Scheduling of Dual-Gripper Robotic Cells With Reinforcement Learning

Hyun-Jung Kim and Jun-Ho Lee

***Abstract*— A dual-gripper robotic cell consists of multiple processing machines and one material handling robot, which can perform an unloading or a loading task one at a time but can hold two parts at the same time. We address a scheduling problem of the robotic cell that determines a robot task sequence when two part types are processed in a different set of machines and all machines have variable processing times within a given interval. The objective is to minimize the makespan. This study proposes a learning-based method, i.e., a reinforcement learning (RL) approach, for the first time, to address a dual-gripper robotic cell scheduling problem. The problem is modeled with a Petri net, a graphical and mathematical modeling tool, which is used as an environment in RL. The states, actions, and rewards are defined by using flow shop scheduling properties, features from a Petri net, and knowledge from previous studies of scheduling robotized tools. Then, the RL approach is compared to the first-in-first-out (FIFO) rule, which is generally used in practice, a swap sequence, which is widely used for cyclic scheduling of dual-gripper robotic cells, and a lower bound. The extensive experiments show that the proposed method performs better than FIFO and the swap sequence; moreover, the gap between the makespan of the proposed method and the lower bound is not large.**

***Note to Practitioners*—We address a scheduling problem of dual-gripper robotic cells with two-part types when all machines have processing time variations. We propose an RL approach to obtain an efficient robot task sequence in order to minimize makespan. The proposed method is performed offline, and a robot task sequence is then obtained instantaneously. The proposed method performs better than the FIFO rule used in practice and the swap sequence used for cyclic scheduling of robotic cells. It can be easily extended for scheduling other configurations of robotic cells.**

***Index Terms*— Dual-gripper robotic cell, reinforcement learning (RL), scheduling, time variations.**

I. Introduction

MODERN manufacturing systems use a computer-controlled material handling robot to transport raw materials or partially completed parts through multiple processing machines to produce a finished product or part. We consider a particular type of automated material handling system used in cellular manufacturing, called robotic cells. Robotic cells consist of a series of processing machines, which conduct different processes on each part, a robot that transports parts between the machines, and an input and an output device [1]. The robot performs loading, unloading, and transporting tasks of parts between the machines that have no intermediate buffer. There are several applications of the robotic cells, such as semiconductor or liquid crystal display (LCD) manufacturing, electroplating, textiles, injection molding of battery components, glass manufacturing and processing, cosmetics, lawn tractors, and 3-D printer-based production systems [1]–[6]. To achieve maximum return on investment while operating such robotic cells, it is essential to obtain effective sequences of the robot tasks and schedules of the parts. Scheduling a robotic cell is similar to a flow shop scheduling problem with blocking and a material handling robot.

Currently, customer order sizes are becoming smaller, and customized products are required. These trends can be identified in semiconductor manufacturing where lot sizes have become significantly small owing to the continual circuit width reductions, larger wafer sizes, and utilization of heterogeneous chips [7]. Automated electroplating lines for processing printed circuit boards (PCBs) must often process a variety of jobs in small quantities that require a different sequence of chemical baths [8]. Several other industries that mostly adopt a make-to-order strategy or produce diverse product types have similar requirements. Because of such small order sizes with different job types, robotic cells are often operated for processing multiple part types at the same time to increase the throughput and efficiency [9], [10]. Cluster tools for semiconductor manufacturing and electric die sorting lines for wafer quality testing are often used by two wafer types concurrently [11]–[13]. In addition, to reduce the setup times and increase the quality, wafer types are normally processed in a different set of machines. There have also been several previous studies on flow shop scheduling problems considering such dedicated machines [14].

In this study, we address a scheduling problem of a buffer-less dual-gripper robotic cell in which two-part types, each containing multiple units, are processed in a different set of machines. Fig. 1 shows a robotic cell layout with six processing machines and a dual-gripper robot. Parts A and B are processed sequentially on machines 1, 2, and 3, and on

Manuscript received October 2, 2020; revised December 4, 2020; accepted December 25, 2020. Date of publication January 15, 2021; date of current version April 7, 2022. This article was recommended for publication by Associate Editor X. Wu and Editor Q. Zhao upon evaluation of the reviewers' comments. This work was supported by the National Research Foundation of Korea (NRF) Grant funded through the Korean Government (MSIT) under Grant 2019R1C1C1004667. *(Corresponding author: Jun-Ho Lee.)*

Hyun-Jung Kim is with the Faculty of Department of Industrial and Systems Engineering, KAIST (Korea Advanced Institute of Science and Technology), Daejeon 34141, Republic of Korea (e-mail: hyunjungkim@kaist.ac.kr).

Jun-Ho Lee is with the Faculty of School of Business, Chungnam National University, Daejeon 34134, Republic of Korea (e-mail: junholee@cnu.ac.kr).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2020.3047924.

Digital Object Identifier 10.1109/TASE.2020.3047924

1545-5955 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

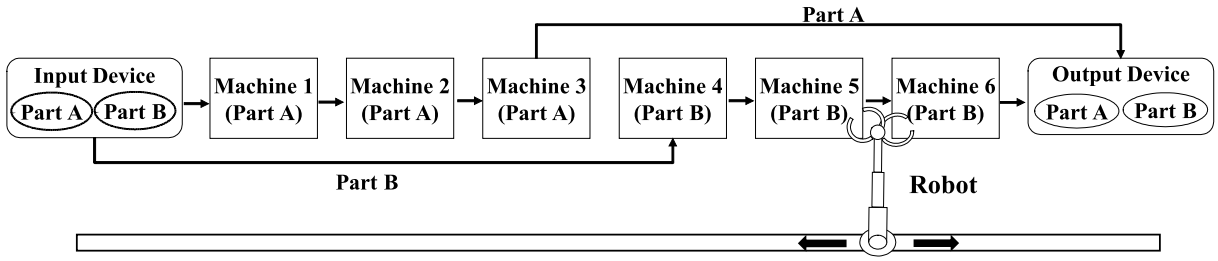


Fig. 1. Dual-gripper robotic cell.

machines 4, 5, and 6, respectively, in the figure. A dual-gripper robot can hold two parts at the same time; however, it performs a loading or an unloading task one at a time. It is known that the throughput of a dual-gripper robot is better than that of a single-gripper robot [15]. The processing time of a unit on each machine is determined randomly between the given minimum and maximum processing times on the machine. Hence, an actual processing time is measured only after the completion of the process. Many manufacturing systems for semiconductors, LCDs, and PCBs demonstrate such processing time variations with a certain distribution, such as uniform or truncated normal distributions [16]–[19]. In fact, the wafer processing times for semiconductor manufacturing are not deterministic in many process steps and vary within a given time range [19]–[21]. Our goal is to determine a robot task sequence for processing all units of two-part types to minimize makespan. The production requirements, such as a dual-gripper robot, two-part types processed in different sets of machines, and variable processing times on all the machines, result in a significantly difficult scheduling problem.

Hence, one of the learning-based methods, reinforcement learning (RL), specifically Q-learning, is adopted for the first time for this problem by using a Petri net and the scheduling properties developed previously. In RL, an agent observes a state from an environment, selects an action based on the state, and receives a reward. It then encounters a new state and subsequently selects an action again. The agent learns the effectiveness of the selected action based on the reward it receives. The RL approach is suitable for our problem because the agent can learn the best action in various states where processing times are changing. Optimization techniques, such as mathematical programming models, branch and bound algorithms, and meta-heuristics can also be used by assuming the average, minimum, or maximum processing time; however, the solutions from those techniques may not be efficient when exact processing times are not provided, and they may need to be solved again if there is even a marginal change in the processing time used.

In practice, priority-based dispatching rules, such as first-in-first-out (FIFO) or longest processing time (LPT) first, are widely used, especially in dynamic scheduling environments. In fact, one of the largest semiconductor fabrication plants in Korea applies such rules to operate machines and achieves good performance when compared to other optimization methods. We note that the FIFO rule is essentially the only efficient and practical method for operating robotic cells with processing time variations when the objective is to minimize the makespan. For cyclic scheduling of dual-gripper robotic cells or dual-armed cluster tools, a swap sequence in which the robot repeats a swap operation (unloading a unit from a machine, switching grippers, and then loading another unit into the machine) for each machine is widely used [5], [17]. However, it has not been defined for noncyclic scheduling of robotic cells with two-part types. We subsequently present a modification of the swap sequence for our problem and compare RL with FIFO, the modified swap sequence, and a lower bound.

II. Literature Review

There have been numerous articles on cyclic scheduling of robotic cells in which the robot repeats a certain sequence to minimize the cycle time. Sethi *et al.* [15] addressed a two-machine robotic cell problem with a dual-gripper robot and identical parts to minimize the steady-state cycle time and showed that a dual-gripper robotic cell can demonstrate productivity that is twice as much as its single-gripper counterpart. Sriskandarajah *et al.* [9] addressed the issues of problem complexity and solvability and identified all potentially optimal robot movement sequences for the part sequencing problem in a two-machine dual-gripper robotic cell. Geismar *et al.* [22] examined a dual-griper robotic cell that processes identical parts and has parallel machines in each processing stage. They obtained a lower bound on the throughput and an optimal solution under certain practical conditions. Gultekin *et al.* [23] considered cyclic scheduling of multiple parts on a robotic cell and proposed a mathematical programming model and meta-heuristics. Dawande *et al.* [24] addressed an interval robotic cell problem with a dual-gripper robot and identical parts where a completed part in a machine must leave the machine within a specified time limit. For two extreme cases, namely no-wait and free-pickup cells, an (asymptotically) optimal solution was obtained in a polynomial time.

There have been some studies on scheduling robotic cells with processing time windows, where a processing time can be chosen within a given time window. Yan *et al.* [25] proposed a branch and bound algorithm for optimal cyclic scheduling in a single-gripper robotic cell by formulating the problem into a set of prohibited intervals of the cycle time. Yildiz *et al.* [26] considered m-machine single-gripper robotic cells with identical parts in which the processing time can be altered by modifying the machining conditions at different manufacturing costs. Yan *et al.* [27] used a tabu search algorithm to solve

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

a cyclic robotic cell scheduling problem with a single-gripper robot where part processing times on each machine are confined to time window constraints. Che *et al.* [28] considered a single-gripper robotic cell scheduling problem with identical parts in order to minimize a cycle time. They assumed that the robot transportation times can have small perturbations and the processing duration of parts can be selected from given intervals. The previous studies on robotic cell scheduling with processing time windows have assumed that engineers can select certain processing times within a given time interval. Hence, processing times are given with certainty in advance. However, in our study, the processing times are uncertain and realized after they are completed. Geismar *et al.* [16] considered a single-gripper robotic cell in which one process has a stochastic processing time; the authors analyzed the time taken for a reverse sequence and the impact of the proximity of the stochastic process to the bottleneck process on the throughput.

Most studies on robotic cell scheduling have assumed a cyclic operation of the robot with identical parts. In addition, many of them have addressed two- or three-machine robotic cells by developing possible robot movement sequences and analyzing the complexity of the problem or optimality conditions of these sequences. The processing time is either fixed or can be chosen by engineers within a given interval, which is different from the random or uncertain processing times we considered in our study. Cyclic schedules require a certain initial state of a robotic cell where some machines have a unit, and each time a robot task sequence for a cycle is performed, the robotic cell returns to the initial state. When identical parts are produced consecutively, cyclic operations of a robotic cell provide optimal cycle times [29]. In addition, the swap sequence provides an optimal cycle time in many practical cases. However, it is not easy to find an optimal robot task sequence when two-part types which do not share a machine are produced simultaneously, with the makespan measure. The robotic cell needs to have a transient period to reach such an initial state of a cyclic schedule and another transient period to empty the robotic cell in practice. When a small number of parts is produced or multiple parts are produced simultaneously in a different set of machines, it is hard to operate the robotic cell in a cyclic manner. Therefore, we assume the noncyclic schedules. There has been no study on noncyclic scheduling of dual-gripper robotic cells with uncertain processing times on all the machines. There have been a few studies on the noncyclic scheduling of robotic cells. Hurink and Knust [30] analyzed the complexity of special cases of a flow shop scheduling problem with a single robot and unlimited buffer space. Carlier *et al.* [31] addressed a robotic flow shop scheduling problem with multiple part types and a single robot by developing a branch and bound algorithm and a genetic algorithm. The authors determined a processing order of n jobs, which was the same for each machine, to minimize the makespan. The readers can refer to [1], [5], [32], [33] for more detailed explanations on scheduling robotic cells.

Cluster tools for semiconductor manufacturing have similar configurations to the robotic cells that we consider. Several studies have addressed cyclic scheduling problems of cluster tools with various requirements [2], [3], [17], [34], [35]. Qiao *et al.* [20] analyzed wafer sojourn times for cyclic scheduling of dual-armed cluster tools when there are processing time variations. Lee *et al.* [11] considered the concurrent processing of two wafer types in the cyclic scheduling of cluster tools and developed optimal robot task sequences. Kim *et al.* [7], [18], [36], [37] developed a mathematical programming model and a branch and bound algorithm based on a Petri net for noncyclic scheduling of cluster tools with the makespan measure. Certain studies have analyzed the completion time of a wafer lot in cluster tools [38], [39]. Most studies on scheduling cluster tools with processing time variations have analyzed the performance or schedulability of a given sequence. A review of scheduling cluster tools can be obtained in [12] and [40]. There have also been some studies on hoist scheduling, which is similar to the robotic cell problem [41]. Che *et al.* [4] addressed a robust optimization problem for a cyclic hoist scheduling problem with hoist transportation time variations. They developed a mixed integer programming model to optimize the cycle time and robustness and identified Pareto-optimal solutions. Amraoui and Elhafsi [8] developed an efficient heuristic algorithm for the hoist scheduling problem with processing time windows to minimize the makespan.

There have been numerous articles on the scheduling of robotic cells, cluster tools, and hoists; however, there has been no work on the scheduling of dual-gripper robotic cells with processing time variations and two-part types processed in a different set of machines, even though there are many practical applications with these conditions, especially in semiconductor, LCD, and automotive parts manufacturing processes. It is significantly difficult to obtain an efficient robot movement sequence in two- or three-machine robotic cells with the makespan minimization measure even if processing time variations or two-part types are not considered. Hence, it is extremely complicated to derive a good robot task sequence with processing time variations on all the machines in addition to two-part types. One of the appropriate techniques for handling this complicated scheduling problem is a learning-based method so that a robot can learn a proper action (in this case a loading or an unloading task) in each state of a robotic cell as the system states change dynamically with uncertain processing times. Hence, we address the problem using an RL approach, specifically Q-learning. RL is extensively used for robot or drone controls, traffic light controls, resource management in computer clusters, games, bidding and advertising, automatic driving, and production scheduling. There are certain studies on scheduling manufacturing systems using RL [42]–[45]. Many of these studies, with the assumption of fixed processing times, have used a selection of dispatching rules as actions, defined states by dividing the remaining processing times into several classes and used a reward as the state transition times or robot transportation times. We define new states, actions, and rewards by considering scheduling properties of dual-gripper robotic cells with a Petri net so that high performance can be achieved with a small number of states and actions and with rewards for reducing the idle times

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

TABLE I
SYMBOLS AND DESCRIPTION

Symbol	Description
M_i	machine i
$\mathbb{M}_\mathbb{A}$, $\mathbb{M}_\mathbb{B}$	sets of machine indexes used by parts A and B, respectively
n_A, n_B	number of units of parts A and B, respectively
n_A', n_B'	number of remaining units of parts A and B, respectively
p_{ij}	processing time of the jth unit on machine i
p_i^{min}, p_i^{max}, p_i^{avg}	minimum, maximum, and average processing times on machine i, respectively
δ	time taken by the robot to move from a machine to the right next machine
τ_u, τ_l	unloading and loading task times, respectively
θ	gripper switching time
$\mathcal{S}$	set of states
$\mathcal{A}(s_t)$	set of actions that can be selected at state s_t
s_i^m	state of machine i
s^d	state of the input device
s^r	state of the robot
S^m	index set of machines having a unit
$p_{i'}$	estimated remaining processing time on M_i assuming p_i^{avg}
$p_{i'}^r$	actual remaining processing time on M_i measured after the process completion
a_i^u, a_i^l	unloading and loading actions on M_i, respectively
M_b^A, M_b^B	bottleneck machines of parts A and B, respectively
s_t, a_t, r_t	state, action, and reward at time step t, respectively

of machines. We now describe our problem in detail. For ease of reference, the symbols used in the following sections are summarized in Table I.

III. PROBLEM DESCRIPTION AND MODELING

A. Problem Description

We consider a robotic cell that has m machines, $M_1, M_2, \ldots, M_m$, without any buffer between machines and an input device and an output device, which are denoted as M_0 and M_{m+1}, respectively. Two part types, A and B, with n_A and n_B units, respectively, are processed in different sets of machines, $\{M_1, M_2, \ldots, M_{m_A}\}$ and $\{M_{m_A+1}, M_{m_A+2}, \ldots, M_{m_A+m_B}\}$, respectively. We denote $\mathbb{M}_\mathbb{A}$ and $\mathbb{M}_\mathbb{B}$ as a set of machine indexes used by parts A and B, respectively, where $\mathbb{M}_\mathbb{A} = \{1, 2, \ldots, m_A\}$ and $\mathbb{M}_\mathbb{B} = \{m_A + 1, m_A + 2, \ldots, m_A + m_B\}$. The dual-gripper robot unloads, loads, and transports all units of parts to be processed in each set of machines. The robot can hold two units at most, one in each gripper and perform an unloading or loading task one at a time. All units of the two parts are stored in the input device at the beginning, and the completed parts are loaded into the output device by the robot. A machine can process one unit at a time, and there is no preemption.

The processing time of the jth unit on M_i, where $1 \le i \le m$, is denoted as p_{ij}, which is determined randomly between $p_i^{\min}$ and $p_i^{\max}$. The symbol, p_i^{avg}, indicates the average processing time on machine i. The values $p_i^{\min}$, $p_i^{\max}$, and p_i^{avg} can be estimated by analyzing the historical operations data. The part type is distinguished by i as parts A and B do not share a machine. Each unit of a part can have a different processing time even on the same machine, and the processing time is measured after the completion of the process. The robot movement time, δ, indicates the time taken by the robot to travel between two consecutive machines M_i and M_{i+1}. Hence, when it travels from M_i to M_k, the moving time of the robot is $\delta|i-k|$. For example, in Fig. 1, if the robot travels from the input device to the output device, it takes a duration of 7δ. The robot unloading and loading times on a machine are denoted as τ_u and τ_l, respectively. The times, τ_u and τ_l, are also used when picking up a unit from the input device and dropping off a unit into the output device, respectively. We use θ for the gripper switching time required to reposition the grippers immediately after one gripper has unloaded a unit from a machine so that the second gripper can load another unit into the same machine. We assume that θ is less than or equal to δ ($\theta \le \delta$) so that when two consecutive machines are served by the different grippers of the robot, the robot can switch the gripper while traveling to the subsequent machine [9]. The objective of this problem is to minimize the makespan, which is the maximum completion time of all units of the two parts. In practice, p_{ij} is much larger than δ, especially in semiconductor or LCD manufacturing. It requires 2–5 s for unloading, loading, and transporting tasks of the robot in cluster tools, whereas it requires 60 to 600 s for several wafer fabrication processes. When θ is very large, the dual-gripper robot is operated as if it has one gripper.

B. Modeling With a Petri Net

To apply an RL approach, we first model the problem with a Petri net. A Petri net, which consists of places, transitions, arcs, and tokens, is used for modeling various discrete event dynamic systems [46]. The places and transitions represent events or activities, arcs indicate the relationship between places and transitions, and tokens are used for entities. A place (or a transition) can have a holding time (or a firing delay) that a token must spend in the place (or a transition). When the input places of a transition have a token and each token stays there for a predefined holding time of each input place, then the transition is enabled and can fire after its firing delay. If the transition fires, the tokens in these input places are eliminated and the output places obtain a token. The system dynamics can be well-modeled with such token firing; therefore, a Petri net model can be used as the environment

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

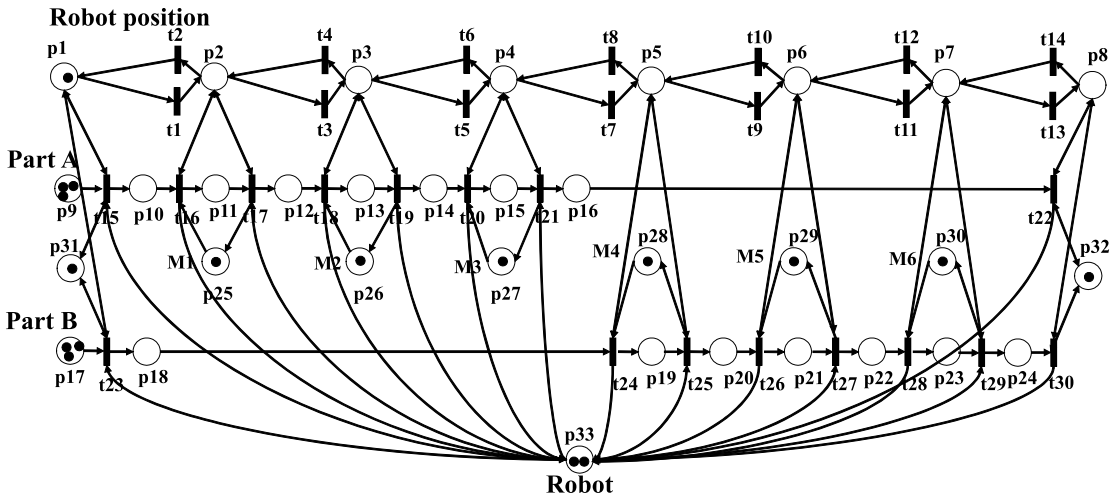


Fig. 2. Petri net model of the robotic cell illustrated in Fig. 1.

that the agent interacts within RL. As it is easy to modify the Petri net model according to different configurations of robotic cells, the proposed RL approach can also be used for other variants of scheduling problems in robotic cells. There have been several existing studies on using a Petri net for modeling and scheduling cluster tools [7], [47], [48].

Fig. 2 shows a Petri net model of the robotic cell in Fig. 1. The places, denoted as pi, where $1 \leq i \leq 33$, indicate the states of the robot, parts, and machines in the robotic cell while the transitions represent the robot tasks, i.e., unloading, loading, and transporting. In particular, pi with a holding time of zero, where $1 \leq i \leq 8$, indicates the robot that is located in front of M_{i-1}; further, p9 and p17, with a holding time of zero, represent the input device with n_A and n_B units of parts A and B, respectively. There are three units of part A and three units of part B in Fig. 2. The places from p10 to p16 represent the states of the units of part A on M_1, M_2, and M_3; p10, p12, p14, and p16 with a holding time of zero denote a unit being transported from the input device to M_1, from M_1 to M_2, from M_2 to M_3, and from M_3 to the output device, respectively; p11, p13, and p15 with a holding time of p_{1j}, p_{2j}, and p_{3j}, respectively, indicate a unit being processed or completed on M_1, M_2, and M_3, respectively. Similarly, the places, p18 to p24 represent the states of the units of part B on M_4, M_5, and M_6, respectively; p18, p20, p22, and p24 with a holding time of zero indicate a unit of part B moving from the input device to M_4, from M_4 to M_5, from M_5 to M_6, and from M_6 to the output device, respectively; p19, p21, and p23 with a holding time of p_{4j}, p_{5j} and p_{6j}, respectively, represent a unit on M_4, M_5, and M_6, respectively. The places from p25 to p30 indicate whether the machines, from M_1 to M_6, are idle or not. The two places, p31 and p32, are used for the gripper switching task in the input and output device, respectively. The places from p25 to p32 have a holding time of θ to indicate the gripper switching time on each machine. The robot availability is represented with p33, which has two tokens at the beginning because the robot does not hold any unit.

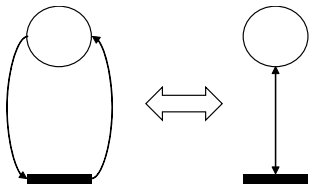


Fig. 3. Modeling with a bidirectional arc.

Note that some places, from p1 to p8 and p31 and p32 in Fig. 2, have arcs with arrows on both sides. Such arcs are used for simpler and better presentation, and it is assumed that the two models in Fig. 3 behave in the same way; one is with two unidirectional arcs and the other is with a bidirectional arc, as illustrated in Fig. 3.

The transitions in Fig. 2, denoted as ti, where $1 \leq i \leq 30$, indicate the robot unloading, loading, and transporting tasks. The transitions, t1, t3, t5, t7, t9, t11, and t13, indicate the movement of the robot from the input device to M_1, from M_1 to M_2, from M_2 to M_3, from M_3 to M_4, from M_4 to M_5, from M_5 to M_6, and from M_6 to the output device, respectively. The transitions, t2, t4, t6, t8, t10, t12, and t14, represent the reverse movement of the robot from M_1 to the input device, from M_2 to M_1, from M_3 to M_2, from M_4 to M_3, from M_5 to M_4, from M_6 to M_5, and from the output device to M_6, respectively. The transitions from t1 to t14 have a firing delay of δ. The transitions from t15 to t30 indicate the unloading and loading tasks of the robot; t15, t17, t19, t21, t23, t25, t27, and t29 with a holding time of τ_u indicate a robot unloading task from the input device, M_1, M_2, M_3, the input device, M_4, M_5, and M_6, respectively; t16, t18, t20, t22, t24, t26, t28, and t30 with a holding time of τ_l represent a robot loading task to M_1, M_2, M_3, the output device, M_4, M_5, M_6, and the output device, respectively. The robot switching task can be measured from two consecutive tasks of unloading a unit from a machine and loading another unit into the machine; the

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

places p25 to p31 have the corresponding holding time. For a dual-gripper robotic cell with two part types processed on m_A and m_B machines, $(m_A+m_B+2)+2(m_A+1)+2(m_B+1)+m_A+m_B+3$ places and $2(m_A+m_B+2)+2(m_A+1)+2(m_B+1)$ transitions are required in a Petri net model.

Before explaining the proposed RL algorithm, we first discuss the derivation of a lower bound on the makespan of the robotic cell scheduling problem for the performance evaluation of our algorithm. We analyze the minimum required time of the resources, which are the machines and the robot, to complete all the processes. Note that $\lceil k \rceil$ denotes a minimum integer value larger than or equal to k.

Theorem 1: A lower bound, LB, on the makespan of a dual-gripper robotic cell with two part types, A and B, processed in different sets of machines is obtained with $\max\{\max_{i\in\mathbb{M}_A} \mathrm{LB}^A_{M_i}, \max_{i\in\mathbb{M}_B} \mathrm{LB}^B_{M_i}, \mathrm{LB}_R\}$, where $\mathrm{LB}^A_{M_i}$, $\mathrm{LB}^B_{M_i}$, and LB_R, which are the minimum required times of M_i for parts A and B and the robot, respectively, are computed with the assumption of $\theta \le \delta$ as follows:

$$\begin{aligned}
\mathrm{LB}^A_{M_i} &= (m_A+1)(\tau_u+\tau_l)+(m_A+m_B+1)\delta \\
&\quad +(\tau_u+\theta+\tau_l)(n_A-1)+\sum_{k=1}^{i-1} p_{k1}+\sum_{j=1}^{n_A} p_{ij} \\
&\quad +\sum_{k=i+1}^{m_A} p_{kn_A}, \quad 1 \le i \le m_A \\
\mathrm{LB}^B_{M_i} &= (m_B+1)(\tau_u+\tau_l)+(m_A+m_B+1)\delta \\
&\quad +(\tau_u+\theta+\tau_l)(n_B-1)+\sum_{k=m_A+1}^{i-1} p_{k1}+\sum_{j=1}^{n_B} p_{ij} \\
&\quad +\sum_{k=i+1}^{m_A+m_B} p_{kn_B}, \quad m_A+1 \le i \le m_A+m_B \\
\mathrm{LB}_R &= (m_A+1)(\tau_u+\tau_l)n_A+(m_B+1)(\tau_u+\tau_l)n_B \\
&\quad +\left\lceil \frac{(m_A+m_B+1)\delta(n_A+n_B)+(m_A+m_B+1)\delta(n_A+n_B-1)}{2} \right\rceil.
\end{aligned}$$

The proof of Theorem 1 is provided in the Appendix. LB can be computed when all the processing times, p_{ij}, for $1 \le i \le m_A$ (or $m_A+1 \le i \le m_A+m_B$), $1 \le j \le n_A$ (or $1 \le j \le n_B$), are realized after the completion of all the processes. We later compare the makespan from the proposed algorithm to the calculated lower bound, LB.

IV. Reinforcement Learning Approach

We consider a standard RL setting where an agent interacts with environment $\mathcal{E}$ over a number of discrete time steps [49]. At time step t, the agent receives state s_t, and selects action a_t according to its policy π, which indicates a distribution over actions given states $\pi(a|s)=\mathbb{P}[a_t=a|s_t=s]$. After selecting and performing an action, the agent receives reward r_t and encounters the next state s_{t+1} at time step $t+1$. The objective is to maximize the expected return from state s_t, where the return is computed by $R_t=\sum_{k=0}^{T}\gamma^k r_{t+k}$ from time step t with a discount factor $\gamma \in (0, 1]$ where time step T is a terminal state. The action value, $Q^\pi(s,a)=\mathbb{E}[R_t|s_t=s,a_t=a]$, indicates the expected return starting from state s, taking action a, and then following the policy π. When the agent reaches a terminal state where all the processes of units of parts A and B are completed, one episode is completed. The initial state is when all units of parts A and B are in the input device and all machines are idle. We denote $\mathcal{S}$ and $\mathcal{A}(s_t)$ as the sets of states and actions that can be selected at state s_t, respectively.

We use a Q-learning algorithm, one of the extensively used off-policy algorithms in RL. In this algorithm, the agent chooses an action using a behavior policy but updates $Q(s_t,a_t)$, an action value of choosing a_t in s_t, toward a value of an alternative action to learn a target policy. We use an ϵ-greedy policy, which chooses an action having the maximum action value with a probability of $1-\epsilon$ and selects another action randomly with a probability of ϵ, for the behavior policy. A greedy policy, which chooses an action with the maximum action value, is used for the target policy in our problem. After an agent selects action a_t and receives reward r_t at time step t, $Q(s_t,a_t)$ is updated to $Q(s_t,a_t)+\alpha[r_t+\gamma \max_{a\in\mathcal{A}(s_{t+1})} Q(s_{t+1},a_{t+1})-Q(s_t,a_t)]$, where α is a learning rate and $\alpha \in (0, 1]$. When α is close to one, Q values are updated quickly whereas they are never updated when α is close to zero. When the discount factor, γ, is close to one, it implies that the future rewards are worth as much as the immediate rewards, whereas the immediate rewards are worth significantly more when γ is close to zero. We now define the states, actions, and rewards for scheduling dual-gripper robotic cells.

A. States

A state that an agent encounters should contain information helpful for choosing an action, i.e., the unloading and loading tasks of the robot. The robot can recognize the robotic cell state based on the following: whether the machines are processing a unit or are idle, the remaining processing times are short, the input device contains units of parts A and B, or the number of units that the robot is holding. The Petri net model that describes the system dynamics can then be used for representing such states of the robotic cell. In Fig. 2, if p11, p15, p21, and p2 have a token, and p9 and p17 have one and two tokens, respectively, we can observe that M_1, M_3, and M_5 have a unit and M_2, M_4 and M_6 are idle; furthermore, the robot is in front of M_1 and there are one and two units of parts A and B, respectively, in the input device.

Hence, a marking that indicates the position of tokens in all places at each time step can be used for defining the system states. Then, the number of states is $2^{30}\times 3\times(n_A+1)\times(n_B+1)$ in Fig. 2. If all of them are required for the robot to choose an action, it is necessary to use an approximate function for $Q(s_t,a_t)$ with neural networks or linear combinations of features owing to a very large number of states. However, only a portion of the places is sufficient for defining states. For example, p11 and p25 in Fig. 2 cannot have a token at the same time; therefore, it is sufficient to have token positions in p11, p13, p15, p19, p21, and p23 for the machines without considering p25, p26, p27, p28, p29, and p30. In addition, since the number of tokens in p33, p10, p12, p14, p16, p18, p20, p22, and p24 must be two, the number of tokens in

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

p33 represents the robot state sufficiently whether it is empty, holding one unit, or two units. The remaining processing times are also helpful for choosing an action of unloading a unit. However, not all machines having a unit are required for choosing an action; only a few machines with short remaining times are required to be considered in many practical cases.

By extracting important information from a Petri net, state s_t at time step t is defined as $[s_1^m, s_2^m, \ldots, s_{m_A+m_B}^m, s^d, s^r]$, where s_i^m, $1 \leq i \leq m_A + m_B$, which represents a state of M_i, takes an integer value between zero and three; further, s^d for a state of the input device, takes zero or one by comparing the workload of parts A and B, and s^r indicates the number of available grippers of the robot. $s_i^m = 0$ if $i = \text{argmin}_{l \in S^m}\{p_{l'} - |k-l|\delta\}$, where S^m is an index set of machines having a unit at time step t. Here, M_k is the machine where the robot is located, and $p_{l'}$ is the estimated remaining processing time on M_l at time step t, which is obtained by assuming p_l^{avg} as the actual processing time. Furthermore, $s_i^m = 1$ if $i = \text{argmin}_{l \in S^m, l \neq i^*}\{p_{l'} - |k-l|\delta\}$, where M_{i^*} is the machine with $s_{i^*}^m = 0$; moreover, $s_i^m = 2$ for all the other machines in S^m. If M_i is not included in S^m, i.e., if there is no unit in M_i, then $s_i^m = 3$. The term, $p_{l'} - |k-l|\delta$, indicates the estimated remaining time of a process on M_l when the robot arrives at M_l from M_k. If the process has already been completed, then $p_{l'}$ becomes zero. Hence, there is a high probability of the process on M_{i^*} finishing at the earliest time; furthermore, a process on M_i with $s_i^m = 1$ might be completed after M_{i^*}. Therefore, the states of machines are identified based on whether they have a unit or not and the estimated remaining time is the shortest or the second shortest time if the machine has a unit.

The input device state is obtained by computing the workload of the remaining units of parts A and B. We use n'_A and n'_B to indicate the remaining number of units of parts A and B, respectively, at time step t. s^d is one if $n'_A \times \max_{i \in \mathbb{M}_\mathbb{A}} p_i^{\text{avg}} \geq n'_B \times \max_{l \in \mathbb{M}_\mathbb{B}} p_l^{\text{avg}}$, and zero otherwise. The term $n'_A \times \max_{i \in \mathbb{M}_\mathbb{A}} p_i^{\text{avg}}$ indicates the remaining workload of part A. Even if n'_A and n'_B are 10 and 5, respectively, it might be better to unload a unit of part B first if $\max_{i \in \mathbb{M}_\mathbb{A}} p_i^{\text{avg}}$ and $\max_{l \in \mathbb{M}_\mathbb{B}} p_l^{\text{avg}}$ are 20 and 100, respectively. Hence, the total workload is used to determine the state of the parts in the input device. s^r is zero if the robot is holding two units, one if one gripper is holding a unit, and two if the two grippers are empty.

B. Actions

When an agent (the robot) observes state s_t, an action of unloading or loading a unit should be chosen. As the robot tasks are represented by the transitions in the Petri net model, the enabled transitions without considering token sojourn times are included in a set of actions, $\mathcal{A}(s_t)$, at state s_t. An enabled transition is identified by checking whether all of its input places (except for places indicating robot locations) have a token; such an enabled transition can fire after the maximum holding time of these places. For example, t15 and t23, indicating unloading a unit of part A and part B, respectively, are enabled in Fig. 2.

TABLE II
STATE TRANSITION TIME

Action	State Transition Time
$a_0^{u_A}$ and $a_0^{u_B}$, $k \neq 0$	$k\delta + \tau_u$
$a_0^{u_A}$ and $a_0^{u_B}$, $k = 0$	$\theta + \tau_u$
a_i^u, $1 \leq i \leq m_A + m_B$, $i \neq k$	$\max\{p_{i'}^r, \lvert k-i\rvert\delta\} + \tau_u$
a_i^u, $1 \leq i \leq m_A + m_B$, $i = k$	$\max\{p_{i'}^r, \theta\} + \tau_u$
a_i^l, $1 \leq i \leq m_A + m_B + 1, i \neq k$	$\lvert k-i\rvert\delta + \tau_l$
a_k^l	$\theta + \tau_l$

A set of all actions can be defined as $[a_0^{u_A}, a_0^{u_B}, a_1^u, a_2^u, \ldots, a_{m_A+m_B}^u, a_1^l, a_2^l, \ldots, a_{m_A+m_B}^l, a_{m_A+m_B+1}^{l_A}, a_{m_A+m_B+1}^{l_B}]$, where $a_0^{u_A}$ (or $a_0^{u_B}$) indicates an unloading task of a unit of part A (or B) from M_0; the input device, $a_{m_A+m_B+1}^{l_A}$ (or $a_{m_A+m_B+1}^{l_B}$) denotes a loading task of a unit of part A (or B) into $M_{m_A+m_B+1}$, i.e., the output device; further, a_i^u and a_i^l represent unloading and loading tasks on M_i, respectively. An unloading action, a_i^u, for M_i includes moving from M_k, where the robot is positioned at time step t, to M_i, waiting if the process on M_i is not completed, and unloading a unit from M_i. A loading action, a_i^l, for M_i includes moving to M_i and loading a unit into M_i.

A state transition time from s_t to s_{t+1} is the time required to perform a_t at time step t. If a_t is $a_0^{u_A}$ when the robot is in front of M_2, it requires a duration of $|2-0|\delta + \tau_u$ to perform the action. If a_t is a_2^u, the robot can unload a unit on M_2 after its process is completed, which takes a duration of $p_{2'}^r + \tau_u$, where $p_{2'}^r$ indicates the actual remaining time on M_2, which is measured after the completion. Hence, when the robot is located at M_k, and an action related to M_i is chosen, the transition time from s_t to s_{t+1} can be obtained as listed in Table II.

In Fig. 2, the transitions from t15 to t30 become a set of all actions; [$a_0^{u_A}$, $a_0^{u_B}$, a_1^u, a_2^u, a_3^u, a_4^u, a_5^u, a_6^u, a_1^l, a_2^l, a_3^l, a_4^l, a_5^l, a_6^l, $a_7^{l_A}$, $a_7^{l_B}$] indicates t15, t23, t17, t19, t21, t25, t27, t29, t16, t18, t20, t24, t26, t28, t22, t30. It is to be noted that t15 (or t23), i.e., $a_0^{u_A}$ (or $a_0^{u_B}$), indicates an unloading action of a unit of part A (or B) from the input device, and t16, i.e., a_1^l, represents a loading action of a unit of part A into M_1.

When the robot selects an action, it should be careful to avoid a deadlock in which the robot cannot perform any task further. A deadlock is demonstrated in a dual-gripper robotic cell only if the robot holding a unit to be processed on M_k unloads another unit for processing M_i even though both M_k and M_i already have a unit. Therefore, when s^r is one and an unloading action, a_{i-1}^u (or $a_0^{u_A}$, $a_0^{u_B}$) is selected, a deadlock avoidance condition indicating whether the number of tokens in the two places representing M_i and M_k is two must be verified before performing the action. We note that the two actions, $a_{m_A}^u$ and $a_{m_A+m_B}^u$, can be selected without checking the condition because the output device can always receive completed units.

C. Rewards

After an agent selects and performs action a_t at time step t, reward r_t is obtained from the environment $\mathcal{E}$. The makespan of basic flow shop scheduling can be computed by the sum of the idle time of the last machine before starting the first

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

TABLE III
REWARD

Action	Reward
$a_0^{u_A}$ and $a_0^{u_B}$, $k \neq 0$	$\min\{0, p_{b'}^r - (k\delta + \tau_u)\}$
$a_0^{u_A}$ and $a_0^{u_B}$, $k = 0$	$\min\{0, p_{b'}^r - (\theta + \tau_u)\}$
a_i^u, where $1 \leq i \leq m_A + m_B$, $i \neq b, k$	$\min\{0, p_{b'}^r - (\max\{p_{i'}^r, \lvert k - i\rvert\delta\} + \tau_u)\}$
a_i^u, where $1 \leq i \leq m_A + m_B$, $i \neq b, i = k$	$\min\{0, p_{b'}^r - (\max\{p_{i'}^r, \theta\} + \tau_u)\}$
a_i^l, where $1 \leq i \leq m_A + m_B + 1, i \neq k, b$	$\min\{0, p_{b'}^r - (\lvert k - i\rvert\delta + \tau_l)\}$
a_k^l, $k \neq b$	$\min\{0, p_{b'}^r - (\theta + \tau_l)\}$
a_b^u	$\min\{0, p_{b'}^r - \lvert k - b\rvert\delta\}$
a_b^l	0

process on the machine, the total processing times of all jobs on the machine, and the waiting time of the last machine measured between consecutive processes [50]. Hence, one possible reward is to use the idle and waiting times of the last machine so that these times on the last machine can be minimized as the total processing times do not depend on the robot task sequence. The idle and waiting times of the other machines can also be used. Another way for defining a reward is to use the state transition time because the sum of state transition times is the makespan.

It is well-known that minimizing the idle and waiting times on the bottleneck process can always increase the throughput or efficiency of the system. Hence, we attempt to increase the utilization of the bottleneck machine by reducing the idle and waiting times on the machine. As two parts are processed concurrently, two bottleneck machines for each part are considered and a negative value of the sum of idle and waiting times on the two machines is used as a reward. Let M_b^A and M_b^B denote the bottleneck machine of parts A and B, respectively, which are determined by $b = \text{argmax}_{i \in \mathbb{M}_\mathbb{A}(\text{or } \mathbb{M}_\mathbb{B})} p_i^{\text{avg}}$. Reward r_t is then defined as $r_t^A + r_t^B$, where r_t^A and r_t^B are rewards from M_b^A and M_b^B, respectively; moreover, each reward depending on an action is computed according to Table III with the assumption of the robot location, M_k, at time step t.

When a_i^u, where $i \neq b$, is selected, if M_b^A (or M_b^B) is in progress until the action is completed, then r_t^A (or r_t^B) of zero is received because $p_{b'}^r \geq \max\{p_{i'}^r, |k - i|\delta\} + \tau_u$. However, if M_b^A (or M_b^B) is empty at time step t, then r_t^A (or r_t^B) of $-(\max\{p_{i'}^r, |k-i|\delta\}+\tau_u)$, which indicates the idle time of M_b^A (or M_b^B), is obtained. The term $\max\{p_{i'}^r, |k-i|\delta\}+\tau_u$ indicates the robot action time or state transition time when performing a_i^u. If a_i^l, for $i \neq k, b$, is chosen, it takes $|k - i|\delta + \tau_l$ time units to perform the action; hence, the idle time, as much as $p_{b'}^r - (|k-i|\delta+\tau_l)$, is computed for M_b^A or M_b^B and used for r_t^A or r_t^B, respectively. If a_k^l, where $k \neq b$, is performed, the robot switches the position of the two grippers and then loads a unit into M_k; hence, $p_{b'}^r - (\theta + \tau_l)$ is used as a reward. When a_b^u is selected, if $p_{b'}^r - |k - b|\delta$ is larger than zero, the robot must wait for the process on M_b^A or M_b^B; then, r_t^A or r_t^B becomes zero, respectively. Otherwise, the machine is required to wait for the robot for a maximum duration of $-p_{b'}^r + |k - b|\delta$. For a_b^l, while loading a unit into M_b^A or M_b^B, r_t^A or r_t^B of zero is received, respectively. Even if the robot task time is large when compared to the processing times, this reward can work well because minimizing the idle and waiting times of the two machines can still result in the reduction of the makespan.

We subsequently present a comparison of the proposed r_t with five other rewards, i.e., r_t^1, r_t^2, r_t^3, r_t^4, and r_t^5; r_t^1 is obtained using r_t^A if $n_A \times \max_{i \in \mathbb{M}_\mathbb{A}} p_i^{\text{avg}} \geq n_B \times \max_{l \in \mathbb{M}_\mathbb{B}} p_l^{\text{avg}}$, or else using r_t^B; r_t^2 is computed using r_t^A if $n'_A \times \max_{i \in \mathbb{M}_\mathbb{A}} p_i^{\text{avg}} \geq n'_B \times \max_{l \in \mathbb{M}_\mathbb{B}} p_l^{\text{avg}}$ or else using r_t^B; r_t^3 is the same as r_t except that only r_t^A (or r_t^B) is used after all n_B (or n_A) units of part B (or A) are completed; r_t^4 is the state transition time, the robot action time; and r_t^5 uses the sum of idle and waiting times of the two last machines, M_{m_A} and $M_{m_A+m_B}$. The results are in the Appendix.

D. Modified Q-Learning Algorithm

In a basic Q-learning algorithm, Q values are updated at each time step of an episode, and the updated Q values in the terminal state of an episode are used as the initial Q values for the next episode. When one iteration consisting of multiple episodes is executed, the final Q values are used for determining an action on each state by choosing the action with the maximum Q value. We modify the basic Q-learning algorithm for scheduling the dual-gripper robotic cell with processing time variations. In the proposed Q-learning algorithm, L iterations, each consisting of E episodes, are repeated to learn the Q values. The E episodes in each iteration have the same set of processing times, which are randomly generated between $p_i^{\min}$ and $p_i^{\max}$ for each unit on machine i. The L iterations have a different set of processing times. In the beginning, the Q values are initialized to zero for each pair of a state and an action and then updated in each time step of an episode. The updated Q values in an episode are used as the initial Q values of the subsequent episode, and the final Q values of the last episode in an iteration are considered in the first episode of the next iteration.

The Q values in the last episode in the Lth iteration are not used for determining a robot task sequence, unlike the basic Q-learning algorithm. We use the Q values that provide the minimum gap from LB among all episodes in all iterations for determining the robot task sequences. We denote Q_l^e as the Q values of the eth episode in the lth iteration and Q_l as the Q values that provide the minimum gap from LB among E episodes in the lth iteration. The gap, $G(Q_l^e)$ (or $G(Q_l)$), is computed by ((makespan with Q_l^e(or Q_l) $-$ LB)/LB). Then, the final Q_{l^*} (also denoted as Q^*) is obtained with $l^* = \text{argmin}_{1 \leq l \leq L}\{G(Q_l)\}$. The following is the modified Q-learning algorithm. It is to be noted that the proposed algorithm outperformed the basic Q-learning algorithm in the preliminary experiments.

In the modified Q-learning algorithm, the parameters γ, α, ϵ, E, and L should be set first. Then, for each time step t of an episode in an iteration, the Q values of the greedy policy are updated while the agent follows the ϵ-greedy policy. After E episodes in the lth iteration, Q_l is computed and Q^* is updated. Even if $e < E$, if the Q values converge, which implies that the maximum difference between the Q values for the same pair of a state and action of two consecutive episodes is less

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Modified Q-Learning Algorithm

Initialize parameters: discount rate $\gamma \in [0, 1]$, learning rate $\alpha \in [0, 1]$, small $\epsilon > 0$, E, and L.
Initialize $Q(s, a)$ to zero for all $s \in \mathcal{S}$, and $a \in \mathcal{A}(s)$ and set $e = 1$ and $l = 1$
Repeat for each iteration:
  Repeat for each episode:
    Initialize s as an empty robotic cell with $n_A + n_B$ units in the input device
    Repeat for each time step t of episode e:
      Choose a_t from s_t using an ϵ-greedy policy
      Take action a_t, observe r_{t+1}, s_{t+1}
      $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t +$
      $\gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
      $s_t \leftarrow s_{t+1}$;
    Until s_t is terminal
    Compute $G(Q_l^e)$ and set $e = e + 1$;
  Until $e == E$ or the Q values converge
  Compute Q_l, update Q^*, and $e = 1$, $l = l + 1$;
Until $l == L$

than 0.0001, the corresponding iteration is finished, and the next iteration starts. When the algorithm is terminated after L iterations, the final Q^* is used for generating a robot task sequence for new problem instances.

V. Experiments

We first performed preliminary experiments for comparing the six reward functions, r_t, r_t^1, r_t^2, r_t^3, r_t^4, and r_t^5 and for determining parameters, α, γ, ϵ, E, and L. The results are illustrated in *B. Parameter Setting* in the Appendix. We use α, γ, and ϵ of 0.3, 0.9, and 0.3, respectively, and E of $(n_A + n_B) \times 500$ and L of 10, which provided the minimum gap in the preliminary experiments.

A. Other Sequences

The proposed RL method is compared to FIFO and the modified swap sequence. It is to be noted that FIFO is widely used in practice with the makespan measure especially when there are processing time variations. The swap sequence has been proven to be optimal in cyclic scheduling of a dual-gripper robotic cell or a dual-armed cluster tool in most practical cases [5], [12], [51].

The robot following the FIFO rule selects an action (unloading or loading task), which can be conducted at the earliest time. When both grippers are empty, a possible action is an unloading task from the input device or the machine with a unit. The robot unloads part from the input device (if s^d is one, it unloads part A; otherwise, it unloads part B) if all the machines with a unit are still processing. If there are multiple machines that have already finished processing, the robot unloads a unit from the machine that completed the process at the earliest time. When one gripper is holding a unit while another is empty, an action of loading the unit is chosen if the corresponding machine is empty. If the machine is not empty but has just finished processing, an action of unloading a unit from the machine is performed. Otherwise, an action of unloading a unit from another machine, which completed the process at the earliest time, is conducted. A deadlock condition is also checked when unloading a unit. When both grippers are holding a unit, loading one of the units into a machine should be the next action. If two machines, where two units on the grippers are to be processed, are empty, the unit that was unloaded first is loaded. If only one machine is empty, the corresponding unit is loaded into the machine.

In a swap sequence, the robot repeats a swap operation of unloading a unit from M_i on a gripper, switching the grippers, loading a unit on another gripper into M_i, and moving to M_{i+1}, for all machines in order. The swap sequence is defined for cyclic operations of robotic cells with identical parts. Therefore, it should be modified for noncyclic scheduling of the robotic cell with two-part types. Assume the Petri net model in Fig. 2. The initial startup period, where all machines are empty at the beginning and then filled with a unit, is performed in the sequence of $a_0^{u_A}$, a_1^l, $a_0^{u_B}$, a_4^l, $a_0^{u_A}$, a_1^u, a_1^l, a_2^l, $a_0^{u_B}$, a_4^u, a_4^l, a_5^l, $a_0^{u_A}$, a_1^u, a_1^l, a_2^u, a_2^l, a_3^l, $a_0^{u_B}$, a_4^u, a_4^l, a_5^u, a_5^l, a_6^l. Once all machines have a unit, the swap operation is performed for each part type one at a time, i.e., SWAP(A): $a_0^{u_A}$, a_1^u, a_1^l, a_2^u, a_2^l, a_3^u, a_3^l, $a_7^{l_A}$ for part A, and SWAP(B): $a_0^{u_B}$, a_4^u, a_4^l, a_5^u, a_5^l, a_6^u, a_6^l, $a_7^{l_B}$ for part B. When $n_A \leq n_B$, SWAP(A) and SWAP(B) are performed alternatively as many as $n_A - m_A$ times, and then only SWAP(B) is conducted as many as $n_B - n_A - m_A$ times. The close-down period, where all machines having a unit become empty, is operated in the sequence of a_1^u, a_2^u, a_2^l, a_3^u, a_3^l, $a_7^{l_A}$, a_4^u, a_5^u, a_5^l, a_6^u, a_6^l, $a_7^{l_B}$, a_2^u, a_3^u, a_3^l, $a_7^{l_A}$, a_5^u, a_6^u, a_6^l, $a_7^{l_B}$, a_3^u, $a_7^{l_A}$, a_6^u, $a_7^{l_B}$. The swap sequence for the startup or close-down period was modified based on previous studies [35], [52]. It is worth noting that developing an efficient and sophisticated sequence by modifying the swap sequence for scheduling robotic cells is beyond the scope of this study.

B. Experimental Results

Experiments were performed on the dual-gripper robotic cell shown in Fig. 1, where parts A and B were processed on M_1, M_2, M_3 and M_4, M_5, and M_6, respectively. Note that the proposed algorithm can also handle other configurations of the robotic cell easily, such as a single-gripper robot with a swap ability and a self-buffered robot [53], [54]. First, the Petri net model should be modified so that the robot which can transport one unit at a time but also swap two units on a machine or the robot with a buffer can be represented. Then we believe that the proposed states, actions, and rewards can be used without significant modification when the makespan minimization is considered.

The proposed algorithm was coded with JAVA and executed in a PC with Intel Core i7-8700K (3.7 GHz) CPU and 16-GB RAM. With the parameters determined in the previous section, the RL algorithm was compared to FIFO, the modified swap sequence, and the calculated lower bound, LB.

1) Balanced Processing Times: Table IV lists 18 cases for the experiments, which were initially divided into two time variation levels of 10% and 20%; then, a combination

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
EXPERIMENTS WITH BALANCED PROCESSING TIMES

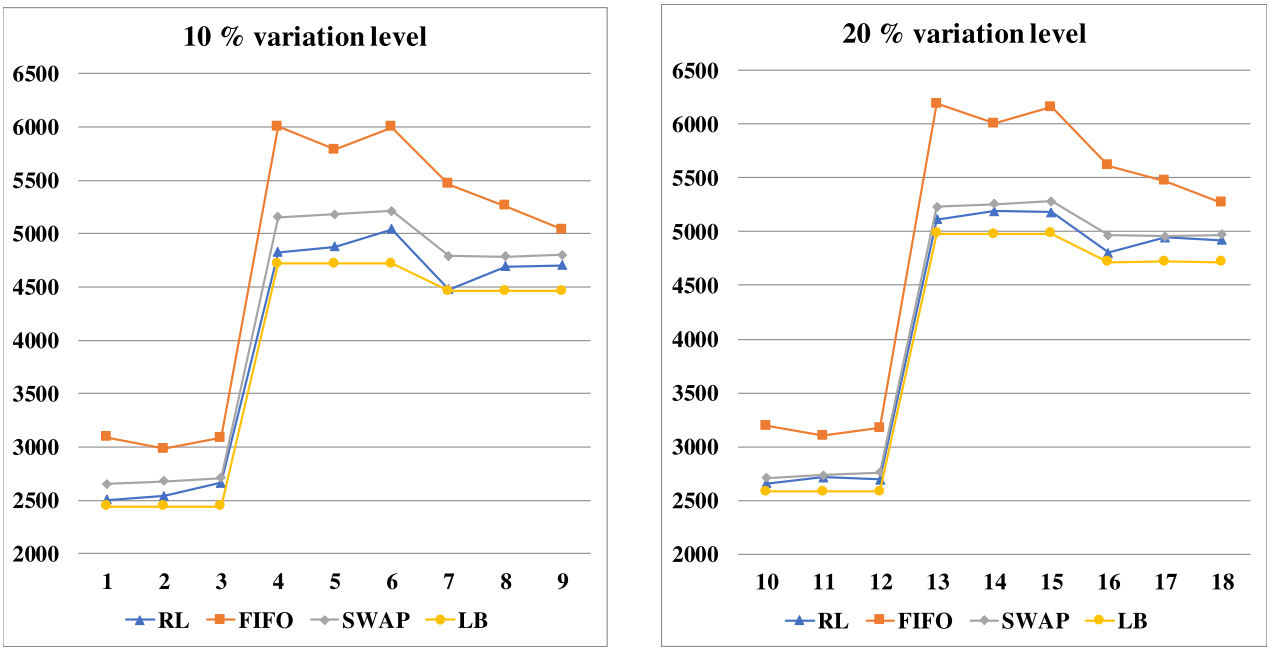
Var	No.	n_A	n_B	P_A / P_B	RL	FIFO	SWAP	LB	Gap(S)	Gap(L)	LT(s)
10%	1	25	25	[65,75],[70,80],[80,90]/ [75,85],[65,75],[60,70]	2504.7	3086.9	2654.0	2444.0	6.0	2.5	220.2
	2	25	25	[80,90],[70,80],[65,75]/ [75,85],[65,75],[60,70]	2540.7	2979.3	2679.1	2443.3	5.4	4.0	215.6
	3	25	25	[80,90],[70,80],[65,75]/ [60,70],[65,75],[75,85]	2665.0	3082.5	2711.0	2444.2	1.7	9.0	206.7
	4	50	50	[65,75],[70,80],[80,90]/ [75,85],[65,75],[60,70]	4824.3	6002.9	5153.9	4718.5	6.8	2.2	936.0
	5	50	50	[80,90],[70,80],[65,75]/ [75,85],[65,75],[60,70]	4874.2	5783.1	5179.4	4718.2	6.3	3.3	941.2
	6	50	50	[80,90],[70,80],[65,75]/ [60,70],[65,75],[75,85]	5038.8	5993.8	5210.2	4718.0	3.4	6.8	947.6
	7	25	50	[65,75],[70,80],[80,90]/ [75,85],[65,75],[60,70]	4477.2	5461.4	4791.0	4458.2	7.0	0.4	501.8
	8	25	50	[80,90],[70,80],[65,75]/ [75,85],[65,75],[60,70]	4691.6	5255.6	4783.4	4460.5	2.0	5.2	493.9
	9	25	50	[80,90],[70,80],[65,75]/ [60,70],[65,75],[75,85]	4703.7	5034.4	4798.2	4458.6	2.0	5.5	492.4
20%	10	25	25	[65,85],[70,90],[80,100]/ [75,95],[65,85],[60,80]	2655.4	3193.2	2708.0	2578.1	2.0	3.0	221.5
	11	25	25	[80,100],[70,90],[65,85]/ [75,95],[65,85],[60,80]	2713.2	3101.1	2732.4	2579.7	0.7	5.2	205.5
	12	25	25	[80,100],[70,90],[65,85]/ [60,80],[65,85],[75,95]	2693.9	3169.8	2760.9	2579.4	2.5	4.4	218.7
	13	50	50	[65,85],[70,90],[80,100]/ [75,95],[65,85],[60,80]	5112.4	6189.6	5232.5	4980.2	2.3	2.7	961.5
	14	50	50	[80,100],[70,90],[65,85]/ [75,95],[65,85],[60,80]	5190.9	6008.3	5255.2	4978.3	1.2	4.3	882.1
	15	50	50	[80,100],[70,90],[65,85]/ [60,80],[65,85],[75,95]	5182.7	6158.1	5284.2	4979.3	2.0	4.1	946.4
	16	25	50	[65,85],[70,90],[80,100]/ [75,95],[65,85],[60,80]	4803.5	5613.6	4967.0	4718.2	3.4	1.8	509.2
	17	25	50	[80,100],[70,90],[65,85]/ [75,95],[65,85],[60,80]	4946.1	5471.8	4958.3	4719.2	0.2	4.8	455.9
	18	25	50	[80,100],[70,90],[65,85]/ [60,80],[65,85],[75,95]	4917.8	5267.6	4969.8	4717.8	1.1	4.2	502.1

of n_A and n_B values and different locations of bottleneck machines were considered. The processing time ranges were determined by reflecting the real data from the semiconductor manufacturing processes and were assumed to follow a uniform distribution. Generally, etching and deposition processes require more than 60 s and other processes require more than 300 s in the manufacturing steps. It is worth noting that when the processing times are large, it is relatively easy to obtain a good robot task sequence; consequently, the gaps from LB are also small, which can be observed in cases 5 and 6 in Table IX in the Appendix. In the preliminary experiments, when processing times were large, the two methods, RL and the modified swap sequence, performed similarly, and the gap from LB was very small. Therefore, in this section we focus on the instances with small processing times as it is difficult to obtain a good robot task sequence for such cases; the processing times were set from 60 to 100 s. All the robot tasks, i.e., τ_l, τ_u, δ, and θ, were assumed to take two time units. For each case in Table IV, we obtained the Q^* values with the proposed algorithm, and 100 instances, which were randomly generated within the given time ranges, were tested with the Q^* of the proposed RL algorithm, FIFO, SWAP (the modified swap sequence), and LB, as listed in Table IV. The experiments were repeated ten times to demonstrate convincing results. The average makespan from each of the methods is listed along with the gap from SWAP (Gap(S)) and LB (Gap(L)) in Table IV. Gap(S) is computed by (((makespan from SWAP − makespan from RL) × 100)/ makespan from RL). The gap from FIFO is not presented as it is larger than Gap(S) in all cases. Gap(L) is computed as (((makespan from RL—LB) × 100)/LB). The last column in Table IV shows a learning time (LT), which is the time the proposed method requires to obtain Q^*, which was performed offline. It takes less than 1 s to obtain robot task sequences for 100 newly generated instances. Fig. 4 shows the average makespan from RL, FIFO, SWAP, and LB for each case.

Cases 1–9 and cases 10–18 in Table IV have the same set of n_A and n_B and the same location of bottleneck machines; however, they have different processing time variation levels of 10% and 20%, respectively. Cases 1–3 (or cases 10–12) have 25 units for both part types; moreover, the bottleneck machines of these cases are M_3, M_4, and M_1, M_4, and M_1, M_6 for parts A and B, respectively. We assume three different sets of processing times (see those of Cases 1–3), and they are repeatedly used for 18 cases.

Gap(S) is large in cases 1 and 2 (or 4 and 5), whereas Gap(L) is large in case 3 (or 6). When M_3 and M_4 or M_1 and M_4 are the two bottleneck machines for parts A and B, respectively, as in cases 1 and 2, performing SWAP(A) and SWAP(B) one at a time may result in more waiting times

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.



(a) Average makespan of Cases 1 to 9

(b) Average makespan of Cases 10 to 18

Fig. 4. Comparison of RL to FIFO, SWAP, and LB. (a) Average makespan of cases 1–9. (b) Average makespan of cases 10–18.

on the machines when compared to the process using RL. In these cases, it might be better to perform actions for parts A and B alternatively, instead of moving units of the same part type consecutively (from M_1 to M_3 or from M_4 to M_6); thus, the waiting times of the bottleneck machines can be reduced. This is also the reason for the large value of Gap(L) in case 3 (or 6) as RL provides a sequence similar to SWAP when M_1 and M_6 have large processing times. When n_A and n_B become large, Gap(S) increases but Gap(L) tends to decrease, as can be seen from cases 1–3 and cases 4–6. As LB becomes tight with large values of n_A and n_B, the gap from LB is reduced. When the workload between two types is not balanced, as in cases 7–9, Gap(S) in case 7 is large but it is small in cases 8 and 9. Gap(L) shows the opposite result as case 7 has a small gap and cases 8 and 9 have large gaps. In case 7, RL provides an almost optimal sequence, which is fairly different from SWAP, whereas RL and SWAP provide similar sequences in cases 8 and 9. When cases 5 and 8 are compared, Gap(S) is decreased and Gap(L) is increased, which implies that the RL algorithm tends to provide a sequence similar to SWAP when the number of units for two-part types is different and M_1 and M_4 are the bottlenecks. We can observe that both Gap(S) and Gap(L) are decreased when comparing cases 6 and 9.

We can also observe in Fig. 4 that FIFO does not work well when the machine loads are relatively well-balanced, as in cases 1–6, when compared to cases 7–9. Cases 10–18 in Fig. 4(b) demonstrate similar features to those in Fig. 4(a); however, the average makespan from RL, SWAP, and FIFO and LB are larger, and the overall gap from SWAP and LB is smaller.

The average values of Gap(S) for cases 1–9 and cases 10–18 are 4.5% and 1.7%, respectively. As the processing time variation becomes larger from 10% to 20%, Gap(S) is smaller. We can see that, with the 20% level, the average processing times of two-part types on machines are increased compared to those of the 10% variation level, whereas the robot task times are the same. Therefore, the robot can have more idle times between tasks, which reduces the impact of robot task sequences on the makespan. Hence, SWAP can also contribute to minimizing the idle or waiting times of the bottleneck machines in this case, which may lead to the smaller Gap(S). The average values of Gap(L) for the first and subsequent nine cases are 4.3% and 3.8%, respectively, which can be considered to be small. The gaps between RL and FIFO for the two variation levels are 17.7% and 15.8%, respectively. The average LT is 547.7 s for all cases. From the overall result, we can observe that the proposed RL algorithm performs better than FIFO and SWAP, which are two of the most popular methods in operating dual-gripper robotic cells or dual-armed cluster tools. It can be noted that the average gaps between SWAP and LB for the 10% and 20% variation levels are 9.0% and 5.6%, respectively. We note that the LB may not be tight because it only contains the minimum workload of machines and the robot. The inevitable idle or waiting times of machines or robots are not included in computing the lower bound. Nevertheless, the average gap for all cases in Table IV is 4.1%, which can be considered to be small. It might be beneficial to develop a tighter lower bound in the future.

We further increased n_A and n_B to 100 and applied the RL algorithm with the processing time ranges in cases 1–3 (10% variation level) and cases 10–12 (20% variation level). Table V lists the results for the six new cases, i.e., cases 19–24. Similar to cases 1–6, Gap(S) increases whereas Gap(L) decreases with the 10% variation level. The average gaps from SWAP and LB with the 20% variation level decrease in cases 23 and 24 and case 22, respectively, while others

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

TABLE V
EXPERIMENTS WITH 100 UNITS OF PARTS A AND B

Var	No.	n_A	n_B	P_A / P_B	RL	FIFO	SWAP	LB	Gap(S)	Gap(L)	LT(s)
10%	19	100	100	[65,75],[70,80],[80,90]/ [75,85],[65,75],[60,70]	9445.2	11810.9	10153.5	9270.4	7.5	1.9	4048.0
	20	100	100	[80,90],[70,80],[65,75]/ [75,85],[65,75],[60,70]	9520.8	11405.4	10179.4	9268.4	6.9	2.7	4056.9
	21	100	100	[80,90],[70,80],[65,75]/ [60,70],[65,75],[75,85]	9630.7	11828.3	10211.1	9267.2	6.0	3.9	4026.3
20%	22	100	100	[65,85],[70,90],[80,100]/ [75,95],[65,85],[60,80]	9947.5	12192.5	10278.7	9780.6	3.3	1.7	4211.6
	23	100	100	[80,100],[70,90],[65,85]/ [75,95],[65,85],[60,80]	10245.1	11816.1	10300.3	9777.1	0.5	4.7	4159.2
	24	100	100	[80,100],[70,90],[65,85]/ [60,80],[65,85],[75,95]	10205.8	12141.1	10329.7	9779.9	1.2	4.3	4198.9

TABLE VI
EXPERIMENTS WITH UNBALANCED PROCESSING TIMES

Var	No.	n_A	n_B	P_A / P_B	RL	FIFO	SWAP	LB	Gap(S)	Gap(L)	LT(s)
10%	25	25	25	[40,50],[50,60],[90,100]/ [85,95],[50,60],[45,55]	2678.6	3168.1	2723.9	2649.1	1.7	1.1	221.2
	26	25	25	[90,100], [50,60],[40,50]/ [45,55],[50,60],[85,95]	2669.5	3030.8	2809.4	2649.5	5.2	0.8	221.4
	27	25	25	[40,50],[50,60],[90,100]/ [40,50],[50,60],[45,55]	2674.1	3147.8	2683.6	2649.3	0.4	0.9	203.4
	28	50	50	[40,50],[50,60],[90,100]/ [85,95],[50,60],[45,55]	5217.4	6215.9	5268.0	5173.3	1.0	0.9	883.2
	29	50	50	[90,100], [50,60],[40,50]/ [45,55],[50,60],[85,95]	5200.3	5939.5	5354.7	5174.0	3.0	0.5	943.0
	30	50	50	[40,50],[50,60],[90,100]/ [40,50],[50,60],[45,55]	5197.3	6190.5	5228.9	5174.0	0.6	0.4	933.6
20%	31	25	25	[40,60],[50,70],[90,110]/ [85,105],[50,70],[45,65]	2830.6	3298.6	2857.2	2784.3	0.9	1.7	204.3
	32	25	25	[90,110],[50,70],[40,60]/ [45,65],[50,70], [85,105]	2828.3	3153.3	2946.5	2783.0	4.2	1.6	209.6
	33	25	25	[40,60],[50,70],[90,110]/ [40,60],[50,70],[45,65]	2808.7	3229.8	2810.7	2783.5	0.1	0.9	205.9
	34	50	50	[40,60],[50,70],[90,110]/ [85,105],[50,70],[45,65]	5508.3	6441.1	5529.0	5433.4	0.4	1.4	884.8
	35	50	50	[90,110],[50,70],[40,60]/ [45,65],[50,70], [85,105]	5488.1	6179.3	5619.2	5433.5	2.4	1.0	905.7
	36	50	50	[40,60],[50,70],[90,110]/ [40,60],[50,70],[45,65]	5457.7	6343.2	5472.2	5434.5	0.3	0.4	873.5

demonstrated a marginal increase. The overall average gap from LB is 3.6%, which is smaller than that with the previous cases.

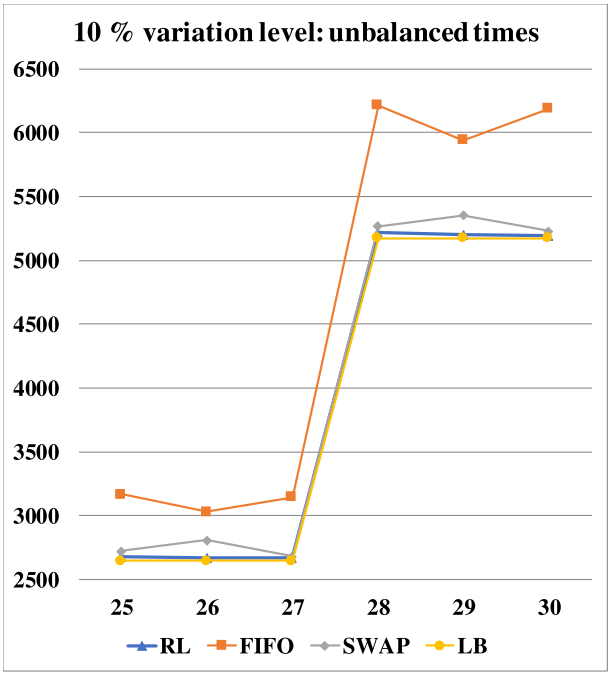
SWAP consists of three robot task sequences for each of the initial startup, steady state, and close-down periods. The robot repeats a swap sequence for each part type during the steady-state period. Therefore, the robotic cell applied with SWAP is operated cyclically except for the startup and close-down periods. Therefore, when 100 units are considered as in Table V, the robot, which follows SWAP, performs cyclic operations for producing more than 90 units of each part type. In this perspective, SWAP may represent the cyclic solutions whereas RL provides noncyclic schedules. We can observe that Gap(S) is large when a 10% variation level is assumed. Of course, a better sequence than SWAP needs to be developed by considering two-part types in the future.

2) Unbalanced Processing Times: The processing times on machines in cases 1–18 are well-balanced, which reflects practical situations in semiconductor manufacturing. Depending on job types and manufacturing environments, certain processes may require a longer time. Hence, we further verified the performance of the algorithm with unbalanced processing times where one machine has a particularly large processing time.

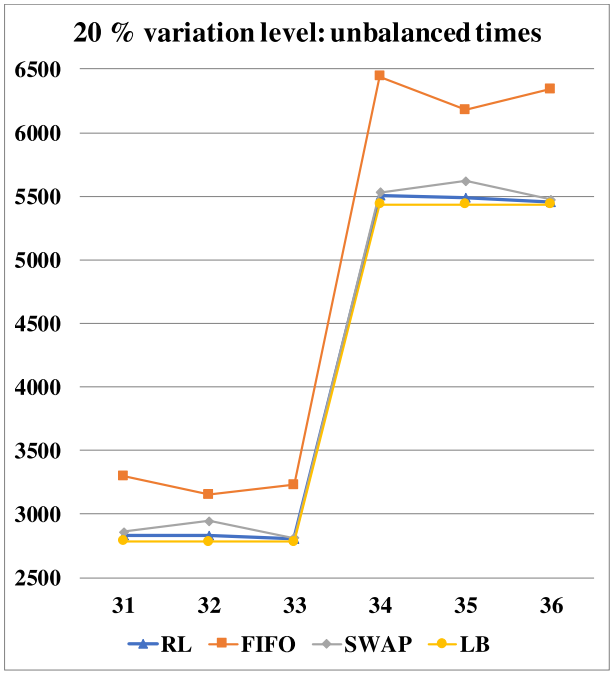
Table VI lists another 12 cases with unbalanced processing times. In cases 25 and 26, M_3 and M_4, and M_1 and M_6 have large processing times, respectively; further, in case 27, all machines have a small processing time except for M_3. Cases 28–30 demonstrate the same set of processing time ranges as cases 25–27 with a different number of units. Cases 31–33 are similar to cases 25–27 except for the processing time variation level.

When processing times are unbalanced, we can expect that the swap sequence works well because when it performs SWAP(A) [or SWAP(B)], the robot can arrive at the bottleneck machine before it finishes processing due to the much smaller processing times on the preceding machines. This can reduce the waiting time of the bottleneck machine. The machines with small processing times need to wait for the robot to unload a unit; hence, the robot can perform tasks on those machines

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.



(a) Average makespan of cases 25 to 30



(b) Average makespan of cases 31 to 36

Fig. 5. Comparison of RL to FIFO, SWAP, and LB with unbalanced processing times. (a) Average makespan of cases 25–30. (b) Average makespan of cases 31–36.

quickly without waiting and then go to the bottleneck machine. Fig. 5 shows the average makespan from RL, FIFO, and SWAP, and LB for all cases listed in Table VI.

Gap(S) and Gap(L) in Table VI are smaller than those in Table IV. The RL approach works well with unbalanced processing times because the minimization of the idle or waiting times of the bottleneck machines has a significant impact on the makespan in this case. The overall average of Gap(S) for the 10% and 20% variation levels are 2.0% and 1.3%, respectively; further, they are 0.8% and 1.2% for Gap(L). This implies that the proposed RL algorithm provides near-optimal solutions and also performs slightly better than SWAP, which is expected to provide good solutions. The average LT is 557 s for all cases. We note that, in Table IV where processing times are relatively well-balanced as in practical production environments, the average gap between RL and SWAP was 3.1% and the maximum gap was 7.0% in case 7. In addition, the average gap was 6.8% when 100 units are produced with a 10% variation level in Table V. Therefore, RL performs much better than SWAP in practical situations but has a similar performance when unbalanced processing times are considered.

When M_3 and M_4 are the two bottlenecks of parts A and B, respectively, or M_3 has the largest processing time as in case 25 or 27 (or case 28 or 30), respectively, Gap(S) and Gap(L) are very small. However, as it can be observed in Fig. VI, when M_1 and M_6 have large processing times as in case 26 or 29, Gap(S) is relatively large whereas Gap(L) is very small, which is the opposite result to case 3 or 6. When SWAP is applied, the robot performs unloading, switching, and loading tasks for M_1, M_2, and M_3 and then for M_4, M_5, and M_6. Hence, in case 26, M_6 may be required to wait for the robot after it finishes processing as M_1 has a larger processing time. In this case, RL provides a sequence different from SWAP and attempts to reduce idle times on both M_1 and M_6; hence, Gap(L) is small. When n_A and n_B are increased to 50, Gap(S) and Gap(L) tend to decrease for both the 10% and 20% variation levels. The gap between RL and FIFO is 17% and 15% with the 10% and 20% variation levels, respectively. Cases 31–36 have features similar to cases 25–30. In Fig. 5, we can observe that when the variation level is increased, the average makespan is increased, and Gap(S) and Gap(L) decrease marginally. FIFO does not work well especially with a large number of units. We also applied the proposed algorithm to a robotic cell with a single type (only part A) to verify the performance of RL with identical units. The average value of Gap(L) for cases 1, 2, 4, 5, 19, and 20 with n_B equal to zero is 0.26%,

3) Comparison to Single-Gripper Robotic Cell: We now compare the performance of the dual-gripper robotic cell with that of a single-gripper robotic cell where the robot can hold one unit at a time. The proposed RL algorithm is applied for both robotic cells, and p33 in Fig. 2 has one token. In a single-gripper robotic cell, s^r in state s_t is not used, and an action of unloading a unit from M_i, transporting it, and loading it into M_l is considered because the only action the robot can perform with a unit on its gripper is to load it into the next machine. Table VII shows the results for cases 19–24, in Table V, which have 100 units of both part types. As we can expect, the average makespan of the dual-gripper robotic cells is smaller than that of the single-gripper robotic cells, and the average gap between dual-gripper and single-gripper

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
EXPERIMENTS WITH 100 UNITS IN DUAL-GRIPPER AND SINGLE-GRIPPER ROBOTIC CELLS

Var.	No.	n_A	n_B	P_A / P_B	RL (Dual)	RL (Single)	Gap (%)
10%	19	100	100	[65,75],[70,80],[80,90]/ [75,85],[65,75],[60,70]	9445.2	11557.8	22.4
	20	100	100	[80,90],[70,80],[65,75]/ [75,85],[65,75],[60,70]	9520.8	11409.6	19.8
	21	100	100	[80,90],[70,80],[65,75]/ [60,70],[65,75],[75,85]	9630.7	11182.7	16.1
20%	22	100	100	[65,85],[70,90],[80,100]/ [75,95],[65,85],[60,80]	9947.5	12087.2	21.5
	23	100	100	[80,100],[70,90],[65,85]/ [75,95],[65,85],[60,80]	10245.1	11667.6	13.9
	24	100	100	[80,100],[70,90],[65,85]/ [60,80],[65,85],[75,95]	10205.8	11223.9	10.0

TABLE VIII
COMPARISON TO OPTIMAL SOLUTIONS

n_A/n_B	No.	Gap(RL)	Gap(SWAP)	Gap(FIFO)	Cplex Comp. Time (s)	Cplex Optimality Gap	LT(s)
3/3	1	2.08	3.37	9.16	10754.02	0.00	10.70
	2	1.30	1.98	6.61	31364.36	0.00	11.15
	3	0.20	9.28	6.40	28525.41	0.00	10.91
4/4	1	0.90	1.13	8.17	43202.91	33.68	20.67
	2	0.14	0.39	5.77	43202.99	37.82	21.13
	3	-2.75	5.07	3.94	43203.22	38.91	20.76

robotic cells is 17.3%. From the extensive experiments, we can conclude that the proposed RL algorithm performs well for dual-gripper robotic cells with processing time variations. It works significantly better than SWAP and FIFO, and the gap from LB is not large. It can also be applied to a dual-gripper scheduling problem with identical units and a single-gripper robotic cell.

4) Comparison to Optimal Solutions: It is not possible to obtain an optimal solution to our problem because we do not know the processing times of units until all of them are actually completed. Therefore, we relaxed our problem with the following assumptions and compared it to an optimal solution: 1) all parts have constant processing times in all machines and 2) the transportation time between M_i and M_j is constant as δ regardless of i and j. Therefore, it takes δ for the robot to move from the output device to the input device. Then we can modify and use the proposed mathematical programming model based on a Petri net in [3]. A more detailed explanation of the formulation can be found in [3]. We use Cplex 12.10.0 for obtaining a solution.

Table VIII shows the results with the instances where processing time ranges follow cases 1–3 in Table IV. In the table, Gap(RL) [or Gap(SWAP), Gap(FIFO)] is computed by (((makespan from RL (or SWAP, FIFO)—makespan from Cplex) $\times$ 100)/makespan from Cplex). We generated ten instances for each case and presented the average gap. The sixth and seventh columns show the computation time of Cplex and its optimality gap when terminated before it reaches an optimal solution. Since it takes a long time even with 3 units of parts A and B, which is 23547.9 s on average, we set the 12-h limit for four units of parts A and B.

We can observe that the average gap between the proposed RL and Cplex with three units of parts A and B is 1.19%, and it is -0.57% with four units of parts A and B. SWAP and FIFO have a larger gap. The learning time of the proposed RL is only around 10 s with three units of parts A and B whereas it is approximately 6.5 h (=23547.9 s) with Cplex. We note that it takes a long time to obtain an optimal solution of dual-armed cluster tools because the robot has many possible actions at each state (two job types, different machines), and there can be many deadlock situations.

5) Discussion of Possible Extension to a Cyclic Case: We think that it is possible to extend the proposed RL method to a cyclic case, but it is not straightforward. We first need to assume an initial state of a robotic cell where some machines have a unit. Then the terminal state should be equal to the initial state after producing a certain number of units (for example one unit of part A and one unit of part B). The objective is to minimize a cycle time. The proposed Petri net needs to be modified to reflect a cycle schedule. The proposed state can be used, but s^d needs to be modified because the robotic cell has an infinite number of units in a cyclic schedule. The action set can be used without modification, and the reward might need to be adjusted so that the cycle time can be minimized. When ten units of parts A and B are produced in our problem, each transition from t15 to t30 needs to fire ten times in a schedule. However, if only one unit of parts A and B are produced in a cyclic schedule, the number of transition firing is reduced significantly. Therefore, the learning time of the RL can be reduced as well.

VI. CONCLUSION

We addressed a scheduling problem of dual-gripper robotic cells with two part types when machines have variable processing times to minimize the makespan. We proposed a learning-based method, RL, for the first time for our problem. We first modeled the problem with a Petri net and defined the states, actions, and rewards by using scheduling properties of flow

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

shops and features from the Petri net. Then, the proposed RL was compared with FIFO and the swap sequence, which are widely used for scheduling robotic cells in practice. We demonstrated that RL performs better than FIFO or the swap sequence and the gap from LB is not large.

This study can be extended to other variants of robotic cell scheduling problems, such as production lines with many different part types or with machines shared between two types. We can also consider parallel machines in each process step and multiple material handling robots. In addition, cluster tools or hoist systems can be good applications in the future. The ϵ-greedy policy or greedy policy in RL can be improved further by considering scheduling properties. The initialization of Q values also must be addressed.

APPENDIX

A. Proof of Theorem 1

Proof: The minimum completion time for each machine to finish n_A or n_B units should be analyzed. We first focused on all the processes on M_1. The earliest possible start time of the first process on M_1 is $\tau_u + \delta + \tau_l$ because the robot must unload a unit of part A from the input device (τ_u), move it (δ), and load it into M_1 (τ_l). After the process takes a time of p_{11}, M_1 can start the second process after $\tau_u + \theta + \tau_l$ time units, which is required for the robot to unload the first unit from M_1 to a gripper, switch the position of the two grippers, and load the second unit from another gripper into M_1. Hence, it takes $\tau_u + \delta + \tau_l + (\tau_u + \theta + \tau_l)(n_A - 1) + \sum_{j=1}^{n_A} p_{1j}$ time units until the processing of the n_Ath unit on M_1 from the beginning. Then, the last unit, the n_Ath unit, is moved and processed on the other machines from M_2 to M_{m_A} and transported to the output device, which takes $\sum_{k=2}^{m_A} p_{kn_A} + m_A(\tau_u + \delta + \tau_l) + m_B\delta$ time units. Hence, the minimum required time of M_1, $\mathrm{LB}^A_{M_1}$, is $(m_A+1)(\tau_u+\tau_l)+(m_A+m_B+1)\delta+(\tau_u+\theta+\tau_l)(n_A-1)+\sum_{j=1}^{n_A} p_{1j} + \sum_{k=2}^{m_A} p_{kn_A}$. For the first process on M_2, the robot is required to unload a unit, transport it, load it into M_1, unload the completed unit from M_1, transport it, and load it into M_2, which takes a duration of $\tau_u + \delta + \tau_l + p_{11} + \tau_u + \delta + \tau_l$. Then, M_2 processes n_A units with $(n_A - 1)$ times of unloading, switching, and loading tasks on M_2, and then the last unit, the n_Ath unit, is processed on the remaining machines and moved to the output device. Hence, it requires a duration of $(m_A + 1)(\tau_u + \tau_l) + (m_A + m_B + 1)\delta + (\tau_u + \theta + \tau_l)(n_A - 1) + p_{11} + \sum_{j=1}^{n_A} p_{2j} + \sum_{k=3}^{m_A} p_{kn_A}$ for M_2 to complete the processing of all n_A units. $\mathrm{LB}^A_{M_i}$, for $3 \le i \le m_A$, and $\mathrm{LB}^B_{M_i}$, for $m_A + 1 \le i \le m_A + m_B$, can be obtained similarly.

LB_R can be computed by analyzing the minimum robot tasks, i.e., unloading, loading, transporting, and switching, required to process $n_A + n_B$ units on the robotic cell. For each unit of part A in the input device, $m_A + 1$ unloading and loading tasks are conducted by the robot, i.e., $(\tau_u + \tau_l)(m_A + 1)$ time units are necessary to process it on m_A machines and to load it into the output device. Similarly, a unit of part B should be unloaded from the input device and loaded into the output device after being processed on m_B machines, which takes $(\tau_u + \tau_l)(m_B + 1)$ time units. Even though the robot can hold two parts simultaneously, the unloading and loading tasks by the robot can be performed only one at a time. Hence, it takes at least $(m_A + 1)(\tau_u + \tau_l)n_A + (m_B + 1)(\tau_u + \tau_l)n_B$ to process $n_A + n_B$ units, excluding the transportation tasks. We now consider a robot moving task by assuming that the robot has one gripper. All units of parts A and B should be transported from the input device to the output device, which takes a duration of $(m_A + m_B + 1)\delta(n_A + n_B)$. In addition, the robot should return to the input device after loading a completed unit into the output device to unload the remaining units, which are as many as $n_A + n_B - 1$. Hence, the total transportation time of the robot is $(m_A + m_B + 1)\delta(n_A + n_B) + (m_A + m_B + 1)\delta(n_A + n_B - 1)$; further, this is divided by two as the robot can move two units simultaneously. ■

B. Parameter Setting

We compared the six reward functions, r_t, r_t^1, r_t^2, r_t^3, r_t^4, and r_t^5, with the six cases listed in Table IX. We used the Petri net model illustrated in Fig. 2. The parameters, α, γ, and ϵ, were set to 0.1, 0.95, and 0.3, respectively, and E and L as $(n_A + n_B) \times 1500$ and 10, respectively, were used. The value of ϵ, which is calculated by $(\epsilon/\sqrt{e})$, decreases as the number of episodes, e, increases. After learning the Q values with each reward function, the average makespan of 100 instances, which were newly generated for each case, were obtained with Q^*, as listed in Table IX. In the table, P_A and P_B indicate the processing time ranges, $[p_i^{\min}, p_i^{\max}]$, on machine i for parts A and B, respectively, and the last column shows the average gap of the makespan from LB while using each of the six reward functions, which is computed by (((makespan from RL - LB) $\times$ 100)/LB). The six cases with both n_A and n_B of 10 were generated by considering the bottleneck location, processing time difference between two-part types, and processing time scales. All the robot tasks, i.e., τ_l, τ_u, δ, and θ, were assumed to require two time units.

In Table IX, we can observe that the gap from LB is large when the processing times are relatively small because multiple machines often compete for the robot to unload a unit; hence, the idle or waiting times on the machines occur frequently, which are not included while computing LB. Hence, the gap from LB is large and a robot task sequence significantly affects makespan in this situation. In addition, when the processing time difference between two-part types is large, the average gap is large, as can be observed in cases 3 and 4 in Table IX. Gap(L), the average gap from LB with each reward function, in the last row of Table IX, shows that r_t, r_t^3, and r_t^4 work well when compared to others. We can also observe that the average gaps with r_t^1 and r_t^5 are relatively large, which implies that using the idle or waiting times of one fixed bottleneck machine or the last machines as rewards do not provide a good robot task sequence. It can be noted that the average gap of cases 5 and 6 with r_t is approximately 1% to 3%, which is significantly small while considering the fact that the gap from LB and not the optimal makespan was used.

We now present a comparison of the three reward functions, r_t, r_t^3, and r_t^4, which provided a good performance, by changing α, γ, and ϵ. The average gap from LB with the minimum makespan of the six cases in Table IX among

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

TABLE IX
PRELIMINARY EXPERIMENTS

No.	n_A	n_B	P_A / P_B	r_t	r_t^1	r_t^2	r_t^3	r_t^4	r_t^5	Avg. Gap
1	10	10	[40, 50], [60, 70], [80, 90]/ [70, 80], [60, 70], [50, 60]	1118.8	1470.3	1175.8	1083.1	1100.8	1182.2	13.2
2	10	10	[80, 90], [60, 70], [40, 50]/ [70, 80], [60, 70], [50, 60]	1110.1	1176.9	1135.3	1137.8	1104.9	1456.5	13.7
3	10	10	[40, 50], [60, 70], [80, 90]/ [35, 40], [30, 35], [25, 30]	1128.9	1198.7	1228.2	1121.7	1074.7	1187.4	10.8
4	10	10	[80, 90], [60, 70], [40, 50]/ [35, 40], [30, 35], [25, 30]	1078.1	1114.6	1117.6	1116.0	1177.2	1195.9	8.5
5	10	10	[80, 100], [120, 140], [160, 180]/ [80, 100], [100, 120], [120, 140]	2095.5	2040.6	2064.4	2093.7	2053.7	2124.2	3.7
6	10	10	[160, 180], [120, 140], [80, 100]/ [80, 100], [100, 120], [120, 140]	2018.5	2076.5	2169.1	2047.9	2261.9	2150.0	5.8
Gap(L) (%)				5.0	13.4	9.5	5.7	7.1	15.1	9.3

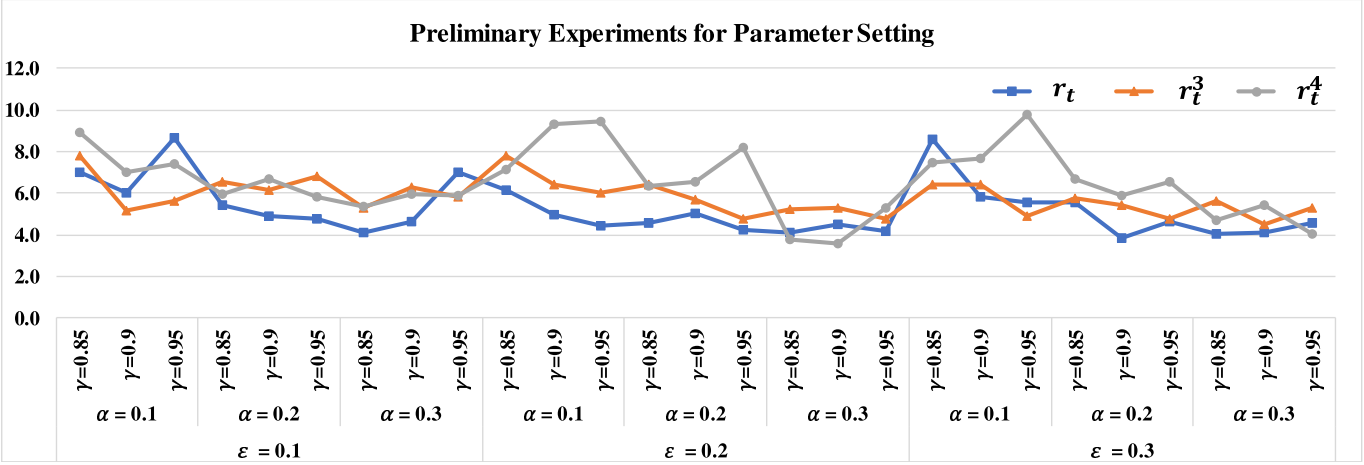


Fig. 6. Parameter setting.

$(n_A + n_B) \times 1500$ episodes having the same set of processing times is illustrated in Fig. 6. α changes from 0.1 to 0.3, γ ranges from 0.85 to 0.95, and ϵ is set to 0.1, 0.2, and 0.3 [42], [43], [45]. We can observe that the average gap with r_t is smaller than that with r_t^3 or r_t^4, and the minimum gap is obtained with α, γ, and ϵ of 0.3, 0.9, and 0.3, respectively. E is set to $(n_A + n_B) \times 500$ as the makespan reduction after $(n_A + n_B) \times 500$ episodes in an iteration was not observed, and L was determined as 10.

REFERENCES

[1] M. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah, "Sequencing and scheduling in robotic cells: Recent developments," *J. Scheduling*, vol. 8, no. 5, pp. 387–426, Oct. 2005.

[2] C. Jung, H.-J. Kim, and T.-E. Lee, "A branch and bound algorithm for cyclic scheduling of timed Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 1, pp. 309–323, Jan. 2015.

[3] C. Jung and T.-E. Lee, "An efficient mixed integer programming model based on timed Petri nets for diverse complex cluster tool scheduling problems," *IEEE Trans. Semicond. Manuf.*, vol. 25, no. 2, pp. 186–199, May 2012.

[4] A. Che, J. Feng, H. Chen, and C. Chu, "Robust optimization for the cyclic hoist scheduling problem," *Eur. J. Oper. Res.*, vol. 240, no. 3, pp. 627–636, Feb. 2015.

[5] C. Sriskandarajah and B. Shetty, "A review of recent theoretical development in scheduling dual-gripper robotic cells," *Int. J. Prod. Res.*, vol. 56, nos. 1–2, pp. 817–847, Jan. 2018.

[6] H.-J. Kim and J.-H. Lee, "Cyclic robot scheduling for 3D printer-based flexible assembly systems," *Ann. Oper. Res.*, to be published, doi: 10.1007/s10479-018-3098-2.

[7] H.-J. Kim, J.-H. Lee, C. Jung, and T.-E. Lee, "Scheduling cluster tools with ready time constraints for consecutive small lots," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 1, pp. 145–159, Jan. 2013.

[8] A. E. Amraoui and M. Elhafsi, "An efficient new heuristic for the hoist scheduling problem," *Comput. Oper. Res.*, vol. 67, pp. 184–192, Mar. 2016.

[9] C. Sriskandarajah, I. Drobouchevitch, S. P. Sethi, and R. Chandrasekaran, "Scheduling multiple parts in a robotic cell served by a dual-gripper robot," *Oper. Res.*, vol. 52, no. 1, pp. 65–82, Jan. 2004.

[10] M. Dawande, M. Pinedo, and C. Sriskandarajah, "Multiple part-type production in robotic cells: Equivalence of two real-world models," *Manuf. Service Oper. Manage.*, vol. 11, no. 2, pp. 210–228, Apr. 2009.

[11] J.-H. Lee, H.-J. Kim, and T.-E. Lee, "Scheduling cluster tools for concurrent processing of two wafer types with PM sharing," *Int. J. Prod. Res.*, vol. 51, no. 12, pp. 3671–3687, 2015.

[12] C. Pan, M. Zhou, Y. Qiao, and N. Wu, "Scheduling cluster tools in semiconductor manufacturing: Recent advances and challenges," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 586–601, Apr. 2018.

[13] Y.-J. Joo, "Operational optimization of an automated electrical die sorting line with single-wafer transfer," M.S. thesis, Korea Adv. Inst. Sci. Technol., Daejeon, South Korea, 2009.

[14] I. Ribas, R. Leisten, and J. M. Framiñan, "Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective," *Comput. Oper. Res.*, vol. 37, no. 8, pp. 1439–1454, Aug. 2010.

[15] S. P. Sethi, J. B. Sidney, and C. Sriskandarajah, "Scheduling in dual gripper robotic cells for productivity gains," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 324–341, Jun. 2001.

[16] H. N. Geismar and M. Pinedo, "Robotic cells with stochastic processing times," *IIE Trans.*, vol. 42, no. 12, pp. 897–914, Sep. 2010.

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.

[17] N. Wu and M. Zhou, "Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 1, pp. 203–209, Jan. 2012.
[18] H.-J. Kim, J.-H. Lee, and T.-E. Lee, "Schedulability analysis for non-cyclic operation of time-constrained cluster tools with time variation," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 3, pp. 1409–1414, Jul. 2016.
[19] Y. Qiao, N. Wu, and M. Zhou, "Petri net modeling and wafer sojourn time analysis of single-arm cluster tools with residency time constraints and activity time variation," *IEEE Trans. Semicond. Manuf.*, vol. 25, no. 3, pp. 432–446, Aug. 2012.
[20] Y. Qiao, N. Wu, F. Yang, M. Zhou, and Q. Zhu, "Wafer sojourn time fluctuation analysis of time-constrained dual-arm cluster tools with wafer revisiting and activity time variation," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 4, pp. 622–636, Apr. 2018.
[21] J.-H. Lee and J.-H. Kim, "Analysis of backward sequence for single-armed cluster tools with processing time variations," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 2167–2174, Oct. 2020.
[22] H. N. Geismar, M. Dawande, and C. Sriskandarajah, "Throughput optimization in constant travel-time dual gripper robotic cells with parallel machines," *Prod. Oper. Manage.*, vol. 15, no. 2, pp. 311–328, Jan. 2009.
[23] H. Gultekin, B. Coban, and V. E. Akhlaghi, "Cyclic scheduling of parts and robot moves in m-machine robotic cells," *Comput. Oper. Res.*, vol. 90, pp. 161–172, Feb. 2018.
[24] M. Dawande, H. N. Geismar, M. Pinedo, and C. Sriskandarajah, "Throughput optimization in dual-gripper interval robotic cells," *IIE Trans.*, vol. 42, no. 1, pp. 1–15, Oct. 2009.
[25] P. Yan, C. Chu, N. Yang, and A. Che, "A branch and bound algorithm for optimal cyclic scheduling in a robotic cell with processing time windows," *Int. J. Prod. Res.*, vol. 48, no. 21, pp. 6461–6480, Nov. 2010.
[26] S. Yildiz, M. S. Akturk, and O. E. Karasan, "Bicriteria robotic cell scheduling with controllable processing times," *Int. J. Prod. Res.*, vol. 49, no. 2, pp. 569–583, Jan. 2011.
[27] P. Yan, A. Che, N. Yang, and C. Chu, "A tabu search algorithm with solution space partition and repairing procedure for cyclic robotic cell scheduling problem," *Int. J. Prod. Res.*, vol. 50, no. 22, pp. 6403–6418, Nov. 2012.
[28] A. Che, V. Kats, and E. Levner, "An efficient bicriteria algorithm for stable robotic flow shop scheduling," *Eur. J. Oper. Res.*, vol. 260, no. 3, pp. 964–971, Aug. 2017.
[29] H. N. Geismar, M. W. Dawande, and S. P. Sethi, "Dominance of cyclic solutions and challenges in the scheduling of robotic cells," *SIAM Rev.*, vol. 47, no. 4, pp. 709–721, Jan. 2005.
[30] J. Hurink and S. Knust, "List scheduling in a parallel machine environment with precedence constraints and setup times," *Oper. Res. Lett.*, vol. 29, no. 5, pp. 231–239, Dec. 2001.
[31] J. Carlier, M. Haouari, M. Kharbeche, and A. Moukrim, "An optimization-based heuristic for the robotic cell problem," *Eur. J. Oper. Res.*, vol. 202, no. 3, pp. 636–645, May 2010.
[32] Y. Crama, V. Kats, J. van de Klundert, and E. Levner, "Cyclic scheduling in robotic flowshops," *Ann. Oper. Res.*, vol. 96, no. 1, pp. 97–124, Nov. 2000.
[33] E. Levner, V. Kats, D. A. López de Pablo, and T. C. E. Cheng, "Complexity of cyclic scheduling problems: A state-of-the-art survey," *Comput. Ind. Eng.*, vol. 59, no. 2, pp. 352–361, Sep. 2010.
[34] N. Qi Wu and M. Zhou, "Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 2, pp. 446–454, Apr. 2012.
[35] D.-K. Kim, H.-J. Kim, and T.-E. Lee, "Optimal scheduling for sequentially connected cluster tools with dual-armed robots and a single input and output module," *Int. J. Prod. Res.*, vol. 55, no. 11, pp. 3092–3109, Jun. 2017.
[36] H.-J. Kim, J.-H. Lee, S. Baik, and T.-E. Lee, "Scheduling in-line multiple cluster tools," *IEEE Trans. Semicond. Manuf.*, vol. 28, no. 2, pp. 171–179, May 2015.
[37] H.-J. Kim, J.-H. Lee, and T.-E. Lee, "Noncyclic scheduling of cluster tools with a branch and bound algorithm," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 690–700, Apr. 2015.
[38] J.-H. Lee and H.-J. Kim, "Completion time analysis of wafer lots in single-armed cluster tools with parallel processing modules," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 4, pp. 1622–1633, Oct. 2017.
[39] H.-J. Kim and J.-H. Lee, "Closed-form expressions on lot completion time for dual-armed cluster tools with parallel processing modules," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 898–907, Apr. 2019.
[40] T.-E. Lee, "A review of scheduling theory and methods for semiconductor manufacturing cluster tools," in *Proc. Winter Simulation Conf.*, Austin, TX, USA, Dec. 2008, pp. 2127–2135.
[41] A. Che, P. Yan, N. Yang, and C. Chu, "Optimal cyclic scheduling of a hoist and multi-type parts with fixed processing times," *Int. J. Prod. Res.*, vol. 48, no. 5, pp. 1225–1243, Mar. 2010.
[42] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Eng. Appl. Artif. Intell.*, vol. 18, no. 1, pp. 73–82, Feb. 2005.
[43] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robot. Auto. Syst.*, vol. 33, nos. 2–3, pp. 167–178, 2000.
[44] C. Chen, B. Xia, B.-H. Zhou, and L. Xi, "A reinforcement learning based approach for a multiple-load carrier scheduling problem," *J. Intell. Manuf.*, vol. 26, no. 6, pp. 1233–1245, Dec. 2015.
[45] K. Arviv, H. Stern, and Y. Edan, "Collaborative reinforcement learning for a two-robot job transfer flow-shop scheduling problem," *Int. J. Prod. Res.*, vol. 54, no. 4, pp. 1196–1209, Feb. 2016.
[46] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[47] N. Wu, F. Chu, C. Chu, and M. Zhou, "Petri net modeling and cycle-time analysis of dual-arm cluster tools with wafer revisiting," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 1, pp. 196–207, Jan. 2013.
[48] Y. Qiao, N. Wu, and M. Zhou, "A Petri net-based novel scheduling approach and its cycle time analysis for dual-arm cluster tools with wafer revisiting," *IEEE Trans. Semicond. Manuf.*, vol. 26, no. 1, pp. 100–110, Feb. 2013.
[49] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. London, U.K.: MIT Press, 2018.
[50] M. L. Pinedo, *Scheduling Theory, Algorithms, and Systems*. New York, NY, USA: Springer, 2012.
[51] N. Geismar, M. Dawande, and C. Sriskandarajah, "Productivity improvement from using machine buffers in dual-gripper cluster tools," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 1, pp. 29–41, Jan. 2011.
[52] T.-K. Kim, C. Jung, and T.-E. Lee, "Scheduling start-up and close-down periods of dual-armed cluster tools with wafer delay regulation," *Int. J. Prod. Res.*, vol. 50, no. 10, pp. 2785–2795, May 2013.
[53] M. Foumani and K. Jenab, "Cycle time analysis in reentrant robotic cells with swap ability," *Int. J. Prod. Res.*, vol. 50, no. 22, pp. 6372–6387, Nov. 2012.
[54] E. Gundogdu and H. Gultekin, "Scheduling in two-machine robotic cells with a self-buffered robot," *IIE Trans.*, vol. 48, no. 2, pp. 170–191, Feb. 2016.

Hyun-Jung Kim received the Ph.D. degree in industrial and systems engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 2013.

She was a Post-Doctoral Researcher with the Department of Industrial Engineering and Operations Research, University of California at Berkeley, Berkeley, CA, USA, and an Assistant Professor with the Department of Systems Management Engineering, Sungkyunkwan University, Republic of Korea. She is currently an Assistant Professor with the Department of Industrial and Systems Engineering, KAIST. Her research interests include modeling, scheduling, and control of production systems.

Jun-Ho Lee received the B.S. and Ph.D. degrees from the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejon, Republic of Korea, in 2008 and 2013, respectively.

He was a Senior Researcher with Systems Engineering Team, Samsung Electronics, Republic of Korea, from 2013 to 2016, and a Post-Doctoral Researcher with the Department of Industrial and Systems Engineering, University of Wisconsin, Madison, WI, USA, from 2016 to 2017. He is currently an Assistant Professor with the School of Business, Chungnam National University, Republic of Korea. His research interests are in modeling, analysis, scheduling, and control of production and service systems.

Authorized licensed use limited to: Guangdong Univ of Tech. Downloaded on April 22,2022 at 12:25:21 UTC from IEEE Xplore. Restrictions apply.