A REINFORCEMENT LEARNING APPROACH TO SCHEDULING DUAL-ARMED CLUSTER TOOLS WITH TIME VARIATIONS

Ji-Eun Roh[(a)], Tae-Eog Lee[(b)]

[(a),(b)]Department of Industrial and Systems Engineering,
Korea Advanced Institute of Science and Technology (KAIST), 305-701 Daejeon,
Republic of Korea

[(a)]rje0531@kaist.ac.kr, [(b)]telee@kaist.ac.kr

ABSTRACT
The cluster tool, which consists of multiple process chambers are widely used in the semiconductor industry. As the process of wafers becomes more sophisticated, the operation of cluster tools is also being improved. To effectively operate cluster tools, several rule-based schedules, such as the swap sequence have been developed. However, scheduling in time variance environment is not fully considered. In this paper, we propose a cluster tool modeling method, which can handle time variance in dual-armed cluster tool. Then, we present a reinforcement learning process based on the proposed cluster tool model to find new operational schedules in specific configurations. To measure the performance of the newly obtained schedule, makespan is compared under the new policy and the swap policy. The makespan reduced compared to the conventional swap policy, which implies that the reinforcement learning well learned the operation schedule in the time variance environment.

Keywords: cluster tool, Markov decision process, scheduling, reinforcement learning

1. INTRODUCTION

Along with the innovative development in the semiconductor manufacturing industry, quality issues in the wafer manufacturing process have been discussed. The technologies of each process have rapidly developed and improved the production quality of the wafers. To avoid quality issues due to batch production, the cluster tool that process one wafer at a time are now widely used in the semiconductor industry (Lee 2008). Chambers in cluster tools do not process wafers in units of batches but process them individually, so they can meet the quality standard. The cluster tool consists of usually four to six processing chambers and one transport robot. In each chamber, only one wafer is processed at a time, and one of the process steps specified before the start of the process is processed. In addition, the transport robot moves the wafers inside the cluster tool. To start the processes, a wafer enters the chamber that is responsible for the first process step. After the process is completed in the chamber, the wafer is transported to the next chamber by a transport robot. The transport robot repeats the process of unloading the processed wafer from the chamber and loading the wafer in the next appropriate chamber which is in charge of the next process step. Once all the required processes are completed with the proper chamber sequence, the wafer process in the cluster tool finally is completed.

Such a configuration leads cluster tools to have several issues in operation. Since cluster tools consist of only processing chambers and a single transport robot, only one robot operation is possible at a time. Hence, the wafers cannot be moved at any time, and can only be moved when the transport robot moves them. Even if the multiple chambers are ready for the process, multiple wafers cannot be delivered at the same time. In addition, the cluster tools do not contain any buffer space in their interior space, so the only way to store the wafers outside the chamber within the cluster tool is to hold it with the transport robot. When the robot loads a wafer into the chamber, the chamber starts the process, and the process ends naturally after the process time. The chamber processes for a period of time without any decision. Therefore, we can say that the overall cluster tool schedule depends on the order decisions of the transport robot. As the transport robot unloads or loads the chambers, the status of the cluster tool system has changed. This can be seen as a discrete event system because the state of the system changes as the robot takes action. Petri nets, finite state machines (FSMs), Markov decision processes (MDPs), etc. are used to model discrete event systems (Murata 1989; Choi and Kang 2013; Puterman 2014).

Several studies about cluster tools scheduling have used a timed Petri net (TPN) model, in which the places or transitions have a time delay. A TPN is classified into deterministic and stochastic TPN whether the time delay is deterministic or stochastic (Murata 1989). Each of them is a modeling technique that deals with different kinds of time delays. Many studies have been conducted on finding policies for the deterministic environment. For example, (Lee, Lee and Shin 2004, Zuberek 2004) have modeled analyzed cluster tools by the deterministic TPNs, and Jung and Lee (2012) proposed a mixed integer programming (MIP) model to find the optimal policies in deterministic TPNs. In stochastic environment, there also has been several studies. To deal with time variation in cluster tool, (Kim and Lee 2008) suggested an extended Petri net and developed a graph-based procedure to verify the schedulability condition. Qiao, Wu, and Zhou (2012) have introduced two-level architecture to deal with time variation, and proposed some heuristic algorithms to find a schedule for one of the architectures. Molly (1982) suggested that the stochastic TPNs are isomorphic to finite Markov processes under the certain conditions.

However, new methods for finding good policies in specific systems are studying nowadays. Reinforcement learning (RL) is widely used in finding policies in many fields such as manufacturing systems, autonomous vehicle control, finance, and games. This reinforcement learning is one of the solutions to find optimal policies in MDP models (Sutton and Barto 1998). Hence, modeling the system behavior by MDPs, then find the policies for the system by RL is widely used (Moody et al. 1998; Mnih et al. 2015).

In this paper, we propose a model using MDPs, which can handle time variations in cluster tools. Then, we report how we learn the new robot sequence using the MDP model to minimize the average makespan in time variation environment, and measure the performance of the obtained sequence. This study is an attempt to schedule the cluster tool using reinforcement learning. Since the model is designed to represent the stochastic process time of the cluster tool, the schedules obtained from reinforcement learning can be expected to be well applied in a stochastic environment.

2. MODELING DUAL-ARMED CLUSTER TOOLS WITH TIME VARIATIONS

To model dual-armed cluster tools with time variations, we use a MDP model. After introducing the problem of configuring the MDP model, the model we proposed is reported.

2.1. Markov Decision Process

A Markov decision process (MDP) is a tuple $< S, A, P, r >$, where S denotes a set of states, A a set of actions, P a state transition probability distribution, and r a reward function, respectively. The process follows the Markov property, which means the transition probability and reward functions depend only on the current state and the action, not the past history. In this paper, we address only the stationary environments, which means the system properties does not change as time goes by. According to Puterman (2014), at every decision epoch, a state $s \in S$, which is a representation of a system, is observed and an action $a \in A_s$ has to be chosen by a decision maker from the set of allowable actions in the state s. As a result of the action, the system state at the next decision epoch changes by some transition probability $P : S \times A \times S \to [0,1]$. $P^{a}_{s,s'}$ is determined by the current state s and the chosen action a. The decision maker gets a reward $r : S \times A \times S \to \mathbb{R}$, and $r^{a}_{s,s'}$ is determined by the current state s, action a, and the next state s'. Here, we have a concept of a decision rule, which describes a procedure for action selection in each state. This rule is called a stationary policy, $d : S \to A_s$. The decision maker passes through a sequence of states s_t , which are determined by transition probabilities $P^{a}_{s,s'}$ and the actions $a_t = d(s_t)$ the decision maker chooses, then the sequence of reward $r^{a}_{s,s'}$ is obtained. The popular performance metric is discounted reward, which is the sum of the discounted reward gained over the entire time horizon when we allow the specific policy. The discounted reward of a policy d starting at state i is defined as $J_d(i) \equiv \lim_{k\to\infty} \mathrm{E}\left[\sum_{j=1}^{k} \gamma^{(j-1)} r^{d(s_j)}_{s_j,s_{j+1}} \middle| s_1 = i\right]$, where $\gamma \in [0,1]$ is a discount factor. By Bertsekas (1995), it is proved that $J_d(i) = \left[\sum_{s'\in S} P^{d(s)}_{s,s'} \left(r^{d(s_j)}_{s_j,s_{j+1}} + \gamma J^d(s')\right)\right]$. The function J_d is called the value function for policy d. There are many variants of MDP model, and one of the them is a semi-Markov decision process (semi-MDP). In semi-MDP, temporal factors are included in the modeling. The original MDP assumes that each transition takes the same time through all the states; however, semi-MDP considers the transition time to follow arbitrary probability distributions.

2.2. Cluster tool modeling with MDP

If we simply insert the state of the chambers and actions the robot takes into MDPs as previous studies have done with deterministic TPN models, the MDPs does not properly represent the behavior of the cluster tools. This is due to characteristics of cluster tools. The cluster tools are not simply changing the systems by a discrete event; cluster tools are changing the systems by the time element. The general MDP assumes that every transition takes the same time, however, the transitions in cluster tools cannot be assumed to be the same. Considering the different transition times between states, we may think of simple ways to model the cluster tool system.

1. Model with MDPs which have a constant transition time, one second. Adjust transition probabilities to represent the time element.
2. Model with semi-MDPs so we can insert time information to the transition distribution.

To verify above two methods, we first consider the example case as shown in Figure 1.

The example represents a status change in a single chamber, which has two states A and B, and two actions 1 and 2. Consider the case where the environment state changes from A to B after 50 seconds. By using the first modeling method listed above, we can set transition probability to 1/50; thus, the state changes from A to B occurs after 50 seconds in average. However, the average cannot express the actual individual transition time explicitly.

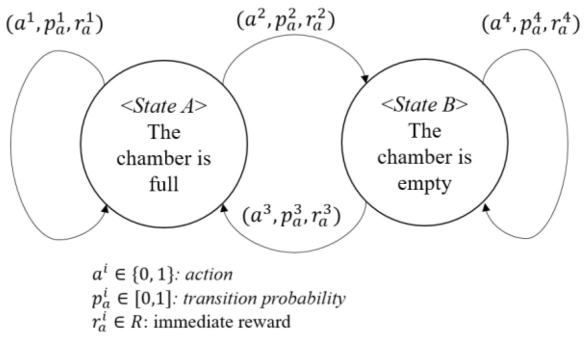
(a^1, p_a^1, r_a^1)
(a^2, p_a^2, r_a^2)
(a^4, p_a^4, r_a^4)
<State A> The chamber is full
<State B> The chamber is empty
(a^3, p_a^3, r_a^3)
$a^i \in \{0,1\}$: action
$p_a^i \in [0,1]$: transition probability
$r_a^i \in R$: immediate reward

Figure 1: Simple Example for Cluster Tool System

By using the second modeling method, we can set transition time distribution as 50 seconds. When using the second modeling method, we can express that the transition occurs after 50 seconds in Figure 1 if we set the transition time distribution to 50 seconds (as a constant). However, the cluster tool is a tool with multiple chambers, so the model should reflect changes in state in other chambers. If 20 seconds have elapsed while the status changes in another chamber, and so the chamber of the example now needs to change its state after 30 seconds, the transition time distribution should be changed to 30 seconds. Every time the transition occurs in the other chamber, we have to calculate and change the distribution; the environment of the system changes, which does not satisfy the markov property.
The two MDP modeling methods above are not appropriate for modeling the cluster tool operations in time variations, since they do not represent the transition time explicitly or do not satisfy the Markov property. To satisfy the Markov property, a state should contain sufficient information, so that the state does not require past history to obtain the next state or reward. Therefore, state is designed to contain information about chamber and robot usage, and the remaining process time.
In this study, We propose to represent the state as $(W, C_1, C_2, \dots, C_n, R, S_1, S_2, Z_1, Z_2, \dots, Z_n)$, where n is the number of processing chambers in cluster tool. W is the number of wafers remaining in the loadlock. C_i is the state of the chamber, which represents whether the ith chamber is empty ($C_i = 0$), full ($C_i = 1$), or completed processing ($C_i = 2$) for $i \in \{1, 2, \dots, n\}$. R is the number of wafers held by the transport robot. Since we are dealing with a dual-armed cluster tool, R can have a value of 0, 1, or 2. S_1 *and* S_2 are the next process steps of the wafers held by the robot. These represent the process steps that each wafer should visit in the next step. Z_i is the expected remaining process time of the ith chamber for $i \in \{1, 2, \dots, n\}$. The value is calculated by subtracting the elapsed time from the initially set process time, because it is not known what the actual process time will be in a time variation environment. In addition, define S to be the state space of the proposed state.
Hence, the proposed state structure contains the information about the chamber and the wafer inside it, and includes information on which chambers the wafers on the robot should be sent. Finally, it roughly contains information when the chambers are going to be finished. Then, the model satisfies the Markov property.
To better understand the structure of the states, we show state representation of Figure 2 by using the proposed states structure. Suppose that the black and white wafers indicate that they are in the first and second process steps, respectively, and the hatched wafer indicates that the process is in progress. According to the proposed structure, the state is $(W = 3, C_1 = 2, C_2 = 1, C_3 = 1, C_4 = 2,\ R = 1,\ S_1 = 2, S_2 = 0, Z_1 = 0,\ Z_2 = 4, Z_3 = 4, Z_4 = 0)$. This means that the three wafers are remained in the loadlock, the process is going on in the second and the third chambers, whereas the first and the forth chamber have completed its process. The robot holds a single wafer waiting to enter the second process step, and the remaining process times of the chambers are four for the second and the third chambers.

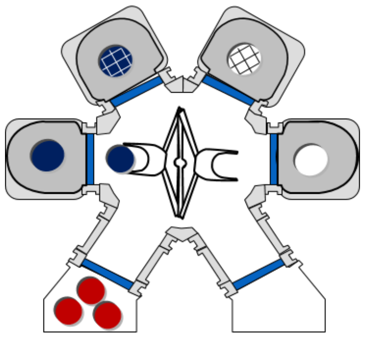

Figure 2. A Dual-armed Cluster Tool with Four Chambers

A is a set of actions that the transport robot can perform, which is $\{Wait, U_j\ , L_j\ , SW_i\}$, where $j \in \{0\ ,1, 2, \dots, n\}$ *and* $i \in \{1, 2, \dots, n\}$. They all represent the robot tasks: $U_j, L_j,$ and SW_i indicate unload, load, and swap operations on the jth or ith chamber. The unload, load, and swap operations are the common robot tasks in other modeling methods. However, in this study, $Wait$ action is added. $Wait$ represents the robot waiting until the state changes, such as finishing the process in a chamber. To make the model transition independent on remaining process time, we suggest $Wait$ to be an action of the robot. If without a $Wait$ action, the state of the cluster tool changes when the process completes, even if no action is selected. Furthermore, for all states $\{Wait, U_j\ , L_j\ , SW_i\}$ can always be selected as an action; therefore, $3n + 3$ actions are available to be chosen in all states.
Transition probability is denoted by $P_{s,s'}^{a}$, where s is the current state, a is the current action and s' is the next state. It stands for probability for transition to state s' when we choose action a in state s. In this study, we do not consider unexpected event occurrence such as robot failure, so the system always transposes exactly to the specified state. Hence, when the action is one of $U_i, L_i,$ *and* SW_i, $P_{s,s'}^{a}$ has a value of 1 for the appropriate s'. Here, the appropriate state s' is when s'

is likely to be the next state in state s. Furthermore, when the s' is not likely to be the next state in s, $P^{a}_{s,s'} = 0$. However, when the action is $Wait$, transition probability $P^{a}_{s,s'}$ does not only have a value of 0 or 1. Assume that the action is $Wait$. If at least one chamber is 'full', which stands for $C_i = 1$, the chambers should be selected which change their state to 'completed processing'. In other words, it chooses which chamber the process ends with. Choosing the chamber with the shortest remaining process time is the simplest and most intuitive. However, since we are dealing with a time variation environment, we cannot say that the process is finished after Z_i, exactly the expected remaining process time. Therefore, the chamber selection should reflect the fact that unexpected chamber processes may end sooner. In addition to selecting the chamber, the transition time needed to be stochastically generated to reflect the time variation environment. In general, the process of the chamber is completed within a range of times, including the average time. Therefore, we choose to generate the transition time using a beta distribution that can generate the average value within the bounded range.

Reward function is set to get 1 if a wafer completes all processes, otherwise, the function is set to get -0.1 in all transitions except for the $Wait$ actions. When the $Wait$ action is selected, the reward is set to be $-0.1 \times (\text{transition time})$.

Detailed transition rules and reward definitions are introduced in Sections 3.2 and 3.3.

3. LEARNING POLICIES FOR ROBOT MOVES IN CLUSTER TOOLS BY REINFORCEMENT LEARNING

There are various ways to solve MDP. First, when we know the model, we can define a total return. Then by dynamic programming, we can find the optimal value function of Bellman optimality equation (Bellman 1954). However, the proposed model has a time element in the state, it is not possible to solve the MDP through dynamic programming due to the curse of dimensionality. Therefore, we applied reinforcement learning (RL) to find the policy for the proposed MDP model. RL requires an environment to interact with. So we built an interactive environment based on the proposed MDP model. We then set up tasks to perform the learning, and learned the robot policy using Q-learning. After learning, we performed another experiment to measure the performance of the policy.

3.1. Reinforcement learning

Reinforcement learning (RL) is useful when the classical dynamic programming is not enough to solve the MDP problems. Dynamic programming is not suitable for the problem, when the environment is difficult to build a perfect model due to its unknown transition probability (curse of modeling), or when the environment has too many states to solve (curse of dimensionality). RL adds the concepts of stochastic approximation, temporal differences, and function approximation to classical dynamic programming, so that it can solve the MDP problem even if the transition probability is not explicitly represented, and the state dimension is too large (Gosavi 2014).

3.2. The environment

Since we find the robot policy by RL, we need an environment that returns the next state and reward for action according to state for learning progress. The environment required for RL is either an environment consisting of real equipment or an environment built by computer simulation. The best way to interact with the environment is to build the real world environment, then conduct the experiments with it. However, using a real world cluster tool is nearly impossible due to its value. A single tool is so expensive and needs large area to install; therefore, even the company is not able to arrange enough cluster tools for their manufacturing facilities. To replace the real world environment, we built a virtual environment, which responds to some stimulation.

The simulation environment mimics the real world; however, not fully, only partially. To make the partially reflected simulation environment to involve the core elements we want, we needed to be careful making the simulation environment. We made some assumptions in building an environment. First, we do not consider machine failure. Process time can vary depending on a machine and time; however, we do not handle machine failure in this study. Second, the environment is stationary, which means system dynamics does not change as time goes by. No matter how much time has passed, they follow the same distribution. Third, the time dynamics of the systems are governed by the rules we set; hence, it may not fully reflect the real world cluster tool behaviors.

The simulation environment shows the change of the environment depending on the agent's action. It shows the appropriate reward and the next state according to the selected action. Some actions are valid only in particular states. For example, robots can take a *Wait* action only when more than one chamber are filled with the wafer, and are not completed processing, which means that chamber i with a C_i value of 1 exists. Detailed rules for robot actions are specified in Table 1.

Table 1: Possible Situations for the Robot Actions

Robot Actions	Possible Situations
Wait	When at least one chamber is filled with a wafer has not been completed the process
Unload	When the chamber has a processed wafer, but still have the wafer inside in it
Load	When the robot holds some wafers, and one of the chambers corresponding to the steps that the wafers are going to enter is empty
Swap	When the chamber has a processed wafer, but still has the wafer inside in it. At the same time, the robot holds only one wafer to enter the chamber

Table 1 shows the possible situations for the action. The action is valid only when the right box condition is satisfied. If we select an action that is not in those possible situations, the state will not change. We set the environment to prevent state transitions when inappropriate actions are taken. We call an action taken in an impossible situation an $invalid\ action$, and an action taken in a possible situation a $valid\ action$. For example, if a chamber has processed wafers but the wafers are still in it and at the same time the robot has only one wafer in its arm, swap action on the chamber is valid action, while the rest actions are not.
All transitions are treated to be deterministic move except for $Wait$ action, as mentioned in Section 2.2. Other actions cause the environment to have the same response each time a state is entered. However, when a $Wait$ action is chosen, the state transition is performed stochastically and the transition probability is defined differently depending on the state. When only one chamber is in the 'full' state, the chamber is selected to complete the process. However, when more than one chambers are in 'full' state, changes in environment is determined by the algorithm below.

- With a probability 0.9, the chamber with the minimum remaining process time completes the process
- With a probability of 0.1, the selected chamber completes the process with a probability of inversely proportional to the remaining process time

Every time the $Wait$ action is chosen, the transition time is set to be stochastically selected along the $beta(50{,}50)$ distribution.

3.3. Task description

In general, a set of 25 wafers becomes a 'wafer cassette', which is the unit of production for cluster tools. Therefore, the goal is to quickly finish a single cassette process rather than a single wafer process. In fact, in real companies usually produces 100 to 200 wafers once, which means produces 4~8 wafer cassette without stopping the operation of the cluster tool. The problem of minimizing the total makespan can be thought of as a reduction in the total makespan itself and a reduction in the makespan of each subtask. When the whole makespan is continuously measured and returned at the end, the reward structure is heavily sparse to learn the policy. Therefore, in this study, we follow the method of minimizing the makespan of each subtask, which is completing a single wafer. In addition, we follow the structure of receiving a reward at the end of each wafer process. In order for all subtasks to ultimately reduce the total makespan, the total return is defined as the discounted reward. So learning is done in the direction of receiving the reward as soon as possible.

Furthermore, after 100 ~ 200 wafers are produced in real world, the equipment configuration may be changed. In the case of an episodic task that ends after the production of a certain number of wafers, the whole process can be divided into start-up, steady, and close-down cycles (Kim, Lee and Kim 2016). However, in this study, we examine whether there is a policy change when the environment changed from deterministic to stochastic in the steady cycle. Therefore, we consider the learning task as a continuous task to learn a steady cycle.

The most important goal of this problem is to learn to choose an action that maximizes the total return. But we should learn to prevent the system from selecting a behavior that makes it deadlock and learn to prevent invalid actions.

3.4. Learning procedure

The cluster tool behavior was learned by computer simulation-based reinforcement learning. Learning is achieved through the interaction of agents and environments. The agent repeats the process of selecting the action, receiving the corresponding response, and updating the Q value for the state and the action. The action selection and Q value update follow the traditional Q-learning method (Sutton and Barto 1998). The algorithm for Q-learning is illustrated in Figure 3. We used an ε-greedy policy to select an action from the current state, where the ε decreased over the time steps.

Even if using the same learning method, depending on how the environment is set, the selectable action set changes and the response to the action changes. In this study, we set the environment to have two chambers, and a wafer flow pattern (1,1) with all process times to be 4 seconds. Here, the wafer flow pattern refers to the number of parallel chambers in each process. The initial state is set to be full loaded, so all the chambers are filled with wafers that have not yet begun to process. Since we only deal with the steady state, we only used $(C_1,\ C_2,\ R, S_1,\ S_2, Z_1,\ Z_2)$ as the state. With this environment settings, the agent continued learning using Q-learning in the direction of maximizing the total return.

```
Initialize $Q(s,a), \forall s \in S, a \in A(s)$, arbitrarily, and $Q(terminal-state,\cdot) = 0$
Repeat (for each episode):
        Initialize $S$
        Repeat (for each step of episode):
                Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
                Take action $A$, observe $R, S'$
                $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
                $S \leftarrow S'$
        until $S$ is terminal
```

Figure 3: Q-Learning Algorithm

However, there are other issues that need to be taken care of by agents: deadlock actions and invalid actions.

We want to ensure that when an agent chooses an action in the current state, it does not choose an invalid action and a deadlock action that causes the next state to become deadlock. The agent goes through the process of defining possible action sets for the current state by a two-step look ahead method. To remove the invalid actions for the current state, the agent requests the environment for the results for all the actions, then saves only which actions are valid (one-step look ahead). For an action that is valid for current state, a second look ahead is needed to check if the deadlock occurs. It is time-consuming to always have a two-step look ahead in every learning step. To further speed up the procedures, we followed the deadlock prevention rules to the second look ahead procedure. As we apply the rules, deadlock action checking is needed only when all of the loading actions for the current state are invalid and the number of the holding wafer (R) is 1. In addition, only unloading actions among valid actions need to be checked. If any of the loading actions are valid, the agent does not need to do a deadlock check. A way to check if a given unloading action is a deadlock is to request a response for an appropriate loading action. If the unloaded wafer has a process step k to be visited, check that it is possible to load to the chambers that are in charge of the kth process step. If any of the loading actions are valid, the given unloading action is not a deadlock action, otherwise, the given unloading action is a deadlock action.

For example, consider the case illustrated in Figure 2, where chamber size (n) is 4, the current state is $(3,2,1,1,2,1,2,0,0,4,4,0)$, and the wafer flow pattern is (2,2). Since the value of n is 4, the cardinality of the available action sets for all states is 15, which is the value of $3n+3$. Among these 15 actions, the agent has to avoid invalid actions and deadlock actions. The agent first removes the invalid actions from the action set, and then removes the deadlock actions from the remaining action set. Through interaction with the environment, the agent gets the invalid actions and stores only the valid actions, which is $\{Wait, U_0, U_1, U_4, SW_4\}$. Since there is no loading action among the valid actions, actions $\{U_0, U_1, U_4\}$ become the candidates for the deadlock actions, by the deadlock checking rule that we mentioned above.

Therefore, the agent needs to check the validity of the appropriate loading actions for states (b), (c), and (d) in Figure 4. In state (b), unloaded wafer has to visit the 1st process step; hence, request responses for actions $\{L_0, L_1\}$, which are in charge of the 1st process step. In this way, responses for actions $\{L_2, L_3\}$ and $\{L_4\}$ are requested to check the deadness of the state (c) and (d), respectively. Then, the agent gets that any loading actions are not valid for state (b) and (c), and $\{L_4\}$ is valid for state (d). Therefore, the unloading actions corresponding to the state (b) and (c) are the deadlock actions, so the actions $\{U_0, U_1\}$ should be removed from the remaining valid action set, $\{Wait, U_0, U_1, U_4, SW_4\}$. Finally, actions $\{Wait, U_4,\ SW_4\}$ can be chosen from the current state S_0.

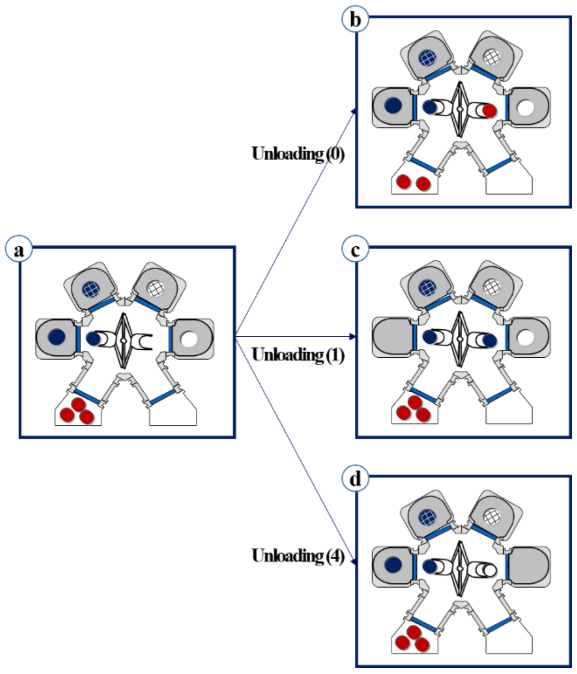

Figure 4: Checking Deadlock Action Candidates

We conducted Q-learning based on the above action selection rule. The agent always chooses a non-invalid and non-deadlock action, regardless of the exploration rate ε. The reason for constructing this additional action selection part before the action selection part in Q-learning is the learning time. In the cluster tool, there are too many invalid or deadlock actions in the whole action. If we do not forbid those inappropriate actions, a computation expense to explore for unnecessary action occurs and unnecessary learning time occurs. Therefore, by adding hand-crafted adjustments specific to this system, learning time has been shortened.

Table 2: Q-values for Some States

	State S $(C_1, C_2, R, S_1, S_2, Z_1, Z_2)$	**Q-value** $[Q(s,a_1), Q(s,a_1), \dots, Q(s,a_8)]$	**Action** $argmax_a Q(s,a)$
1	(0, 0, 0, 0, 0, 0, 0)	[-1., 4.43624915, -1., -1., -1., -1., -1, -1, -1]	1
2	(0, 0, 1, 0, 1, 0, 0)	[-1., 4.62024937, -1. , -1., 4.59535455, -1., -1., -1., -1.]	1
3	(0, 0, 1, 0, 2, 0, 0)	[-1., 5.17879609, -1., -1., -1., 5.16569827, -1., -1., -1.]	1
4	(0, 0, 1, 1, 0, 0, 0)	[-1., 4.62429555, -1., -1., 4.5976508, -1., -1., -1., -1.]	1
5	(1, 0, 2, 3, 2, 3, 0)	[6.16012302, -1., -1., -1., -1., 6.64482254, 6.63412537, -1., -1.]	5
6	(1, 0, 2, 3, 2, 4, 0)	[6.05787908, -1., -1., -1., -1., 6.42254697, 6.42956666, -1., -1.]	6
7	(1, 2, 1, 1, 0, 4, 0)	[5.84405062, -1., -1., 5.71535707, -1., -1., -1., -1., -1.]	0
8	(1, 2, 1, 2, 0, 0, 0)	[6.39532166, -1., -1., 6.0765975, -1., -1., -1., -1., 6.40542711]	8
9	(1, 2, 1, 2, 0, 1, 0)	[6.31353027, -1., -1., 6.31195477, -1., -1., -1., -1., 6.48067444]	8
10	(2, 1, 0, 0, 0, 0, 4)	[5.67589726, 5.68395603, 5.53326838, -1., -1., -1., -1., -1., -1.]	1
11	(2, 1, 1, 0, 1, 0, 1)	[6.15144211, -1., 6.06476367, -1., -1., -1., -1., 6.23425531, -1.]	7
12	(2, 2, 1, 2, 0, 0, 0)	[-1., -1., -1., 6.30849807, -1., -1., -1., -1., 6.46170915]	8
13	(2, 2, 1, 3, 0, 0, 0)	[-1., 7.02139286, 6.88018823, 6.88533456, -1., -1., 7.03105732, -1., -1.]	6

3.5. Performance measurement

After the learning is performed based on the proposed MDP model, $Q(s,a)$ values for every state $s \in S$ and action $a \in A$, are obtained. The policy we obtained is a greedy policy that selects the action with the highest Q-value in each state, i.e., $max_a Q(s,a)$. To report the policy, we have to list the selected actions for states as shown in Table 2. However, there are too many states to list all of them as a table, and it is difficult to intuitively understand what actions to take in some cases. Therefore, after we obtained the policy, we used the measures that represent the performance of the policies, then compared the performance of the acquired policy against the existing swap sequence.

Through learning, $Q(s,a)$ for $\forall s \in S, \forall a \in A$ is continuously updated; define $Q(s,a)$ for each thousand wafers produced as $q_0, q_1, \dots, q_t \in \mathbb{R}^{|S| \times |A|}$. However, we do not use all $q_0, q_1, \dots, q_t$ to get the policy; we only use the $Q(s,a)$ value after the convergence. To verify the convergence of a sequence $q_0, q_1, \dots, q_t$, $|q_k - q_{k-1}|$ for $k \in \{1, \dots, t\}$ was checked. If all elements in the matrix $|q_k - q_{k-1}|$ is relatively small, the q_k can be considered to be converged. Figure 5 shows the maximum element in matrix $|q_k - q_{k-1}|$ for $k \in \{1, \dots, 543\}$. The value converges to 0, which means the sequence $q_0, q_1, \dots, q_t$ converged. We used the converged $Q(s,a)$ value after the 543th iteration, i.e., used q_{543} to obtain the policy.

To indicate the performance of the obtained policy, the time it takes to process 50 wafers, makespan, is used. Makespan is measured in the same environment as the learning period. We used two processing chambers, wafer flow pattern (1,1), average process time of four, and the chamber process time is set to follow the $beta(50,50)$ distribution. Because the process time is randomly generated, makespan can be measured differently, even if the measurement is implemented under the same policy. Therefore, we measured makespan a thousand times under the designated policies and compared the averages.

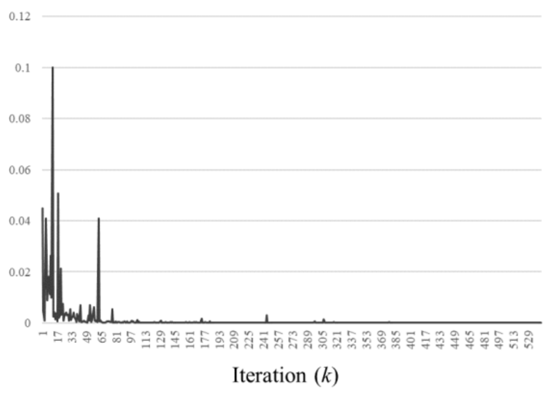

Figure 5: Q-value difference maximum value

4. ANALYSIS OF ROBOT POLICY OBTAINED BY LEARNING

Figure 6 represents the average q-values of the obtained policy and the swap policy. Figure 7 and Table 3 represent the makespan differences of the two policies as a histogram and a table, respectively. M_p and M_s represent makespan under the obtained policy and the swap policy.

We confirmed that the makespan of the policy obtained through Q-learning based on the proposed MDP model is on average shorter than the swap policy as shown in Figure 6.

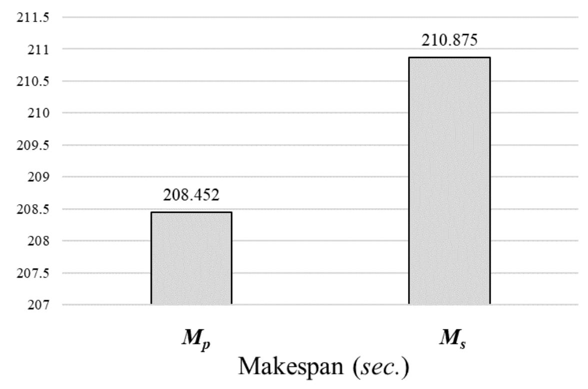

Figure 6: Average Q-values of Two Policies

Except for 2.3% cases, the others acquired the same or shorter makespan with the obtained policy. This means that the above reinforcement learning has found a good robot sequence and obtained a slightly better policy than the widely used swap policy. The average of makespan is not significantly different as illustrated in Figure 6. It seems that the small difference is due to the settings, the average process time and the number of the chambers in our learning environment. Both of the settings may have reduced the time variation effect to the environment. If we were to increase the average process time and number of chambers as much as the actual cluster tool, then the time variance would have increased, and the difference in makespan may have increased.
However, the fact that makespan is reduced in most cases means that as a policy run in a time variance environment, the policy obtained through learning in a time variance environment is better than the policy obtained in a deterministic environment.

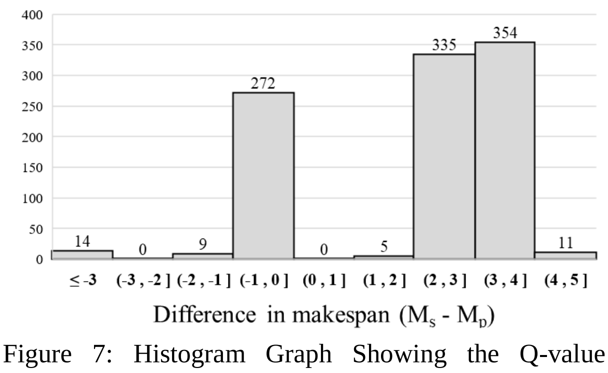

Figure 7: Histogram Graph Showing the Q-value Differences

Table 3: Frequency Distribution of the Q-value Difference

Frequency Distribution of $(M_s - M_p)$			
$M_s < M_p$ Case		$M_s \geq M_p$ Case	
Bin	Count	Bin	Count
[-5, -4)	1	[0, 1)	272
[-4, -3)	10	[1, 2)	0
[-3, -2)	3	[2, 3)	5
[-2, -1)	0	[3, 4)	335
[-1, 0)	9	[4, 5)	354
		[5, 6)	11
23 (2.3%)		977 (97.7%)	

5. CONCLUSIONS

Cluster tool scheduling has been usually studied under the deterministic environment. In this paper, we modeled dual-armed cluster tool behaviors as a Markov decision process in the time variance environment, then applied reinforcement learning to obtain robot policies. The newly obtained robot policies reduces the makespan of 50 wafer processes compare to the conventional swap sequence in most cases. From this study, it seems that the proposed MDP models can adequately express the cluster tool behaviors in time variance environment. This kind of view makes it possible to look at the cluster tool from a slightly different perspective; hence we can continue to try scheduling cluster tools through reinforcement learning.

However, since the above model includes the remaining time in the state in order to consider the time variance, the state space can be very large. Furthermore, the environment settings we used in learning couldn’t fully express the real world cluster tool behaviors. Therefore, the learning with more general environment settings is needed. To conduct such learning in further research, functional approximation seems to be applied due to the time elements in the states.

ACKNOWLEDGMENTS

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2015R1D1A1A01057131).

REFERENCES

Bellman, R., 1954. The theory of dynamic programming (No. RAND-P-550). RAND CORP SANTA MONICA CA.

Bertsekas, D. P., 1995. Dynamic programming and optimal control (Vol. 1, No. 2). Belmont, MA: Athena scientific.

Choi, B. K., & Kang, D., 2013. Modeling and simulation of discrete event systems. John Wiley & Sons.

Gosavi, A., 2003. Simulation-based optimization. parametric optimization techniques and reinforcement learning.

Jung, C., & Lee, T. E., 2012. An efficient mixed integer programming model based on timed Petri nets for diverse complex cluster tool scheduling problems. IEEE Transactions on Semiconductor Manufacturing, 25(2), 186-199.

Kim, D. K., Lee, T. E., & Kim, H. J., 2016. Optimal scheduling of transient cycles for single-armed cluster tools with parallel chambers. IEEE Transactions on Automation Science and Engineering, 13(2), 1165-1175.

Kim, J. H., & Lee, T. E., 2008. Schedulability analysis of time-constrained cluster tools with bounded time variation by an extended Petri net. IEEE Transactions on Automation Science and Engineering, 5(3), 490-503.

Lee, T. E., Lee, H. Y., & Shin, Y. H., 2004, December. Workload balancing and scheduling of a single-armed cluster tool. In Proceedings of the 5th APIEMS Conference (pp. 1-15). Gold Coast, Australia.

Lee, T. E., 2008, December. A review of scheduling theory and methods for semiconductor manufacturing cluster tools. In Proceedings of the 40th conference on winter simulation (pp. 2127-2135). Winter Simulation Conference.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S., 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

Molloy, M. K., 1982. Performance analysis using stochastic Petri nets. IEEE Transactions on computers, 31(9), 913-917.

Moody, J., Wu, L., Liao, Y., & Saffell, M., 1998. Performance functions and reinforcement learning for trading systems and portfolios. Journal of Forecasting, 17(56), 441-470.

Murata, T.,1989. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, 77(4), 541-580.

Puterman, M. L., 2014. Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.

Qiao, Y., Wu, N., & Zhou, M., 2012. Real-time scheduling of single-arm cluster tools subject to residency time constraints and bounded activity time variation. IEEE Transactions on Automation Science and Engineering, 9(3), 564-577.

Sutton, R. S., & Barto, A. G., 1998. Reinforcement learning: An introduction (Vol. 1, No. 1). Cambridge: MIT press.

Zuberek, W. M., 2004. Cluster tools with chamber revisiting-modeling and analysis using timed Petri nets. IEEE Transactions on semiconductor manufacturing, 17(3), 333-344.