


Deep Reinforcement Learning-Driven Scheduling in Multijob Serial Lines: A Case Study in Automotive Parts Assembly

Sanghoon Lee , Jinyoung Kim , Gwangjin Wi , Yuchang Won ,
Yongsoo Eun , Senior Member, IEEE, and Kyung-Joon Park , Senior Member, IEEE

Abstract—Multijob production (MJP) is a class of flexible manufacturing systems, which produces different products within the same production system. MJP is widely used in product assembly, and efficient MJP scheduling is crucial for productivity. Most of the existing MJP scheduling methods are inefficient for multijob serial lines with practical constraints. We propose a deep reinforcement learning (DRL)-driven scheduling framework for multijob serial lines by properly considering the practical constraints of identical machines, finite buffers, machine breakdown, and delayed reward. We analyze the starvation and the blockage time, and derive a DRL-driven scheduling strategy to reduce the blockage time and balance the loads. We validate the proposed framework by using real-world factory data collected over six months from a tier-one vendor of a world top-three automobile company. Our case study shows that the proposed scheduling framework improves the average throughput by 24.2% compared with the conventional approach.

Index Terms—Multijob serial lines, production scheduling, reinforcement learning (RL), smart manufacturing.

I. INTRODUCTION

SMART and flexible manufacturing is gaining in great popularity with the development of the Internet of Things, 5G, big data, and most of all, artificial intelligence (AI) and machine learning (ML) [1], [2]. For example, a multijob production (MJP) serial line widely used in practice is a flexible manufacturing system that produces multiple products within the same serial line system [3].

Efficient production scheduling is crucial for improving the throughput of a production line without structural changes. However, it is challenging to derive an efficient scheduling solution

within a reasonable time because most production scheduling problems are nondeterministic polynomial-time (NP) hard with complex practical constraints [4].

The AI and ML techniques have been applied to production scheduling problems, e.g., [5] and [6]. In particular, a scheduling problem, once formulated as a Markov decision process (MDP), can be solved with reinforcement learning (RL) [7]. Recently, deep reinforcement learning (DRL) has been applied to solve scheduling problems in assembly lines [8], [9], [10]. However, applying DRL-driven scheduling to serial lines under practical constraints is still challenging. Most of all, MDP modeling of production scheduling has not been well established [11]. The MDPs in recent studies are primarily for production systems with a parallel structure [11], [12], [13], and cannot be directly applied to MJP serial lines.

In this article, we propose DRL-driven scheduling in MJP serial lines under practical constraints. The contributions of this article are listed as follows.

- 1) We formulate an MJP serial line as an MDP model, which includes the following practical constraints: identical machines, finite buffers, machine breakdown, and delayed reward. To the best of authors' knowledge, this is the first study on MJP serial line scheduling with RL that considers all of these practical constraints.
- 2) We propose a DRL-driven scheduling framework for the MJP serial line. We adopt the double dueling deep Q-learning network (DDQN) as a main learning algorithm. Unlike existing studies, we use a queueing strategy to consider the delayed reward of a serial line.
- 3) We validate our proposed scheduling framework using real-world factory data collected over six months. Our performance evaluation shows that the proposed algorithm improves the average throughput by 24.2 % with reduced variance compared with the conventional approach.

Our target manufacturing system is an MJP serial line with a single serial structure, which is widely used in practice. In this structure, in order to improve the throughput, identical machines that perform the same task are often placed in series. Scheduling of identical machines has been substantially studied. However, most of them have considered identical machines in parallel [14], [15]. In the meantime, little research has been conducted on the scheduling of identical machines in series, which is more complicated than the parallel case. For example, if there are

Manuscript received 30 November 2022; revised 27 April 2023; accepted 25 June 2023. Date of publication 8 August 2023; date of current version 19 January 2024. This work was supported by the DGIST R&D Program of the Ministry of Science and ICT, South Korea, under Grant 23-DPIC-16. Paper no. TII-22-4898. (Corresponding author: Kyung-Joon Park.)

The authors are with the Department of Electrical, Engineering, and Computer Science, Daegu Gyeongbuk Institute of Science and Technology, Daegu 42988, South Korea (e-mail: leesh2913@dgist.ac.kr; tndnjs101@dgist.ac.kr; wgj2050@dgist.ac.kr; yuchang@dgist.ac.kr; yeun@dgist.ac.kr; kjp@dgist.ac.kr).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TII.2023.3292538>.

Digital Object Identifier 10.1109/TII.2023.3292538

two identical machines, M_A and M_B in a parallel placement, the scheduler only needs to consider the buffer states of both machines. However, in a serial placement, the scheduler further needs to consider the order of machines. Assume that the two machines are placed in the order of M_A-M_B . Then, even when M_B 's buffer is empty, M_B cannot be assigned a job until M_A 's job is completed. Therefore, serial placement of the identical machines introduces additional starvation and blockage, which further complicates scheduling.

In our target system, both the model to be produced and the pattern of the machine employed must be scheduled at the same time. In addition, the system has finite buffers, and may also suffer machine breakdown. Thus, each machine experiences either starvation or blockage, depending on the state of the buffers and the machines. Because of the serial nature of the line, the system response to scheduling or so-called the reward for an action is delayed and should be carefully considered in MDP modeling of RL. Existing approaches do not properly deal with these practical constraints.

II. RELATED WORK

An approach for improving the throughput of production lines is to update the structure of the production lines [16]. However, practical constraints, such as production halts, space limitations, and budgetary concerns make it difficult to alter the line structure. A practical approach without structural changes is efficient production scheduling.

Because most MJP scheduling problems are NP-hard, it is difficult to derive optimal solutions within a reasonable time. Therefore, production scheduling using ML techniques is introduced to derive near-optimal scheduling within a reasonable time [5], [6]. Recently, following the development of DRL [17], DRL-driven scheduling has been widely studied in assembly production lines [8], [9], [10].

DRL-driven production scheduling has specific considerations, depending on the target production system. In semiconductor manufacturers, the order of workplace reservation is determined by considering waiting operations, setup status, action history, and utilization history [8]. The aero-engine assembly scheduling problem considers the waiting queues, emergency level of the product, remaining processing time, and machine utilization to the appropriate machine [9]. In a shipyard assembly production system, the loading sequence of blocks is determined by considering the remaining processing time of each block [10]. However, none of these studies consider the practical constraints of identical machines, finite buffers, machine breakdown, and delayed reward.

If the MDP describing the production system is complex, it is difficult to implement DRL-driven scheduling in practice. A lack of theoretical methodology on MDP modeling leads to a variety of approaches for modeling the MDP of production systems [11], [12], [13]. In particular, the use of an effective reward function significantly affects the performance of DRL-driven scheduling [18]. In job shop scheduling, the reward is defined as the critical ratio of the selected action to the cost of the job held by the system between two consecutive actions [11]. In gantry

work cells, production loss attribution (PLA) and production loss risk are used for the reward [12]. The researchers in [13] use a reward table for different action properties. However, their reward function is inapplicable to our case, which has a serial structure.

Our case can be modeled as a serial line system, a type of Bernoulli serial line that includes a finite buffer and machine failures [19]. In addition, our case includes a constraint of serial placement of identical machines. Due to the characteristics of the serial line and the constraint of identical machine placement, it is not feasible to utilize the MDP model of existing production systems. To the best of authors' knowledge, there is no MDP modeling for the serial line of our case for production efficiency. Yan and Zheng [20] proposed an MDP model for the serial line for energy consumption optimization, but only included evaluation of system production loss in the reward function, and was scheduled for maintenance. Wang et al. [21] proposed an MDP model for the serial line, but targeted real-time control of each machine instead of scheduling of multiple tasks and parts.

As mentioned previously, the adoption of DRL scheduling methods in production lines contributes to improving production throughput. However, a DRL-based scheduling method that appropriately considers the real constraint conditions of a serial line system has not been studied. In this study, we propose a DRL-based scheduling method for MJP serial lines by designing an MDP that considers the characteristics and real constraint conditions of the serial line.

III. TARGET PRODUCTION SYSTEM

The production system in this study is an automotive parts assembly system used by a tier-one vendor of a world top-three automobile company. The data on the assembly line is collected over six months from May to October 2021. In this section, we describe the steps involved in processing the collected data and the target serial line system in detail.

A. Assembly Line

The target system produces a total of 29 models of A/T Levers, which are levers used to adjust gears in automatic transmissions. Each of the 29 models is designed for a specific automotive model. Models go through several processes, including screw fastening, assembly, performance testing, pressing, calibration, and end-of-line final inspection.

The assembly line is an automated conveyor system consisting of 20 machines and 19 buffers. The parts move in one direction, and there is no reentrant process. Workers put the parts of the product model into M_1 and collect the finished products from M_{20} . The movement between the machines and the buffers is fully automated by a pallet on the conveyor. When a machine finishes assembling, the part moves through the conveyor to the next buffer. If the next buffer is already full, the part waits in the machine, which is logged as blockage by sensor data collected every second. Pallets are collected at M_{19} and resupplied to M_1 . So, there is no loss of productivity due to an insufficient or excessive number of pallets.

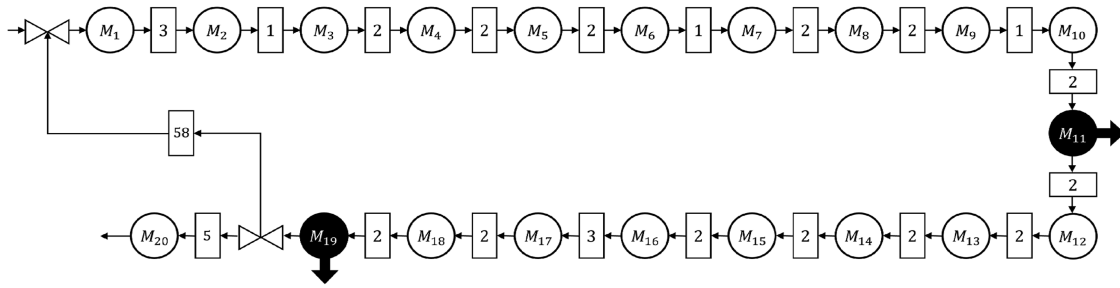


Fig. 1. Structure of the serial assembly line.

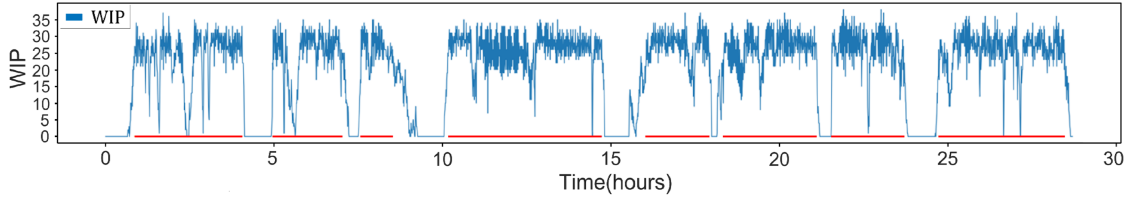


Fig. 2. WIP of the assembly line.

Fig. 1 shows the structure of the assembly line, where the circles and squares indicate the machines and the buffers, respectively. The numbers in the squares denote the capacity of the buffer, and M_i denotes the i th machine. The machines in black eliminate defective parts. The branch at the valve symbol after M_{19} toward M_1 denotes pallet carrying, and there is a buffer with the capacity of 58 for pallets.

This assembly line can be used to produce multiple models. Each machine includes processing modules that can perform processes for multiple models on the same machine. For example, M_3 performs solenoid cover assembly for all models, and M_5 performs console and lever assembly. When the target model is changed, the radio frequency identification (RFID) tag containing information on the target production model is placed on the pallet and passes through the machines. When the information is read by a machine, the settings are automatically altered. Therefore, there is no additional time delay caused by model changes.

The machine breakdown is managed by the fault monitoring system of the line. All machines transmit operation data to the fault monitoring system every second and issue warnings in the event of a breakdown. This fault monitoring system has been developed in our previous research [22]. If a machine breakdown, the assembly process of the machine stops and the part turns defective. Machine breakdown require repair, and defective parts can also occur without machine breakdown. In the case of breakdowns requiring repair, machine repair is performed by workers. Stopping the entire line for one machine repair significantly reduces productivity. Therefore, during the repair of a malfunctioning machine, other machines continue to operate normally.

Machines following the block after service or block before service rules operate automatically until blockage occurs. Consequently, if the repair time becomes longer, the machines before the broken one experience blockage, and subsequent machines

experience starvation. The assembly line machines can handle defective parts and transfer them to inspection machines M_{11} and M_{19} for disposal, if only defective parts are found. Machines except M_{11} and M_{19} identify defective parts and do not proceed with assembly processing, but have a task time similar to that of normal parts due to the automated line process.

B. Data Processing

The data from the assembly line are collected every second using RFID sensors, which includes information on the machine status, pallet number, inspection status, working time, logging time, vehicle model, part number, and specifications. The collected data are saved every hour in a CSV file format.

1) *Work Time*: Because the data are collected all the time, it also includes information when the plant is not operational. Fig. 2 shows the work-in-process (WIP) of the assembly line for one specific day. From the figure, we can know when the line is interrupted. The causes of interruptions include worker break time, machine breakdown, and product changes. Since intentional interruptions are excluded when calculating the efficiency of the machines, it is crucial to identify whether the interruptions are intentional or not from the data. Interruptions within the regular work schedule are considered unintentional. The red horizontal segments at WIP = 0 in Fig. 2 denote the duration of each work time between intentional breaks. The graph in blue denotes the WIP of the line. If this graph hits WIP = 0 in a certain work time, it is considered as an unintentional interruption due to machine breakdown.

2) *Machine Parameters*: In this study, the system parameters of the task time, the average uptime, the average downtime, and efficiency are extracted from factory data and used for performance evaluation. The blockage, starvation, and machine breakdown can shut down machines. However, blockage and starvation are flow problems that do not affect the downtime of

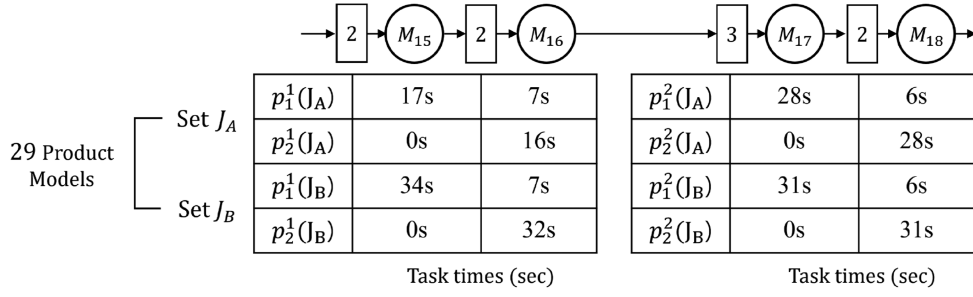


Fig. 3. Multiple working patterns between M_{15} and M_{18} due to pairs of identical machines.

TABLE I
EFFICIENCY OF MACHINES FOR MODEL 46700-H2100

Machine	M_1	M_2	M_3	M_4	M_5
e (%)	100	99.602	99.6	98.585	99.999
Machine	M_6	M_7	M_8	M_9	M_{10}
e (%)	99.21	99.208	99.975	99.573	99.653
Machine	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
e (%)	100	100	99.918	99.784	100
Machine	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}
e (%)	100	100	100	99.995	99.993

the machine. Hence, only machine interruptions due to machine breakdown are regarded as the downtime.

We calculate the average uptime and the average downtime from the collected data. The average uptime represents the average value of the operation time from one breakdown to the next breakdown, and the average downtime represents the average from the breakdown time to the time when the repair is completed. Thus, the efficiency e of the machine can be expressed as $u/(u + d)$, where u is the average uptime and d is the average downtime.

The efficiency e of the machine calculated from actual data varies according to which product model is produced. For example, the efficiency of the machine M_4 is 98.58% for the model 46700-H2100, but 99.37% for the model 46700-M6210. Therefore, we model the target system by calculating the efficiency of the machine for each product model. For example, the efficiency of the model 46700-H2100 is shown in Table I.

3) Buffers: The buffer is the space between machines where the part is held before the next step. The buffer state can be obtained from the difference between the serial numbers of parts in progress for each machine. All parts must pass through M_1 to M_{20} in the order in which they are loaded into the machines. Therefore, there is no scheduling between buffers and modules.

C. Line Characteristics

A distinctive behavior pattern on the assembly line exists between M_{15} and M_{18} . Budget and space constraints prevent the assembly line from operating in parallel. Instead, it is designed using a series of identical machines that perform the same task. In particular, (M_{15}, M_{16}) and (M_{17}, M_{18}) are two pairs that handle the same task, respectively. The calibration process is performed at M_{15} and M_{16} , and the force testing process is performed at

M_{17} and M_{18} . Each model belongs to one of two sets, J_A or J_B . The models in set J_A require the calibration process once, while those in J_B require the calibration process twice. In addition, the two sets have different task times in the force testing process.

Fig. 3 shows the pattern of task times between M_{15} and M_{18} . Each machine pair, (M_{15}, M_{16}) and (M_{17}, M_{18}) , has two patterns: p_1 and p_2 . In p_1 , the parts are assembled at the former machine, M_{15} and M_{17} , and spend a brief verification time at the latter machine, M_{16} and M_{18} . In p_2 , the parts pass the former machine and are assembled in the latter.

Therefore, the working pattern set P for model j is given as

$$P(j) = \{(p_x^1(j), p_y^2(j)) \mid x \in \{1, 2\}, y \in \{1, 2\}\} \quad (1)$$

where $p_x^1(j)$ is the working pattern for (M_{15}, M_{16}) , $p_y^2(j)$ is the working pattern for (M_{17}, M_{18}) , and j is the product model.

D. Scheduling Problem

This study aims to minimize the makespan of the entire stock by selecting the product model and its working pattern for the next loaded part of M_1 . The schedule for each scheduling step $i = 0, \dots, N_S$ is defined as

$$a_i = \{(j_i, p_i) \mid j \in J, p \in P(j)\} \text{ s.t. } \sum_{j \in J} N_j = N_S \quad (2)$$

where N_S denotes the total stocks to be produced, and N_j denotes the stock of model j . J denotes the set of all product models. In our case, $|J|$ is 29, and N_S is 51 200. In addition, Tables II and III show N_j for each model j . $P(j)$ denotes the working pattern set defined in (1). Pattern p_i is applied after M_{15} . However, because the machine's setting in the factory is determined by the RFID tag inserted in M_1 , the product model and the working pattern should be selected at the beginning of each scheduling step i . Each scheduling occurs whenever M_1 is uptime and available, therefore the length of scheduling steps is not fixed.

Moreover, scheduling continues normally even in the case of blockage or breakdown in the line. This is because our system is fully connected in a serial manner, and the insertion of parts does not cause additional blockages. For example, if a breakdown or blockage occurs at M_{i+1} , parts accumulate in the buffer between M_i and M_{i+1} . If the buffer between M_i and M_{i+1} becomes full, inserting a new part causes a blockage in M_i . However, if no new part is inserted, M_i will experience starvation. As M_1 to M_{20} are connected in a serial manner, this relationship holds

TABLE II
MODEL INFORMATION - PART 1

j	N_j	M_1	M_2	M_3	M_4	M_5	M_6
j_1	300	17	15	18	16	7	12
j_2	300	16	16	16	16	16	12
j_3	300	16	15	18	8	11	11
j_4	300	17	16	16	16	11	13
j_5	300	17	14	18	16	7	14
j_6	400	16	15	17	16	7	13
j_7	500	16	13	16	16	7	11
j_8	500	18	16	33	15	3	16
j_9	500	16	15	18	15	7	13
j_{10}	500	17	15	17	16	11	14
j_{11}	500	17	14	33	17	11	12
j_{12}	700	16	23	33	8	7	7
j_{13}	900	16	16	17	17	13	21
j_{14}	900	17	15	16	17	14	12
j_{15}	1000	16	15	15	4	7	12
j_{16}	1000	16	14	18	17	10	41
j_{17}	1000	16	15	36	17	9	27
j_{18}	1000	17	15	17	17	8	12
j_{19}	1300	17	16	18	16	7	15
j_{20}	1300	16	15	10	15	10	12
j_{21}	1500	17	15	29	16	16	12
j_{22}	1600	16	15	16	18	10	29
j_{23}	1700	5	14	34	15	8	12
j_{24}	3200	16	15	33	16	7	13
j_{25}	4300	16	15	17	16	10	11
j_{26}	5100	16	15	16	16	10	19
j_{27}	5500	16	16	14	15	10	12
j_{28}	7200	16	13	9	17	7	13
j_{29}	7600	16	13	18	16	7	12

TABLE III
MODEL INFORMATION - PART 2

j	N_j	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}
j_1	300	17	12	17	11	30	16
j_2	300	16	16	13	9	30	17
j_3	300	16	17	10	30	7	16
j_4	300	11	13	11	29	17	15
j_5	300	16	17	9	29	17	16
j_6	400	16	13	10	30	16	16
j_7	500	13	14	10	29	16	17
j_8	500	16	13	9	31	15	16
j_9	500	15	16	9	30	15	15
j_{10}	500	12	13	9	30	18	16
j_{11}	500	12	15	10	33	17	16
j_{12}	700	7	7	9	32	7	12
j_{13}	900	14	13	13	30	16	17
j_{14}	900	12	13	10	29	16	16
j_{15}	1000	13	13	9	30	16	16
j_{16}	1000	11	13	11	30	17	13
j_{17}	1000	13	17	10	34	16	14
j_{18}	1000	16	17	12	30	16	16
j_{19}	1300	17	17	12	31	16	17
j_{20}	1300	16	13	10	30	16	16
j_{21}	1500	12	4	10	29	16	14
j_{22}	1600	13	13	10	30	16	17
j_{23}	1700	13	13	10	33	16	16
j_{24}	3200	16	13	11	33	16	16
j_{25}	4300	13	13	9	31	16	16
j_{26}	5100	13	13	9	30	16	18
j_{27}	5500	8	16	9	30	16	16
j_{28}	7200	17	13	9	32	16	15
j_{29}	7600	15	13	9	30	16	19

for all i . Therefore, the insertion of a new part does not cause additional delays in the line.

All the machines (except M_{14}, \dots, M_{18}) have different task times for each of the 29 products. Therefore, the scheduling is to select one out of 116 scheduling actions, which is one of four patterns for each of 29 products. This is different from simply scheduling of the 116 products. For example, a part j with a total of N_j stock has four possible cases of $P(j)$. In order to consider them as four different parts j , the constraint of $N_{j_1} + N_{j_2} + N_{j_3} + N_{j_4} = N_j$ must be satisfied. This leads to an additional optimization problem to find appropriate combinations of $(N_{j_1}, N_{j_2}, N_{j_3}, N_{j_4})$. If stock is N and the number of patterns is r , the complexity becomes $\binom{N+r-1}{r-1}$. Therefore, the constraint of identical machines adds significant complexity to scheduling, which highlights the essential need for using DRL-based scheduling.

IV. DEEP REINFORCEMENT LEARNING SCHEDULING

In this section, we introduce DRL-driven scheduling based on the DDDQN. First, we describe the agent and environment of RL. Then, we formulate the target production system as an MDP and introduce a DRL-driven scheduling framework.

A. Agent and Environment of Reinforcement Learning

In RL, the agent takes an action in the environment and learns through feedback. Here, the environment is a virtual production line constructed by the collected data from the actual production line in Section III. Based on the machine efficiency obtained by the factory data, the breakdown event of a machine is implemented as an exponential model. We assume that the breakdown of a machine follows an exponential distribution. The exponential model is one of the most widely used probability models in the related studies [23], [24]. The probability of breakdown Υ is calculated as follows:

$$\Upsilon_M^j(t) = 1 - \exp(-\lambda_E \cdot t). \quad (3)$$

Here, $\lambda_E = -\ln e$, where e is the efficiency of machine M for product model j , and t is the machine's uptime. Because, the breakdown of each machine is monitored in every second by the fault monitoring system, the unit of t in (3) is 1 s.

In our system, the agent is the scheduler that determines the next schedule a_i at every scheduling step i . We assume that the scheduler observes the state of the production system before determining the schedule. This assumption is practically reasonable because the actual system can monitor the production state in real time using RFID sensors.

B. Markov Decision Process

Here, we formulate the production system as an MDP. We consider the practical constraints of identical machines, finite buffers, machine breakdown, and delayed reward.

1) *State*: The state s_i of the production system is observed at every scheduling step i , which is updated when M_1 is empty in its uptime. Table IV gives the components of the state. In the table, N_M denotes the total number of machines in the assembly

TABLE IV
COMPONENTS OF STATE s_i

	Features	Descriptions	Dimension
Machines	Item	Information on the item being processed on the machine containing τ with the applied pattern	$(N_M)^2$
	Residual time	Time left for the machine to finish its job	N_M
	Breakdown	Whether the machine is down or not	N_M
Buffers	Item	Information on the item held in the buffer containing τ with the applied pattern	$N_B \times N_M$
Models	Stock	Remaining stocks by the model	$ J $

τ means the task times of each machine.

line, N_B is the total capacity of the buffer, and $|J|$ is the number of elements in J . The state consists of information on the parts in progress for each machine and whether they are in assembly process or stored in a buffer.

This state design enables recognition of machine breakdowns and defective parts. The breakdown status of each machine is recognized through the breakdown feature, made possible by sending operation data to the breakdown monitoring system every second. If defective parts are removed at M_{11} , the agent can recognize information about their disposal through the item feature of machines and buffers.

We use the task times of each machine, τ (with applying working pattern) for product model identifier. $\tau(a_i, M_i)$ denotes the task times of the product that enters machine M_i through a_i . For example, if a part of schedule a_2 is processed in machine M_3 , the cycle time is $\tau(a_2, M_3)$. The product model identifier for a_2 is defined as $\tau(a_2, M_1), \tau(a_2, M_2), \dots, \tau(a_2, M_{20})$. Using the task times of machines as a model identifier eliminates the need for additional normalization or preprocessing to identify the product model.

2) Action: The action of agent is determining a_i based on the observed s_i . Therefore, the dimension of the possible action is $|J| \times N_P$, where $|J|$ is the number of product models and N_P is the number of possible working patterns. When the agent performs action a_i at s_i , the environment returns the next actionable state s_{i+1} . It should be noted that an empty action $a_i = 0$ is used after $N_S < i$ because there are no more parts in the stock. However, even in this case, there are still parts in the machine and the buffer.

3) Reward: Typically, the reward r_i according to action a_i is determined between s_i and s_{i+1} . Most research on production scheduling formulates an MDP that determines the reward r_i between s_i and s_{i+1} . However, it is ineffective to determine the reward r_i between s_i and s_{i+1} for a serial system. Due to the characteristics of serial production lines, the action a_i , which load a part in M_1 , does not immediately affect the system.

The action a_3 that triggers a state transition from s_3 to s_4 initially affects the machine M_1 only. Then, during the part inserted into the system by a_3 is under production, a_3 affects the products, starvation, and blockage. For example, if the part inserted by a_3 has become a product between s_{40} and s_{41} , then it

means that the system is affected by the action a_3 from s_4 to s_{40} . Therefore, the reward r_3 with respect to the action a_3 cannot be determined immediately after the action, but can be determined only after the action is no more in effect to the system, which is considered as a delayed reward.

The major changes between s_i and s_{i+1} caused by the sequence of previous actions A_p are given as $A_p \subset \{a_k \mid \max(0, i - (N_B + N_M)) < k < i + 1\}$, where i is the current scheduling step, N_B and N_M are the total number of buffers and machines, respectively. Consequently, serial systems are characterized by a delayed environmental reward. In fact, a delayed reward is common in the real world, which means that the reward for an action is available after a certain time period, rather than immediately.

Delayed reward refers to the situation where the agent does not receive an immediate reward for any action it takes, but instead must wait for a certain amount of time step k before receiving feedback on the reward. The delay in delayed reward is typically assumed to be constrained by a polynomial function, representing the time interval during which the agent occupies a state and receives feedback (state observation and reward) [25].

A delayed reward results in bias in temporal difference learning and high deviation in Monte Carlo learning [26]. Traditional solutions for delayed environmental rewards include wait agent, memoryless policies, and the augmented approach [27]. However, these approaches are practically difficult to apply in our case. Wait agent, waiting for k steps to observe the reward for the current action is not feasible, and the use of memoryless policies leads to poor learning performance. Moreover, the augmented approach constructs an MDP with a larger state space of $S \times A_k$ [28], leading to an exponential increase in the state space with respect to k , making it difficult to apply in practice.

Walsh et al. [27] use planning approaches in environments with delayed feedback. The planning approaches rely on the estimation of the most probable current state to determine the transition kernel. State estimation is sensitive to the size of the state space, making it infeasible in domains with large state spaces. Derman et al. [29] propose a method where future k steps are inferred before each decision, and the tuple $(s_{t+k}, r_{t+k}, a_t, s_{t+k+1})$ resulting from executing action a_t is used for learning. However, this approach has the drawback of requiring a fixed value of k to infer states. In addition, Han et al. [30] propose an off-policy method that explicitly decomposes the value function into components consisting of history (H) and current (C) step information. However, this method cannot use general RL algorithms due to the non-Markovian property of the modeling.

In this context, we propose a method to solve the scheduling problem in an environment with delayed rewards by prestoring s_i , a_i , and s_{i+1} and get the reward r_i after k steps for saving it in a replay buffer. The reason we can use this method is that in the MJP serial line environment, we can clearly define the range of time steps that directly affect the action, allowing us to dynamically determine the step k . In the serial line, we can observe the effect of action a_i until step k , where $i + N_M < k < i + N_M + N_B$ is the step until the production

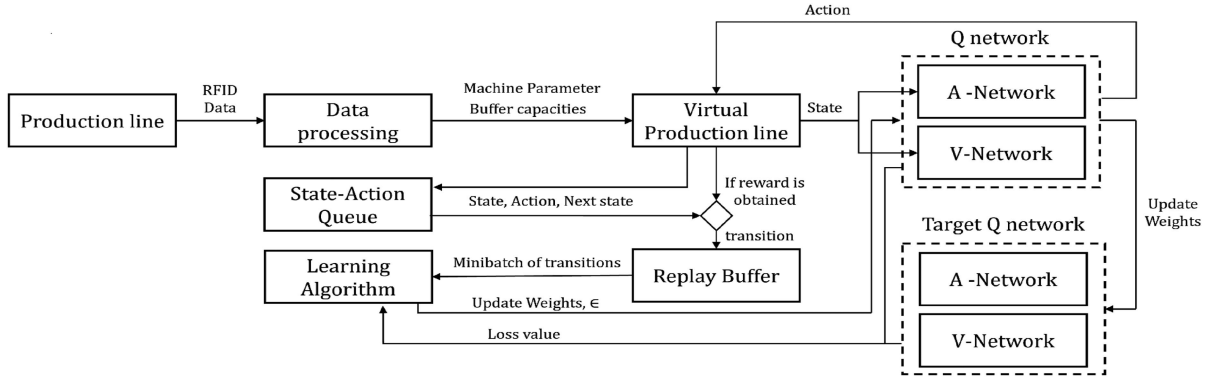


Fig. 4. Overall framework of DRL-driven scheduling.

of action a_i is completed. Therefore, we can extract the reward r_i from the steps separated from s_i and a_i .

Because the objective of scheduling is to minimize the makespan of the entire stock, the sum of the rewards should be designed to minimize T , which is the time required to produce the entire stock N_S . To design the reward, we determine the task time of the part. The loaded part in M_l at step i is expressed as a_i in (2) and the task time of a_i in M_k is expressed as $\tau(a_i, M_k)$. Hence, the sum of every task time of a_i for each machine is given as

$$T_i = \sum_{l=1}^{N_M} \tau(a_i, M_l). \quad (4)$$

Subsequently, the reward r_i for action a_i is expressed as

$$r_i = T_i - (\tau_{\text{out}}(a_i) - \tau_{\text{in}}(a_i)) \quad (5)$$

where τ_{out} and τ_{in} are the end and the start times of production, respectively. The reward r_i for action a_i is not determined between s_i and s_{i+1} , but determined between s_k and s_{k+1} , where $i + N_M < k < i + N_M + N_B$. Because, we can observe $\tau_{\text{out}}(a_i)$ between s_k and s_{k+1} . In (5), the reward r_i is defined as the negative value of the total delay time of the product loaded at step i . Thus, maximizing r_i for all scheduling step i minimizes the total production time T .

We do not include rewards for machine breakdowns and defective parts in our reward function r_i . The causal relationship between machine breakdowns, defective parts, and scheduling in the target system is unknown. Therefore, including machine breakdowns and defective parts in the scheduling reward is not appropriate. The purpose of our reward design is solely to reduce the total production time through scheduling, not to reduce the breakdown rate.

C. DRL-Driven Scheduling

In this study, we choose the DDDQN as the RL method, which is a learning algorithm based on the Deep Q Network (DQN) introduced in [17]. The DDDQN is a double DQN [31] that adopts a dueling method [32].

In DDDQN, the agent selects the action with the highest Q-value by entering the state of the environment into the neural

networks (NN) of the Q-network. The NN is suitable for expressing the possible states of high dimensions. The Q network in DDDQN consists of an advantage network (A-network) and a value network (V-network). Using dueling Q-network for the A-network and the V-network allows the network to learn which state is valuable or not. We adopt a target Q-network to prevent overestimation of the Q-value. The DDDQN uses the Q-network to select the action and obtain the Q-value from the target network with the selected action. The weight of the Q-network is periodically copied to the target Q-network for the stationary target network.

Fig. 4 shows the overall structure of the proposed DRL-driven scheduling framework. The virtual production line is a data-driven event simulator that provides the state of the production line to the Q-network and receives an appropriate action. The state s_i containing the machine and line information is an input into the Q-network, and the Q-value for each action to be executed at the next time step is returned as the output. The Q-value of a_i in s_i is represented as $Q(s_i, a_i; \theta)$ in the Q-network with the weight of θ . The action a_i is selected from the Q-network and the learning algorithm. The details are presented in Algorithm 1.

Algorithm 1 describes the overall DRL-driven scheduling. We use the ϵ -greedy policy in Line 4. With a probability of ϵ , a_i performs a random action. With a probability of $1 - \epsilon$, the action with the highest Q-value is selected. As the episode progresses, ϵ decreases. However, the minimum value of ϵ requires further exploration. The process in Lines 6–18, which determines the part and the pattern to be loaded, is performed when the first machine M_1 accepts the next part.

After taking the action a_i in the state s_i , the state s_{i+1} is returned by the environment. However, the reward r_i is not obtained between s_i and s_{i+1} , but between s_j and s_{j+1} , where $i + N_M < j < i + N_M + N_B$. To resolve this delayed reward issue, the state-action queue SA is adopted. The tuple (s_i, a_i, s_{i+1}) is stored in SA when s_{i+1} is observed. After r_i and d_i are determined in step j , the transition $(s_i, a_i, s_{i+1}, r_i, d_i)$ is saved in the replay buffer B as a batch. Here, d_i is the signal that the episode ends. If there is no stock left, and all buffers and machines have no parts, $d_i = 1$. Otherwise, $d_i = 0$.

The training process in Lines 26–32 is performed if the replay buffer B exceeds a certain threshold. For effective learning,

Algorithm 1: DRL-Driven Scheduling.**Input:** Selection problem**Output:** Q -network

```

1: Initialization: Set  $Q$ -network ( $V$ -network and
    $A$ -network) with random weight  $\theta$ , target network  $\hat{Q}$ 
   with  $\hat{\theta}$ 
2: for  $e = 1, 2, \dots, N_E$  do
3:   Reset virtual product line (Machine, Buffer, Stock)
4:    $\epsilon = \max(0.1, 0.8 - 0.1(e/200))$ 
5:    $i = 0$ 
6:   while  $d_i \neq 1$  do
7:     Observe  $s_i$ 
8:      $x \leftarrow$  random value between 0 and 1
9:     if  $J \neq \emptyset$  then
10:      if  $x < \epsilon$  then
11:         $a_i \leftarrow$  randomly in  $J$  and  $P$ 
12:      else
13:         $a_i \leftarrow \arg\max Q(s_i, a_i; \theta_i)$ 
14:      end if
15:    else
16:       $a_i \leftarrow 0$ 
17:    end if
18:    Load  $a_i$  in line
19:    if  $r_i, d_i$  is determined then
20:      pop  $(s_i, a_i, s_{i+1})$  from  $SA$ 
21:      store transition  $(s_i, a_i, r_i, s_{i+1}, d_i)$  in  $B$ 
22:    end if
23:    Observe  $s_{i+1}$ 
24:    push  $(s_i, a_i, s_{i+1})$  in  $SA$ 
25:  end while
26:  if Size of  $B > N_{tr}$  then
27:    for  $t = 1, 2, \dots, N_{tr}$  do
28:      Sample transitions  $(s_u, a_u, r_u, s_{u+1}, d_u) \in B$ 
29:       $q_u \leftarrow Q(s_u, a_u; \theta_u)$ 
30:       $a^{\max}(s_{u+1}; \theta_u) = \arg \max_{a_{u+1}} Q(s_{u+1}, a_{u+1}) \theta_u$ 
31:       $y_u = r_u + \gamma * \hat{Q}(s_{u+1}, a^{\max}(s_{u+1}; \theta_u); \theta') * d_i$ 
32:      Calculate loss  $L$  from (8)
33:      Perform gradient descent step on  $L$  with  $\theta$ 
34:    end for
35:  end if
   replace  $\hat{Q} = Q$  every  $N_u$  episodes
36: end for
37: return  $Q$ -Network

```

B must be sufficiently large. The learning algorithm trains the Q -network using a minibatch of transitions at the end of each episode. Randomly selected transitions in B are used for learning because of the problem of correlation between samples in Line 28. The learning is repeated N_{tr} times for each episode.

During the training process, the Q -value is calculated as

$$Q(s, a; \theta) = V(s; \theta) + \left\{ A(s, a; \theta) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta) \right\}. \quad (6)$$

The Q -value of the sampled transition is given as q_u in Line 29. The maximum Q -value of the next state s_i is calculated using the target network for a stationary target in Line 31. The loss l is obtained using a smooth $L1$ function in Line 32 [33] as

$$f(y_u, q_u) = \begin{cases} 0.5(y_u - q_u)^2/\beta, & \text{if } |y_u - q_u| < \beta \\ |y_u - q_u| - 0.5\beta, & \text{otherwise} \end{cases} \quad (7)$$

where, y_u represents the actual value, q_u represents the predicted value, and β is a threshold hyper-parameter. This function operates as an L2 loss function, i.e., squared difference between y_u and q_u , when the prediction error is less than β . This enables faster convergence similar to the conventional L2 loss function. On the other hand, when the difference between y_u and q_u exceeds β , the function operates as an L1 loss function, i.e., absolute difference between y_u and q_u . This method helps to reduce the impact of outliers. Moreover, when the difference between y_u and q_u exceeds β , the function adds -0.5β to make it converge to zero. In this study, we set β to 1. As the value of β increases, the function behaves similarly to the L1 loss, and when β equals to 0, it behaves as an L1 loss.

In line 33, the weight θ of the V and A networks is updated using gradient descent update rule. The gradient descent update rule using ADAM is as follows:

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (8)$$

where η is the step size, and \hat{m} and \hat{v} are bias corrected estimators for the first and second momentum, respectively. \hat{m} and \hat{v} are calculated using the moving average and the variance of the slope of the batch, respectively. Here, ϵ is a very small number to avoid dividing by zero. More details can be found, for example, in [21, p. 2]. In Line 35, for the stationary target, the target network is a copied Q -network at the intervals of N_u episodes, rather than every episode. Once the training is finished, the trained Q -network model is returned.

V. EXPERIMENTAL RESULTS

In this section, we validate the proposed scheduling framework based on real-world factory data. First, we describe the datasets and training details. Then, we compare the throughput performance of the proposed scheduling method with that of the conventional rigid scheduling, deterministic scheduling, and random scheduling.

A. Data Sets and Training Details

We use the data collected for six months from a tier-one vendor of a world top-three automobile company. The production line produces 51 200 products for 29 types of models. The Q -network is trained using ADAM [34] with a gradient descent algorithm. The hyperparameters used in the experiment are listed in Table V.

For training, we use collected data with the conventional scheduling method that is simple and repetitive. It should be noted that it is formidable to train the RL scheduling by

TABLE V
HYPERPARAMETERS FOR RL

Hyperparameters	Value
Number of hidden layers	4
Number of nodes in each hidden layers	1 240 930 610 320
ADAM learning rate	5×10^{-3}
Discount rate	0.98
Target Q -network update frequency	20 episodes
Replay buffer size	50 000
Mini-batch size N_{tr}	32
Epsilon convergence	0.01

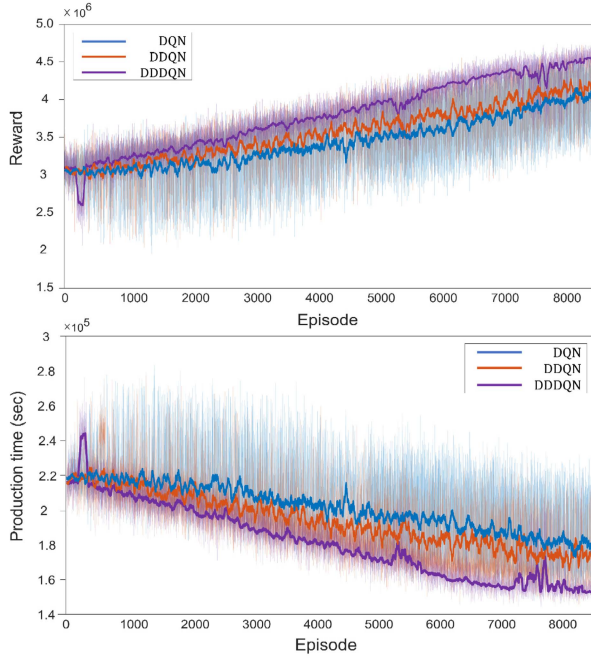


Fig. 5. Production time and reward with respect to the episode.

applying it to the actual factory without complete verification on its effectiveness. Therefore, we conducted the training of RL policies through data obtained from production with the conventional rigid scheduling. In the meantime, it is also possible to train the RL policies with data using other policies because the DQN is an off-policy algorithm. A detailed description of conventional rigid scheduling is provided in Section V-C.

Fig. 5 depicts the learning process with accumulated episodes. Each bold line in Fig. 5 indicates the moving average of the reward and production time, respectively. The blurred lines are the raw data of the reward and production time. As learning progresses, the reward gradually increases, and the production time tends to decrease. This is the same for DQN, DDQN, and DDDQN. Overall, DDDQN shows better learning performance than DQN and DDQN.

B. Effect of the State-Action Queue

We validate the effect of adopting a state-action queue in serial line scheduling by comparing with the case without a state-action queue. The reward function used for comparison

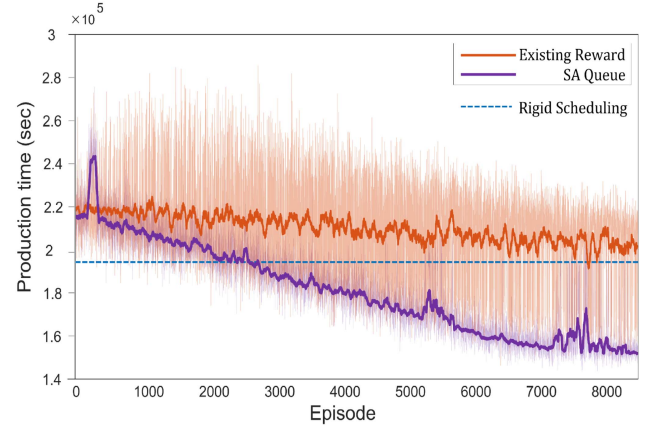


Fig. 6. Production time with respect to the episode. Proposed versus conventional.

is given as

$$r_i = \begin{cases} 0, & N_{\text{out}} = 0 \\ \sum_{k=1}^{N_{\text{out}}} T_k - (\tau_{\text{out}}(a_k) - \tau_{\text{in}}(a_k)), & N_{\text{out}} > 0 \end{cases} \quad (9)$$

where N_{out} denotes the number of products produced between s_i and s_{i+1} . The definitions of the other symbols are the same as those in (5).

The reward design is appropriate for the objective function, which is minimizing the total production time T . However, the learning process with accumulated episodes shows that this reward design is ineffective. Fig. 6 shows the learning process of DDDQN-driven scheduling using different reward functions. The orange and purple lines indicate production times of the scheduling using the reward function in (5) and (9), respectively. The blue dotted horizontal line represents the average production time of rigid scheduling for comparison, which is the conventional scheduling method. We can know from Fig. 6 that the conventional reward design is inefficient because its production times are always above the blue dotted line. On the other hand, scheduling using the state-action queue is effective because its production times become lower than the blue dotted line.

C. Throughput Performance Evaluation

We compare the throughput performance of the proposed DRL-driven scheduling method with those of the conventional scheduling, deterministic scheduling, and random scheduling. The throughput of the scheduling methods are measured through a data-driven discrete-event simulator. We build the simulator model based on the actual data collected for six months from a certain factory line. We use production systems engineering (PSE) theory to model the discrete event of factory line [35]. The convergence of the numerical algorithms in PSE theory has been analytically proven and applied to various real-manufacturing systems. PSE theory has been used for analysis of engine assembly lines and block production lines in Toyota factories [36], which share many similarities with our system. The modeling process of our target system through PSE theory is described in

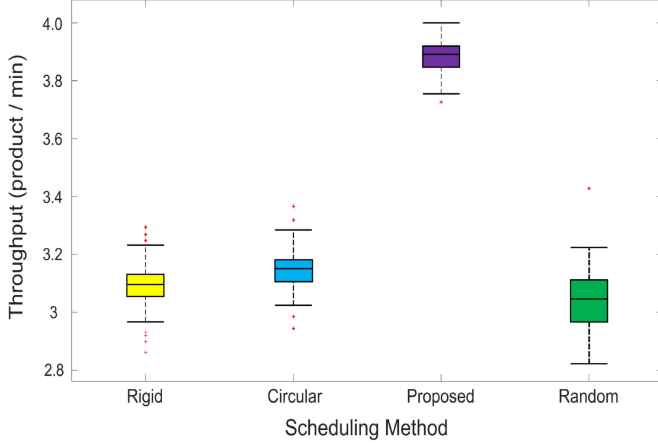


Fig. 7. Throughput performance of the scheduling algorithms.

our previous study [22], [37]. As a result of the validation using actual data, the average throughput of the simulator gives an error of less than 5% with the actual throughput. The data used for validation in this context is distinct from the data used for modeling. In addition, DRL scheduling is implemented using Python's deep learning module PyTorch.

The conventional rigid scheduling is from the raw data analysis of the actual factory. In the rigid method, a_i is determined using the following simple rules:

$$a_i = \begin{cases} (j_f, (p_1^1(j_f), p_1^2(j_f))) & \text{if } i \bmod 2 = 0 \\ (j_f, (p_2^1(j_f), p_2^2(j_f))) & \text{if } i \bmod 2 = 1 \end{cases} \quad (10)$$

where j_f is the first model of the remaining stock J . Thus, until the stock j_f is executed, a_i always chooses model j_f . If scheduling step i is odd, the working pattern is $(p_1^1(j_f), p_1^2(j_f))$. If it is even, the working pattern is $(p_2^1(j_f), p_2^2(j_f))$.

The deterministic scheduling circulates the product model one by one, which is hence titled the circular method. Here, a_i is determined as

$$a_i = \begin{cases} (j_c, (p_1^1(j_c), p_1^2(j_c))) & \text{if } i \bmod 2 = 0 \\ (j_c, (p_2^1(j_c), p_2^2(j_c))) & \text{if } i \bmod 2 = 1 \end{cases} \quad (11)$$

where $c = i \bmod |J|$ is the number of cycles, which equals the types of model left in the stock. Therefore, the model of a_i does not overlap in one cycle. Here, we use the same working pattern scheduling as in (10). Finally, random scheduling is a scheduling method that randomly chooses j_i and p_i .

Fig. 7 shows the throughput performance of the scheduling algorithms. To eliminate the bias caused by failure, we conduct 100 runs for each method. As a result of the experiment, the proposed scheduling method achieves an average performance improvement of approximately 24.2% over the conventional rigid method. As reported in [3], circular scheduling has a higher throughput than rigid scheduling with the same working pattern p_i . In the meantime, random scheduling degrades the throughput performance compared with the rigid method.

TABLE VI
THROUGHPUT COMPARISON OF SCHEDULING METHODS

Method	Average throughput (# of products/min)	Variance	Average improvement over Rigid
Proposed	3.8415	0.0027	24.2%
Circular	3.1475	0.0045	1.7%
Rigid	3.0938	0.0063	0%
Random	3.0376	0.0104	-1.8%

The throughput performance evaluation is summarized in Table VI. DRL-driven scheduling improves the average throughput over rigid scheduling by 24.2% and reduces the variance by approximately 57%. This is because the next schedule a_i in the proposed scheme is dynamically determined by observing the system state s_i .

We compare the blockage and starvation times of each machine to analyze the performance of the proposed scheduling in detail. Blockage occurs when a part cannot be released because the next buffer is full, even though the machine has completed its operation. Starvation is a state in which the machine cannot perform any work even though the machine is in uptime and empty because the previous buffer is empty.

Fig. 8 compares the blockage and the starvation for each machine, respectively. In most machines, the proposed method results in less blockage. In particular, the blockage times from M_6 to M_{16} are significantly reduced. In the meantime, the starvation times from M_7 to M_{16} tend to increase. This is because our reward function aims to minimize the production time of each product. Therefore, the Q-network chooses an action to minimize the blockage time of each product because the blockage time affects the production time of each product more significantly than the starvation time. Therefore, we conclude that DRL-driven scheduling increases the average throughput by dramatically reducing the blockage time at the expense of an increase in the starvation time.

In the rigid method, the starvation times of M_{16} and M_{18} are longer than those of M_{15} and M_{17} . This means that the working pattern of the rigid method causes more loads to be allocated to the former machines M_{15} and M_{17} in each pair of the identical machines, respectively. In the proposed scheduling, the loads on identical machines are well balanced. In addition, the starvation times of M_{15} and M_{16} are similar.

VI. CONCLUSION

In this article, we have investigated the scheduling problem of a multijob serial line under practical constraints of identical machines, finite buffers, machine breakdown, and delayed reward. First, we have formulated an MDP model considering the delayed reward. Based on the formulated MDP, we have proposed a DRL-driven scheduling framework with the DDDQN.

We have evaluated the throughput performance of the proposed DRL-driven scheduling algorithm using real-world factory data collected over six months. Our evaluation results have shown that the proposed scheduling method improves the throughput performance by 24.2% over the conventional one.

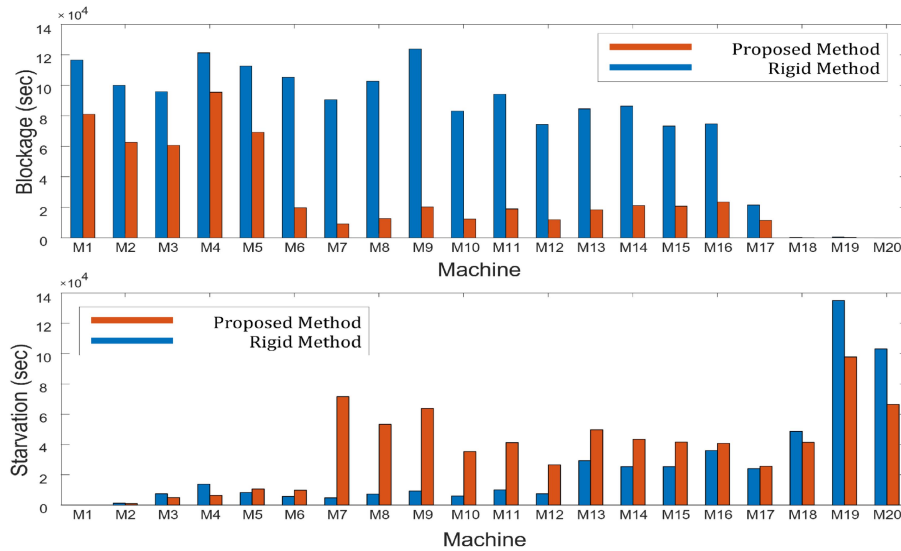


Fig. 8. Blockage time and the starvation time of each machine with respect to scheduling.

Our scheduling results can be widely applicable to industrial processes that adopt MJP serial lines. We expect that our scheduling results will contribute to improving productivity in various industrial fields, such as automotive parts assembly processes [36] and multiproduct furniture manufacturing processes [38], [39].

REFERENCES

- [1] J. Wan et al., "Reconfigurable smart factory for drug packing in healthcare industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 15, no. 1, pp. 507–516, Jan. 2019.
- [2] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep Q network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4276–4284, Jul. 2019.
- [3] P. Alavian, P. Denno, and S. M. Meerkov, "Multi-job production systems: Definition, problems, and product-mix performance portrait of serial lines," *Int. J. Prod. Res.*, vol. 55, no. 24, pp. 7276–7301, 2017.
- [4] C. Dimopoulos and A. M. Zalzal, "Recent developments in evolutionary computation for manufacturing optimization: Problems, solutions, and comparisons," *IEEE Trans. Evol. Comput.*, vol. 4, no. 2, pp. 93–113, Jul. 2000.
- [5] M. Saidi-Mehrabad and P. Fattahi, "Flexible job shop scheduling with tabu search algorithms," *Int. J. Adv. Manuf. Technol.*, vol. 32, no. 5, pp. 563–570, 2007.
- [6] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, 2008.
- [7] R. S. Sutton et al. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.
- [8] I. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 3, pp. 1420–1431, Jul. 2020.
- [9] H. Wang, B. R. Sarker, J. Li, and J. Li, "Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning," *Int. J. Prod. Res.*, vol. 59, no. 19, pp. 5867–5883, 2021.
- [10] J. H. Woo, Y. I. Cho, S. H. Nam, and J.-H. Nam, "Development of a reinforcement learning-based adaptive scheduling algorithm for block assembly production line," in *Proc. Winter Simul. Conf.*, 2021, pp. 1–12.
- [11] T. Zhang, S. Xie, and O. Rose, "Real-time job shop scheduling based on simulation and Markov decision processes," in *Proc. Winter Simul. Conf.*, 2017, pp. 3899–3907.
- [12] X. Ou, Q. Chang, and N. Chakraborty, "A method integrating Q-learning with approximate dynamic programming for gantry work cell scheduling," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 1, pp. 85–93, Jan. 2021.
- [13] D. Shi, W. Fan, Y. Xiao, T. Lin, and C. Xing, "Intelligent scheduling of discrete automated production line via deep reinforcement learning," *Int. J. Prod. Res.*, vol. 58, no. 11, pp. 3362–3380, 2020.
- [14] D. Anghinolfi, M. Paolucci, and R. Ronco, "A bi-objective heuristic approach for green identical parallel machine scheduling," *Eur. J. Oper. Res.*, vol. 289, no. 2, pp. 416–434, 2021.
- [15] J.-H. Lee and H.-J. Kim, "A heuristic algorithm for identical parallel machine scheduling: Splitting jobs, sequence-dependent setup times, and limited setup operators," *Flexible Serv. Manuf. J.*, vol. 33, no. 4, pp. 992–1026, 2021.
- [16] X. Xie and J. Li, "Modeling, analysis and continuous improvement of food production systems: A case study at a meat shaving and packaging line," *J. Food Eng.*, vol. 113, no. 2, pp. 344–350, 2012.
- [17] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [18] X. Ou, Q. Chang, and N. Chakraborty, "Simulation study on reward function of reinforcement learning in gantry work cell scheduling," *J. Manuf. Syst.*, vol. 50, pp. 1–8, 2019.
- [19] L. Zhang, C. Wang, J. Arinez, and S. Biller, "Transient analysis of Bernoulli serial lines: Performance evaluation and system-theoretic properties," *IIE Trans.*, vol. 45, no. 5, pp. 528–543, 2013.
- [20] C.-B. Yan and Z. Zheng, "Problem formulation and solution methodology for energy consumption optimization in Bernoulli serial lines," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 2, pp. 776–790, Apr. 2021.
- [21] X. Wang, Y. Dai, L. Wang, and Z. Jia, "Transient analysis and scheduling of Bernoulli serial lines with multi-type products and finite buffers," *IEEE Trans. Autom. Sci. Eng.*, 2022, early access, Oct. 5, 2022, doi: [10.1109/TASE.2022.3210259](https://doi.org/10.1109/TASE.2022.3210259).
- [22] Y. Won, S. Kim, K.-J. Park, and Y. Eun, "Continuous productivity improvement using IoE data for fault monitoring: An automotive parts production line case study," *Sensors*, vol. 21, no. 21, 2021, Art. no. 7366.
- [23] S. Kim, Y. Won, K.-J. Park, and Y. Eun, "An indirect estimation of machine parameters for serial production lines with Bernoulli reliability model," in *Proc. 59th IEEE Conf. Decis. Control*, 2020, pp. 5540–5545.
- [24] S. Kim, Y. Won, K.-J. Park, and Y. Eun, "A data-driven indirect estimation of machine parameters for smart production systems," *IEEE Trans. Ind. Informat.*, vol. 18, no. 10, pp. 6537–6546, Oct. 2022.
- [25] T. J. Walsh, A. Nouri, L. Li, and M. L. Littman, "Planning and learning in environments with delayed feedback," in *Proc. Mach. Learn.: ECML: 18th Eur. Conf. Mach. Learn.*, 2007, pp. 442–453.
- [26] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: Return decomposition for delayed rewards," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 13544–13555.

- [27] T. J. Walsh, A. Nouri, L. Li, and M. L. Littman, "Learning and planning in environments with delayed feedback," *Auton. Agents Multi-Agent Syst.*, vol. 18, pp. 83–105, 2009.
- [28] K. V. Katsikopoulos and S. E. Engelbrecht, "Markov decision processes with delays and asynchronous cost collection," *IEEE Trans. Autom. Control*, vol. 48, no. 4, pp. 568–574, Apr. 2003.
- [29] E. Derman, G. Dalal, and S. Mannor, "Acting in delayed environments with non-stationary Markov policies," in *Proc. Int. Conf. Learn. Representations*, 2021. [Online]. Available: <https://openreview.net/forum?id=j1RMMKeP2gR>
- [30] B. Han, Z. Ren, Z. Wu, Y. Zhou, and J. Peng, "Off-policy reinforcement learning with delayed rewards," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 8280–8303.
- [31] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, vol. 30, pp. 2094–2100.
- [32] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
- [33] P. J. Huber, "Robust estimation of a location parameter," *Ann. Math. Statist.*, vol. 35, no. 1, pp. 73–101, 1964.
- [34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, vol. 500, 2015.
- [35] J. Li and S. M. Meerkov, *Production Systems Engineering*. Berlin, Germany: Springer, 2008.
- [36] J. Li, "Continuous improvement at Toyota manufacturing plant: Applications of production systems engineering methods," *Int. J. Prod. Res.*, vol. 51, no. 23/24, pp. 7235–7249, 2013.
- [37] Y. Won et al., "Toward implementation of production systems engineering (PSE) method for industry 4.0," Ph.D. dissertation, Daegu Gyeongbuk Inst. of Sci., Technol., Daegu, South Korea, 2022.
- [38] C. Zhao and J. Li, "Analysis and improvement of multi-product assembly systems: An application study at a furniture manufacturing plant," *Int. J. Prod. Res.*, vol. 52, no. 21, pp. 6399–6413, 2014.
- [39] C. Zhao, J. Li, N. Huang, and J. A. Horst, "Flexible serial lines with setups: Analysis, improvement, and application," *IEEE Robot. Autom. Lett.*, vol. 2, no. 1, pp. 120–127, Jan. 2017.



Sanghoon Lee received the B.S. degree in computer science from the School of Undergraduate Studies, Daegu Gyeongbuk Institute of Science and Technology, Daegu, South Korea, in 2022, where he is currently working toward the Ph.D. degree in computer science with the Department of Electrical Engineering and Computer Science.

His research interests include industrial cyber-physical systems and industrial artificial intelligence.



Jinyoung Kim received the B.S. degree in engineering and the M.S. degree in information and communication engineering from the Daegu Gyeongbuk Institute of Science and Technology, Daegu, South Korea, in 2019 and 2021, respectively.

He is currently a Technical Research Personnel with Begas, Seoul, South Korea. His research interests include data-mining and industrial artificial intelligence.



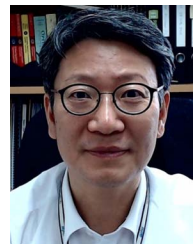
Gwangjin Wi received the B.S. degree in computer science and engineering from Kookmin University, Seoul, South Korea, in 2019, and the M.S. degree in electrical engineering and computer science from the Daegu Gyeongbuk Institute of Science and Technology, Daegu, South Korea, in 2023.

He is currently a Technical Research Personnel with LIG Nex1, Pangyo, South Korea. His research interests include scheduling and resource management in wireless networks.



Yuchang Won received the B.E. degree in computer engineering from Gachon University, Seongnam, South Korea, in 2014, and the M.S. and Ph.D. degrees in information and communication engineering from the Daegu Gyeongbuk Institute of Science and Technology, Daegu, South Korea, in 2016 and 2022, respectively.

He is currently a Senior Engineer with SEMES, Hwaseong, South Korea. His current research interests include production systems engineering, digital twin, digital transformation, and automated material handling systems.



Yongsoo Eun (Senior Member, IEEE) received the B.A. degree in mathematics and the B.S. and M.S.E. degrees in control and instrumentation engineering from Seoul National University, Seoul, South Korea, in 1992, 1994, and 1997, respectively, and the Ph.D. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, MI, USA, in 2003.

From 2003 to 2012, he was a Research Scientist with Xerox Innovation Group, Webster, NY, USA, where he worked on technologies in the xerographic marking process and production inkjet printers. Since 2012, he has been with the Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu, South Korea, and is currently a Professor with the Department of Electrical Engineering and Computer Science and also the Director of DGIST Resilient Cyber-Physical Systems Research Center. His research interests include control systems with nonlinear sensors and actuators, control of quadrotors, communication network, Industry 4.0 production systems, railroad vehicle platooning, and resilient cyber-physical systems.



Kyung-Joon Park (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 1998, 2000, and 2005, respectively.

From 2005 to 2006, he was a Senior Engineer with Samsung Electronics, Suwon, South Korea. From 2006 to 2010, he was a Postdoctoral Research Associate with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA. He is currently a Professor with the Department of Electrical Engineering and Computer Science, Daegu Gyeongbuk Institute of Science and Technology, Daegu, South Korea. His research interests include resilient cyber-physical systems and smart production systems.