A Reinforcement Learning Approach to Robust Scheduling of Semiconductor Manufacturing Facilities

IB Park et al.

IEEE Transactions on Automation Science and Engineering(2021)

Speaker : Min Joon Kim

Jan 20th, 2022

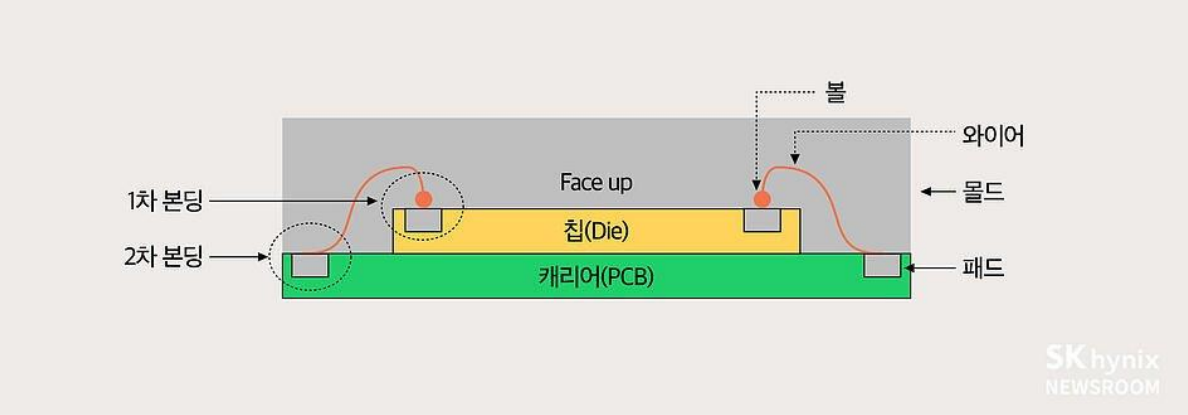
Background

▪ Packaging process

- Packaging is the one of the semiconductor manufacturing process.

  Bonding : Connecting eletrical signals

  Molding : Protect the chips



(Source : SK hynics newroom)

- Many operations in the bonding process.(needed to be scheduled)
- Setup time will be needed.
- Limitation of computation time

Background

▪ Job shop scheduling

- **Job shop scheduling problem** is an optimization problem.
- In a general job scheduling problem, there are n **jobs** which need to be scheduled on m **machines** while trying to **minimize the makespan.**
- Each job consists of a set of **operations** O_1**,** O_2**, …,** O_n which need to be processed in a specific order.

✓ **(Objective)** Minimizing makespan, tardiness, idle time, …

✓ **(Decision)** 1) Determining which operation to process next

2) Determining which machine to assign

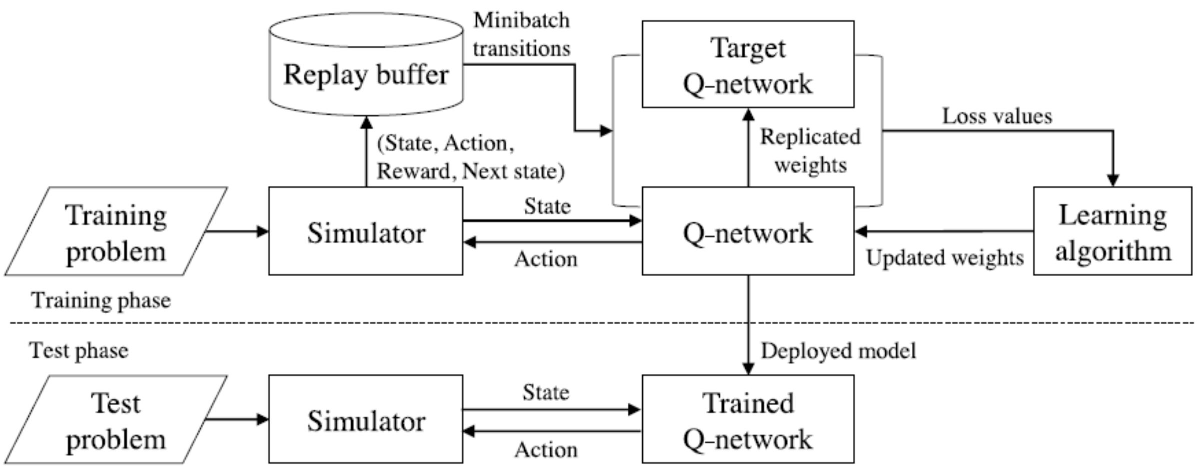
✓ **(Constraints)** 1) Each operation can only be processed on one machine at a time.

2) Each machine can only perform one operation at a time

⋮

Overview

- **Reinforcement learning to solve Job-Shop Scheduling**
  - They want to solve the job shop scheduling problem with reinforcement learning.
  - They applied Deep Q-Network to solve the problem.
  - The performance was greater than metaheuristic and rule-based methods.



[Proposed framework]

Related works

Subject	Author	Paper
Meta-heuristic	Shen (2018)	Solving the flexible job shop scheduling problem with sequence-dependent setup times
	Chung (2014)	Setup change scheduling for semiconductor packaging facilities using a genetic algorithm with an operator recommender
	Defersha (2010)	A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups
Rule-based	Jia (2018)	A performance analysis of dispatch rules for semiconductor assembly & test operations,"
	Wang (2007)	A lot dispatching strategy integrating WIP management and wafer start control

- Metaheuristic method need **a lot of computations** to find a near-optimal schedule.
- Rule-based method **cannot gurantee the high-quality solution.**

Problem definition

▪ Scheduling problem for die attach and wire bonding stages

- There are jobs that belong to one of $\boldsymbol{N_J}$ **job types.**
- Jobs are processed by $\boldsymbol{N_M}$ **machines** of which the $l\,th$ machine is denoted as M_l.
- Let **P($\boldsymbol{J_J}$) be the total number of jobs** of J_J to be scheduled, indicating the production requirement of J_J
- A job of J_J consists of **N($\boldsymbol{J_J}$) operations** that need to be processed in the predetermined order, $O_{j,1}, \ldots, O_{j,N(J_J)}$.
- The kth operation type of J_J is represented as $O_{j,k}$. (the number of operation types N_O)
- **(Setup time constraint)** If an operation of $O_{j',k'}$ is assigned to the machine whose setup type is $O_{j,k}$,

  the operation of $O_{j',k'}$ can be processed at the machine only after the **setup change time, as $\boldsymbol{\sigma_{j,k,j',k'}}$**
- **The objective function** is to **minimize the makespan, $\boldsymbol{C_{max}}$ which is the completion time of the last finished operation**

Example

- **Scheduling problem for die attach and wire bonding stages**

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

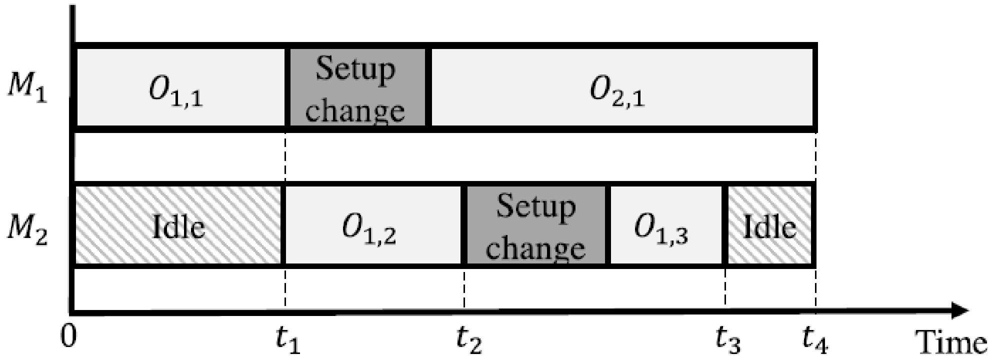


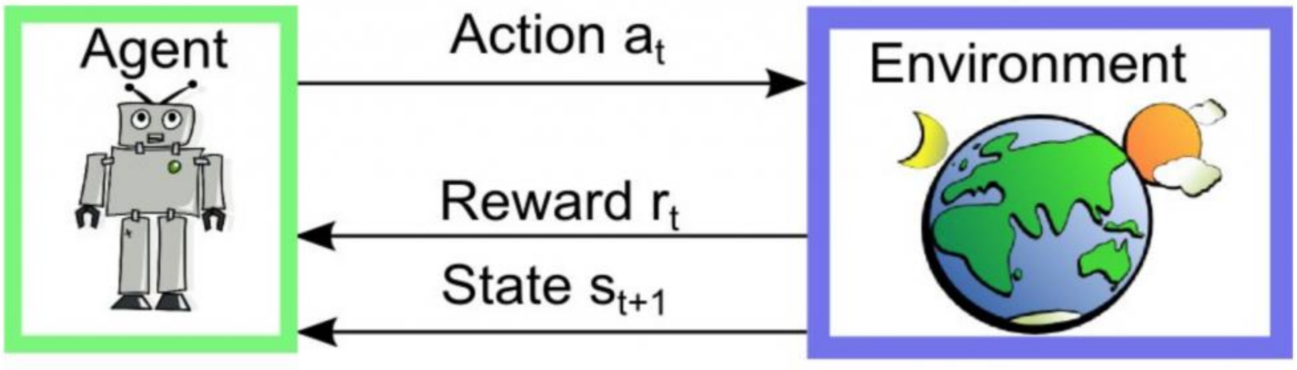
Fig. 3. Schedule obtained from the example.

Minimize makespan

Methodology

- **MDP(Markov Decision Process)**
  - MDP is **a tuple** $< \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{\gamma} >$, where ths **state space S** and **action space A**.
  - The **transition probability** $\boldsymbol{P}: \mathrm{SxAxS} \rightarrow [0, 1]$ represents the probability of the next state given the current state and action.
  - The **reward function** $\boldsymbol{R}: \mathrm{SxAxS} \rightarrow [\mathrm{r_{min}}, \mathrm{r_{max}}]$
  - **Discount factor** $\boldsymbol{\gamma} \rightarrow [0, 1]$
  - RL considers a sequential decision making problem as MDP and solve the Bellman equation by iterative learning.
  - So, the objective of RL agent is **to learn a policy that maximizes the expected cumulative sum of rewards.**

Methodology



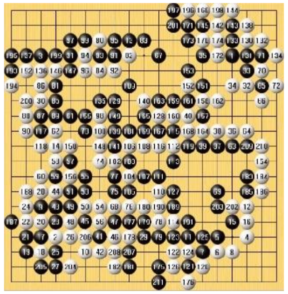
Learning through trial and error!

- **AlphaGo**

  State : Board, score

  Action : Drop the stone

  Reward : win(+) / lose(-)

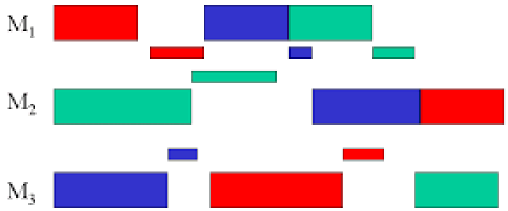


- **Scheduling**

  State : Production environment

  Action : Assign the operation to the machine

  Reward : -(Makespan)



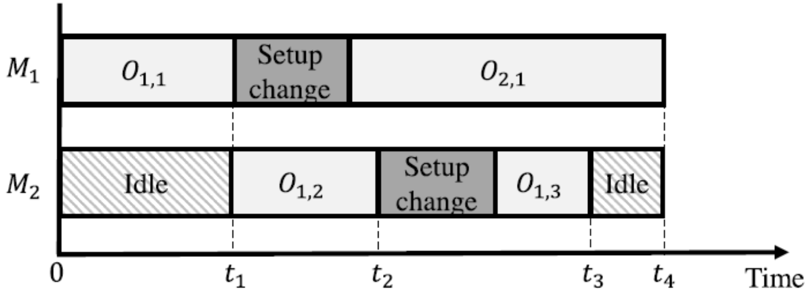
State, Action, Reward

▪ State

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

▪ Action

- Assigning an operation(Dimension $N_O + 1$)
  - ✓ Including do-nothing action as δ_0



(Caculation example)

Machine1 $(a_2 = O_{2,1})$: $-(t4 - t1 - p_{2,1})$

▪ Reward

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

Indicating setup or idle time

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

Maximize sum of all rewards(R)
is equivalent to minimize C_{max}

Q-Network

- **State-action value(Q) function approximation**

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$ (Bellman equation)

- Q value is the cumulative reward when we are at state s and do action a
- In dynamic programming, we can get the optimal policy through Q-value table.

Game Board:

Current state (s): 000 010

Q Table: γ = 0.95

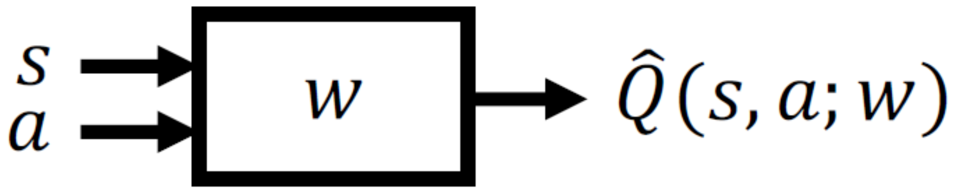
action	000 100	000 010	000 001	100 000	010 000	001 000
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

→ state

- However, if the dimension of state and action is large, the curse of dimensionality problem occurs.

Q-Network

- **State-action value(Q) function approximation**
  - To overcome curse of dimensionality, we can use neural network to approximate the Q-value



ε −greedy exploration

▪ ε-greedy policy

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability 1-ε choose the greedy action(choose the best action, exploitation)
- With probability ε choose an action at random(exploration)

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

DQN

- **Experience replay and fixed Q-targets**
  - To overcome correlations between samples, experience replay was suggested.
  - For learning stability, seperate two Q-Network idea was suggested.(fixed Q-targets)

✓ Take action a_t according to ε-greedy policy

✓ Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D

✓ Sample random mini-batch of transitions (s, a, r, s′) from D

✓ Compute Q-learning targets old, fixed parameters w^-

✓ Optimize MSE(Mean Squared Error) between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

↓ fixed target network

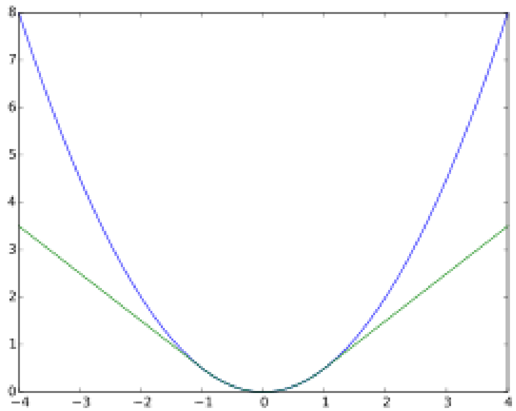
DQN

- **Loss function**
  - They applied Huber loss instead of MSE error.
  - **(Huber Loss)** Quadratic for small difference and linear for large difference.

$$q_u = Q\left(s_u, a_u; \theta\right)$$
$$y_u = r_u + \gamma \mathbb{1}_F(s_{u+1}) \max_{a'} \hat{Q}(s_{u+1}, a'; \hat{\theta})$$

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

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

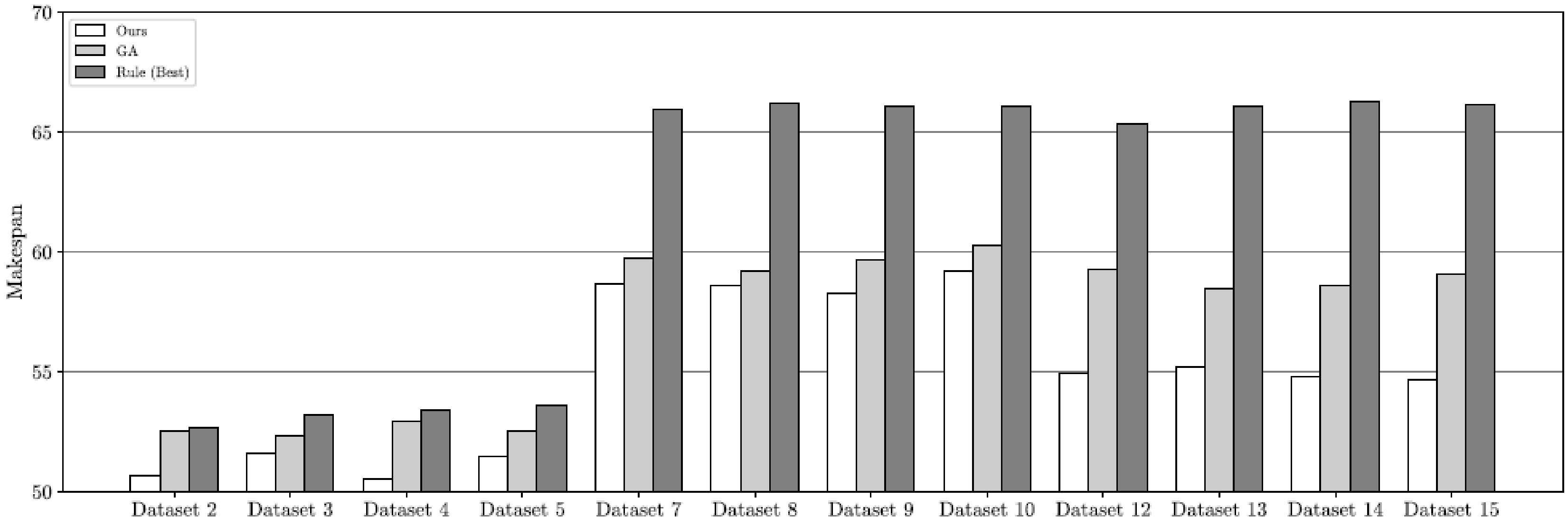


Blue : MSE loss
Green : Huber loss

Experiments

▪ Comparing with other methods

- Proposed method was outperformed the other methods in every given dataset.

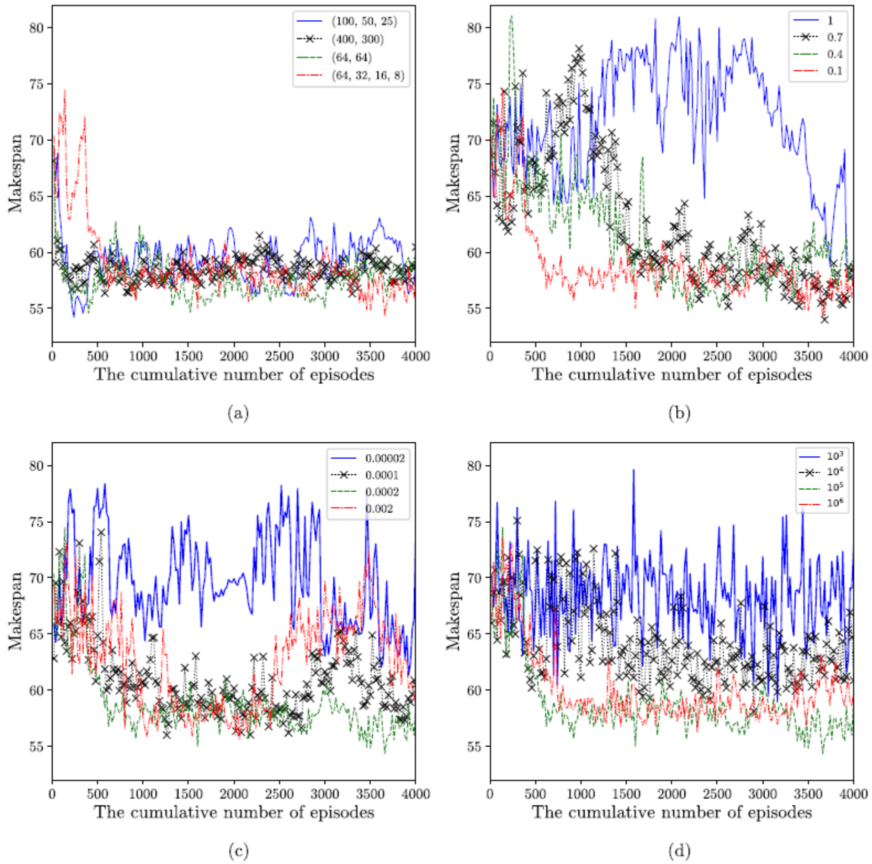


*Rule based
shortest setup time (SSU), shortest sum of processing time and setup time (SPTSSU)
most operation remaining (MOR), most work remaining (MWR), shortest processing time (SPT)

Experiments

▪ Sensitivity analysis

- To investigate the sensitivity of hyperparameters, they compared the makespan for each result.



(a) Network structure(Q-network)

(b) Epsilon(ε-greedy policy)

(c) Learning rate

(d) Replay buffer size(experience replay)

(a) 64, 32, 16, 8 nodes for hidden layers

(b) epsilon = 0.1

(c) 0.0002

(d) 10^5

Experiments

- **Computation time**

  - Compared to GA(Genetic Algorithm), the computation time was decreased.(about 100 times)
  - The computation time of proposed method was less than 120s.

Dataset No.	Best Rule	Ours	GA
2	6.84	17.75	1705.75
3	13.48	35.12	3881.79
4	22.20	59.20	5561.81
5	33.31	88.08	8938.19
7	7.78	19.84	1734.74
8	15.22	39.76	3968.45
9	25.15	69.36	5702.72
10	37.779	100.04	8991.76
12	8.19	19.66	1765.32
13	16.20	38.67	4070.17
14	26.19	65.46	5919.82
15	39.47	98.85	9015.66

Conclusion

- **Conclusion**

  - They proposed reinforcement learning approach to scheduling of semiconductor manufacturing
  - The performance was greater than metaheuristic and rule-based method.

- **Contribution**

  - Machine setup status was considered in this paper.
  - Applying reinforcement learning in semiconductor manufacturing scheduling.

- **Evaluation**

  - The overall structure of this paper was great and I thought the comparing with the optimal scheduling should be needed.