; \theta) \right)$
10: **end for**
11: Select $l = \operatorname{argmax}_l q^l$, $s = s^l$, and $a = a^l$
12: **end if**
13: **return** s, a, and M_l

be selected to execute an action. In this case, we select the machine that yields the maximum Q-value among the idle machines via ε-greedy policy. This is elaborated in Algorithm 2 where s^l, a^l, and q^l indicate the current state of M_l, the action that maximizes its Q-value given s^l, and Q-value of a^l, respectively. Once the state s, action a, and M_l on which a is executed are determined (Line 13), (s_i, a_i) is retrieved from $H(M_l)$ (Line 14).

Based on (s_i, a_i), r_i is observed, (s_{i+1}, a_{i+1}) is set to (s, a), and $H(M_l)$ is updated to (s_{i+1}, a_{i+1}) (Lines 15–16). Then, transition (s_i, a_i, r_i, s_{i+1}) is stored in the replay buffer (Line 17). The replay buffer and its size are denoted as B and N_B, respectively.

In line 17, it is checked whether the number of stored transitions equals to N_B or not. If it is equal to N_B, new transitions replace the oldest ones. To train the Q-network, N_{TR} transitions (s_u, a_u, r_u, s_{u+1}) are randomly sampled from B regardless of the recent stored transitions (Line 18), where N_{TR} is the number of sampled transitions.

Given the sampled transitions, a loss is calculated from a Q-value and target value [53] (Lines 19–21). In line 20, we remark that γ is a discount factor [52], $a' \in A(s_{u+1})$, and $\mathbb{1}_F(s)$ is 1 if s is a final state, and 0, otherwise. In line 22, the learning algorithm performs a gradient descent step with respect to θ on the huber loss L [84], denoted as

$$L = \frac{1}{N_{TR}} \sum_{u=1}^{N_{TR}} f\left(y_u, q_u\right) \tag{3.3}$$

where $f\left(y_u, q_u\right)$ is the loss function given by

$$f\left(y_u, q_u\right) = \begin{cases} \frac{1}{2}\left(y_u - q_u\right)^2 & \text{if } |y_u - q_u| < 1 \\ |y_u - q_u| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{3.4}$$

(3.4) improves the stability of the learning algorithm more than the mean squared error [53].

Lines 13–23 are repeated until E_t becomes empty. At $t = C_{\max}$, the last transition of each machine is observed and stored in B (Lines 27–31). In line 32, the weights of the target Q-network, denoted as $\hat{\theta}$, are periodically replaced to those for the Q-network for ensuring stable convergence as in [53]. The update period is denoted as N_U. Finally, the trained Q-network is returned (Line 34).

In the test phase, the trained network is used to solve a test scheduling problem

Algorithm 3 Test process of decentralized scheduler

Input: Test scheduling problem, trained Q-network

Output: Schedule

1: Initialize jobs and N_M machines
2: $t \leftarrow 0$
3: **for** $l = 1, 2, \ldots, N_M$ **do**
4: Observe s_1 as the first state of M_l
5: Execute action $a_1 = \underset{a \in A(s_1)}{\operatorname{argmax}} Q(s_1, a)$
6: $H(M_l) \leftarrow (s_1, a_1)$
7: **end for**
8: **while** $t < C_{\max}$ **do**
9: Build E_t
10: **repeat**
11: Get s, a, and M_l using Algorithm 2
12: $(s_i, a_i) \leftarrow H(M_l)$
13: Observe r_i and set $(s_{i+1}, a_{i+1}) = (s, a)$
14: $H(M_l) \leftarrow (s_{i+1}, a_{i+1})$
15: $E_t \leftarrow E_t \setminus \{M_l\}$
16: **until** $E_t = \emptyset$
17: Set t to the earliest time when there is a finished operation
18: **end while**
19: Replace $\hat{\theta} = \theta$ every N_U episodes
20: **return** Q-network

even when the production requirements, the number of machines, and the initial setup status of the problem are different from those of training problems. We remark that random actions (line 7 in Algorithm 3 and lines 3–5 in Algorithm 2) are not

executed during the test scheduling process any more. The rest of the process is identical to Algorithm 3 except the lines 17–22 and 27–31 which are required to train a Q-network.

Chapter 4

Deep Reinforcement Learning Based Scheduler: Centralized Approach

In this chapter, we introduce centralized scheduling methods using deep reinforcement learning. The difference between decentralized and centralized approaches is the assumption that one agent accesses all observations and allocates each job to one of the machines. For the clarity of the proposed approach, a semi-markov decision process (SMDP) is defined. Besides, novel definitions of state, action, state transition, and reward for the centralized approach are represented.

Based on the state, action, and reward definitions, we present two DRL-based centralized schedulers. The first one is a rule selection scheduler (CS-R). It determines the best rule-based method among the existing ones at the time when there are more than one waiting operation and one idle machine. After the rule is selected, one of the waiting operations is assigned to one of the idle machines. Unfortunately, CS-R has a limitation when a tie break is required after selecting a rule. To overcome this limitation, a centralized scheduler that selects a job-machine pair given a state (CS-P) is secondly proposed. Whenever a scheduling decision is required, it evaluates all feasible job-machine pairs and selects the best one among the existing pairs. To achieve this, Wolpertinger policy [30] and an actor-critic deep reinforcement learning (ACDRL) method [29] are adopted. The details of CS-R and CS-P including

scheduling frameworks and algorithms are described after introducing SMDP.

4.1 Scheduling framework

In this section, we introduce the definitions of state, action, state transition, and reward for the centralized approach. Since actions for CS-R and CS-P are different from each other, their policies are described in the following sections. The interactions between an agent and the environment are depicted in Fig. 4.1. The environment indicates a manufacturing system with the constraints considered in this thesis. Different from the decentralized approach introduced in the previous chapter, one agent determines all actions at the end of a terminal state.

The scheduling process based on a centralized approach can be modeled as SMDP. Different from markov decision process where decisions can be made only at fixed time steps $t = 0, 1, ...$, an agent model based on SMDP assumes that time intervals between successive decisions are not fixed but random [85]. This assumption is appropriate for our scheduling model based on a centralized approach since time intervals between successive job allocations vary according to states when scheduling decisions are made.

SMDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{T}, r, \gamma, \pi)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ indicates the action space, $\mathcal{P}$ denotes the transition probability that the next state will be s' if action a is executed at state s: $\mathcal{P}(s'|s, a)$, $\mathcal{T}$ presents the expected time until the next decision if action a is executed at state s: $\mathcal{T}(s, a)$, r is a reward function, γ is a discount factor, and π indicates a policy.

Different from the previous chapter, the state, action, and reward are globally defined. We denote the t^{th} state, action, reward as s_t, a_t, and r_t, respectively. Additionally, $\tau(s_t)$ indicates the time when s_t is observed. The details of the state, action, state transition, and reward are presented in the following.

State

To ensure no loss of information from the environment, the state needs to be designed by containing observations of all operations and machines. Unfortunately, this design leads to the result that the state space grows as the numbers of operations and machines increase. Hence, we abstract the observations to reduce the complexity of state space while entailing less sacrifice in its loss.

The state comprises three components which are N_O-dimensional vectors, respectively, as presented in Table 4.1. First, the number of waiting operations for each operation type is represented. Second, the number of in-process operations for each operation type are counted. They indicate operations which are being processed by a machine. Finally, we include the number of idle machines for each setup type. As a result, the state cannot capture observations on the completion rate of in-process operations. To reduce the effect caused by using different scales across state components, their elements are normalized to the values between 0 and 1. In detail, we divide elements of the first two components by the number of operations and the last one by N_M.

Action

An action is executed by allocating an waiting operation to an idle machine. Accordingly, the agent can execute an action when there are more than one idle machine and one operation that can be process on the machine. Since an operation is distinguished by its type, an action is executed by selecting a pair of operation type and machine. We assume that an operation of $O_{j,k}$ is assigned to M_k whose setup type is $O_{j',k'}$ at t. Then, this assignment is valid when satisfying the following three

Table 4.1: Components of a global state

Features	Descriptions	Dimension
Waiting operations	The number of waiting operations of $O_{j,k}$ which can be processed by the machine	N_O
In-process operations	The number of operations of $O_{j,k}$ which are being processed on a machine	N_O
Idle machines	The number of idle machines whose setup type is $O_{j,k}$	N_O

requirements: an operation of $O_{i,j}$ is one of the waiting operations at s_t, M_k is one of the idle machines at s_t, and $M_l \in \mathcal{M}_{j,k}$. We denote the set of possible actions at s_t as A_t. If it is defined one by one, the number of elements that belong to $A(t)$ is equal to $N_O \cdot N_M$ in the worst case. From the perspective of RL, the larger action space leads to difficulties in function approximation and exploration [86, 87, 88, 89, 90]. Therefore, we propose two actions that reduce the complexity of the action space, as will be described in the following two sections.

State transition

After a_t is executed, $\tau(s_{t+1})$ is defined as the earliest time when there is more than there is more than one element in A_{t+1}. This indicates the condition that there are one waiting operation and idle machine. Note that although the state is changed from s_t to s_{t+1}, $\tau(s_t)$ is equal to $\tau(s_{t+1})$ if more than two elements in A_t.

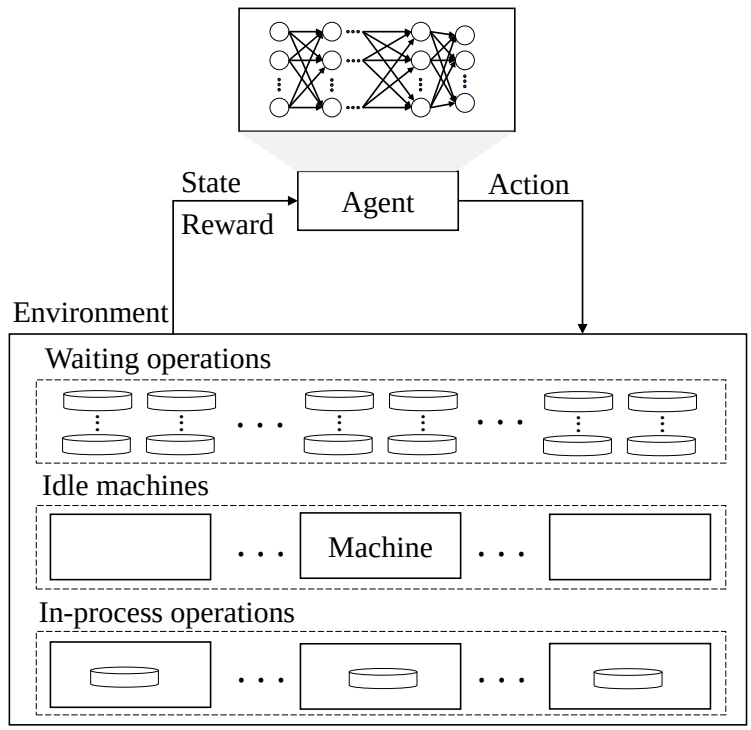


Figure 4.1: Interactions between agent and environment of the centralized approach

Reward

The goal of RL is to learn a policy which maximizes the discounted sum of rewards, denoted as R, received by the agent. It can be formulated as

$$R = \sum_{t=1}^{T} \gamma^t r_t \tag{4.1}$$

where T is a final timestep and γ is a discount factor [52]. Since T is finite, we set γ to 1, which indicates that future rewards are not discounted. As mentioned in the previous chapter, r_t is designed to guarantee that maximizing R is equivalent to minimizing the objective function $C_{\max}$. To satisfy this statement, r_t is defined as

follows.

$$r_t = -\left(\sigma_{j',k',j,k} + \sum_{l=1}^{N_M} d_l(t, t+1)\right) \tag{4.2}$$

where $d_l\,(t, t+1)$ is idle time of M_l between $\tau(s_t)$ and $\tau(s_{t+1})$. Besides, the following equation holds for any schedule.

$$\sum_{j=1}^{N_J} \sum_{k=1}^{N(J_j)} p_{j,k} \times P(J_j) + \sigma_{total} + \sum_{t=1}^{T-1} \sum_{l=1}^{N_M} d_l(t, t+1) = N_M C_{\max} \tag{4.3}$$

where σ_{total} is the sum of total setup time. Equation 4.3 means that the sum of processing time, setup time, and idle time across all machines until $C_{\max}$ is equal to $N_M C_{\max}$. As a result, the following equation is valid:

$$N_M C_{\max} = \sum_{j=1}^{N_J} \sum_{k=1}^{N(J_j)} p_{j,k} \times P(J_j) - \sum_{t=1}^{T} r_t \tag{4.4}$$

Due to the fact that N_M and the sum of total processing time are constants regardless of the scheduling result, the statement is clearly shown from (4.4).

4.2 Rule selection scheduler

The proposed rule selection method is presented in this section. Fig. 4.2 depicts the scheduling process of the rule selection method. We employ a fully-connected NN architecture for the Q-network. Different from the decentralized approach, the outputs of Q-network for CS-R are rule-based methods. Specifically, the input and output of the Q-network are the state and the predicted values of the individual rules, respectively. Given a state from the environment, a rule is obtained by the Q-network among its candidates. By deploying the selected rule, one of the waiting operations is allocated to an idle machine. Next, we describe the details of action and rule-based methods considered in the following.

Action

a_t is defined as the selected rule by the Q-network given s_t. Table 4.2 shows the rule-based methods considered in CS-R. Since 8 rules are considered for the Q-network, the dimension of its input and output are $3N_O$ and 8, respectively, and these rules are elements of A_t. The number of elements of A_t is equal to 8 since it is assumed that all rules are always applicable on any state. When deploying a_t, we select the operation and machine whose indexes are the smallest among waiting operations and idle machines, respectively, which indicates the tie-break criterion.

Training phase

CS-R consists of two phases: training and test, as depicted in Fig. 3.1 of the previous section. Algorithm 4 indicates the training phase of the rule selection scheduler. Given scheduling problems for training the Q-network, lines 2–16 present a CS-

Table 4.2: Descriptions of rule-based methods

Rules	**Descriptions**
SSU (shortest setup unit)	Select an operation and a machine with shortest setup time
SPTSSU (shortest sum of processing time and setup time)	Select an operation and a machine with minimum sum of processing and setup time
MOR (most operation remaining)	Select an operation with the largest number of successor operations
LOR (least operation remaining)	Select an operation with the smallest number of successor operations
MWR (most work remaining)	Select an operation with the largest amount of remaining processing time
LWR (least work remaining)	Select an operation with the smallest amount of remaining processing time
SPT (shortest processing time)	Select an operation with the shortest processing time
LPT (longest processing time)	Select an operation with the longest processing time

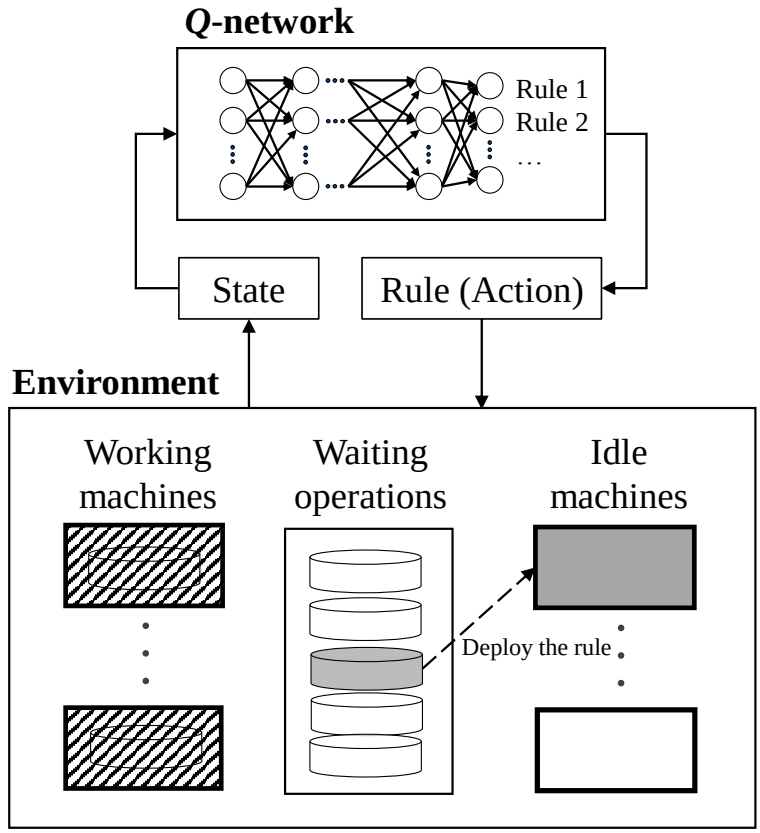


Figure 4.2: Scheduling process of the rule selection method

R's scheduling process which is terminated when all operations are finished. The process continues until e reaches N_E. At the starting time of an episode, each of N_M machines with an initial setup status and each of jobs are initialized (Line 3), and the agent observes the first state s_1 (Line 4). The agent executes actions until t reaches T, where T is the timestep when $\tau(s_T) = C_{\max}$. In this phase, an action is

Algorithm 4 Training process of rule selection scheduler

Input: Scheduling problems

Output: Q

Initialization : Set network network Q with random weight θ, target network $\hat{Q}$ with $\hat{\theta} = \theta$, and replay buffer B to size N_B
for $e = 1, 2, \ldots, N_E$ **do**
  Initialize jobs and N_M machines
  Observe s_1
  for $t = 1, 2, \ldots, T$ **do**
    Execute a_t according to (4.5)
    Observe s_{t+1} and r_t
    Store (s_t, a_t, r_t, s_{t+1}) in B
    Sample a random minibatch of N_{TR} transitions $(s_u, a_u, r_u, s_{u+1}) \in B$
    Calculate loss L from (4.6)-(4.9)
    Perform a gradient descent step on L with respect to θ
  end for
  Replace $\hat{\theta} = \theta$ every N_U episodes
end for
return Q

executed by ε-greedy policy [52]

$$a_i = \begin{cases} \text{rand}\left(A(s_i)\right) & \text{with probability } \varepsilon \\ \underset{a \in A(s_i)}{\text{argmax}}\, Q(s_i, a; \theta) & \text{otherwise} \end{cases} \tag{4.5}$$

where rand$(\cdot)$ outputs a random element that belongs to the input set, ϵ is a probability that a random action is selected, and θ indicates the weights of Q-network.

According to (4.5), a_t is executed (Line 6). As the state transition takes place from s_t to s_{t+1}, r_t is observed (Line 7). Then, transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer B (Line 8). Note that B and N_B indicate the replay buffer and its size, respectively. To train the Q-network, N_{TR} transitions (s_u, a_u, r_u, s_{u+1}) are randomly sampled from B where N_{TR} is the number of sampled transition (Line 9). Given the sampled transitions, a loss is calculated from a Q-value and target value (Line 10). The Q-value, called q_u, is

$$q_u = Q\left(s_u, a_u; \theta\right) \tag{4.6}$$

and the target value y_u [53] is

$$y_u = \begin{cases} r_u & \text{if } s_{u+1} \text{ is terminal} \\ r_u + \gamma \max\limits_{a'} \hat{Q}(s_{u+1}, a'; \hat{\theta}) & \text{otherwise} \end{cases} \tag{4.7}$$

where γ is a discount factor [52] and $a' \in A(s_{u+1})$. In line 11, the learning algorithm performs a gradient descent step with respect to θ on the huber loss L [84], denoted as

$$L = \frac{1}{N_{TR}} \sum_{u=1}^{N_{TR}} f\left(y_u, q_u\right) \tag{4.8}$$

where $f\left(y_u, q_u\right)$ is the loss function given by

$$f\left(y_u, q_u\right) = \begin{cases} \dfrac{1}{2}\left(y_u - q_u\right)^2 & \text{if } |y_u - q_u| < 1 \\ |y_u - q_u| - \dfrac{1}{2} & \text{otherwise} \end{cases} \tag{4.9}$$

Algorithm 5 Test process of the rule selection scheduler

Input: Scheduling problem, trained rule selection Q-network

Output: Schedule

1: Initialize jobs and N_M machines
2: Observe s_1
3: **for** $t = 1, 2, \ldots, T$ **do**
4: Execute $a_t = \underset{a \in A(s_t)}{\operatorname{argmax}} Q(s_t, a)$
5: Obtain the selected operation O_* and machine M_* from a_t
6: Add $(O_*, M_*, \tau(s_t))$ to schedule
7: Observe s_{t+1} and r_t
8: **end for**
9: **return** Schedule

(3.4) improves the stability of the learning algorithm more than the mean squared error [53]. In line 13, the weights of target Q-network, denoted as $\hat{\theta}$, are replaced every N_U episodes to those of the Q-network for ensuring stable convergence as in [53]. Finally, the trained Q-network is returned (Line 15).

Test phase

In this phase, the trained Q-network is used to solve a test scheduling problem even when the production requirements, the number of machines, and the initial setup status of the problem are different from those of training problems. Given a test scheduling problem, Algorithm 5 describes the scheduling process that is performed by using a trained Q-network. Different from the training phase, random actions are not executed during the scheduling process any more (Line 4). Whenever an action is executed on a machine at time $\tau(s_t)$, the triple of the operation, machine, and

time is stored to obtain a schedule which contains action sequences of all machines (Lines 5 and 6). The rest of the process is identical to Algorithm 4.

4.3 Job-machine pair selection scheduler

In this section, we introduce the job-machine pair selection scheduler based on actor-critic deep reinforcement learning (ACDRL). The agent consists of two neural networks called the actor and critic. Fig. 4.3 illustrates the scheduling process based on interactions between the environment and the actor. The environment indicates the considered FJSP with the constraints and assumptions presented in the previous section. Given a state from the environment, the actor helps to decide one of the possible actions. An action is executed by allocating an operation to one of its alternative machines that is currently idle. Then, a reward and next state are observed from the environment. To train actor and critic neural networks, a transition, which is composed of a state, action, reward, and next state, is stored in the replay buffer. The details of action and policy are proposed in the following.

Action

As described in Section 5.1, an action is executed by selecting a pair of operation type and machine. However, the number of all possible actions is $N_O \cdot N_M$ in the worst case if it is defined one by one. To reduce the complexity of the action space, we utilize four continuous features in which an action is represented into a 4-dimensional vector regardless of N_O and N_M. a_t is defined as

$$a_t = \left(p_{j,k}, \sigma_{j',k',j,k}, N(J_j) - k, \sum_{u=k+1}^{N(J_j)} p_{j,v} \right) \tag{4.10}$$

Each dimension indicates processing time, setup time, the remaining number of operations, and remaining processing time, respectively. To alleviate the impact

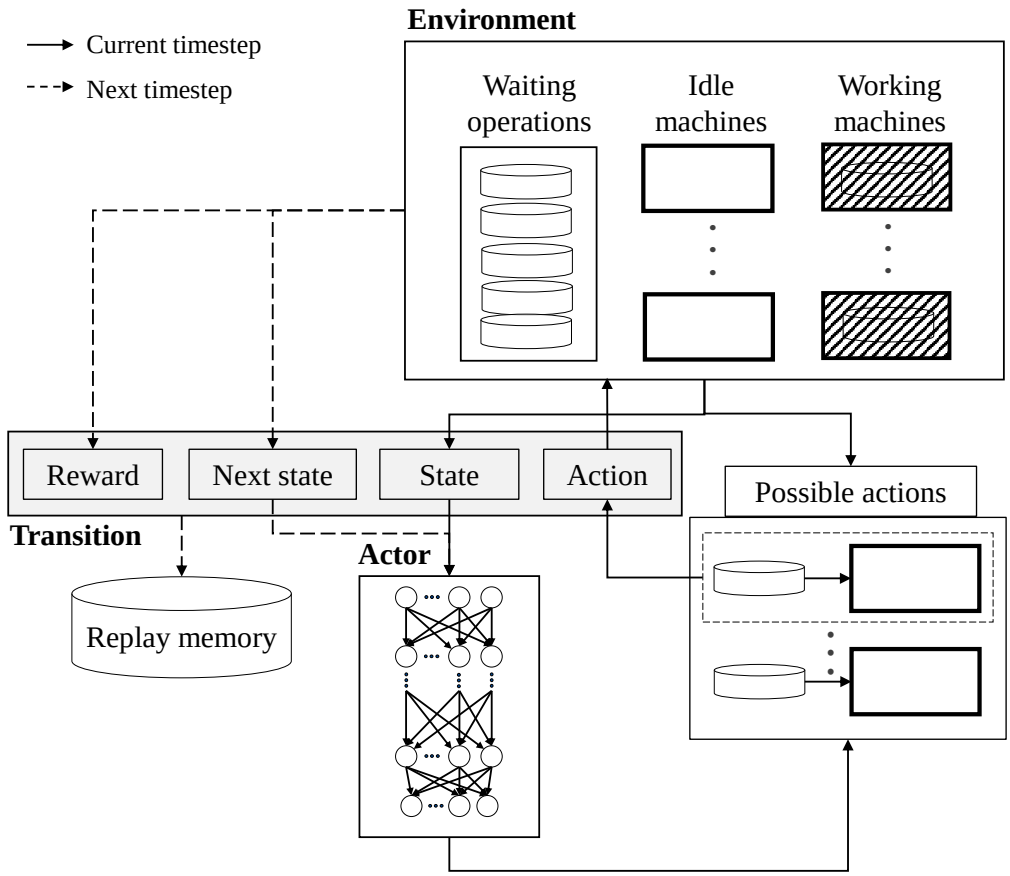


Figure 4.3: Scheduling framework of job-machine pair selection scheduler

caused by use of different scales across features, we utilize the min-max normalization [91] as follows:

$$f' = 2 \cdot \frac{f - f_{min}}{f_{max} - f_{min}} - 1 \tag{4.11}$$

where f and f' respectively stand for a feature value and a normalized feature value, and f_{min}, f_{max} refer to the minimum and maximum values among the possible values of f.

Policy

A policy is a mapping from the state space to the action space. Let $\hat{a}_t$ be an action provided by the actor at s_t. Although $\hat{a}_t$ is successfully represented in a continuous space $\mathbb{R}^4$, it is not likely to be a valid action, i.e. $\hat{a}_t \notin A_t$. To accommodate the inconsistency between a_t and $\hat{a}_t$, we adopt the wolpertinger policy [30]. It is defined as mapping from $\hat{a}_t$ to an element in A_t by using the following L_2 distance:

$$a_t = \underset{a \in A_t}{\operatorname{argmin}} \left(\| a - \hat{a}_t \|_2 \right) \tag{4.12}$$

where $\|\cdot\|_2$ is L_2-norm. To clarify the difference between a_t and $\hat{a}_t$, we call $\hat{a}_t$ a proto-action at t.

Training phase

We utilize a fully-connected NN for both actor and critic networks. Let μ, Q, μ', and Q' be the actor, critic, target actor, and target critic networks, respectively. In addition, we denote the weights of the actor, critic, target actor, and target critic networks as θ_Q, θ_μ, θ'_Q, and $\theta_{\mu'}$, respectively. The actor network receives a state s and provides a proto-action $\hat{a}$. Meanwhile, the input of critic network is the state s and action a, and its output is $Q(s, a)$ that indicates the predicted action-value. Accordingly, the input and output dimensions of actor and critic networks are $3N_O$ and 4, and $3N_O + 4$ and 1, respectively.

Algorithm 6 describes the training phase of the job-machine pair selection scheduler. Given a scheduling problem for training the actor and critic networks, μ and Q, lines 3–35 present a scheduling process which is terminated when all operations

are finished. We define the process as an episode, and e indicates the index of the episode currently being performed. Lines 3–19 continue until e reaches the number of training episodes N_E. At the starting time of an episode, each of N_M machines with an initial setup status and each of jobs are initialized (Line 3), and the agent observes the first state s_1 to execute an action (Line 4). After observing s_1, lines 6–18 are repeated until t reaches T.

When an action is executed in this phase, we utilize an Ornstein-Uhlenbeck process [92] to efficiently explore continuous action spaces. Specifically, an proto-action at timestep t is determined by adding noise sampled from an Ornstein-Uhlenbeck process, denoted as $\mathcal{N}_t$, to actor's output $\mu(s_t;\theta_\mu)$ (Line 6). This enables the agent to generate temporally correlated exploration, which has been adopted in DRL with continuous action spaces [29]. After a_t is executed by employing Wolpertinger policy (Line 7), s_{t+1} and r_t are observed (Line 8). To train both actor and critic networks, transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer B (Line 9), and N_{TR} transitions (s_u, a_u, r_u, s_{u+1}) are randomly sampled from B (Line 10).

Given the sampled transitions, a target action is obtained by using target actor and wolpertinger policy (Lines 11–12). Note that A_{u+1} can be built from s_{u+1} since the state contains observations of waiting operations and idle machines. To compute the loss, a Q-value and target value are calculated (Line 13). The Q-value, called q_u, is

$$q_u = Q\left(s_u, a_u; \theta_Q\right) \tag{4.13}$$

Algorithm 6 Training process of job-machine pair selection scheduler

Input: Scheduling problems

Output: μ and Q

Initialization : Set critic network Q with random weight θ_Q, actor network μ with random weight θ_μ, target critic network Q' with $\theta_{Q'} = \theta_Q$, target actor network μ' with $\theta_{\mu'} = \theta_\mu$, and replay buffer B with size N_B

for $e = 1, 2, \ldots, N_E$ **do**

  Initialize jobs and N_M machines

  Observe s_1

  for $t = 1, 2, \ldots, T$ **do**

    Obtain $\hat{a} = \mu(s_t; \theta_\mu) + \mathcal{N}_t$

    Execute $a_t = \underset{a \in A_t}{\operatorname{argmin}} (\|a - \hat{a}\|_2)$

    Observe s_{t+1} and r_t

    Store (s_t, a_t, r_t, s_{t+1}) in B

    Sample a random minibatch of N_{TR} transitions $(s_u, a_u, r_u, s_{u+1}) \in B$

    Obtain $\hat{a} = \mu'(s_{u+1}; \theta_{\mu'})$

    Select action $a_{u+1} = \underset{a \in A_{u+1}}{\operatorname{argmin}} (\|a - \hat{a}\|_2)$

    Calculate y_u and q_u from (4.13)-(4.14)

    Compute the huber loss L from (4.8)-(4.9)

    Update Q by minimizing L

    Update μ using the sampled gradient in (4.16)

    $\theta_{Q'} \leftarrow \omega\theta_Q + (1 - \tau)\theta_{Q'}$

    $\theta_{\mu'} \leftarrow \omega\theta_\mu + (1 - \tau)\theta_{\mu'}$

  end for

end for

return μ and Q

and the target value y_u [29] is

$$y_u = \begin{cases} r_u & \text{if } s_{u+1} \text{ is terminal} \\ r_u + \gamma Q'\left(s_{u+1}, a_{u+1}; \theta_{Q'}\right) & \text{otherwise} \end{cases} \tag{4.14}$$

where γ is a discount factor [52]. In line 14, the learning algorithm performs a gradient descent step with respect to θ on the huber loss L [84], as described in the previous subsection. Moreover, θ_μ is updated by computing the following gradient [59] with respect to θ_μ (Line 16):

$$\nabla_{\theta_\mu} J = \nabla_{\theta_\mu} \mathbb{E}_{s_1}[R] \approx \mathbb{E}[\nabla_{\theta_\mu} Q(s, a; \theta_Q)] \tag{4.15}$$

where $s = s_t$, $a = \mu(s_t; \theta_\mu)$. Note that the definition of R is described in (4.1) and J indicates the expected return from the initial state s_1. By applying the chain rule to J, the following sampled gradient can be obtained:

$$\nabla_{\theta_\mu} J \approx \frac{1}{N_{TR}} \sum_{u=1}^{N_{TR}} \nabla_a Q(s, a; \theta_Q) \nabla_{\theta_\mu} \mu(s; \theta_\mu) \tag{4.16}$$

where $s = s_u$ and $a = \mu(s_u)$. In [59], it was proved that (4.16) is the policy gradient which indicates the gradient of the policy's performance. In lines 17–18, the target actor and critic networks are updated. The updating weights of target networks is denoted as ω. By setting $\omega << 1$, $\theta_{\mu'}$ and $\theta_{Q'}$ are updated by having them slowly track the original networks, μ and Q. This improves the stability of training actor and target networks sicne the weights of target networks are constrained not to change quickly [29], Finally, μ and Q are returned (Line 21).

Test phase

The test phase of job-machine pair selection scheduler is similar to that of CS-R, as described in the previous section. Given a test scheduling problem, Algorithm 7 describes the scheduling process that is performed by using a trained μ. Note that the critic network is not used in this phase since it only contributes to training μ. Additionally, a noise $\mathcal{N}$ is not used during the scheduling process any more (Line 4). Whenever an action is executed on a machine at time $\tau(s_t)$, the triple of the operation, machine, and time is stored to obtain a schedule which contains action sequences of all machines (Lines 5 and 6). The rest of the process is identical to Algorithm 6.

As a result, the trained actor network is used to solve test scheduling problems even when the production requirements, the number of machines, and the initial setup status of the problem are different from those of training problems. Furthermore, the complexity of action exploration does not grow even when N_O increases since the output dimension of actor network is independent of N_O.

Algorithm 7 Test process of job-machine pair selection scheduler

Input: Scheduling problem and μ

Output: Schedule

1: Initialize jobs and N_M machines
2: Observe s_1
3: **for** $t = 1, 2, \ldots, T$ **do**
4: Obtain $\hat{a} = \mu(s_t)$
5: Execute $a_t = \underset{a \in A_t}{\operatorname{argmin}} (\|a - \hat{a}\|_2)$
6: Obtain the selected operation O_* and machine M_* from a_t
7: Add $(O_*, M_*, \tau(s_t))$ to schedule
8: Observe s_{t+1} and r_t
9: **end for**
10: **return** Schedule

Chapter 5

Experimentation results

5.1 Datasets

In the experiments, we examined the proposed methods on real-world FJSP with sequence dependent setups. The considered problem is multi-chip products (MCPs) scheduling for real-world semiconductor packaging lines described in [6, 7]. Semiconductor packaging lines include four major production stages which are backlap, saw, die attach (DA), and wire bond (WB) [44, 93], as presented in Fig. 5.1. MCPs are produced by stacking multiple chips in a single package to achieve the increase in density and the reduction in size compared to single-chip products [48]. As a result, when producing MCPs, it is required to revisit DA and WB stages repeatedly. This leads to forming bottlenecks and increasing setup changes when producing various types of MCPs [7]. Since the frequent setup changes might incur increasing makespan, scheduling semiconductor packaging facilities becomes one of the most important research issues for semiconductor manufacturers [44]. Furthermore, DA and WB stages in a semiconductor packaging line become significantly complicated due to the presence of several constraints related to sequence dependent setups and alternative machines [3, 4, 5]. Therefore, we select the semiconductor packaging line scheduling problems to examine the effectiveness and robustness of the proposed

methods.

During the training phase of the proposed networks, it takes a considerable amount of time to train the networks for large-scale scheduling problems. To reduce the training time, we attempt to train the networks using small-scale problems which are generated by proportionally scaling down the number of machines and production requirements of large-scale problems. We prepared 15 datasets to verify this attempt, as shown in Table 5.1. The number of operations is averaged over the scheduling problems and set to require more than 2 days to complete the operations, as described in [7]. Datasets 1, 6, 11, and 16 are prepared to train and validate the Q-networks, and the other datasets are used to test the robustness of the Q-networks in terms of the variabilities in production requirements, the number of machines, and initial setup status. Specifically, datasets 1, 6, 11, and 16 were built by varying N_J and N_O while maintaining the number of machines. Datasets 2 to 5, datasets 7 to 10, datasets 12 to 15, and datasets 17 to 20 were generated by multiplying both the number of machines and the average number of operations in datasets 1, 6, 11, and 16 by 2 to 5 times, respectively.

Each dataset has 30 scheduling problems whose production requirements of job types were perturbed by 10 percent and the initial setup status was randomly generated. For each of datasets 1, 6, 11, and 16, only one problem in a dataset was used to train a Q-network while the others were used to validate the trained Q-network. In particular, the initial setup status of a training problem was set randomly for every episode, which has the effect of training a Q-network with a number of problems whose initial setup statuses are different from each other. This enables the trained Q-network to accommodate the various initial setup status of scheduling problems

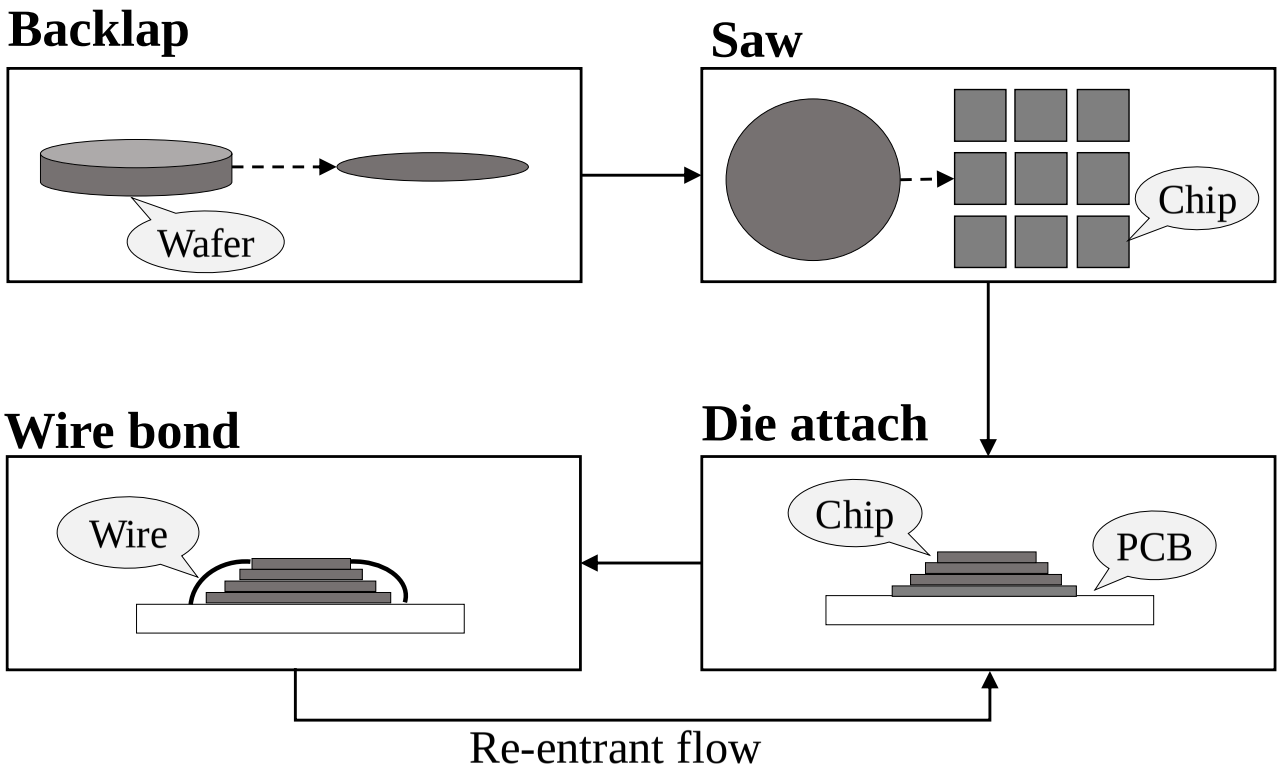


Figure 5.1: Main production stages of semiconductor packaging lines.

Table 5.1: Description on datasets for performance comparison

Dataset No.	N_M	N_J	N_O	The average number of operations
1	35	7	28	610
2	70	7	28	1220
3	105	7	28	1830
4	140	7	28	2440
5	175	7	28	3050
6	35	10	47	620
7	70	10	47	1240
8	105	10	47	1860
9	140	10	47	2480
10	175	10	47	3100
11	35	12	64	590
12	70	12	64	1180
13	105	12	64	1770
14	140	12	64	2360
15	175	12	64	2950
16	35	15	100	600
17	70	15	100	1200
18	105	15	100	1800
19	140	15	100	2400
20	175	15	100	3000

in the test phase. Experiments were conducted on a Core i7 3.6 GHz PC with 4-GB memory.

5.2 Training details

In this section, we introduce the training details of DS, CS-R, and CS-P. In RL, determining the values of hyperparameters is crucial for the performance of the networks. Unfortunately, it is challenging to find the optimal values due to the large search space of hyperparameters. Therefore, by performing the random search [94], we found the values that yielded the best performance and they are presented in Tables 5.2 and 5.3.

Since DS and CS-R utilize the Q-network, the same hyperparameters are used in the experiments, as presented in Table 5.2. The Q-network was trained by RMSProp [95] as the gradient decent algorithm. It has the rectifier linear unit [96] as an activation function except for the last layer which is responsible for representing negative action-values.

On the other hand, the hyperparameters for training actor and critic networks are shown in Table 5.3. The networks was trained by RMSProp [95] as the gradient decent algorithm. Different from DS and CS-R, DS uses the leaky rectifier linear unit [97] as an activation function except for the last layer which is responsible for representing negative action-values. Furthermore, the layer normalization [98] is performed for each layer in the actor and critic networks.

Table 5.2: Hyperparameters for DS and CS-R

Hyperparameters	Value
The number of hidden layers	4
The numbers of nodes in first, second, third, and fourth hidden layers	64, 32, 16, 8
Learning rate of RMSProp	2×10^{-4}
Decay factor of learning rate in RMSProp	0.99
Momentum of RMSProp	0
Min sqaured gradient of RMSProp	10^{-6}
The probability that random action is selected, ε	0.1
Decay factor of ε per episode	10^{-4}
Replay buffer size, N_B	10^5
Minibatch size, N_{TR}	64
Discount factor, γ	0.9
Target Q-network update frequency, N_U	20 episodes

Table 5.3: Hyperparameters for CS-P

Hyperparameters	Value
The number of hidden layers in the actor network	3
The numbers of nodes in first, second, and third hidden layers of the actor network	64, 64, 64
Learning rate of the actor network	10^{-4}
The number of hidden layers in the critic network	3
The numbers of nodes in first, second, and third hidden layers of the critic network	64, 64, 64
Learning rate of the critic network	5×10^{-4}
Decay factor of learning rate in RMSProp	0.99
Momentum of RMSProp	0
Min sqaured gradient of RMSProp	10^{-6}
Mean value of the Ornstein-Uhlenbeck process	0.2
Standard deviation of the Ornstein-Uhlenbeck process	0.2
Replay buffer size, N_B	10^5
Minibatch size, N_{TR}	64
Discount factor, γ	0.9
The updating weights of target networks, ω	10^{-3}

During the training phases of our methods, the networks were stored whenever the target networks were updated. Subsequently, scheduling problems for validation were solved by using the stored networks. The validation phase was carried out due to the following two reasons: First, it helps to reduce overfitting in the networks. Second, in the training phase, it is difficult to identify which network yields the best performance in terms of $C_{\max}$. To identify the best performance, random explorations were not performed. Specifically, during the validation phases, ε was set to 0 for DS and CS-R, and a random noise was not added for CS-P. Consequently, we selected networks with the lowest $C_{\max}$ at the validation phase and used the selected networks to test their performance.

$C_{\max}$ (in hours) curves of the training and validation phases for DS, CS-R, and CS-P are shown in Figs. 5.2, 5.3, and 5.4, respectively. In the plots in Fig. **??**, the x and y coordinates of each point indicate the cumulative number of episodes and the average $C_{\max}$ of scheduling problems, respectively. We present the convergence performance up to 4,000 episodes since the performance changes of the validation phase were negligible when the number of training episodes exceeds 4,000. As shown in Figs. 5.2 and 5.3, it was observed that $C_{\max}$ decreased at different speeds for each dataset. This observation can be attributed to the difference in the complexity of obtaining schedules of low $C_{\max}$. This implies that it is difficult to know in advance the number of training episodes necessary for producing satisfactory $C_{\max}$. Furthermore, the number even varies according to the datasets.

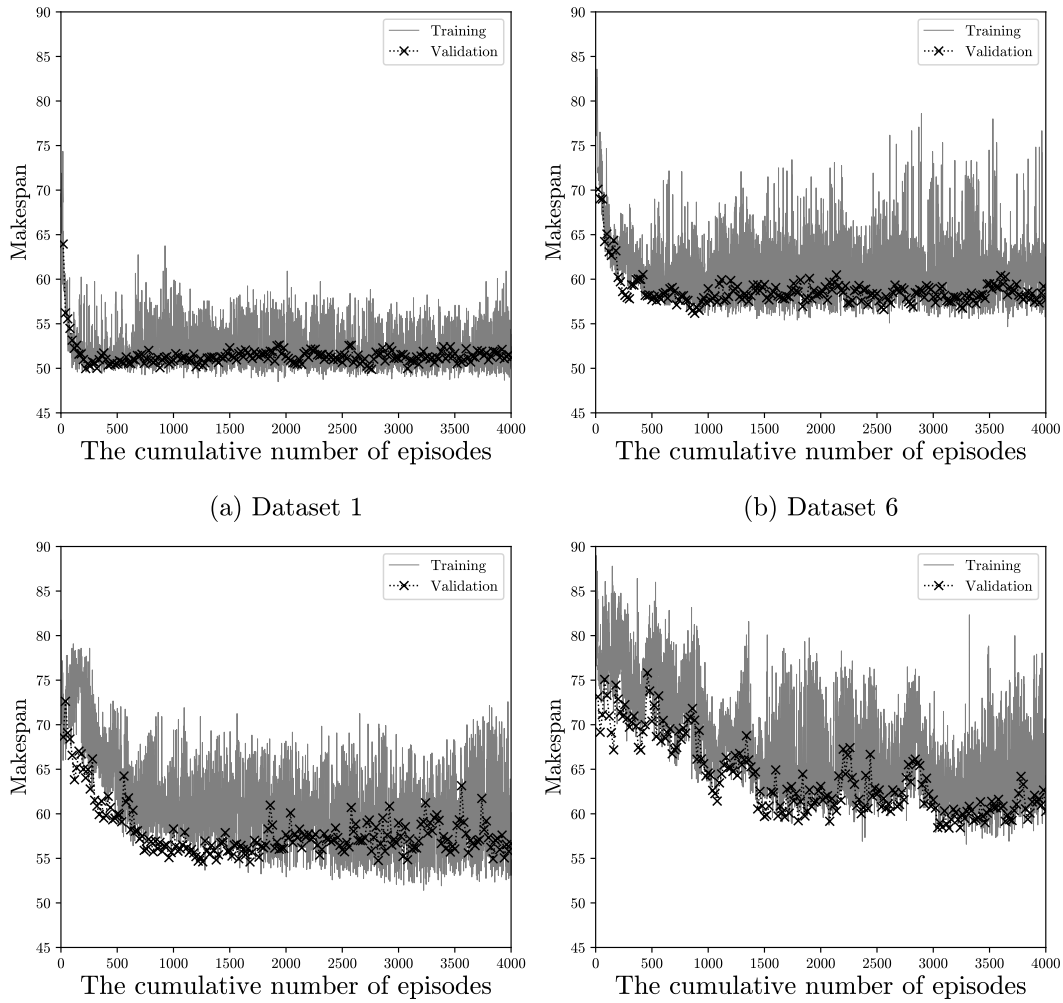
Meanwhile, fluctuations were observed in both the training and validation curves. Fluctuations of the training curves were caused by the ε-greedy policy and the randomly generated initial setup status at each episode. From Fig. 5.2a to Fig. 5.2d

Table 5.4: Training time (in hours) results of DS, CS-R, and CS-P

Dataset No.	DS	CS-R	CS-P
1	10.24	10.28	27.15
6	11.49	10.93	30.71
11	11.77	11.23	29.54
16	12.37	12.27	36.59

and Fig. 5.3a to Fig. 5.3d, the amplitude of fluctuations becomes larger. This is because more diverse schedules in terms of $C_{\max}$ are likely to be produced as N_O increases. On the other hand, the validation curve tends to show less fluctuations and more decreasing tendency with respect to $C_{\max}$ than the training curve.

In Fig. 5.4, fluctuations of the training curves were caused by the Ornstein-Uhlenbeck process and the randomly generated initial setup status at each episode. From Fig. 5.4a to Fig. 5.4d , the amplitude of fluctuations seems to be similar. Compared to DS and CS-R, lower $C_{\max}$ results are obtained with less training episodes. This may be attributed to continuous action features and the Wolpertinger policy which help the actor and critic networks to effectively solve the scheduling problems although more diverse schedules in terms of $C_{\max}$ are likely to be produced as N_O increases. For datasets 1, 6, 11, and 16, we selected the best performing networks, and they were used to conduct performance comparisons with the conidered baseline methods. Additionally, training time (in hours) results of the proposed methods are presented in Table 5.4. It was observed that the training time of CS-P was longer than those of DS and CS-R. This may be attributed to the additional calculation of euclidean distances between each available action and a proto-action.



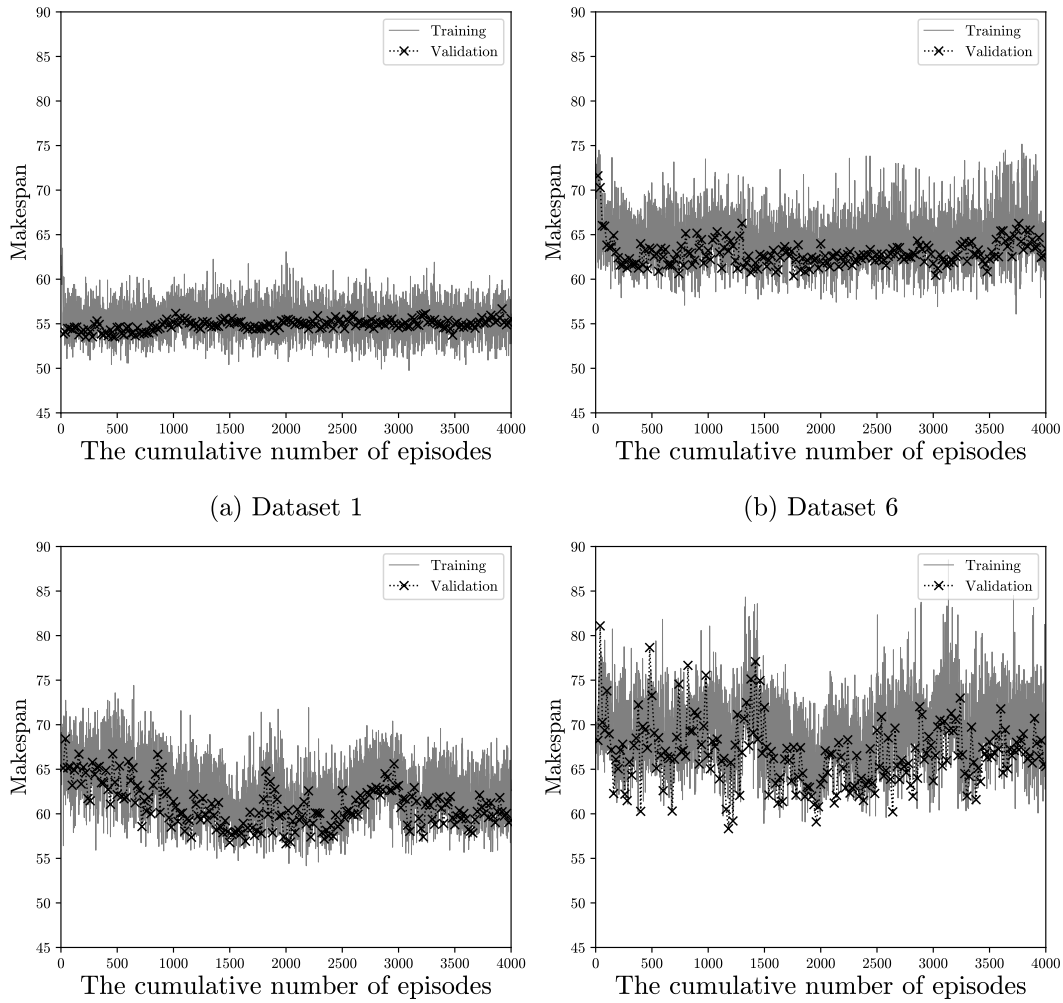
(a) Dataset 1

(b) Dataset 6

(c) Dataset 11

(d) Dataset 16

Figure 5.2: Training and valiation results of DS with respect to the cumulative number of episodes



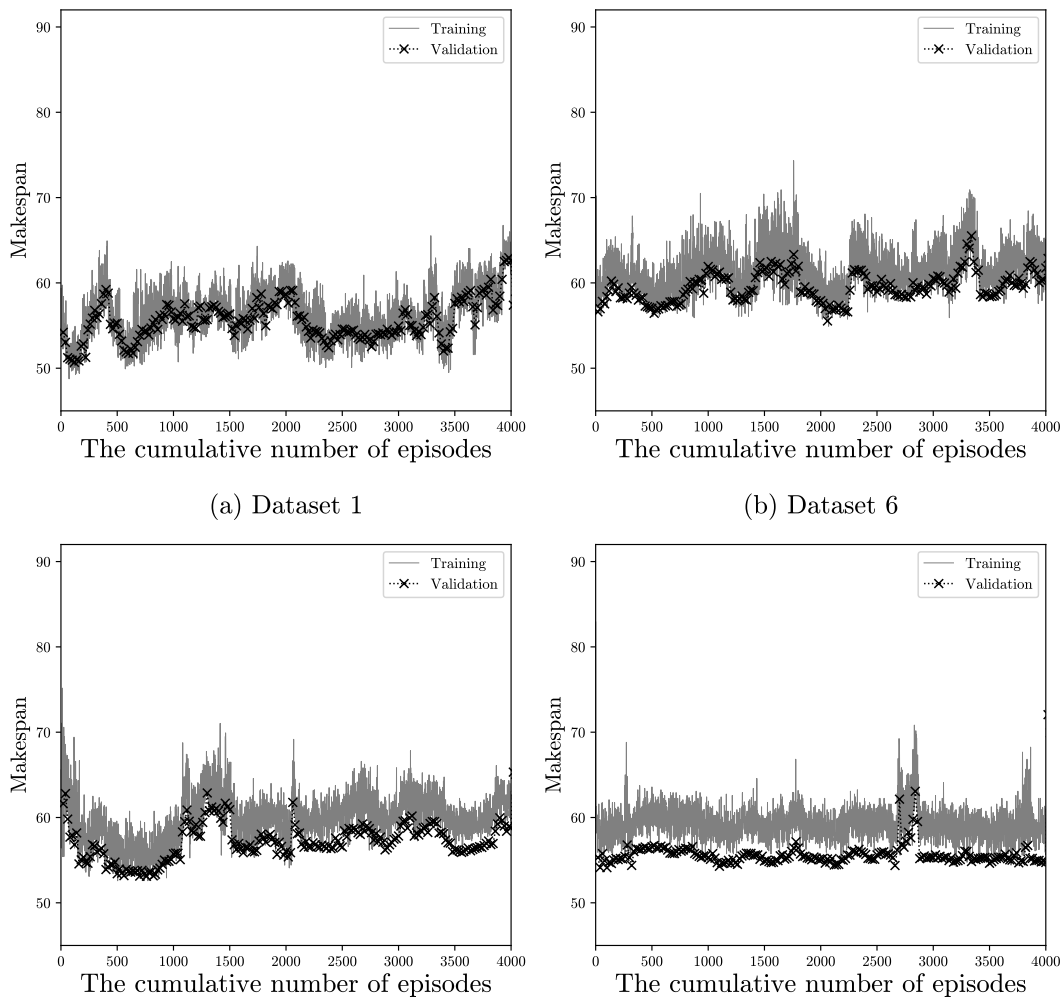
(a) Dataset 1

(b) Dataset 6

(c) Dataset 11

(d) Dataset 16

Figure 5.3: Training and valiation results of CS-R with respect to the cumulative number of episodes



(a) Dataset 1

(b) Dataset 6

(c) Dataset 11

(d) Dataset 16

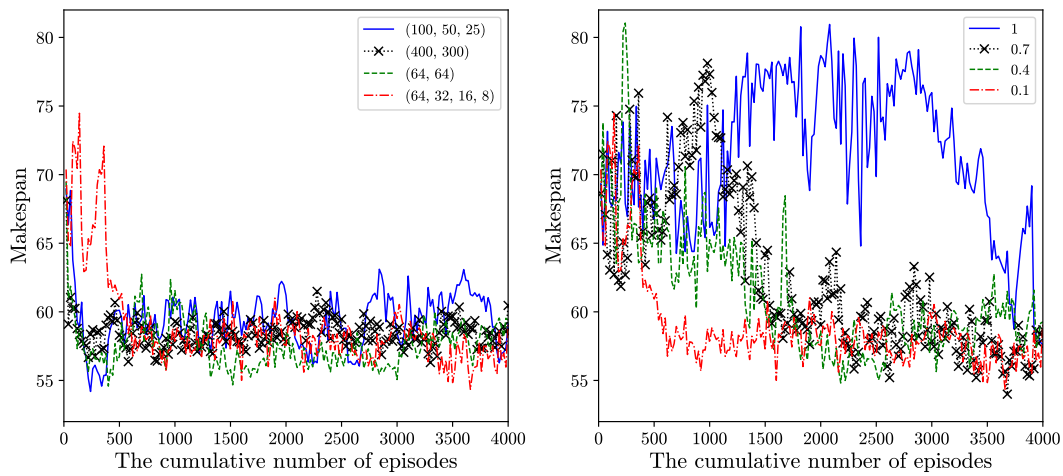
Figure 5.4: Training and valiation results of CS-P with respect to the cumulative number of episodes

5.3 Sensitivity analysis of hyperparmeters

To investigate the sensitivity of hyperparameters, extensive experiments were conducted by changing their values. Due to the large search space, the experiments were carried out on a dataset by using the same method. As a result, we provide the validation results of DS on dataset 11, as shown in Fig. 5.5. Specifically, Figs. 5.5a, 5.5b, and 5.5c illustrate $C_{\max}$ results according to the changes in the network structure, ε, and learning rate, respectively. The rest of the hyperparameters were the same as Table 5.2.

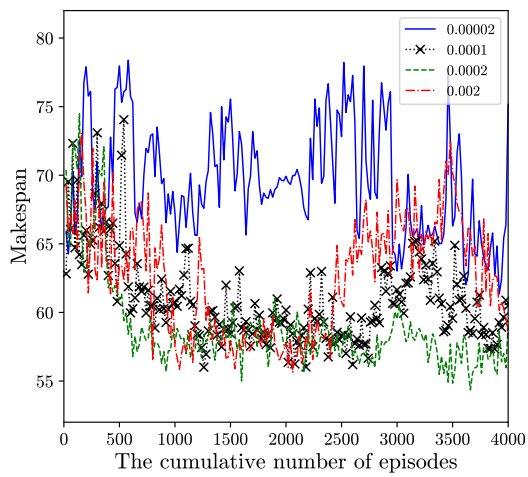
In Fig. 5.5a, we compared the architecture (64, 32, 16, and 8) presented in Table 5.2 with those considered in [99]. After the 500th episode, $C_{\max}$ results observed from all architectures became similar. Fig. 5.5b indicates that $C_{\max}$ tends to decrease slowly as ε became larger. As shown in Fig. 5.5c, the performance difference between the learning rates, 0.0001 and 0.0002, was not significant. On the other hand, there was a significant performance difference when the learning rate increased or decreased 10 times compared to the one used in Section IV. Based on the above observations, the proposed method tends to be more sensitive to ε and learning rate than to the network architecture.

Finally, it was observed that the performances for the replay buffer sizes 10^3 and 10^4 were worse than those of 10^5 and 10^6, as shown in Fig. 5.5d. Although the replay buffer size 10^5 appears to be effective on the scheduling problems considered in this paper, the size need to be re-determined when the configurations of a manufacturing system change. Based on the above observations, the proposed method tends to be more sensitive to the learning rate as well as the replay buffer size than to the network architecture.

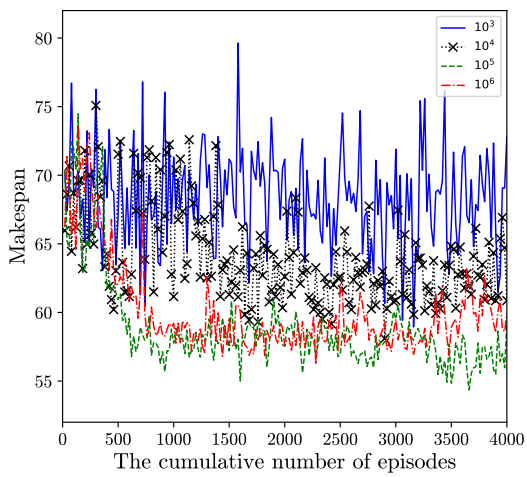


(a) Network architecture

(b) Epsilon



(c) Learning rate



(d) Replay buffer size

Figure 5.5: Validation results of DS with changes in hyperparameters on dataset 11

5.4 Performance comparison

In order to examine the effectiveness of the proposed method in terms of $C_{\max}$, we compared the proposed method with a genetic algorithm (GA) presented in [7]. The parameters of GA were the same as those in [7]. Since the number of generations was set to 100, GA terminates after generation 100 generations. Furthermore, we made comparisons between our method and rule-based methods, including shortest setup time (SSU), shortest sum of processing time and setup time (SPTSSU), most operation remaining (MOR), most work remaining (MWR), shortest processing time (SPT), and longest processing time (LPT) [100]. In particular, SSU and SPTSSU are known to be effective when solving scheduling problems with sequence dependent setup time [101].

Table 5.5 presents the improvement rate of CS-P over the other methods in terms of $C_{\max}$. The negative values in the table indicate $C_{\max}$ results of CS-P are longer than those of the other methods. The results show that $C_{\max}$ of DS and CS-P were lower than those of the baseline methods across all datasets. In particular, $C_{\max}$ of CS-P was lower than those of the other methods except for that of CS-R on datasets 7 and 8.

Furthermore, the proposed methods significantly outperformed MOR, MWR, and SPT. On the other hand, it was observed that the performance difference between CS-P and two rules, SSU, SPTSSU, was less than 12% under datasets 2 to 5. However, the difference was greater than 16% for the other datasets. This is because myopic decisions that reduce setup time do not guarantee high performance as the problem becomes more complex. Although GA outperformed rule-based methods, it yielded slightly longer $C_{\max}$ than DS and CS-P for all datasets.

Table 5.5: Percent improvement of CS-P over GA, the rules, DS, and CS-R in terms of $C_{\max}$

Dataset No.	DS	CS-R	GA	SSU	SPTSSU	MOR	MWR	SPT	LPT
2	0.7%	7.6%	4.5%	4.8%	10.8%	25.9%	23.8%	29.5%	21.7%
3	2.1%	7.6%	3.6%	5.4%	10.5%	26.2%	23.8%	29.0%	22.7%
4	0.3%	10.3%	5.1%	6.0%	10.6%	26.7%	24.6%	29.7%	21.8%
5	2.1%	9.0%	4.3%	6.4%	11.1%	26.8%	24.5%	29.7%	23.6%
7	3.6%	10.1%	5.5%	23.3%	16.4%	27.7%	26.0%	31.3%	46.3%
8	4.8%	9.7%	5.8%	25.9%	18.3%	29.1%	27.6%	33.0%	47.1%
9	3.8%	12.3%	6.3%	26.2%	17.7%	29.5%	27.8%	32.9%	49.0%
10	5.5%	12.0%	7.5%	26.9%	17.8%	28.7%	29.4%	33.0%	47.8%
12	2.8%	8.4%	10.9%	28.3%	24.5%	22.3%	22.9%	32.4%	56.1%
13	2.8%	7.5%	8.9%	27.8%	23.2%	23.1%	23.3%	33.0%	55.1%
14	2.3%	8.2%	9.3%	29.1%	23.7%	23.8%	23.7%	33.1%	58.5%
15	1.7%	9.0%	9.8%	26.6%	24.1%	23.0%	23.6%	34.2%	53.0%
17	8.7%	7.9%	15.2%	49.5%	28.1%	24.0%	21.8%	39.6%	57.0%
18	12.8%	8.3%	17.1%	52.0%	28.1%	23.5%	24.3%	40.4%	63.1%
19	10.1%	7.6%	18.0%	52.1%	29.0%	24.5%	24.3%	39.4%	61.2%
20	12.0%	8.8%	18.2%	51.7%	28.0%	25.3%	24.7%	38.2%	60.9%

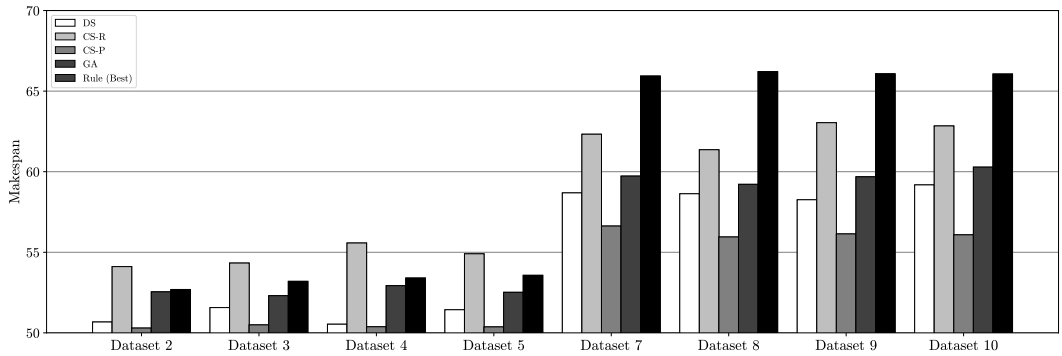


Figure 5.6: $C_{\max}$ results of the proposed methods, GA, and the best rule on datasets 2 to 5 and 7 to 10

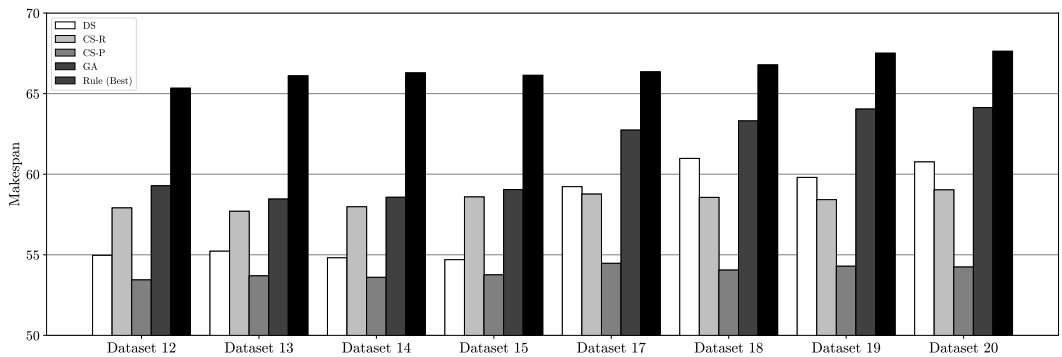
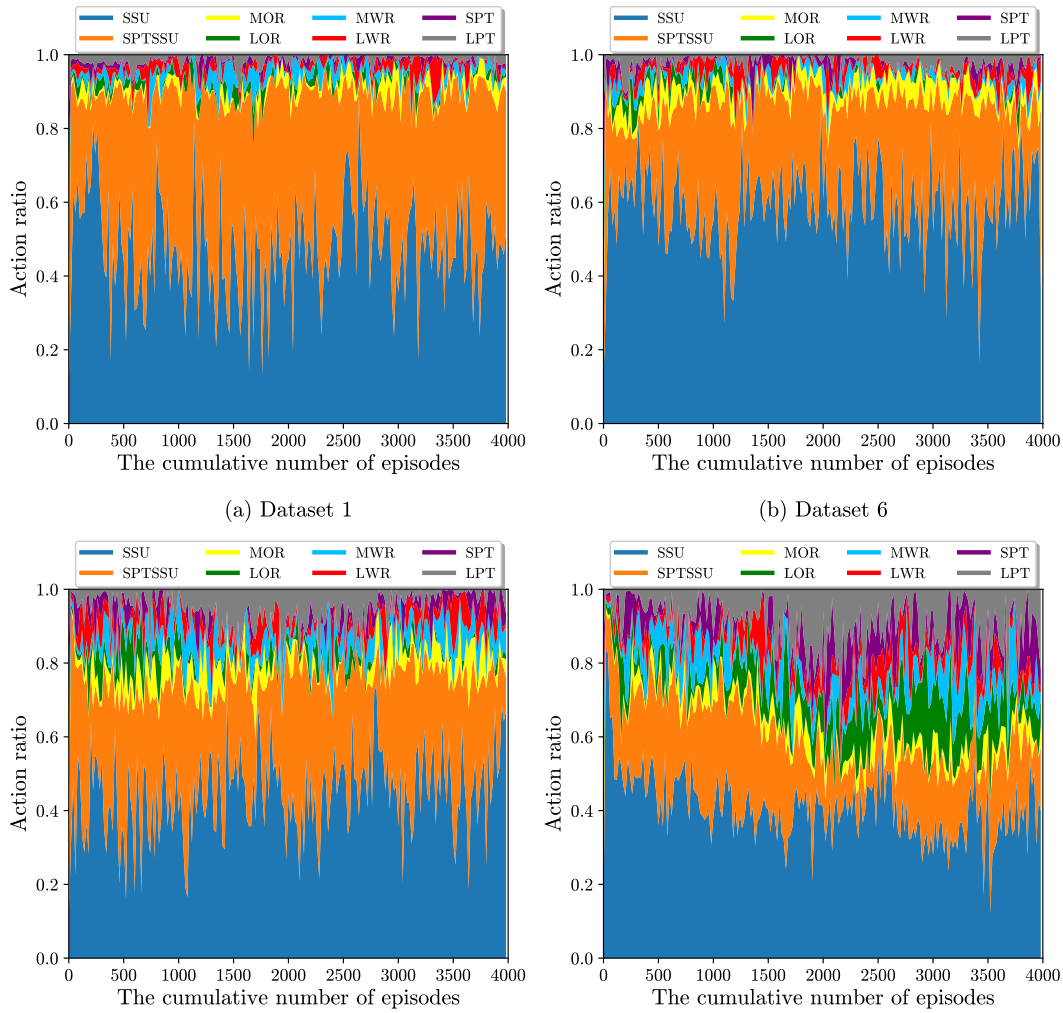


Figure 5.7: $C_{\max}$ results of the proposed methods, GA, and the best rule on datasets 12 to 15 and 17 to 20

We depict the results in a different way to investigate the effects of problem size depending on the production requirements and the number of machines. Figs. 5.6 and 5.7 highlight $C_{\max}$ (in hours) results of DS, CS-R, CS-P, GA, and the best rule whose $C_{\max}$ is the lowest among the rule-based methods. Based on these results, the proposed methods appear to achieve better results in terms of $C_{\max}$ although the production requirements and the number of machines increase.



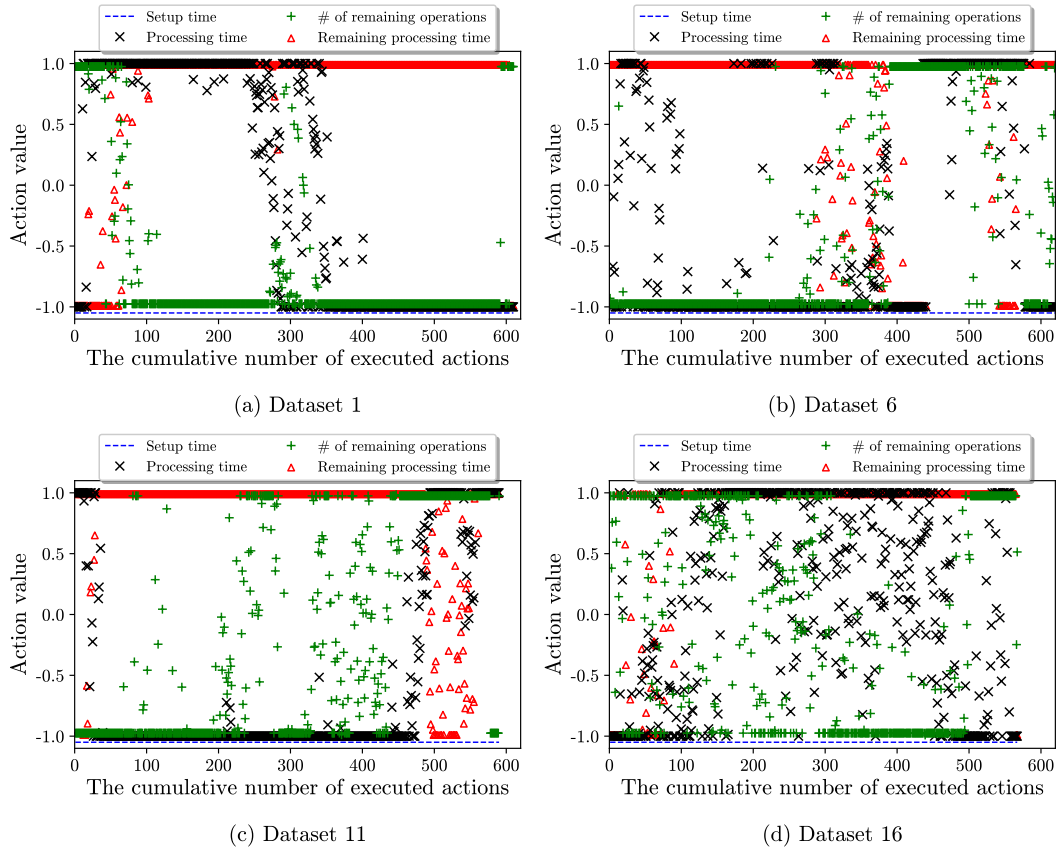
(a) Dataset 1

(b) Dataset 6

(c) Dataset 11

(d) Dataset 16

Figure 5.8: Action selection ratio results of CS-R with respect to the cumulative number of episodes



(a) Dataset 1

(b) Dataset 6

(c) Dataset 11

(d) Dataset 16

Figure 5.9: Action values of CS-P with respect to the cumulative number of executed actions

Figs. 5.8a, b, c, and d depicts the action selection ratio results of CS-R on datasets 1, 6, 11, and 16, respectively. Based on the observations, SSU and SPTSSU are mostly selected across the datasets. On the other hand, the selected ratios of SSU and SPTSSU are decreasing as N_O increases. This is due to the fact that the well-performed rule is different according to each dataset. Since $C_{\max}$ results of SSU and SPTSSU become longer as N_O increases, as presented in Table 5.5, it can be said that CS-R selects the appropriate rule on each state according to the datasets.

To investigate CS-P more specifically, we present Figs. 5.9a, b, c, and d that

indicates proto-action values of CS-P on datasets 1, 6, 11, and 16, respectively. Note that each action of CS-P was normalized between -1 and 1. Based on the results, the proto-action values which represents the setup time were converged into -1. Other values were changed according to changes in the state. This reveals that CS-P outputs an appropriate proto-action given each state.

Table 5.6: Computation time results (in seconds) of the proposed methods, GA, and the best rule

Dataset No.	Best Rule	DS	CS-R	CS-P	GA
2	6.84	17.75	13.81	15.58	1705.75
3	13.48	35.12	26.69	29.36	3881.79
4	22.20	59.20	43.73	47.77	5561.81
5	33.31	88.08	64.35	70.54	8938.19
7	7.78	19.84	15.45	17.14	1734.74
8	15.22	39.76	29.16	32.41	3968.45
9	25.15	69.36	47.77	53.41	5702.72
10	37.78	100.04	71.22	79.43	8991.76
12	8.19	19.66	15.09	16.37	1765.32
13	16.20	38.67	28.77	31.43	4070.17
14	26.19	65.46	46.92	51.01	5919.82
15	39.47	98.85	69.47	77.33	9015.66
17	8.19	20.77	16.22	18.34	1735.41
18	16.20	40.97	31.00	35.25	3890.55
19	26.19	69.02	50.84	57.66	6129.79
20	39.47	109.86	74.21	87.82	9151.38

Furthermore, the computation time (in seconds) required to obtain a schedule is

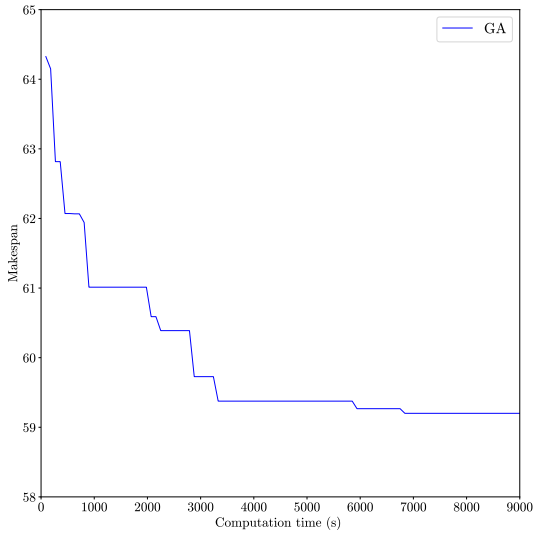


Figure 5.10: Computation time results of GA on dataset 20

calculated against the test datasets to demonstrate the computational efficiency of the proposed methods. We remark that training time of the proposed methods was excluded. Table 5.6 presents the average computation time of our methods, GA, and the best rule whose average computation time is the smallest among the rules. To investigate the computation time in more detail, Fig. 5.11 depicts the computation time results of DS, GA, and the best rule on datasets 2 to 5, 7 to 10, and 12 to 15. It has been observed that the computation time increases as the production requirements increase.

As shown in Table 5.6, the computation time for solving problems with the proposed methods was less than 120 seconds. In detail, the computation time results of centralized schedulers are similar each other, and the results of DS are longer than those of the proposed centralized methods. This may be attributed to the difference of action dimension between decentralized and centralized methods.

Compared to GA, the computation time of DS was decreased by 110 and 82 times

Table 5.7: Percent improvement of CS-P over DS, CS-R, TPDQN, GA, and the rules in terms of $C_{\max}$

Dataset No.	DS	CS-R	TPDQN	GA	SSU	SPTSSU	MOR	MWR	SPT
1	-0.1%	7.3%	24.4%	3.2%	5.3%	9.6%	22.4%	20.9%	30.6%
6	0.2%	9.4%	22.1%	4.5%	26.1%	17.2%	25.7%	24.3%	28.5%
11	4.1%	5.5%	30.9%	7.0%	27.0%	20.2%	20.4%	20.9%	30.5%
16	6.2%	7.8%	28.0%	11.8%	42.7%	25.3%	19.3%	18.9%	34.9%

in the best and worst cases, respectively. To examine changes in $C_{\max}$ according to the computation time, scheduling results on a scheduling problem from dataset 20 were presented in Fig. 5.10. Based on the observations, $C_{\max}$ yielded by GA decreased slowly over time, which indicates that GA can be terminated earlier with less sacrifice in $C_{\max}$. The computation time of GA is more than 30 times longer than that of DS on dataset 20 even when GA is stopped at 4000 seconds.

On the other hand, the computation time of DS was 2.4 to 2.8 times longer than that of the best rule. The difference is caused due to the fact that our method requires additional time for constructing state features and calculating action-values from a Q-network. Nevertheless, the proposed methods seem to be acceptable to practitioners in terms of the computation time taken to obtain a schedule since re-scheduling is usually conducted on an hourly basis in many real-world semiconductor manufacturing systems [6].

We additionally carried out performance comparisons between the proposed methods and a RL method [51], named the two-phase deep Q-network (TPDQN). Since TPDQN is required to be re-trained when the number of operations or the number of machines is changed, the performance comparison was carried out on

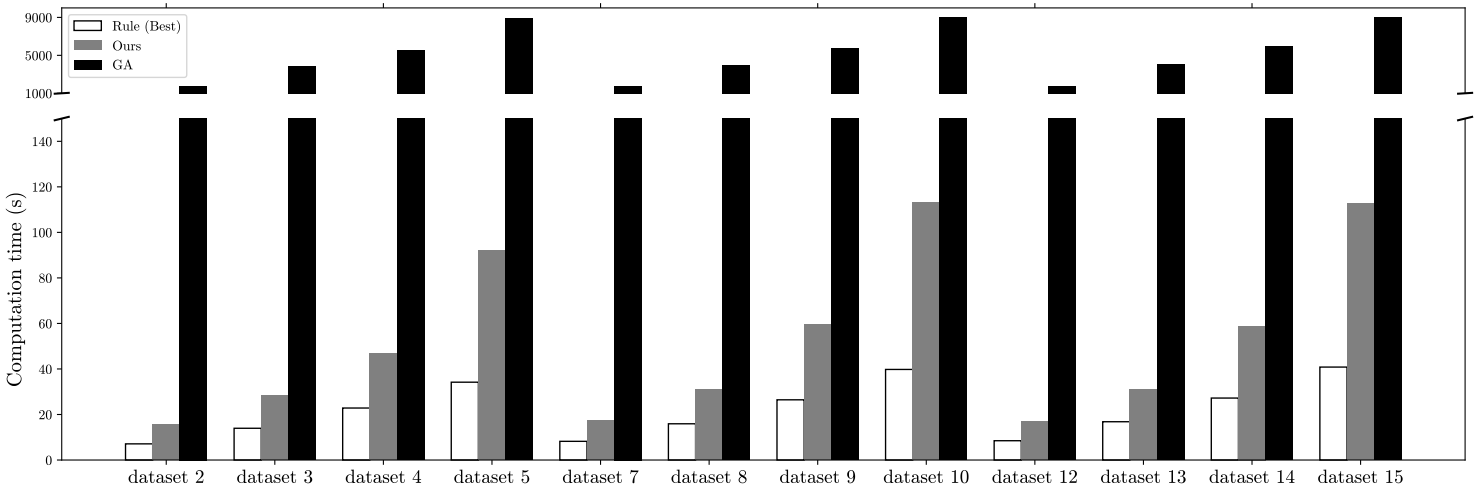


Figure 5.11: Average computation time results of DS, GA, and the best rule

scheduling problems for which those changes did not occur. TPDQN and our method were trained by randomly setting initial setup statuses at each episode. For each dataset, we additionally generated 30 test scheduling problems whose initial setup statuses are different from those of the training problem. Moreover, TPDQN was modified for our scheduling problems because it was originally proposed for minimizing delay from the due date. Specifically, we used the negative setup time as a reward, and SSU was utilized as the dispatching heuristic in the pre-training phase. The rest of the training details were the same as those in [51].

Table 5.7 shows how much improvements were achieved by CS-P over the other methods. TPDQN yielded longer $C_{\max}$ than SPTSSU for all datasets. This reveals that TPDQN does not appear to be robust against the change of the initial setup status. On the other hand, $C_{\max}$ results of CS-P and DS are still lower than those of GA, the rules, and TPDQN. In particular, CS-P outperforms the other methods except for DS on dataset 1.

To examine the robustness of the proposed method, experiments were carried out under stochastic processing time. In detail, the processing time of $O_{j,k}$ is uniformly distributed between $0.8p_{j,k}$ and $1.2p_{j,k}$. For each of the datasets 1, 6, 11, and 16, the proposed methods and TPDQN were trained in the stochastic environment, and a test scheduling problem was solved 30 times with the same random seeds. Besides, the rules and the schedule obtained by GA were evaluated with the same seeds.

Fig. 5.12 depicts the average, minimum, and maximum $C_{\max}$ values of our methods, GA, TPDQN, and the best rule whose average $C_{\max}$ is the lowest among the rule-based methods. DS and TPDQN show a tendency that the difference between maximum and minimum $C_{\max}$ values becomes larger as N_O increases. This obser-

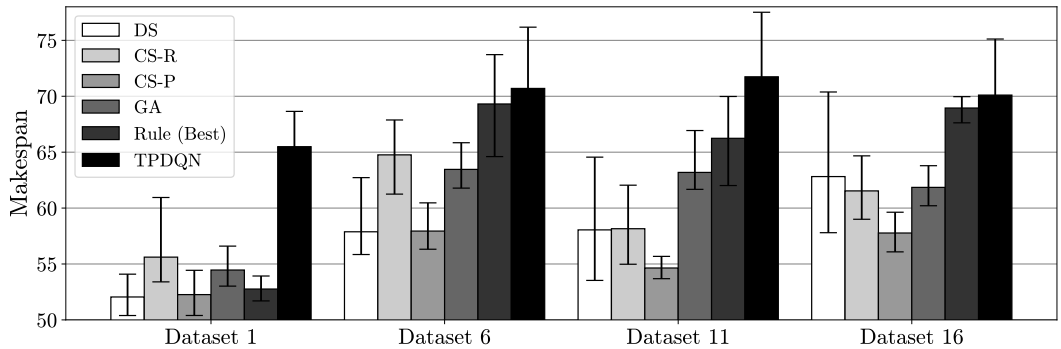


Figure 5.12: $C_{\max}$ results of the proposed methods, GA, the best rule, and TPDQN under stochastic processing time

Table 5.8: p-values for $C_{\max}$ differences in $C_{\max}$ between DS and the other methods for the scheduling problems with stochastic processing time.

Dataset No.	CS-R	CS-P	GA	Rule (Best)	TPDQN
1	4.28×10^{-13}	0.40	1.81×10^{-16}	1.68×10^{-4}	2.84×10^{-33}
6	6.81×10^{-26}	0.83	3.6×10^{-24}	1.11×10^{-27}	1.44×10^{-36}
11	0.87	2.34×10^{-6}	6.53×10^{-10}	3.41×10^{-16}	4.23×10^{-24}
16	0.03	4.53×10^{-11}	0.08	5.38×10^{-13}	1.86×10^{-14}

vation can be attributed to the fact that impacts of uncertainty on processing time increases with the size of state and action spaces. The average $C_{\max}$ values of DS and CS-P are lower than those of the other methods. In particular, the differences between maximum and minimum $C_{\max}$ values of CS-P are comparable to those of GA.

Tables 5.8, 5.9, 5.10 presents p-value results obtained by conducting the t-test between DS, CS-R, and CS-P and the other methods considered at the 5% level of significance, respectively. Although there were no significant differences between DS (CS-R) and the baseline methods in some cases, all p-values yielded by conducting

Table 5.9: p-values for $C_{\max}$ differences in $C_{\max}$ between CS-R and the other methods for the scheduling problems with stochastic processing time.

Dataset No.	DS	CS-P	GA	Rule (Best)	TPDQN
1	-	3.88×10^{-12}	0.18	2.94×10^{-10}	1.03×10^{-29}
6	-	4.94×10^{-27}	2.11×10^{-3}	5.46×10^{-12}	3.26×10^{-19}
11	-	5.41×10^{-12}	1.45×10^{-17}	2.17×10^{-20}	4.55×10^{-25}
16	-	3.61×10^{-3}	0.35	3.78×10^{-25}	8.18×10^{-19}

Table 5.10: p-values for $C_{\max}$ differences in $C_{\max}$ between CS-P and the other methods for the scheduling problems with stochastic processing time.

Dataset No.	DS	CS-R	GA	Rule (Best)	TPDQN
1	-	-	3.03×10^{-12}	0.02	3.99×10^{-35}
6	-	-	1.02×10^{-28}	5.52×10^{-25}	3.48×10^{-32}
11	-	-	5.86×10^{-37}	1.84×10^{-22}	7.17×10^{-24}
16	-	-	4.61×10^{-23}	8.54×10^{-49}	1.69×10^{-22}

t-test between CS-P and the baseline methods are less than 0.05, which reveals that the proposed method outperforms the others in terms of $C_{\max}$.

Chapter 6

Conclusions

6.1 Summary and contributions

This thesis studies a flexible job shop scheduling problem with sequence dependent setups. Due to the variabilities in production requirements, the number of available machines, and initial setup status, it is challenging for a scheduler to produce high quality schedules within a specific time limit using existing approaches. To enhance the robustness against the variabilities while achieving performance improvements, we first presented a decentralized scheduling method, called DS. Novel representations of state, action, and reward were introduced to cope with the variabilities in production requirements and initial setup status. In particular, each agent of DS executes an action in a decentralized manner and learns a centralized policy by sharing an NN among the agents to address the variability in the number of machines.

Furthermore, we introduce a centralized approach in which an agent that determines actions given observations for jobs and machines. To reduce the complexity of state space inherent to the centralized learning, a novel definition of the state is developed by abstracting observations from an environment. Two centralized schedulers, called CS-R and CS-P, were proposed. Given a state, CS-R employs the Q-network that determines a rule-based method and CS-P utilizes the actor-critic deep

reinforcement learning method and Wolpertinger policy to select job-machine pairs. As a result, the dimensionality of action in CS-R and CS-P is independent of the number of operations types.

Through the experiments on solving scheduling problems of real-world semiconductor packaging lines, we trained the networks on small-scale problems and tested those on large-scale problems without re-training the networks. In order to examine the robustness of the proposed method, the networks trained on small-scale scheduling problems were used to solve large-scale scheduling problems with changes in production requirements, the number of machines, and initial setup status. The experimental results on the various datasets demonstrated that the proposed method outperformed the other baseline methods considered in terms of the makespan even when there exists stochasticity in processing time.

6.2 Limitations and future research

Through the thesis, it was shown that decentralized and centralized approaches were performed well for solving various scheduling problems. Yet, training the Q-networks for decentralized approaches might take a long time for the scheduling problems with a wide variety of operation types as the number of weights in the Q-network increases linearly with that of operation types. In addition, the networks for both approaches are subject to re-training when operation types, machine types, and production flows change since the dimensionality of the state is dependent of such changes. This poses a limitation on application of the proposed methods to the manufacturing environment where semiconductor consumers frequently demand new products. Therefore, the future work will focus on developing new methods that do not require re-training the network in the aforementioned scenario.

Although the performances of the proposed methods were satisfactory for solving scheduling problems with stochastic processing time, it is still required to accommodate various types of uncertainties in manufacturing systems such as the machine breakdown and preventive maintenance and investigate stable training methods for deep neural networks. Furthermore, we plan to improve our method by incorporating other deep learning models. For example, recurrent and graph neural networks can be helpful for training historic information and accommodating the variability in product types, For the decentralized approach, a cooperation mechanism between agents proposed in the recent MARL methods will be investigated.

Finally, it is expected that the proposed method is applicable to solving various scheduling problems in real-world manufacturing systems such as thin film transistor-liquid crystal display, wafer growing, and tire production lines. To uti-

lized our methods in such problems, further research should deal with various objective functions. For example, minimizing the mean tardiness and flow time will be investigated. In particular, minimizing the tardiness might be more difficult than minimizing the makespan of semiconductor packaging systems due to the sparsity problem of rewards. To address this problem, new exploration techniques and training auxiliary tasks for DRL will be sought.

Bibliography

[1] L. Shen, S. Dauzère-Pérès, and J. S. Neufeld, "Solving the flexible job shop scheduling problem with sequence-dependent setup times," *European Journal of operations research*, vol. 265, no. 2, pp. 503–516, Mar. 2018.

[2] M. Saidi-Mehrabad and P. Fattahi, "Flexible job shop scheduling with tabu search algorithms," *The International Journal of Advanced Manufacturing Technology*, vol. 32, no. 5-6, pp. 563–570, Mar. 2007.

[3] S. J. Lee and T. E. Lee, "Scheduling a multi-chip package assembly line with reentrant processes and unrelated parallel machines," in *Proceedings of the 40th Conference on Winter Simulation*, Austin, TX, USA, Dec. 2008, pp. 2286–2291.

[4] J. T. Lin and C.-M. Chen, "Simulation optimization with GA and OCBA for semiconductor back-end assembly scheduling," in *2015 International Conference on Industrial Engineering and Operations Management*, Dubai, UAE, 2015, pp. 1–8.

[5] Y. Hur, J. F. Bard, and R. Chacon, "Hierarchy machine set-up for multi-pass lot scheduling at semiconductor assembly and test facilities," *International Journal of Production Research*, pp. 1–20, Sep. 2017.

[6] J. Lim, M.-J. Chae, Y. Yang, I.-B. Park, J. Lee, and J. Park, "Fast scheduling of semiconductor manufacturing facilities using case-based reasoning," *IEEE Transactions on Semiconductor Manufacturing*, vol. 29, no. 1, pp. 22–32, Feb. 2016.

[7] B. S. Chung, J. Lim, I. B. Park, J. Park, M. Seo, and J. Seo, "Setup Change Scheduling for Semiconductor Packaging Facilities Using a Genetic Algorithm With an Operator Recommender," *IEEE Transactions on Semiconductor Manufacturing*, vol. 27, no. 3, pp. 377–387, Aug. 2014.

[8] F. M. Defersha and M. Chen, "A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups," *The International Journal of Advanced Manufacturing Technology*, vol. 49, no. 1-4, pp. 263–279, 2010.

[9] Z. Wang, Q. Wu, and F. Qiao, "A lot dispatching strategy integrating wip management and wafer start control," *IEEE Transactions on Automation Science and Engineering.*, vol. 4, no. 4, pp. 579–583, Oct. 2007.

[10] H. Zhang, Z. Jiang, and C. Guo, "Simulation-based optimization of dispatching rules for semiconductor wafer fabrication system scheduling by the response surface methodology," *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 1-2, pp. 110–121, Apr. 2009.

[11] S. Jia, D. J. Morrice, and J. F. Bard, "A performance analysis of dispatch rules for semiconductor assembly & test operations," *Journal of Simulation*, pp. 1–18, Mar. 2018.

[12] F. Qiao, Y. Ma, M. Zhou, and Q. Wu, "A novel rescheduling method for dynamic semiconductor manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018.

[13] C. Pickardt, J. Branke, T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness," in *Proceedings of the 2010 Winter Simulation Conference*, Baltimore, MD, USA, Dec. 2010, pp. 2504–2515.

[14] M. H. F. Zarandi, A. A. S. Asl, S. Sotudian, and O. Castillo, "A state of the art review of intelligent scheduling," *Artificial Intelligence Review*, pp. 1–93, Nov. 2018.

[15] L. Li, Z. Sun, M. Zhou, and F. Qiao, "Adaptive dispatching rule for semiconductor wafer fabrication facility," *IEEE Transactions on Automation Science and Engineering.*, vol. 10, no. 2, pp. 354–364, Apr. 2013.

[16] L. Li, S. Zijin, N. Jiacheng, and Q. Fei, "Data-based scheduling framework and adaptive dispatching rule of complex manufacturing systems," *The International Journal of Advanced Manufacturing Technology*, vol. 66, no. 9-12, pp. 1891–1905, Jun. 2013.

[17] H.-S. Min and Y. Yih, "Selection of dispatching rules on multiple dispatching decision points in real-time scheduling of a semiconductor wafer fabrication system," *International Journal of Production Research*, vol. 41, no. 16, pp. 3921–3941, Jan. 2003.

[18] J. Huh, I. Park, S. Lim, B. Paeng, J. Park, and K. Kim, "Learning to dispatch operations with intentional delay for re-entrant multiple-chip product assembly lines," *Sustainability*, vol. 10, no. 11, p. 4123, Nov. 2018.

[19] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73–82, Feb. 2005.

[20] A. Xanthopoulos, D. E. Koulouriotis, V. D. Tourassis, and D. M. Emiris, "Intelligent controllers for bi-objective dynamic scheduling on a single machine with sequence-dependent setups," *Applied Soft Computing*, vol. 13, no. 12, pp. 4704–4717, Dec. 2013.

[21] Z. Zhang, L. Zheng, and M. X. Weng, "Dynamic parallel machine scheduling with mean weighted tardiness objective by q-learning," *The International Journal of Advanced Manufacturing Technology*, vol. 34, no. 9-10, pp. 968–980, Oct. 2007.

[22] Z. Zhang, L. Zheng, N. Li, W. Wang, S. Zhong, and K. Hu, "Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning," *Computers and Operations Research*, vol. 39, no. 7, pp. 1315–1324, Jul. 2012.

[23] Z. Zhang, W. Wang, S. Zhong, and K. Hu, "Flow shop scheduling with reinforcement learning," *Asia-Pacific Journal of Operational Research*, vol. 30, no. 05, p. 1350014, Jul. 2013.

[24] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, "Real-time scheduling for a smart factory using a reinforcement learning approach," *Computers & Industrial Engineering*, vol. 125, pp. 604–614, Nov. 2018.

[25] Z. Cao, C. Lin, M. Zhou, and R. Huang, "Scheduling semiconductor testing facility by using cuckoo search algorithm with reinforcement learning and surrogate modeling," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 825–837, Apr. 2019.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[27] J. N. Foerster, C. A. S. de Witt, G. Farquhar, P. H. Torr, W. Boehmer, and S. Whiteson, "Multi-agent common knowledge reinforcement learning," *arXiv preprint arXiv:1810.11702*, 2018.

[28] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, Dec. 2016, pp. 2137–2145.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[30] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[31] Z. Wu and M. X. Weng, "Multiagent scheduling method with earliness and tardiness objectives in flexible job shops," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 2, pp. 293–301, 2005.

[32] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[33] Y. Yuan and H. Xu, "Multiobjective flexible job shop scheduling using memetic algorithms," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 336–353, 2013.

[34] K. Gao, F. Yang, M. Zhou, Q. Pan, and P. N. Suganthan, "Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm," *IEEE transactions on cybernetics*, vol. 49, no. 5, pp. 1944–1955, 2018.

[35] M. Shahgholi Zadeh, Y. Katebi, and A. Doniavi, "A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times," *International Journal of Production Research*, vol. 57, no. 10, pp. 3020–3035, 2019.

[36] Z. Cao, L. Zhou, B. Hu, and C. Lin, "An adaptive scheduling algorithm for dynamic jobs for dealing with the flexible job shop scheduling problem," *Business & Information Systems Engineering*, vol. 61, no. 3, pp. 299–309, 2019.

[37] M. Mousakhani, "Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness," *International Journal of Production Research*, vol. 51, no. 12, pp. 3476–3487, 2013.

[38] G. Vilcot and J.-C. Billaut, "A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem," *European Journal of Operational Research*, vol. 190, no. 2, pp. 398–411, 2008.

[39] K. F. Guimaraes and M. A. Fernandes, "An approach for flexible job-shop scheduling with separable sequence-dependent setup time," in *2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5. IEEE, 2006, pp. 3727–3731.

[40] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[41] A. Bagheri and M. Zandieh, "Bi-criteria flexible job-shop scheduling with sequence-dependent setup times—variable neighborhood search approach," *Journal of Manufacturing Systems*, vol. 30, no. 1, pp. 8–15, 2011.

[42] T. F. Abdelmaguid, "A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times," *Applied Mathematics and Computation*, vol. 260, pp. 188–203, 2015.

[43] Z. Cao, C. Lin, and M. Zhou, "A knowledge-based cuckoo search algorithm to schedule a flexible job shop with sequencing flexibility," *IEEE Transactions on Automation Science and Engineering*, 2019.

[44] J. Potoradi, O. S. Boon, S. J. Mason, J. W. Fowler, and M. E. Pfund, "Semiconductor manufacturing: using simulation-based scheduling to maximize demand fulfillment in a semiconductor assembly facility," in *Proceedings of the*

34th conference on Winter simulation: exploring new frontiers. Winter Simulation Conference, 2002, pp. 1857–1861.

[45] A. K. Gupta and A. L. Sivakumar, "Semiconductor manufacturing: simulation based multiobjective schedule optimization in semiconductor manufacturing," in *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*. Winter Simulation Conference, 2002, pp. 1862–1870.

[46] A. I. Sivakumar and C. S. Chong, "A simulation based analysis of cycle time distribution, and throughput in semiconductor backend manufacturing," *Computers in Industry*, vol. 45, no. 1, pp. 59–78, 2001.

[47] S. Werner, S. Horn, G. Weigert, and T. Jahnig, "Simulation based scheduling system in a semiconductor backend facility," in *Proceedings of the 2006 winter simulation conference*. IEEE, 2006, pp. 1741–1748.

[48] L. Y. Hsieh and C.-B. Cheng, "Efficient due-date quoting and production scheduling for integrated circuit packaging with reentrant processes," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 8, no. 8, pp. 1487–1495, 2018.

[49] J. Huh and J. Park, "Artificial neural network based multi-objective rule selection dispatcher for re-entrant multiple-chip product assembly lines," *Journal of Korean Institute of Information Technology*, vol. 17, no. 2, pp. 1–11, 2019.

[50] W. Yoo, J. Seo, D. Lee, D. Kim, and K. Kim, "Scheduling generation model on parallel machines with due date and setup cost based on deep learning," *The Journal of Society for e-Business Studies*, vol. 24, no. 3, pp. 99–110, 2019.

[51] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Deep reinforcement learning for semiconductor production scheduling," in *Proc. 29th. Annu. SEMI Adv. Semicond. Manuf. Conf.*, Saratoga Springs, NY, Jun. 2018, pp. 301–306.

[52] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. Cambridge, MA, USA: MIT press, 1998, vol. 135.

[53] 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, p. 529, 2015.

[54] H. Yang and X. Xie, "An actor-critic deep reinforcement learning approach for transmission scheduling in cognitive internet of things systems," *IEEE Systems Journal*, 2019.

[55] T. Tan, F. Bao, Y. Deng, A. Jin, Q. Dai, and J. Wang, "Cooperative deep reinforcement learning for large-scale traffic grid signal control," *IEEE transactions on cybernetics*, 2019.

[56] Y. Dai, D. Xu, S. Maharjan, Z. Chen, Q. He, and Y. Zhang, "Blockchain and deep reinforcement learning empowered intelligent 5g beyond," *IEEE Network*, vol. 33, no. 3, pp. 10–17, 2019.

[57] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor–critic deep

reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2061–2073, 2018.

[58] M. Vecerik, O. Sushkov, D. Barker, T. Rothörl, T. Hester, and J. Scholz, "A practical approach to insertion with variable socket position using deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 754–760.

[59] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

[60] X. Li, J. Zhang, J. Bian, Y. Tong, and T.-Y. Liu, "A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 980–988.

[61] L. Pan, Q. Cai, Z. Fang, P. Tang, and L. Huang, "A deep reinforcement learning framework for rebalancing dockless bike sharing systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1393–1400.

[62] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1774–1783.

[63] A. Xanthopoulos, A. Kiatipis, D. E. Koulouriotis, and S. Stieger, "Reinforcement learning-based and parametric production-maintenance control policies

for a deteriorating manufacturing system," *IEEE Access*, vol. 6, pp. 576–588, 2017.

[64] I. Hwang and Y. J. Jang, "Q (λ) learning-based dynamic route guidance algorithm for overhead hoist transport systems in semiconductor fabs," *International Journal of Production Research*, pp. 1–23, 2019.

[65] Q. Xiao, Z. Cao, and M. Zhou, "Learning locomotion skills via model-based proximal meta-reinforcement learning," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019, pp. 1545–1550.

[66] C. Hong and T.-E. Lee, "Multi-agent reinforcement learning approach for scheduling cluster tools with condition based chamber cleaning operations," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 885–890.

[67] Z. Zhang, L. Ma, K. Poularakis, K. K. Leung, J. Tucker, and A. Swami, "Macs: Deep reinforcement learning based sdn controller synchronization policy design," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019, pp. 1–11.

[68] X. Y. Lee, A. Balu, D. Stoecklein, B. Ganapathysubramanian, and S. Sarkar, "A case study of deep reinforcement learning for engineering design: Application to microfluidic devices for flow sculpting," *Journal of Mechanical Design*, vol. 141, no. 11, 2019.

[69] Y.-C. Wang and J. M. Usher, "A reinforcement learning approach for developing routing policies in multi-agent production scheduling," *The International*

Journal of Advanced Manufacturing Technology, vol. 33, no. 3-4, pp. 323–333, 2007.

[70] T. Gabel and M. Riedmiller, "Adaptive reactive job-shop scheduling with reinforcement learning agents," *International Journal of Information Technology and Intelligent Computing*, vol. 24, no. 4, 2008.

[71] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[72] T. Gabel and M. Riedmiller, "Distributed policy search reinforcement learning for job-shop scheduling tasks," *International Journal of Production Research*, vol. 50, no. 1, pp. 41–61, 2012.

[73] B. Yuan, L. Wang, and Z. Jiang, "Dynamic parallel machine scheduling using the learning agent," in *2013 IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE, 2013, pp. 1565–1569.

[74] A. Kim, "Ensemble-based quality classification and deep reinforcement learning-based production scheduling: Ensemble-based quality classification and deep reinforcement learning-based production scheduling," Ph.D. dissertation, Ph. D. dissertation, KyungHee University, 2018.

[75] W. Yoo, J. Seo, D. Kim, and K. Kim, "Machine scheduling models based on reinforcement learning for minimizing due date violation and setup change," *The Journal of Society for e-Business Studies*, vol. 24, no. 3, pp. 19–33, 2019.

[76] J. Wang, S. Qu, J. Wang, J. O. Leckie, and R. Xu, "Real-time decision support with reinforcement learning for dynamic flowshop scheduling," in *Smart Sys-*

Tech 2017; European Conference on Smart Objects, Systems and Technologies. VDE, 2017, pp. 1–9.

[77] S. Qu, J. Wang, and J. Jasperneite, "Dynamic scheduling in large-scale stochastic processing networks for demand-driven manufacturing using distributed reinforcement learning," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 433–440.

[78] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multi-class deep q network," *IEEE Transactions on Industrial Informatics*, 2019, doi: 10.1109/JPROC.2010.2070470.

[79] S. Qu, J. Wang, and J. Jasperneite, "Dynamic scheduling in modern processing systems using expert-guided distributed reinforcement learning," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2019, pp. 459–466.

[80] 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*, 2019.

[81] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*, São Paulo, Brazil, May 2017, pp. 66–83.

[82] A. Sprecher, R. Kolisch, and A. Drexl, "Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem," *European Journal of Operations Research*, vol. 80, no. 1, pp. 94–102, Jan. 1995.

[83] D. Harris and S. Harris, *Digital design and computer architecture*. San Mateo, CA: Morgan Kaufmann, 2010.

[84] P. J. Huber *et al.*, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[85] H. C. Tijms, *A first course in stochastic models*. John Wiley and sons, 2003.

[86] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3562–3573.

[87] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf, "Deep reinforcement learning with an unbounded action space," *arXiv preprint arXiv:1511.04636*, vol. 5, 2015.

[88] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[89] Y. Tang and S. Agrawal, "Discretizing continuous action space for on-policy optimization," *arXiv preprint arXiv:1901.10500*, 2019.

[90] C. Tessler, T. Zahavy, D. Cohen, D. J. Mankowitz, and S. Mannor, "Action assembly: Sparse imitation learning for text based games with combinatorial action spaces," *arXiv preprint arXiv:1905.09700*, 2019.

[91] A. Jain, K. Nandakumar, and A. Ross, "Score normalization in multimodal biometric systems," *Pattern recognition*, vol. 38, no. 12, pp. 2270–2285, 2005.

[92] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[93] Y.-H. Han and J. Y. Choi, "A gspn-based approach to stacked chips scheduling problem," *IEEE Transactions on Semiconductor Manufacturing*, vol. 23, no. 1, pp. 4–12, 2009.

[94] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, Feb. 2012.

[95] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for Machine Learning*, vol. 4, no. 2, pp. 26–31, 2012.

[96] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *International Conference on Machine Learning*, vol. 30, no. 1, 2013, p. 3.

[97] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[98] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[99] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[100] R. Haupt, "A survey of priority rule-based scheduling," *OR Spectrum*, vol. 11, no. 1, pp. 3–16, Mar. 1989.

[101] V. Vinod and R. Sridharan, "Scheduling a dynamic job shop production system with sequence-dependent setups: An experimental study," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 3, pp. 435–449, Jun. 2008.

국문초록

본 논문은 순서 의존적 셋업이 있는 유연 잡샵 스케줄링 문제를 연구한다. 이 문제는 제조 시스템에서 존재하는 재유입 공정과 같은 제약 조건이 있을 때 상당히 복잡해진다. 동시에, 생산 요구량, 사용 가능한 설비 및 초기 셋업 상태의 변동성에 효과적으로 대응하기 위해 스케줄링 문제를 자주 풀어야 한다. 변동성에 대응하기 위해 시간 단위로 스케줄링을 수행해야 하므로 대형 제조 시스템의 경우 시간제한 내에 고품질의 스케줄을 획득하기가 어렵다.

순서 의존적 셋업이 있는 유연 잡샵의 최대 완료시간 최소화를 위해 심층 강화학습을 이용한 세 가지 셋업 스케줄링 기법을 제시하였다. 첫째, 각 에이전트가 분산형 방식으로 셋업을 의사결정하고, 설비 수의 변화에 대처하기 위해 에이전트와 신경망을 공유함으로써 정책을 배우는 분산형 스케줄러를 제안한다. 또한 생산 요건 및 초기 설정 상태의 가변성을 다루기 위해 상태, 행동 및 보상에 대한 새로운 정의를 제안한다.

둘째로, 모든 작업과 설비에 대한 관찰을 토대로 의사결정하는 방식인 중앙집중화된 스케줄링 접근법을 도입한다. 중앙집중식 학습시 발생하는 상태 공간의 복잡성을 줄이기 위해, 관찰된 정보를 추상화함으로써 상태에 대한 새로운 정의를 제안한다. 이를 토대로, 규칙 기반 스케줄러와 작업-설비 쌍을 하나의 행동으로 선택하는 스케줄러를 제안한다. 전자는 심층 큐 신경망을 토대로 중앙 집중식 정책을 배우고, 후자는 행위자-비평가 심층 강화 학습 방법과 울퍼팅어 정책을 활용하여 작업-설비 쌍의 연속적인 특징을 근사하게 만든다.

제안된 방법의 강건성을 검증하기 위해, 소규모의 스케줄링 문제들에 대해 훈련된 신경망을 사용하여 대규모 스케줄링 문제들을 해결한다. 현실 반도체 패키징 라인 스케줄링 문제에 대한 실험을 통해, 제안된 기법들이 규칙 기반, 메타 휴리스틱 및 다른

강화학습 기법을 능가하는 동시에 고려된 메타 휴리스틱보다 짧은 계산 시간을 요구한다는 것을 보여준다. 또한, 훈련된 신경망은 확률적 생산 시간이 있음에도 학습하지 않은 실제 규모 문제를 해결하는 데 좋은 성능을 보이며, 이는 실제 반도체 패키징 라인에 제안된 방법을 적용할 수 있음을 보여준다.

주요어: 유연 잡 샵 스케줄링, 반도체 패키징 라인, 멀티 칩 제품, 강건한 스케줄링, 심층 강화학습, 인공 신경망

학번: 2012-21056