공학석사학위논문

Temporary Capacity Expansion Policy in Semiconductor FAB using Reinforcement Learning

반도체 공장 내 일시적인 생산 용량 확장 정책 제안

2020 년 12 월

서울대학교 대학원

산업공학과

이 희 재

Temporary Capacity Expansion Policy in Semiconductor FAB using Reinforcement Learning

반도체 공장 내 일시적인 생산 용량 확장 정책 제안

지도교수 박 건 수

이 논문을 공학석사 학위논문으로 제출함

2020 년 12 월

서울대학교 대학원

산업공학과

이 희 재

이희재의 공학석사 학위논문을 인준함

2020 년 12 월

위 원 장	문 일 경	(인)
부위원장	박 건 수	(인)
위 원	이 재 욱	(인)

Abstract

Temporary Capacity Expansion Policy in Semiconductor FAB using Reinforcement Learning

Hee Jae Lee

Department of Industrial Engineering

The Graduate School

Seoul National University

Due to the instability of the capacity of the semiconductor process, there are cases in which the production capacity temporarily becomes insufficient compared to the capacity allocated by the initial plan. To respond, production managers require capacity to other lines with compatible equipment. This decision can have an adverse effect on the entire line because the processes are connected in a sequence. In particular, it becomes more problematic when the machine group is a bottleneck process group. Therefore, this study proposes a capacity expansion policy learned by reinforcement learning algorithms in this environment using a FAB simulator built upon a WIP balancing scheduler and a machine disruption model. These policies performed better than policies imitating human decision in terms of throughput and machine efficiency.

Keywords: Capacity Management, Reinforcement Learning, Industrial engineering
Student Number: 2019-23474

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Semiconductor factories are consisted of several lines which are classified by the product type it manages to produce. Each line is assigned its own Work In Process (WIP) and machines to manage. When a monthly production target is given at the company-wide production planning stage, the production target for each line is determined based on the target amount for each product. If it is difficult to achieve the production target, the affiliation of the machines may be changed on a weekly and monthly basis through discussions between lines. Reallocation of production capacity is done by reassigning the managing line of each machine. It is possible because most machines are compatible for different product types and the total FAB is connected by Over Head Transports (OHT) and trucks.

Even after such capacity redistribution, cases of insufficient capacity occur. This is because semiconductor facilities frequently experience unexpected disruptions that are difficult to predict. When such disruption occurs in the bottleneck process, additional capacity expansion is required because waiting WIP can be maintained at a high level for a long time. To cope with this, production managers make a decision to temporarily expand production capacity in shift units.

In this thesis, we consider the problem of temporarily sharing capacity of other lines which is called Send FAB. Send FAB is a major task that is required and requested frequently for production managers in semiconductor FABs. The decision, however, is made by the managers' experiences on the WIP status. They seek to request as much capacity as possible to the amount they think they need. They struggle to release waiting WIP as fast as possible. The more they experienced a tight production schedule, their belief on maximizing the quantity is stronger.

Send FAB decisions should be made very carefully. It is because semiconductor FABs has properties such as complicated sequence of re-entrant process, multi-objective system, and highly unpredictable machine status. The best way to make a careful decision is to consider all status regarding to the production objective. It is why our thesis introduces a Markov Decision Process. The fact that managers cannot see WIP status, machine status, and machine dedication all together made us think of creating a policy that considers all these states as an input. The stochastic nature of FAB aggregates limits our model to be solved by deterministic optimization models.

The environment of our Send FAB problem is built on a FAB simulator. Since the real FAB data is not available, we try to conform a FAB environment containing the most important features of Send FAB problem. The simulator consists of a scheduler that decides production schedule every shift, and a disruption model of machines to model machine uncertainty. The simulator reads the beginning WIP and machine status of each shift and returns the ending WIP and machine status.

We use Reinforcement Learning to solve MDP on our FAB environment. Since the problem with continuous state variables make the state space dimension infinite, and the transition probability is not deterministic, we cannot use tabular methods. Instead, we use a function approximation approach, Deep Q Network. We train the Deep Q Network in several FAB settings that are distinguished by different machine disruption scenarios and machine dedication scenarios.

We compare the performance of our trained policy and the policy that imitates the production managers. Since there are no officially known rule for a the Send FAB decision of the production manager, we make a policy that best describes the manager's decision. The results prove that our policy improves the performance in a big scale and that the policy imitating the human decision can harm the performance in some occasions. We start by describing the details of the problem in Section 1.1.

1.1 Problem Description

Send FAB is a contract between two lines of sharing one's capacity for a promised period and quantity. It is triggered by the production manager of the line with insufficient capacity on a process group, which is a group of machines that operate similar processes. The contract is agreed when both lines achieve benefits. For the requiring line, they get to deal with issues coming from capacity shortage. The offering line, on the other hand, can benefit by increasing their machine efficiency.

The actual operation is done by enrolling the contract to the Automatic Material Handling System (AMHS). Every lot and machine in the FAB have owning line tags. AMHS prevents track-in events of lots to machines with different owning line tags. When the Send FAB contract is enrolled, however, the AMHS allows the requiring line's lot to track-in to machines of the offering line. The lots of the requiring line is then recognized as lots of the offering line and competes with the offering line's lots by the dispatching score. This tag returns to its original status when the promised period or amount is reached.

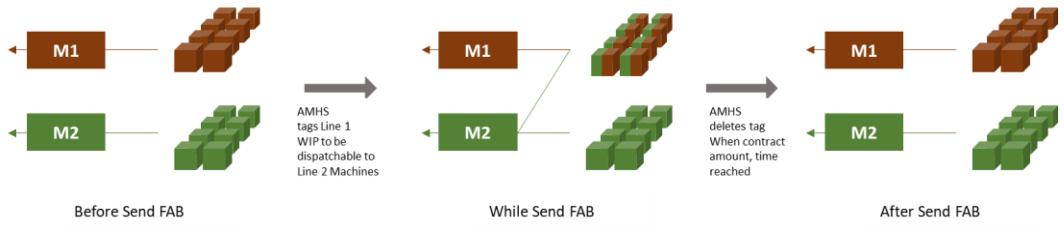

M1
M2
Before Send FAB
AMHS tags Line 1 WIP to be dispatchable to Line 2 Machines
M1
M2
While Send FAB
AMHS deletes tag When contract amount, time reached
M1
M2
After Send FAB

Figure 1.1: An illustration of how Send FAB works

The period of a Send FAB contract is usually one to three shifts. It is because decision makers are not sure of their decisions and to be able to deal with uncertainties of the future capacity status. The number of processes in a Send FAB contract is usually one. It is because the more different steps a machine processes, the higher the setup time would be. Therefore, the requiring line's manager usually selects one process step of the process group for Send FAB.

The selection of the process step is done by selecting the most delayed layer. In reality there are no Standard Operation Procedure (SOP) for the decision. It is mostly chosen by the manager's instinct based on years of experience. Still, for comparison of our RL policy, we used a rule-based policy that selects the most delayed layer which is the best out of just using the WIP information. The manager calculates the lateness of each layer of the problematic process group and selects the process step with the largest late quantity.

The manager of the requiring line asks as much as possible of the late quantity of the problematic layer's process. The final amount is decided after negotiation with the offering line's manager. If the late quantity is x_{late} and the affordable quantity of the offering line is $x_{affordable}$, the quantity is decided as $\min(x_{late}, x_{affordable})$.

For a better explanation let's suppose line A is the line with insufficient capacity and line B, C are the lines requested for some capacity sharing and the process group in capacity shortage is P. A scenario for a Send FAB is presented below.

1) Production manager of line A checks the WIP status of every layer $P_1, P_2, \ldots, P_N$ in process group P.
2) The manager calculates the late quantity $l_1, l_2, \ldots, l_N$ of each layer and decides the layer to require a Send FAB by $p = argmax_i \; l_i$.
3) The manager requires usage of compatible machines of line B, C with layer p for l_p wafers per shift.
4) Managers of line B, C check the status of their own WIP status on P and calculates the amount of capacity o_B, o_C they can provide.
5) o_B appears to be zero, which means support is unavailable from line B, and o_C appears to be non-zero but smaller than l_p.
6) The manager of line C enrolls a Send FAB contract of $\min(l_p, o_C)$ per shift in the system.

Deciding the process step, quantity of Send FAB based on the manager's instinct can cause future adverse effects. It is because the FAB is a multi-objective system having objectives such as throughput, cycle time, efficiency etc. Some have a trade-off relation. Even if the quantity of Send FAB is relatively low, it changes the aggregate of the FAB and could some time bring a massive damage to the whole system.

1.2 Research Motivation and Contribution

In order to deal with emergent situations on bottleneck process groups, Send FAB request come and go frequently between lines. The managers, however, are unavailable to grasp the FAB aggregate and the future effect of the decision. Even if by asking the most of the late quantity may seem to benefit by releasing waiting WIP, the cumulative performance may decrease due to problems that may occur at a future point.

We believe a Send FAB policy should be built on considering the FAB aggregate and its stochastic dynamics on machine uncertainty. Therefore, we develop a Markov Decision Process for Send FAB policy and derive a policy that can replace the manager's decision.

The thesis contributes by suggesting a better solution for the Send FAB problem. By showing results of human imitated policy damaging the production performance after all, we claim that a new policy is needed. The results show that our policy performs better in terms of throughput and machine efficiency which are the key performance measure for production management.

1.3 Organization of the Thesis

The thesis is composed of 5 chapters. In Chapter 2, we review literatures related to the problem. In Chapter 3, we introduce our proposed approach and methodology. In Chapter 4, results of computational experiments are presented. Finally, in Chapter 5, we give concluding remarks and possible future research directions of this thesis.

Chapter 2

Literature Review

To the best of my knowledge, there are no previous research on the Send FAB problem dealt in our research. Therefore, in this section we review researches on FAB scheduling and dynamic production control that we referred while developing our thesis.

2.1 Review on FAB Scheduling

Early research on FAB scheduling, which deal with lot releasing, due-date scheduling, and WIP balancing were focused on each single operational issue. There were researches on lot releasing methods attempting to avoid starvation of bottleneck machines by using the flow rate of a layer. The research of Lin and Lee [1] introduces an appropriate total WIP level in order to achieve better throughput rate while keeping the corresponding cycle time relatively low. They introduced a queueing network based algorithm to develop a FAB environment with a fixed-WIP control policy.

Researches that integrated approaches to focus on multiple performance measures are based on the flow rate control procedure. The Two Boundary algorithm was introduced by Lou and Kager [2]. The algorithm determines the Target Production Quantity (TPQ) by trying to make the difference between planned and actual production zero. Our scheduler was based on the research of Chung and Jang [3] which applies the concept of TPQ in order to solve the scheduling problem that can deal with WIP balancing on process layers, machines together. They introduced a new WIP balancing method called Toolset Available WIP Balancing (TAWB). By adding the concept of Average Available WIP for each machine to the WIP Balancing objective, they showed that it can prevent machine starvation of bottleneck machines.

2.2 Review on Dynamic Production Control

In order to deal with varying production requirements of various products, the FAB operates with multifunctional machines. This additional flexibility helps dealing with demand uncertainty of various products. Ever since its appearance, researchers have started to developed dynamic production controlling methods. Simulation approaches and Markov Decision Process approaches have been widely applied in solving the dynamic capacity allocation problem.

Toba [4] has proposed a load balancing method among multiple FAB lines using predictive scheduling results. It assumes the capacity sharable situation among multiple lines and tries to minimize the transportation cost between two process steps. Urayama, Fu and Marcus [5] adopts a hierarchical control model: long-term and capacity model and short-term job release control model to deal with the dynamic job release control. It applies simulation methods for estimating the parameters for each control model. Kim, Ko and Shin [6] applied a semi-Markov Decision Process and proposed a reinforcement learning method together with the fab simulator to obtain near optimal dispatching policy. They introduced a policy that learns the appropriate weight for selecting multiple dispatching policies together. Kim, Lim and Lee [7] introduced a dynamic scheduling system based on Deep Learning that can consider the Automated Material Handling System (AMHS) constraints. They suggested a new frame of applying trained Neural Networks replaceable of rule based dispatching policies.

Chapter 3

Proposed Approach and Methodology

This section describes how we develop our RL policy on the Send FAB problem. Then, we introduce the FAB simulator we used as our RL environment. Finally, we introduce the methodology on RL.

3.1 Proposed Approach

Our model is built upon the fact that production managers cannot see the whole FAB status. Therefore, we build a Markov Decision Process (MDP) that uses the FAB aggregate as the state and Send FAB decision as the action. In order to solve the MDP with RL, we use a FAB simulator based on preceding WIP Balancing Scheduler along with a disruption model as the RL environment.

We train the RL agent having a goal to meet the production target for the upcoming week. The time period is set as one week since production target usually varies by a week. Along with the fact that Send FAB is negotiated in a shift unit period, we consider every 21 shifts of one week to be the time horizon. The goal of the RL agent is to benefit the FAB aggregate using Send FAB contracts. We start by introducing the concepts of layers which we use to represent our problem.

3.1.1 Layer

The FAB environment is represented by layers of the bottleneck process group. Although there are dozens of process groups in semiconductor production FABs, this study focuses only on the bottleneck process group. It is because most of the Send FAB decisions are made on the bottleneck process group and whenever it causes an adverse effect in the future, it will be on the group itself. Based on the fact that the bottleneck process group controls the throughput rate [1], many researches ([2],[3]) on FAB bottleneck processes have modeled the entire FAB process as a series of loops re-entering the bottleneck process group. We define all processes between these loops as a layer. The processes starting from the process right after the bottleneck process to the next bottleneck group process forms a layer. This is illustrated as in Figure 3.1.

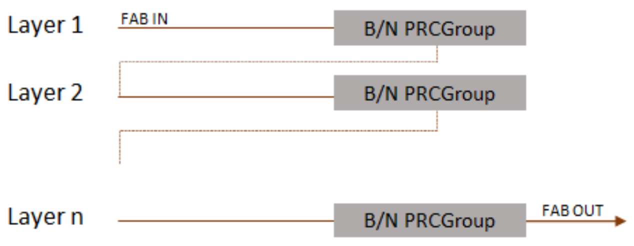


Figure 3.1: An illustration of layers

If the bottleneck process group has n processes, the model will have n layers. The flow time of a layer will be defined as the approximated turnaround time for all processes in the layer. The WIP level of a layer is be the total WIP in a layer. Also, each layer will be designated a demand proportional to their flow times starting from the last layer.

3.1.2 FAB Aggregate

The FAB aggregate represents the whole WIP status of each layer, machine status of each machine and the lateness of production for each layer. For every layer $j \in J_i$ for product $i \in I$, we define the beginning WIP as b_{ij} as the WIP waiting in all process steps in product i and layer j. The machine status for machine $k \in K$ at the beginning of the shift is defined as $m_k \in \{0,1\}$ where 0,1 indicates 'Down', 'Up' status.

The lateness of scheduled production target for each product and layer is defined as l_{ij}. In order to calculate the lateness of production, we compare the rolling demand with the rolling beginning WIP. The demand d_{ij} for product i and layer j represents the needed WIP for the layer. Using the weekly production target w_n and flow time c_{ij} we calculate the demand for product i and layer j as in (3.1) and (3.2). If the flow time of the layer is covered by one weekly target, we use (3.1). If the flow time of a layer is covered by two weekly target production volumes, we use (3.2). c_{ij_1} and c_{ij_2} each indicated the parts of c_{ij} in the first and second weeks, respectively.

$$d_{ij} = \frac{w_n c_{ij}}{7}, if\ 7(n-1) < c_{ij} \leq 7n \quad (3.1)$$

$$d_{ij} = \frac{w_n c_{ij_1}}{7} + \frac{w_{n+1} c_{ij_2}}{7}, if\ 7(n-1) \geq c_{ij}\ or\ c_{ij} \leq 7n \quad (3.2)$$

The demand indicates the wanted amount of WIP in order to meet the weekly production target. Since layers are sequentially connected, we roll the demand and WIP to compare the lateness. We define the lateness l_{ij} for product i, layer j as (3.3). We roll the demand and beginning WIP from the last layer to the next layer. This means difference between the amount that was supposed to be produced and sent to next layers, and the amount that has been produced and sent to next layers.

$$l_{ij} = \sum_{k=j+1}^{|J_i|} d_{ik} - \sum_{k=j+1}^{|J_i|} b_{ik} \quad (3.3)$$

3.1.3 MDP Modeling of Send FAB

In this section we introduce how the Send FAB problem was formulated into a MDP. Our MDP model is designed to consider the important factors and settings of the Send FAB problem. By reflecting the components of the FAB situation related to production KPIs such as WIP status of every layer, machine status of every machine, and the lateness of production by layers, we make it able to consider the FAB aggregate. We also set an appropriate time horizon that can embrace the settings of the Send FAB problem.

Since the production of semiconductors are nearly all time operating, the selection of the time horizon can be a major issue. Unlike MDP models having a terminal state that ends the episode, our problem does not have a specific goal that it heads for. With the no-break production and no-terminal state situation, we need to set a time horizon that appropriately leads the solution to fulfill the production manager's needs. In our model, considering the fact that Send FAB decisions are made to deal with a short-period (few shifts) capacity shortage, and that production target usually fluctuate on a weekly basis, we set the time horizon to 21 shifts, $T = \{1,2,\dots,21\}$. Therefore, we terminate the episode when it reaches the 21th shift (one week). This way, the solution of the MDP can provide a good Send FAB decision for the production manager considering the weekly production target.

The state space of our MDP is a set of vectors that represent the FAB aggregate. The vector is shown in following equations:

$$S = [\vec{W}, \vec{L}, \vec{M}] \tag{3.4}$$

$$\vec{W} = \left[\vec{W}_1, \vec{W}_2, \dots, \vec{W}_{|I|}\right], \vec{W}_i = [b_{i1}, b_{i2}, \dots, b_{i|J_i|}] \tag{3.5}$$

$$\vec{L} = \left[\vec{L}_1, \vec{L}_2, \dots, \vec{L}_{|I|}\right], \vec{L}_i = \left[l_{i1}, l_{i2}, \dots, l_{i|J_i|}\right] \tag{3.6}$$

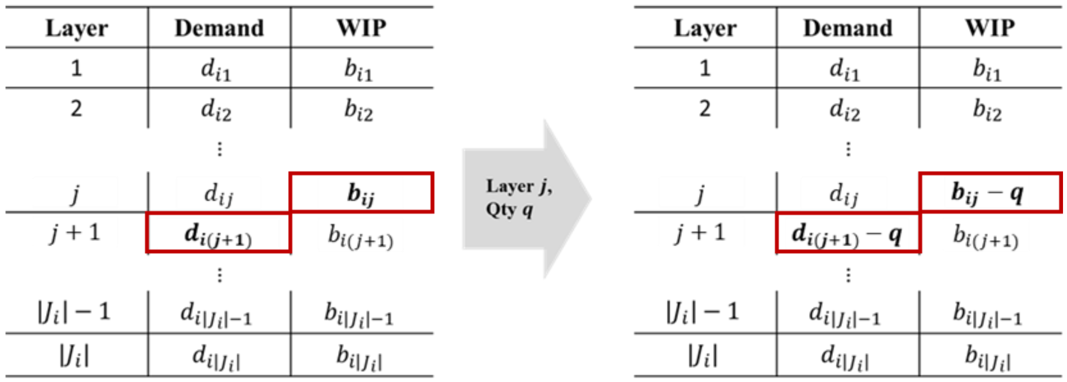
$$\vec{M} = \left[m_1, m_2, \dots m_{|K|}\right] \tag{3.7}$$

$$l_{ij} = \sum_{k=0}^{|J_i|-j-1} d_{i(|J_i|-k)} - \sum_{k=0}^{|J_i|-j-1} b_{i(|J_i|-k)} \tag{3.8}$$

$\vec{W}$ is the state vector of beginning WIPs for all products. Each vector $\vec{W_\iota}$ consists of initial WIP for each $|J_i|$ layers. $\vec{L}$ is the state vector of late production quantities for each product. Each vector $\vec{L_\iota}$ consists of late quantities for all $|J_i|$ layers. They are calculated by subtracting rolling WIP from rolling demand as shown in (3.8). The rolling demand indicates the amount that should have been produced and the rolling WIP is the amount that was produced. $\vec{M}$ is the state vector of machine status for $|K|$ machines.

The action space is a consisted of vectors of all possible Send FAB decision. Each Send FAB decision vector (i, j, q) is consisted of the product type i, layer number j and quantity q. Since Send FAB wafers are carried in a lot which has a size of 25 wafers, we discretized the possible range of sending quantity. The range is chosen to be smaller than 20% of the average layer demand to apply the fact that Send FAB quantities are small relative to regular production quantities. For example, if the average layer demand is 20,000 wafers we limit the maximum Send FAB quantity to be 400 wafers and the possible collection of Send FAB quantities would be $\{0,25,50,\dots,375,400\}$. In this case, the dimension of the action vector space is $17|I||J|$ where $|I|$ is the number of products, $|J|$ is the number of layers, and 17 is the number of Send FAB quantity selections.

The action for Send FAB on product i, layer j, with quantity q is applied by changing the WIP of and demand status of each layer. The beginning WIP b_{ij} and the demand for the next layer $d_{i(j+1)}$ is decreased by q. Figure 3.2 shows how the FAB aggregate changes.



Layer	Demand	WIP
1	d_{i1}	b_{i1}
2	d_{i2}	b_{i2}
	$\vdots$	
j	d_{ij}	$\boldsymbol{b_{ij}}$
$j+1$	$\boldsymbol{d_{i(j+1)}}$	$b_{i(j+1)}$
	$\vdots$	
$\lvert J_i\rvert - 1$	$d_{i\lvert J_i\rvert-1}$	$b_{i\lvert J_i\rvert-1}$
$\lvert J_i\rvert$	$d_{i\lvert J_i\rvert}$	$b_{i\lvert J_i\rvert}$

Layer	Demand	WIP
1	d_{i1}	b_{i1}
2	d_{i2}	b_{i2}
	$\vdots$	
j	d_{ij}	$\boldsymbol{b_{ij} - q}$
$j+1$	$\boldsymbol{d_{i(j+1)} - q}$	$b_{i(j+1)}$
	$\vdots$	
$\lvert J_i\rvert - 1$	$d_{i\lvert J_i\rvert-1}$	$b_{i\lvert J_i\rvert-1}$
$\lvert J_i\rvert$	$d_{i\lvert J_i\rvert}$	$b_{i\lvert J_i\rvert}$

Figure 3.2: An illustration of Send FAB action changing FAB aggregate

The reward is defined by whether there was improvement of the WIP Balancing Scheduler's objective value. Reward for choosing action A at current FAB aggregate S_t is defined as in (3.9) and (3.10). p is the penalty variable that prevents choosing more quantity than the beginning WIP. We give -5 as the penalty value for these cases, and 0 for the cases that does not violates this condition. Improvement and deterioration regard the change in the objective value of the WIP Balancing Scheduler. It checks how helpful the action was with respect to WIP balancing. The condition 'improvement' indicates that $O(S'_{t+1}) - O(S_{t+1}) > 0$, where $O(S)$ is the objective value of the WIP Balancing Scheduler and S'_{t+1} is the next state effected by Send FAB action. 'deterioration' means negative effects on the objective value, and 'large improvement' indicates to times when the improvement quantity is the largest among previous shifts.

$$R(S_t, a) = \begin{cases} 10 + p & \mathit{if} \text{ large improvement} \\ 1 + p & \mathit{if} \text{ improvement} \\ 0 + p & \mathit{if} \text{ no improvement} \\ -10 + p & \mathit{if} \text{ deterioration} \end{cases} \tag{3.9}$$

$$p = \begin{cases} -5 & \mathit{if}\ q \geq b_{ij} \\ 0 & \mathit{if}\ q < b_{ij} \end{cases} \tag{3.10}$$

The objective of our MDP is to find a policy π that maximizes cumulative sum of rewards as shown in (3.11). The cumulative reward can be represented as the bellman optimality equation of action value function $Q(s, a)$ as shown in (3.12).

$$E[\sum_{t=0}^{20} \gamma^t R(S_t, \pi(S_t))] \tag{3.11}$$

$$Q_{\pi^*}(s, a) = R(s, a) + \gamma \max_{a' \in A} Q_{\pi^*}(s', a') \tag{3.12}$$

In order to solve the bellman optimality equation, we need to know the transition probability from prevision state $S_t = [\overrightarrow{W_t}, \overrightarrow{L_t}, \overrightarrow{M_t}]$ to next state $S_{t+1} = [\overrightarrow{W}_{t+1}, \vec{L}_{t+1}, \overrightarrow{M}_{t+1}]$ for choosing action $A_t = [i, j, q]$. The process, however, requires a large amount of computation to cover all possible transitions.

Figure 3.3 shows the process of transition when chosen an action. When initial state S_t is given and action A_t is chosen, the WIP status and lateness status changes as shown in Figure 3.2. The changed FAB aggregate $S_t{}'$ then goes through a scheduling process that decides the production quantity X_t which is a matrix consisted of the production variables x_{ijk} assigned for each product i, layer j, machine k. Then the initial machine status $\overrightarrow{M_t}$ goes through a machine disruption scenario which is based upon a Continuous Markov Chain model (3.2.2 describes details). It returns $\vec{R}_t$ which is a vector consisted of running rates of each machine throughout the shift. $\vec{R}_t$ is then applied to the planned production X_t to acquire the actual production X_t'. Then X_t' is used to calculate the next WIP status $\overrightarrow{W}_{t+1}$and lateness $\vec{L}_{t+1}$. The initial machine status $\overrightarrow{M}_{t+1}$ is directly acquired from the machine disruption model.

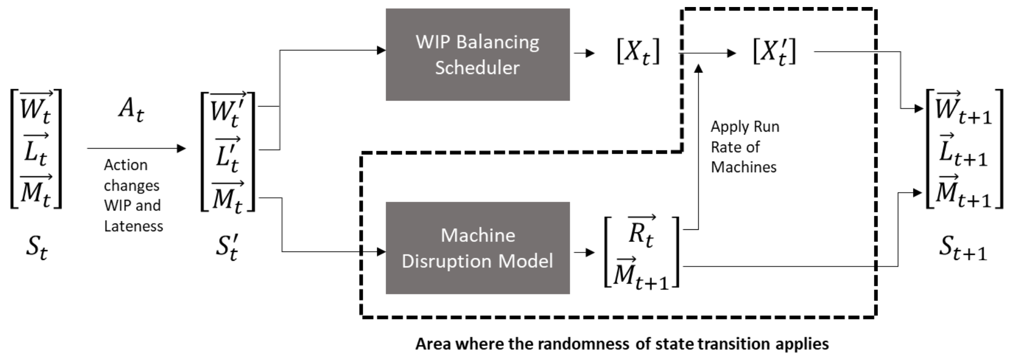

WIP Balancing Scheduler
$[X_t]$
$[X_t']$
$\begin{bmatrix}\overrightarrow{W_t}\\ \overrightarrow{L_t}\\ \overrightarrow{M_t}\end{bmatrix}$
S_t
A_t
Action changes WIP and Lateness
$\begin{bmatrix}\overrightarrow{W_t'}\\ \overrightarrow{L_t'}\\ \overrightarrow{M_t}\end{bmatrix}$
S_t'
Apply Run Rate of Machines
Machine Disruption Model
$\begin{bmatrix}\overrightarrow{R_t}\\ \overrightarrow{M}_{t+1}\end{bmatrix}$
$\begin{bmatrix}\overrightarrow{W}_{t+1}\\ \vec{L}_{t+1}\\ \overrightarrow{M}_{t+1}\end{bmatrix}$
S_{t+1}
Area where the randomness of state transition applies

Figure 3.3: An illustration of how transition probabilities are calculated

In order to calculate the transition probability, we discriminate deterministic processes and random processes. The deterministic processes do not influence the probability. The transition probability is only affected by the random processes (dotted box area in Figure 3.3). $\overrightarrow{W}_{t+1}$, $\vec{L}_{t+1}$ is obtained using the actual production X_t' which is determined by the random vector $\vec{R}_t$, and $\vec{M}_{t+1}$ is the next machine status given the previous initial status $\vec{M}_t$. Therefore, we only consider the probability $P\{\vec{R}_t, \vec{M}_{t+1} | \vec{M}_t\}$. In other words, $P\{S_{t+1}|S_t, A_t\} = P\{(\vec{R}_t, \vec{M}_{t+1})|S_t'\}$. $P\{(\vec{R}_t, \vec{M}_{t+1})|S_t'\}$ can be calculated from the Continuous Time Markov Chain we define as the machine disruption model (3.2.2).

The values of $\vec{R}_t$, however, requires to solve a Mixed Integer Programming (MIP) problem and a Linear Programming (LP) problem each time for a pair of state and action. The MIP problem is needed to get the scheduled production amount X_t and LP is used to solve the corresponding r_{ij} values that generates X_t' which eventually leads to the final state S_{t+1}. In order to solve our MDP with Dynamic Programming, we need to obtain all possible transition probabilities between states. Also, the states have continuous values that makes it impossible to approach the problem in a tabular method. Even we go through discretization on state values, the two required optimization in calculating a single transition probability would lead to a large amount of computation time. Therefore, we instead use a model-free Reinforcement Learning Approach.

3.1.4 Learning Send FAB policy

Our MDP is defined in a state space with infinite dimension. Therefore, we use a Q function approximating approach instead of a tabular method which save all Q values. Among the function approximating approach, we use Deep Q Network (DQN) [8]. The DQN algorithm works by selecting the greatest estimated Q value for a given state and action. The neural network uses the state of our MDP model as the input variable and returns the Q value for every possible action for the given state. In order to fit the neural network, we train the model as in Algorithm 1.

Algorithm 1 Learning DQN for Send FAB

```
Initialize FAB environment
Initialize action value function $Q_{online}$ with random weights
Copy $Q_{online}$ as $Q_{target}$
Create dequeing structure $D$ for memory
For episode $= 1, M$:
  Initialize weekly demand and allocate layer demand
  Initialize FAB aggregate $s_1$ and preprocess into $\phi_1 = \phi(s_1)$
  For $t = 1,21$:
      With probability $\epsilon$ select a random action $a_t$
      otherwise select $a_t = \max_a Q^*_{online}(\phi(s_t), a; \theta)$
      increase $\epsilon$ by $10^{-5}$ and $\epsilon \leftarrow \max(0.01, \epsilon)$
      if $a_t$ is no Send FAB then:
        $r_t = 0$
      else:
        Create $s'_t$ which is the updated FAB aggregate after $a_t$
        Solve WIP Balancing MIP for $s_t, s'_t$ and get $x_{t+1}, O(s'_t), O(s_t), E(s'_t), E(s_t)$
        $r_t = O(s'_t) - O(s_t) + e\,(E(s'_t) - E(s_t))$.
      Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
      if $|D| >$ minibatch size then:
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set $y_j = r_j + \gamma \max_{a'} Q_{target}(\phi_{j+1}, a'; \theta)$
        Perform a gradient descent step on $\left(y_j - Q_{online}(\phi_j, a_j; \theta)\right)^2$
```

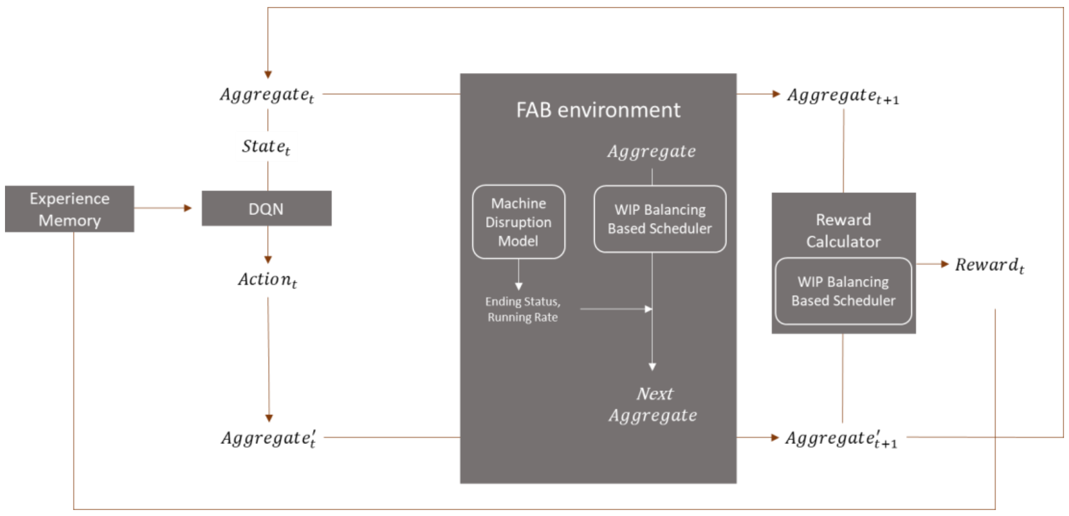


Figure 3.4: An illustration of learning the Send FAB policy

DQN is fitted by numerous episodes that is performed as shown in Figure 3.3. The RL agent chooses the action with the largest estimated Q value from the current DQN. The action is performed to change the FAB aggregate for the current shift. The changed FAB aggregate along with the original FAB aggregate are used as the initial FAB status for production scheduling. After applying a machine disruption scenario, the next FAB aggregate is obtained. For reward assignment we compare the WIP Balancing scheduler's objective of the next shift's aggregate with and without the action. Then the state, action and reward are then saved in an experience memory which has a LIFO structure with a given memory size. The DQN is fitted with a batch of experiences extracted from the experience memory and repeats the process until the end of episode. We used an architecture for the Neural Network as shown in Figure 3.4. The learning rate was set 0.001, and epsilon increases by 10^{-5} for every shift.

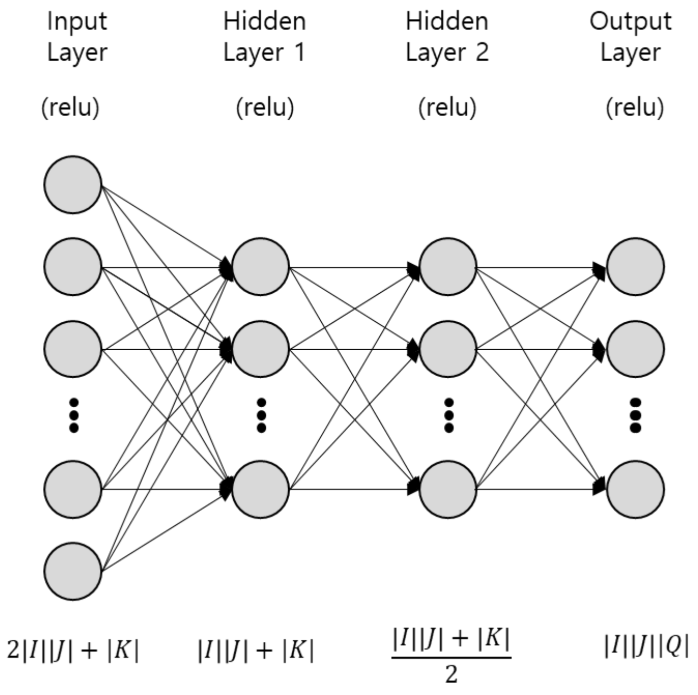


Figure 3.5: An illustration of our Deep Q Network

3.2 FAB simulator

The FAB simulator is composed of a WIP Balancing Scheduler and a Machine Disruption Model. Since real production data is not available for this study, we make a simulator that can represent the key elements of the Send FAB problem. We use a WIP Balancing Scheduler of Chung and Jang [4]. It decides the production quantities and machine usage on bottleneck layers. To deal with machine status on making a Send FAB decision, we made a Machine Disruption Model that models the status of the machine and its running time during a shift.

3.2.1 WIP Balancing Scheduler

In this section we introduce the Mixed Integer Program (MIP) problem of Chung and Jang [4]. The scheduler decides how much to produce for each product and layer. It seeks to minimize lateness of production and starvation of machines. It considers constraints regarding to machine dedication and production capacity. It uses WIP status and machine status of the previous shift along with machine dedication, process time, and setup time as input data.

Unlike the original MIP of Chung and Jang, we remove the constraints related to lithographic process groups. It is a constraint that prevents the number of assigned machines exceeding the number of masks for each layer. Unlike Chung and Jang, we do not assume the bottleneck process group is always a lithographic process group. We intend our model to be applied to any process group that currently is the bottleneck. Therefore, we remove the mask constraints from the original MIP. The following notations are used in this model.

1) Data sets

I:	Set of products.
J_i:	Set of layers of product $i, i \in I$.
K:	Set of machines.
N:	Set of weeks for demands.

2) Input Data

S:	Setup time of machine.
c_{ij}:	Flow time of product i, layer j.
C_{ij}:	Sum of flow time of product i from layer j to final layer.
p_{ijk}:	Wafer processing time of product i, layer j on machine k.
r_{ijk}:	Initial processing layer index. If machine k is processing product i, layer j at the beginning of the current shift, $r_{ijk} = 1$; otherwise $r_{ijk} = 0$.
b_{ij}:	WIP level of product i, layer j at the beginning of the current shift.
e_{ij}:	WIP level of product i, layer j at the end of the current shift.

w_n: Target production volume for week n.

d_{ij}: Layer demand for product i, layer j.

h_{ijk}: Toolset dedication. If product i, layer j can be processed at machine k, $h_{ijk} = 0$.

u_{ij}: Upper-limit production quantity (UPQ) for product i, layer j.

l_{ij}: Current lateness(days) of WIP of product i, layer j

3) Decision Variables

x_{ijk}: Production quantity for product i, layer j from machine k during current shift.

y_{ijk}: Production assignment. If $x_{ijk} > 0$, $y_{ijk} = 1$; otherwise $y_{ijk} = 0$.

The WIP Balancing problem is as follows.

$$\text{minimize} \quad \sum_{i\in I}\sum_{j\in J}(u_{ij} - \sum_{k\in K} x_{ijk}) + G\sum_{k\in K}|MAW_k - AAW| \qquad (3.13)$$

$$\text{subject to} \quad \mathrm{WPM_{ij}} = \frac{e_{ij}}{\sum_{k\in K} h_{ijk}} \qquad i \in I, j \in J_i \qquad (3.14)$$

$$MAW_k = \sum_{i\in I}\sum_{j\in J} MPT_{ij} h_{ijk} \qquad i \in I, j \in J_i,\ k \in K \qquad (3.15)$$

$$\mathrm{AAW} = \frac{\sum_{k\in K} MAW_k}{|K|} \qquad k \in K \qquad (3.16)$$

$$\textstyle\sum_{k\in K} x_{ijk} \le b_{ij} \qquad i \in I, j \in J_i \qquad (3.17)$$

$$x_{ijk} \le M y_{ijk} \qquad i \in I, j \in J_i \qquad (3.18)$$

$$\textstyle\sum_{i\in I}\sum_{j\in J} x_{ijk}\, p_{ijk} + S\big(\sum_{i\in I}\sum_{j\in J} y_{ijk} - 1\big) \le H \qquad k \in K \qquad (3.19)$$

$$e_{ij} = b_{ij} + \textstyle\sum_{k\in K} x_{i(j-1)k} - \sum_{k\in K} x_{ijk} \qquad i \in I, j \in J_i - \{0\} \qquad (3.20)$$

$$e_{i0} = b_{i0} + \textstyle\sum_{k\in K} x_{i|J|k} - \sum_{k\in K} x_{i0k} \qquad i \in I \qquad (3.21)$$

$$\textstyle\sum_{k\in K} x_{ijk} \le u_{ij} \qquad i \in I, j \in J_i \qquad (3.22)$$

$$y_{ijk} = 0 \quad if \quad h_{ijk} = 0 \qquad i \in I, j \in J_i,\ k \in K \qquad (3.23)$$

$$y_{ijk} = 1 \quad if \quad r_{ijk} = 1 \qquad i \in I, j \in J_i,\ k \in K \quad (3.24)$$

$$x_{ijk} \geq 0 \qquad i \in I, j \in J_i,\ k \in K \quad (3.25)$$

$$y_{ijk} \in \{0,1\} \qquad i \in I, j \in J_i,\ k \in K \quad (3.26)$$

(3.13) is the objective function for the problem. It is consisted of two parts: $\sum_{i\in I}\sum_{j\in J}(u_{ij} - \sum_{k\in K} x_{ijk})$ is the part to assure production on product i, layer j be close to the Upper Production Quantity (UPQ) u_{ij} and $G\sum_{k\in K}|TAW_k - AAW|$ is to minimize the variation on WIP per machine which can help prevent machine starvation for unbalanced WIP distribution. UPQ is defined as in Equation (3.27). $\sum_{l=j}^{|J|} d_{il}$ is the rolling demand, $\sum_{l=j+1}^{|J|} b_{il}$ is the rolling WIP, and Q_{ij} is the pulling demand which is the upcoming layer's demand for 3 days. Therefore, UPQ is the WIP amount that is behind schedule plus the WIP that are soon coming into the buffer.

$$u_{ij} = \sum_{l=j}^{|J|} d_{il} - \sum_{l=j+1}^{|J|} b_{il} + Q_{ij} \qquad (3.27)$$

For the WIP balancing among machines, the objective minimizes the difference between the average available WIP (AAW) and each machine's average WIP (MAW_k). Equations (3.14), (3.15), and (3.16) shows how the ending inventory (e_{ij}) and machine dedication (h_{ijk}) is used to define these concepts.

Equation (3.17) is the production quantity constraint that makes production quantity not to exceed the available beginning WIP. Equation (3.18) is the machine setup constraint that makes $y_{ijk} > 0$ if there is production during a scheduled shift. Where M is a large number. Equation (3.19) is the machine time constraint that makes total process time and setup time not exceed a shift. Equation (3.20), (3.21) are the balance equation constraints. The ending inventory of a layer is the sum of beginning inventory and incoming WIP minus the WIP produced in the layer this shift. The final layer production is released back into the first layer and makes the relation as in Equation (3.21). The model assumes a fixed total WIP release policy and therefore releases the same amount produced at the last layer. Equation (3.22) is the UPQ constraint that prevents production exceeding UPQ. Equation (3.23) is the machine dedication constraint that blocks unavailable layer-machine production. Equation (3.24) is the current processing layer constraint that makes the running status continue from the previous shift. If a machine was running in the previous state, the machine starts in a usage status for the current shift. Equation (3.25), (3.26) are the constraints on the decision variables.

3.2.2 Machine Disruption Model

The scheduler itself, cannot work as a simulator. The solution of the scheduler is a scheduled quantity, to get the actual production results we need a disruption scenario. In order to adapt a disruption scenario, we need to generate the next status and the running rate during the period when given the initial status. We assume the machine status follows a Continuous Time Markov Chain (CTMC) with two states 'Up' and 'Down'. We define the machine status at shift t as $X(t)$. The transition rate of 'Up' becoming 'Down' is defined as $P\{X(t+dt) = Down|X(t) = Up\} = \lambda_1 dt$. The transition rate of 'Down' becoming 'Up' is defined as $P\{X(t+dt) = Up|X(t) = Down\} = \lambda_2 dt$. The transition diagram is shown in Figure 3.5.

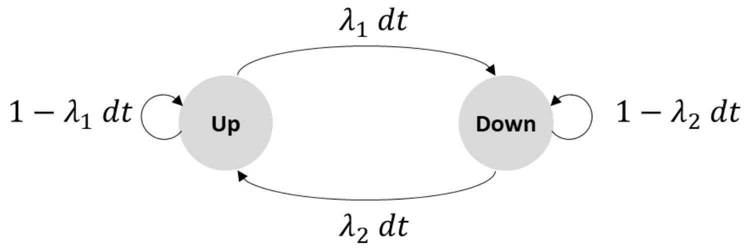
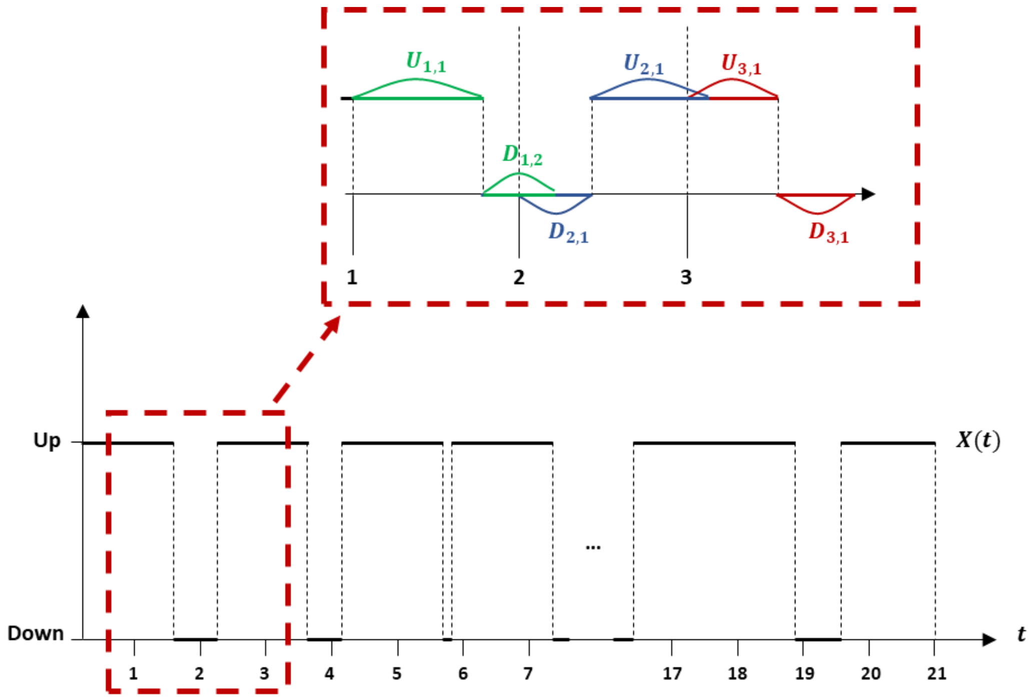


Figure 3.6: An illustration of transition diagram of machine status



$m_{k(t+1)}$	Up	Down	Up	Down	Up	Up	Up	...	Up	Up	Down	Up	Up
r_{kt}	1	0.75	0.8	0.65	0.95	0.95	1		0.7	1	0.95	0.45	1

Figure 3.7: An illustration of how the CTMC provides disruption scenarios

The purpose of the disruption model is to generate the machine status of the next shift and the running rate during the shift. Since all machines follow a CTMC we use the initial machine status for each shift t and sample interarrival times to generate the next initial status $m_{k(t+1)}$ and the running rate r_{kt}. Figure 3.6 shows how the CTMC provides the disruption scenarios on a shift time basis. Here, m_{kt} is the status of machine k at the beginning of shift t, and r_{kt} is the running rate during shift t.

We can acquire $m_{k(t+1)}, r_{kt}$ by sampling interarrival times for given current status of the machine. The upper part of Figure 3.6 shows that every start of a shift, we sample the corresponding interarrival random variable $U\sim \exp(\lambda_1)\ or\ D\sim\exp(\lambda_2)$ and seize the machine status when it reaches the next shift. The following algorithm shows the process of sampling running time and recovery time according to the given initial status. It keeps sampling until the remaining time of the current shift becomes zero. The last status reaching the next shift becomes the initial shift for the next shift. Running rate r_{kt} which is the cumulated running time during a shift, can be given by the sum of running time throughout the repeated process. The Pseudo code is as follows.

Algorithm 2 Sampling from CTMC

```
**SAMPLECTMC**$(S, r, c)$
  **if** $S = UP$ **then**
    $uptime = sample\_exp\ (\lambda_1)$
    **if** $uptime < r$ **then**
      $c \leftarrow c + uptime$
      $downtime = sample\_exp(\lambda_2)$
      **if** $uptime + downtime > r$ **then**
        $return\ [DOWN, c]$
      **else**
        $return$ **SAMPLECTMC**$(UP, r - uptime - downtime, c)$
    **else**
      $c \leftarrow c + r$
      $return\ [\mathrm{UP}, \mathrm{c}]$
  **else**
    $downtime = sample\_exp\ (\lambda_2)$
    **if** $downtime > r$ **then**
      $return\ [Down, c]$
    **else**
      $uptime = sample\_exp\ (\lambda_1)$
      **if** $uptime + downtime > r$ **then**
        $c \leftarrow c + r - downtime$
        $return\ [UP, c]$
      **else**
        $return$ **SAMPLECTMC**$(UP, r - uptime - downtime, c)$

$m_{k(t+1)}, r_{kt}$ = **SAMPLECTMC**$(m_{kt}, 1, 0)$
```

Instead of controlling transition rate values, we decided to use a combination of probabilities $(t_{run}, t_{recover})$. It is because our model requires the running rate for the current shift and the ending status at the end of the shift. We wanted to assign a probability of machine running more than 1 shift given the initial status 'Up', along with the probability of machine taking less than 1 shift to recover from the initial 'Down' status. Equations bellow explain the relation between $t_{run}, t_{recover}$ and λ_1, λ_2. U, D are the interarrival times of 'Up' to 'Down' and 'Down' to 'Up' respectively. By our defined CTMC each follows an exponential distribution with λ_1, λ_2 respectively.

$$t_{run} = P\{U \geq 1\} = \int_1^\infty \lambda_1 e^{-\lambda_1 t} dt \tag{3.27}$$

$$\lambda_1 = -\ln\,(t_{run}) \tag{3.28}$$

$$t_{recover} = P\{D \leq 1\} = \int_0^1 \lambda_2 e^{-\lambda_2 t} dt \tag{3.29}$$

$$\lambda_2 = -\ln\,(1 - t_{recover}) \tag{3.30}$$

We have composed total 9 combinations of $(t_{run}, t_{recover})$ to experiment on various machine stability conditions. The combination is the product set of {0.5,0.7,0.9} and {0.5,0.7,0.9}. Table 3.1 shows the corresponding λ_1, λ_2, Mean Time Between Failures (MTBF), Mean Time To Recovery (MTTR) of corresponding values of $t_{run}, t_{recover}$. Some cases such as t_{run} having value 0.5 might be unrealistic, however, we tried to compare the performance of RL from the human imitating policy in various machine stability conditions.

Table 3.1: λ_1, λ_2, MTBF, MTTR for corresponding $t_{run}, t_{recover}$ values

t_{run}	λ_1	MTBF
0.5	0.69	1.44
0.7	0.36	2.80
0.9	0.11	9.49

$t_{recover}$	λ_2	MTTR
0.5	0.69	1.44
0.7	1.20	0.83
0.9	2.30	0.43

In order to check the effect of various $t_{run}, t_{recover}$ combinations on the simulator, we have sampled the average running rate r_{kt} given the initial state m_{kt} (Up or Down) using Algorithm2. The results are shown in Table 3.2. We also approximated the probability of the next shift being 'Up' given the initial machine status. The results were calculated by calculating the proportion of 'Up' among the 10,000 samples for each case.

Table 3.2: Average running rate r_{kt} observed for corresponding $t_{run}, t_{recover}$ values

t_{run}	$t_{recover}$	Average Running Rate $E[r_{kt} \vert m_{kt}]$	
		$m_{kt} = Up$	$m_{kt} = Down$
0.5	0.5	0.66	0.21
	0.7	0.74	0.29
	0.9	0.88	0.44
0.7	0.5	0.78	0.29
	0.7	0.84	0.4
	0.9	0.93	0.58
0.9	0.5	0.87	0.36
	0.7	0.91	0.52
	0.9	0.96	0.71

Table 3.3: Probability of next shift being 'Up' for corresponding $t_{run}, t_{recover}$ values

t_{run}	$t_{recover}$	Probability of next shift being 'Up' $P[m_{k(t+1)} = Up \vert m_{kt}]$	
		$m_{kt} = Up$	$m_{kt} = Down$
0.5	0.5	0.52	0.47
	0.7	0.66	0.66
	0.9	0.87	0.87
0.7	0.5	0.68	0.57
	0.7	0.78	0.76
	0.9	0.92	0.92
0.9	0.5	0.81	0.65
	0.7	0.87	0.83
	0.9	0.96	0.95

3.3 Reinforcement Learning Approach

3.3.1 Markov Decision Process and Reinforcement Learning

In Reinforcement Learning (RL), the problem to resolve is described as a Markov Decision Process (MDP). Since the theoretical results of RL rely on the MDP description, the more the problem is acceptable as a MDP problem, the better RL would work as a good solution. A MDP is composed of objects $< T, S, A, P(\cdot\,|s,a), R(s,a) >$ where T is a discrete time horizon, S is a state space, A is an action space, $P(\cdot\,|s,a)$ are the state transition probabilities and $R(s,a)$ is a reward function. For our problem to be suitable for a MDP problem, we must assume the Markov property in Equation (3.31) applies to the transition probabilities. This means the transition to the next period state only depends on the previous state and action.

$$P(s_{t+1} = s'|s_t, a_t) = P(s_{t+1} = s'|\, s_t, a_t, s_{t-1}, a_{t-1}, \ldots s_0, a_0) \qquad (3.31)$$

The goal of a MDP is to find a good policy, which is a function $\pi : S \rightarrow A$ that specifies the action $\pi(s)$ to choose given state s. A good policy is to maximize the expected sum of random future rewards. With a discount factor satisfying $0 \leq \gamma \leq 1$, the goal would be to maximize $V_\pi(s) = E_\pi[\sum_{j=0}^{\infty} \gamma^t\, R_{t+j+1}|\, S_t = s)]$. The policy that maximizes this function is called the optimal policy for the MDP and is denoted as π^*.

When we know the transition probabilities $P(s'|s,a)\ for\ \forall s, s' \in S, a \in A$, we solve it through Dynamic Programming. Finding the optimal policy goes under policy evaluation and policy iteration. Policy evaluation evaluates all state values represented as the state value function $V_\pi(s)$ and policy iteration improves the policy by updating the policy $\pi \rightarrow \pi',\ where\ \pi'(s) = greedy(v_\pi)$. This process is called value iteration and it is known to converge to an optimal policy. By Richard E. Bellman, this relation was shown as an equation called Bellman Optimality Equation. The Bellman Optimality Equation for value function and action value function is represented in Equation (3.32), (3.33) respectively.

$$V_{\pi^*}(s) = \textstyle\sum_{s' \in S} P(s'|s,a)\,(R(s,a) + \gamma V_{\pi^*}(s')) \qquad (3.32)$$

$$Q_{\pi^*}(s,a) = R(s,a) + \gamma \max_{a' \in A} Q_{\pi^*}(s', a') \qquad (3.33)$$

On the other hand, if we do not know the dynamics of the environment, we use RL. RL algorithms are based on the fact that we do not know the transition probabilities. Therefore, it updates the policy based on sampled experiences. The Monte Carlo (MC) method samples the whole return for each episode and uses it to evaluate the value function. In other words, it replaces the expectation on $V_\pi(s)$ or $Q_\pi(s, a)$ to an empirical mean return. Temporal Difference method does not sample every reward to the end of an episode. Instead, it samples few steps ahead and updates the value function $V_\pi(S_t)$ or action value function $Q_\pi(S_t, A_t)$towards an estimated return $R_{t+1} + \gamma V_\pi(S_{t+1})$ or $R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1})$. This method has benefit on the fact that it can learn while experiencing. One well known example for TD method is $SARSA$. The algorithm works by sampling rewards one step ahead and uses the estimated return to evaluate and improve the policy. For policy iteration, it uses epsilon greedy method. The algorithm is as follows.

Algorithm 3 SARSA

```
Initialize $Q(s, a)$ arbitrarily
For each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy from Q($\epsilon -$ greedy)
    For each episode:
      Take action $A$, observe $R, S'$
      Choose $A'$ from $S'$ using policy derived from $Q(\epsilon -$ greedy)
      $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma\, Q(S', A') - Q(S, A))$
      $S \leftarrow S'; A \leftarrow A'$;
      until $S$ is terminal
```

In $SARSA$, the maximum reward for the next state is not necessarily used for updating the Q values. Instead, a new action is selected using the same policy that determined the original action. This makes the policy exploit more the greedy selections. Algorithms that set the behavior policy and target policy same are called On-policy algorithms. The opposite concept is Off-policy. It overcomes the low exploration level and can learn about the optimal policy while following an exploratory policy. By separating the behavior policy and the target policy the agent can explore more trajectories. Q learning is the representative Off-policy algorithms and the pseudo code is as follows.

Algorithm 4 Q-Learning

```
**Initialize** $Q(s,a)$ arbitrarily
**For each episode**:
  **Initialize** $S$
    **For each episode**:
      **Choose** $A$ from $S$ using policy derived from $Q(\epsilon - \text{greedy})$
      **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
```

Both $SARSA$ and Q-learning are tabular methods and the curse of dimension can happen with large problems. It is because we need to save and read all updating Q values in a table. Therefore, for large environments such as FAB environment, requires a method called Value Function Approximation.

$$\widehat{V_\pi}(s,w) \approx V_\pi(s) \tag{3.34}$$

$$\widehat{Q_\pi}(s,a,w) \approx Q_\pi(s,a) \tag{3.35}$$

$$\nabla_w J(w) = \left(\frac{\partial J(w)}{\partial w_1}, \frac{\partial J(w)}{\partial w_2}, \dots, \frac{\partial J(w)}{\partial w_n}\right) \quad for\ |w| = n \tag{3.36}$$

$$\nabla \mathrm{w} = -\frac{1}{2}\alpha \nabla_w J(w) \tag{3.37}$$

$$J(w) = E_\pi\left[\left(V_\pi(s) - \widehat{V_\pi}(s,w)\right)^2\right] \tag{3.38}$$

$$\nabla w = \alpha E_\pi[\left(G_t - \widehat{V_\pi}(s,w)\right)\nabla_w \widehat{V_\pi}(s,w)] \tag{3.39}$$

$$\nabla w = \alpha E_\pi[\left(R_{t+1} + \gamma \widehat{V_\pi}(S_{t+1},w) - \widehat{V_\pi}(s,w)\right)\nabla_w \widehat{V_\pi}(s,w)] \tag{3.40}$$

Approximating the value function is done by parameterizing the value function as in Equation (3.34) and (3.35). The function can be in any regression form such as linear regression, Random Forests, Neural Networks etc. The parameter vector w is updated through gradient descent on the loss function $J(w)$. The gradient of the loss function, $\nabla_w J(w)$ is shown in Equation (3.21). The gradient for the parameter vector ∇w is defined as in Equation (3.22). The loss function is defined as in Equation (3.38) which leads to parameter update of MC, TD to be done as in Equation (3.39), (3.40) respectively.

3.3.2 Deep Q Network (DQN)

DQN is a Deep RL algorithm that uses Neural Networks as the Q value approximation function. This study refers to Minh et al (2013), which introduced Deep Neural Networks as a Q value approximation function for learning policies of playing Atari games. Since the state of game environments are in forms of image frames, using Convolution Neural Networks helped to understand the state of the player at some specific moment. The algorithm is based on Q-learning except the fact that they use an approximation function. The pseudo code is as follows.

Algorithm 5 Deep Q Network with Experience Replay

```
Initialize replay memory $D$ to capacity $N$
Initialize action value function $Q$ with random weights
For episode $= 1, M$:
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1, T$:
      With probability $\epsilon$ select a random action $a_t$
      otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
      $Execute\ action\ a_t\ and\ observe\ r_t, x_{t+1}$
      Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
      Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
      Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for nonterminal } \phi_{j+1} \end{cases}$
      Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$
```

They introduced a method called Experience Replay that stores experiences in a memory and sample batches to update the target policy. This was to make sure that the training data is independent and identically distributed, which is a fundamental requirement for Stochastic Gradient Descent optimization. Since sequentially learning from a sequence of an experience causes correlation among training data, by creating a large buffer of experience and randomly sampling a subset of these experiences help keep samples independent.

Chapter 4

Computational Experiments

We compared the performance of our RL policy and human imitated policy (HIP). Average throughput and machine efficiency were used as the performance measures. The two policies are described in Table 4.1. $\hat{Q}$ is the trained Deep Q Network and l_{ijt} is the lateness for product i, layer j at shift t.

Table 4.1: Policy of RL and HIP

Policy	Product, Layer	Quantity
RL	product, layer of $argmax_a \hat{Q}(s_t, a)$	quantity of $argmax_a \hat{Q}(s_t, a)$
HIP	$argmax_{i,j} l_{ijt}$	$min(l_{ijt}, M)$

4.1 Experimental Settings

The virtual FAB to be used in the experiment is a small FAB with two products, five layers, and ten machines. The size of the virtual FAB was chosen to be small since the WIP Balancing Scheduler takes a long time (15 minutes per shift for a FAB with four products, 25 layers, and 50 machines) for solving an optimal solution for one shift. Considering the required number of shifts to learn the Q function approximator and that the purpose of our research is to show the existence of a better policy than human decision concluded us that it will be better to experiment in a small FAB environment.

The weekly demand was sampled from a normal distribution with mean 4,000 wafers/week and standard deviation 500. The process time for each product and layer was set equal as 300 seconds per wafer. The setup time was set 10 minutes/setup. The initial WIP at the beginning of every experiment was set Rd_l where d_l is the layer demand and $R \sim uniform(0.5,0.7)$. The machine running status r_{ijk} for the initial shift was set zero. The machine dedication h_{ijk} was generated by randomly assigning layers to have 5 or 6 machines available to process.

We used the Python-MIP package to solve the WIP Balancing Scheduling problem and Pytorch package to build and train DQN. Experiments were performed on a computer with Ubuntu 18.04 as operating system, processor Intel(R) Core(TM) i7-7700, GeForce gtx 1080 and 16G of RAM.

4.2 Test Instances

The experiment was done in the following test instance sets shown in Table 4.2. They were generated to compare the results in different disruption scenarios and different machine dedication scenarios. The disruption scenarios were generated by using the product of $\{0.5, 0.7, 0.9\}$ itself as the space for $(t_{run}, t_{recover})$. Here we fixed the Machine Numbers Per Layer (MNPL) to $\{5,6\}$. For comparing various machine dedication scenarios, we selected MNPL from $\{2\ or\ 3, 5\ or\ 6, 8\ or\ 9\}$ and $(t_{run}, t_{recover})$ from $\{(0.5,0.5), (0.7,0.7)\}$.

Table 4.2: Description of test instances

Set	t_{run}	$t_{recover}$	$MNPL$
1	0.5	0.5	5 or 6
2	0.5	0.7	5 or 6
3	0.5	0.9	5 or 6
4	0.7	0.5	5 or 6
5	0.7	0.7	5 or 6
6	0.7	0.9	5 or 6
7	0.9	0.5	5 or 6
8	0.9	0.7	5 or 6
9	0.9	0.9	5 or 6
10	0.9	0.9	5 or 6
11	0.9	0.9	5 or 6
12	0.9	0.9	5 or 6
13	0.5	0.5	2 or 3
14	0.7	0.7	2 or 3
15	0.5	0.5	5 or 6
16	0.7	0.7	5 or 6
17	0.5	0.5	8 or 9
18	0.7	0.7	8 or 9

4.3 Test Results

We performed two tests for our thesis. First was to show that our policy can perform better than human imitated policy (HIP). This was performed in various machine disruption scenarios. Secondly, by applying various machine utilization levels we showed that the more complicated the layer-machine relation is makes HIP perform worse than not applying it. Each performance test was done by running 1,000 tests for each test instances.
Test results for different machine scenarios are shown in Table 4.1, Table 4.2. We compared the average throughput and average machine efficiency for each policy. This verifies that the proposed method RL is more effective than HIP.

Table 4.3: Average Throughput comparison by different machine disruption scenarios

t_{run}	$t_{recover}$	Throughput (wf/shift)			
		RL	*HIP*	*RL↔HIP(%)*	*No SendFAB*
0.5	0.5	83.59	48.96	70.7	52.78
	0.7	92.87	67.54	37.5	68.31
	0.9	113.70	91.23	24.6	87.77
0.7	0.5	92.97	68.50	35.7	69.62
	0.7	106.74	82.57	29.3	80.18
	0.9	117.09	99.50	17.7	93.01
0.9	0.5	115.87	84.92	36.4	83.02
	0.7	119.83	93.8	27.7	88.86
	0.9	121.63	105.37	15.4	98.4

The result shows that RL policy performs better than HIP in terms of average throughput and machine efficiency both. The throughput of RL was in average 30% better than HIP and the efficiency was better in average by 1.02%. We performed a two-sided independent samples t-test in order to see the mean difference of efficiency between each policy. Table 4.4 and Table 4.5 compare RL with HIP and No SendFAB in respect to machine efficiency. The result tells RL and HIP have statistically significant mean difference of machine efficiency. RL and No SendFAB, however, had similar machine efficiency results. Considering the fact that Send FAB deteriorates its own machine efficiency by using other lines' machines, it appeared that RL policy prevents this effect by improving WIP balancing among machines.

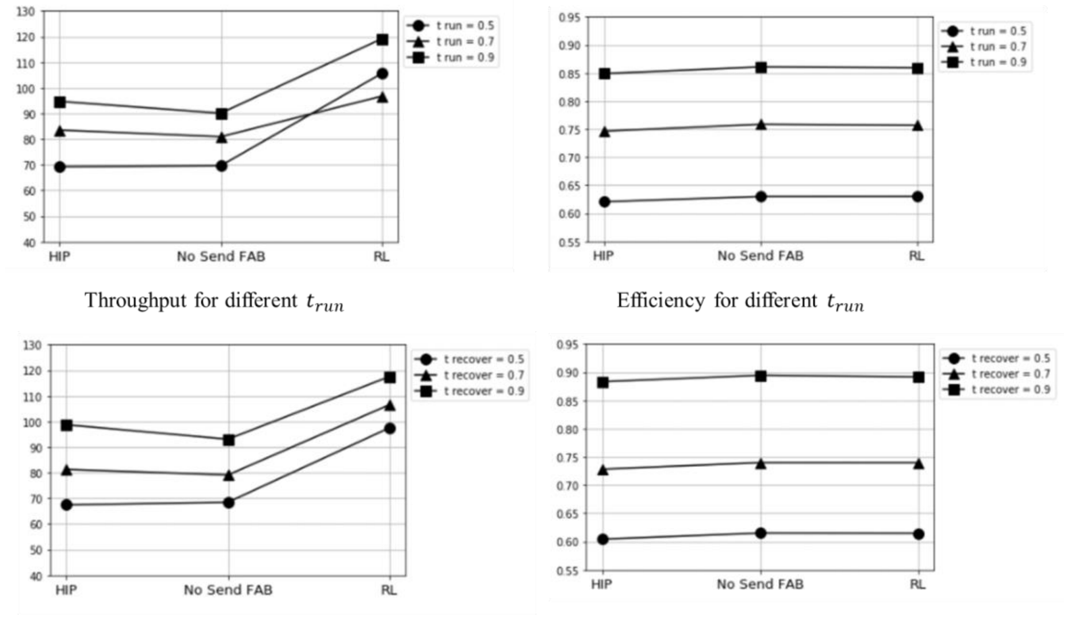
Table 4.4: Average Efficiency comparison between RL and HIP by different machine disruption scenarios

t_{run}	$t_{recover}$	Efficiency(%)				
		RL	*HIP*	*Diff*	t	p
0.5	0.5	45.36	44.56	0.8	6.56	0.000
	0.7	60.69	59.73	0.96	4.01	0.000
	0.9	82.95	81.87	1.08	6.46	0.000
0.7	0.5	61.98	60.77	1.21	4.42	0.000
	0.7	75.15	73.9	1.25	11.59	0.000
	0.9	89.98	89.16	0.82	6.01	0.000
0.9	0.5	77.07	75.95	1.12	4.12	0.000
	0.7	86.11	84.83	1.28	7.06	0.000
	0.9	94.57	93.88	0.69	18.07	0.000

Table 4.5: Average Efficiency comparison between RL and No Send FAB by different machine disruption scenarios

t_{run}	$t_{recover}$	Efficiency(%)				
		RL	*No SendFAB*	*Diff*	t	p
0.5	0.5	45.36	45.27	0.09	-0.07	0.950
	0.7	60.69	60.72	-0.03	-0.14	0.890
	0.9	82.95	82.99	-0.04	-0.28	0.780
0.7	0.5	61.98	62.15	-0.17	-0.66	0.510
	0.7	75.15	75.03	0.12	-0.45	0.650
	0.9	89.98	90.39	-0.41	-9.87	0.000
0.9	0.5	77.07	77.07	0	0.008	0.990
	0.7	86.11	86.16	-0.05	-0.32	0.750
	0.9	94.57	94.95	-0.38	-6.84	0.000

It showed that the throughput improvement was higher when the machine was more unstable (lower values of $t_{run}, t_{recover}$). The average throughput by different t_{run}, $t_{recover}$ groups are shown in Figure 4.1.



Throughput for different t_{run}

Efficiency for different t_{run}

Throughput for different $t_{recover}$

Efficiency for different $t_{recover}$

Figure 4.1: Performance by $t_{run}, t_{recover}$

We have also discovered that the performance improvement gets bigger when the machine is in a more unstable configuration. The average performance improvement of throughput is show in Figure 4.2. RL was more beneficial as the machine became more unstable.

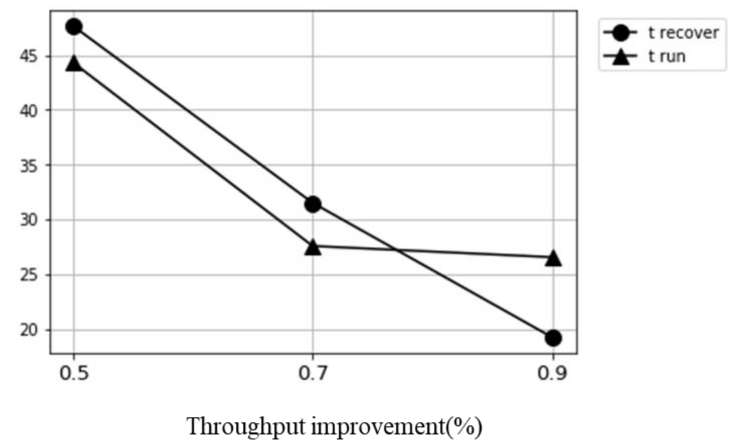


Figure 4.2: Throughput improvement by different levels of $t_{run}, t_{recover}$

The results for applying different machine utilization is shown in Figure 4.3. The result shows that when machine utilization increases the performance relative to policy without Send FAB increases. This indicates that HIP harms the performance and our policy improves the performance when the layer-machine relation gets more complicated.

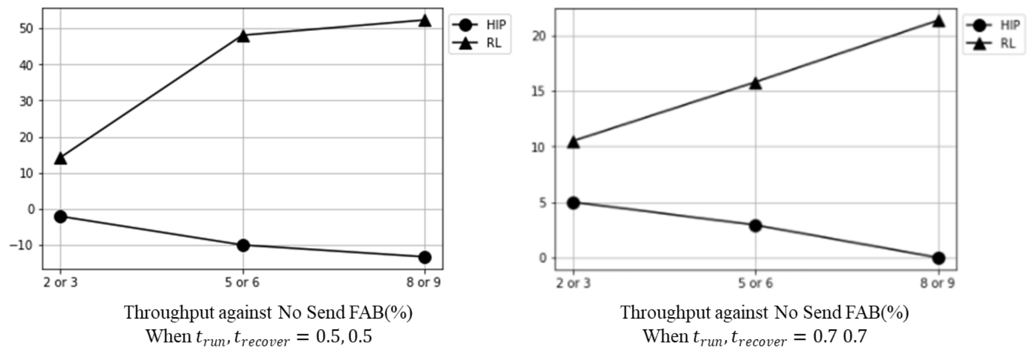


Figure 4.3: Throughput against No Send FAB by different levels of machine utilization

Chapter 5

Conclusions

Through experiments upon different test instances, we have seen that a new policy rather than a decision policy based on human experience is needed. We also checked that the more complex the system is, and the more unstable the machines are leads to a greater performance improvement when using our policy. This indicates that in a real FAB environment which is much more complex with more components to consider, would be able to benefit from applying our framework.

Our thesis provides a guideline for applying Reinforcement Learning to the Send FAB problem. It can be easily applied for any FAB environment. A FAB production manager can apply their own FAB scheduler and machine disruption model and build their own FAB environment.

Our FAB environment, however, assumes that the offering line receives all the required quantity unless it exceeds a limit. The research on the Send FAB problem can be expanded by modeling several lines. Our thesis assumes that the offering line of Send FAB accepts the whole requested amount unless it is below the maximum quantity available. In reality, however, the managers of each line negotiate the amount that can be processed. In other words, the offering line checks the capacity status of their own line and decides an amount that would benefit in machine efficiency while not harming their other production performances. Therefore, modeling several lines together with their own bottleneck layers could help develop a full automation on Send FAB contracts without any negotiation between line managers.

Bibliography

[1] Lin, Y.H. and Lee, C.E., 2001. *A total standard WIP estimation method for wafer fabrication*, European Journal of Operational Research, 131(1), 78-94.

[2] Lou, S.X.C. and Kager, P.W., 1989. *A robust production control policy for VLSI wafer fabrication*, IEEE transactions on semiconductor manufacturing, 2(4), 159-164.

[3] Toba, H., Izumi H., Hatada, H., Chikushima, T., 2005. *Dynamic Load Balancing Among Multiple Fabrication Lines Through Estimation of Minimum Inter-Operation Time,* IEEE transactions on semiconductor manufacturing, 18(1), 202-213.

[4] Chung, J. and Jang, J., 2009. *A WIP Balancing Procedure for Throughput Maximization in Semiconductor Fabrication*, IEEE transactions on semiconductor manufacturing, 22(3), 381-390.

[5] Urayama, K., Fu, M.C., and Marcus, S.T.,2015. *Simulation-Based Work Load and Job Release Control for Semiconductor Manufacturing*, IEEE 54th Conference on Decision and Control, Osaka, Japan, 15-18 Dec.

[6] Lee, W.J., Kim, B.H., Ko, K. and Shin, H., 2019. *Simulation based multi-objective FAB scheduling by using reinforcement learning*, Proceedings of the 2019 Winter Simulation Conference, National Harbor, MD, USA, 8-11 Dec.

[7] Kim, H. Lim, D.E., and Lee, S. 2020. *Deep Learning-Based dynamic scheduling for semiconductor manufacturing with high uncertainty of Automated Material Handling System capability*, IEEE transactions on semiconductor manufacturing, 33(1), 13-22.

[8] Mnih, V. Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, L., Wlerstra, D., and Riedmiller, M., 2013. *Playing Atari with Deep Reinforcement Learning*,. arXiv preprint arXiv: 1312 5602.

국문초록

반도체공장은 설비 용량의 불안정성 때문에 초기 계획하여 할당된 설비 용량에 비해 일시적으로 생산 용량이 부족해지는 경우가 발생한다. 이를 대응하기 위해 생산 담당자들은 다른 라인에 호환가능한 설비를 공유하는 것을 요청하는데, 가능한 많은 양의 WIP에 대한 요청을 한다. 이러한 의사결정은 공정이 순차적으로 연결된 점 때문에 라인 전체 측면에서는 오히려 WIP Balancing을 악화시킬 수 있다. 특히 해당 공정군이 병목공정군인 경우 더 문제가 된다. 따라서 본 연구에서는 병목공정군을 중심으로 한 WIP Balancing scheduler를 이용하여 FAB simulator를 만든 뒤 이러한 환경속에서 강화학습 알고리즘으로 학습한 생산 용량 확장 정책을 제안한다. 이러한 정책은 throughput, machine efficiency 측면에서 사람의 의사결정을 모방한 정책보다 좋은 성과를 보였다.

주요어: 생산용량관리, 강화학습, 산업공학

학번: 2019-23474