IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 April 2023, accepted 11 May 2023, date of publication 18 May 2023, date of current version 31 May 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3277529*

RESEARCH ARTICLE

# A Generation and Repair Approach to Scheduling Semiconductor Packaging Facilities Using Case-Based Reasoning

**IN-BEOM PARK[1], JAESEOK HUH[2], AND JONGHUN PARK[3]**
[1]Department of Industrial and Management Engineering, Myongji University, Yongin 17058, South Korea
[2]Department of Business Administration, Tech University of Korea, Siheung-si, Gyeonggi-do 15073, South Korea
[3]Department of Industrial Engineering, Institute for Industrial Systems Innovation, Seoul National University, Seoul 08826, South Korea

Corresponding author: Jaeseok Huh (jshuh@tukorea.ac.kr)

This work was supported by the 2021 Research Fund of Myongji University.

**ABSTRACT** As the demand for multi-chip products with high capacity and small size increases, semiconductor packaging facilities have been faced with complicated constraints such as re-entrant flows, sequence dependent setups, and alternative routes, which leads to difficulties in scheduling semiconductor manufacturing operations. Furthermore, due to the frequent variations in the relative importance between objectives as well as the variabilities in initial setup status, available machines, and production requirements, practitioners are obliged to obtain a schedule within a short amount of computation time. In this paper, we propose a novel two-phase framework that aims to quickly produce a schedule of semiconductor packaging facilities by using case-based reasoning for minimizing the weighted sum of machine loss time and waiting time of jobs. Specifically, in the case generation phase, a case database is constructed by solving case scheduling problems using an existing solver. The case reasoning phase is responsible for repairing operation type sequences in the cases to produce a schedule for an unseen scheduling problem whose production requirements, available machines, initial setup status, and weight between performance measures are different from those of cases. The extensive experimental results demonstrated that the proposed approach requires a short computation time similar to the rule-based methods while maintaining the quality of the schedules comparable to that of the existing metaheuristics.

**INDEX TERMS** Semiconductor packaging facilities, flexible job shop scheduling, case-based reasoning, sequence dependent setups, schedule repair.

## I. INTRODUCTION

As semiconductor industries become automated, scheduling is one of the key decision-making problems in the semiconductor manufacturing systems [1]. The goal of the scheduling is to find an operation sequence for each machine to optimize specific objectives. Recently, semiconductor manufacturers have been concentrating on producing multi-chip products (MCPs) to satisfy customer demands for high capacity and small devices [2]. This forces semiconductor packaging facilities to accompany complicated constraints such as re-entrant flows, sequence dependent setups, and alternative routes, which leads to difficulties in obtaining schedules while achieving multiple objectives at the same time.

In the contemporary semiconductor packaging facilities, maximizing machine utilization and minimizing the flow time jobs are very important objectives [3], [4]. The former improves the production efficiency and the latter enables manufacturers to flexibly respond to the fluctuations in demand of semiconductor markets [5]. Unfortunately, it is challenging to optimize these two performance measures at the same time [6], [7]. Specifically, machine utilization can be maximized by providing a large amount of work-in-process (WIP). However, this increases the waiting time for jobs in the

The associate editor coordinating the review of this manuscript and approving it for publication was Yingxiang Liu.

next operations, which leads to the increase in the flow time of jobs [8]. On the contrary, if there does not exist a sufficient amount of WIP in order to reduce the flow time, machines in semiconductor packaging facilities become frequently idle, which results in the reduction of the machine utilization [2].

Meanwhile, the importance of between machine utilization and flow time of jobs depends on the policy of enterprises or the opinion of manufacturing line managers [9]. If the manufacturers want to focus on reducing the depreciation cost of machines, the preference for increasing machine utilization is greater than that of decreasing the flow time. In the case of the enterprise that needs to quickly satisfy the various production requirements of customers, minimizing the flow time may be appropriate to achieve the goal of the enterprise. Therefore, in the real-world semiconductor packaging facilities, it is a common practice to optimize the weighted sum of the two objectives according to the given situation [10].

Furthermore, due to the variabilities in initial setup status, available machines, and production requirements, adherence to the generated schedule is not possible. For instance, a machine failure is one of the events that occur frequently in manufacturing lines, which requires modifying a schedule to accommodate the changed capacity of manufacturing lines. Therefore, in order to manage such variabilities, semiconductor manufacturers are responsible for performing scheduling decisions in a short time, which makes solving a scheduling problem more challenging than ever [11].

Motivated by the viewpoints above, in this paper, we propose a generation and repair approach for scheduling the semiconductor packaging facilities. The proposed framework is devised to obtain a schedule that aims to minimize the weighted sum of the loss time of machines and the waiting time of jobs. The contributions of this paper are three-fold.

- In order to quickly generate schedules, we propose a novel framework by utilizing case-based reasoning (CBR) that attempts to solve a new scheduling problem based on past experience called a case which includes information related to previous problems and corresponding solutions [12].
- The case repair and machine allocation algorithms are developed to solve an unseen scheduling problem whose production requirements, available machines, initial setup status, and weight between performance measures are different from those of cases. Due to the reuse of the schedules obtained in advance, case reasoning phase efficiently fixes retrieved cases to a given scheduling problem by modifying them, making it capable of being performed in a timely manner.
- To verify the robustness and efficiency of the proposed framework, experiments were conducted on six datasets that imitate the real-world semiconductor packaging lines. The proposed framework was able to obtain a schedule whose quality is comparable to that of the considered meta-heuristics while requiring a short computation time.

This paper is organized as follows. In the next section, we review the related work. Section III presents the problem definition and the notations that appear in this paper. The proposed framework, including case generation and reasoning steps, is described in Section IV. Section V reports datasets used in the experiments and performance comparison results of the proposed scheduling method. Finally, we conclude this work in Section VI.

## II. RELATED WORK

### A. A FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SETUPS

The scheduling problem considered in this paper can be modeled as a flexible job shop problem (FJSP), which has been proven to be an NP-hard. This implies that there is no polynomial-time algorithm that can solve the problem efficiently [13]. For solving the scheduling problem while reducing the computation time and implementation efforts, rule-based methods have been still widely adopted [14], [15], [16]. However, it is difficult to devise a single well-performing rule whenever new scheduling problems are given.

To overcome this drawback of rule-based methods, the evolutionary method based on genetic programming (GP) was proposed to automatically evolve dispatching rules for FJSP. GP was utilized to generate composite dispatching rules that outperformed single dispatching rules in terms of total tardiness [17]. Zhang et al. [18] proposed stochastic dispatching rules to make a best decision according to the condition of FJSP. The authors represent decisions using probability distribution, including multinomial distribution and the distribution generated by softmax function. Despite of these efforts, due to the lack of the capability to search solution space, these methods are not likely to be acceptable for solving large-scale FJSPs.

In order to obtain schedules through exploring the solution space enough, the metaheuristics based methods have been investigated [19], [20]. Chung et al. [21] proposed a genetic algorithm-based sequence optimizer (GASO) for FJSP with sequence dependent setup time. Their framework successfully maximizes the machine utilization and minimizes the total setup time at the same time. Defersha and Rooyani [22] developed a two-stage genetic algorithm (TSGA) where the first stage is designed to generate good initial solutions in a short time and the next stage is responsible for improving the quality of the generated solutions by searching solution space that might have been excluded from the former stage.

A hybrid metaheuristic algorithm combining GA with Tabu search (TS) was developed to solve FJSPs that involve job lag times [23]. In this method, GA is responsible for the global diversification, and TS conducts an exhaustive local search. Fan et al. [24] improved GA by employing iterated local search to minimize total weighted tardiness of FJSPs considering reconfigurable manufacturing systems. Although they obtain a high-quality schedule, the main drawbacks of these attempts are owing to their large amount of computation

time, which results in difficulties in applying them to real time scheduling environment where the variabilities exist [25].

More recently, there has been considerable interest in using machine learning-based methods for solving FJSP with sequence dependent setups [1], [26]. One of the advantages of this line of approach is that it requires a short amount of computation time to yield a schedule using the trained model. Especially, reinforcement learning (RL) based scheduling methods have become at the center of attention in both academia and industry since an RL is well known to efficiently find a well-performed policy by exploitation and exploration of solution space [11], [25], [27], [28]. Specifically, an agent is trained to conduct actions that can maximize the expectation of cumulative rewards using state and reward obtained from an environment.

A Q-learning based scheduler using a shared neural network (NN) was presented to minimize the makespan of semiconductor packaging facilities [11]. To address the variation in the number of machines, an agent is designed to conduct an action in a decentralized manner while learning a centralized policy by the shared NN. Park and Park [27] extended the work in [11] by developing the new action design which includes four continuous features of a job-machine pair. This enables the dimensionality of the action space to be kept constant regardless the number of machines, jobs, and operation types. Although such RL-based approaches successfully find a high-quality schedule, however, the agent is required to be re-trained when the design of reward changes due to the changes in the objective function.

To the best of our knowledge, the existing studies might be difficult to respond to the variabilities in the initial setup status, available machines, production requirements, as well as the objective function. Meanwhile, the goal of this work focus on quickly generating a schedule in for real-world semiconductor packaging facilities while accommodating such variabilities.

### B. CASE-BASED REASONING

CBR is a knowledge reasoning method that utilizes the past experience of similar problems to solve a current problem [29]. In CBR, the current problem and the problem encountered in the past are regarded to as the target and the historical cases, respectively. The strategy of CBR is to retrieve the historical cases most related to the target case and modify retrieved cases into a solution for a given problem [30]. The bulk of research has been conducted toward the application of CBR to various domains, including scheduling [31], [32], warehouse system [33], classification [34], [35], nurse rostering [36], and health care [37].

Huh et al. [33] proposed a CBR-based algorithm for determining travel routes of large-scale warehouses. They were successful to yield travel routes with a short travel time without exhaustively searching for all possible combination in the whole solution space. A number of research for classification approach using CBR have been actively examined. The classification algorithm based on CBR was developed to cover mixed dataset that contains both categorical and numerical data [35]. Another line of research presented a CBR system that aims to detect the faults on the injection molding machine [34].

There were relatively few studies on solving scheduling problems in manufacturing systems using CBR compared to ones that applied CBR to other fields. Chang et al. [38] developed a two-stage CBR scheduling framework to obtain a solution for dynamic scheduling of complex steel-making process. Chang et al. [39] presented a case-injected GA method to solve single machine sequencing problems, where GA is responsible for constructing the initial casebase. For solving dynamic scheduling problem, another research attempted to employ CBR as a learning module to automatically select the parameters of metaheuristics [40].

More pertinently, Lim et al. [32] employed CBR to a scheduling problem similar to the one that we address in this paper. The authors defined a schedule as a sequence of cases each of which includes the states of a manufacturing line and the decisions conducted at the corresponding states. Given an input state of a scheduling problem, only feasible cases are retrieved by filtering process and then the one most similar to the target case is converted into a solution by an encoding algorithm. This results in a successful reduction of the computation time of their method while requiring little sacrifice in the machine utilization. However, both the changes in the number of machines and waiting time of jobs are not considered in their method.

## III. PROBLEM DEFINITION

This section introduces a scheduling problem for the semiconductor packaging facilities considered in this paper. The scheduling problem is formulated as FJSP with sequence dependent setup time, and the mathematical formulations can be found in [21]. The descriptions of the scheduling problem with the notations and assumptions are listed as follows.

- There are $N_J$ jobs and $N_M$ machines, where the $l^{th}$ job is denoted as $J_l$ and the $k^{th}$ machine is denoted as $M_k$. Each job belongs to one of $N_{JT}$ job types, where the $i^{th}$ job type is denoted as $JT_i$. For each of the $i^{th}$ job type, $P_i$ jobs are required to be scheduled, which represents the production requirements of the $i^{th}$ job type. As a result, the following equation holds for the total number of jobs to be scheduled:

$$N_J = \sum_{i=1}^{N_{JT}} P_i. \tag{1}$$

- The job of type $JT_i$ consists of $N(O_i)$ operations that are required to be performed in a predetermined order. Let $O_{i,j}$ be the $j^{\text{th}}$ operation type of a job whose job type is $JT_i$. In addition, the processing time of an operation of type $O_{i,j}$ is defined as $p_{i,j}$. Then, $p_{tot}$, indicating the total sum of the processing time in a schedule, is computed as

below:

$$p_{tot} = \sum_{i=1}^{N_J} \sum_{j=1}^{N(O_i)} p_{i,j} \tag{2}$$

Note that $p_{tot}$ is the same regardless of the scheduling result. Furthermore, we denote $A_k$ as a set of operations that can be performed on $M_k$. To perform an operation of type $O_{i,j}$ on $M_k$, $O_{i,j}$ must belong to $A_k$, and $M_k$ is required to be set up for type $O_{i,j}$. We remark that an operation type also indicates the setup status for a machine. If the setup type of $M_k$ was $O_{i',j'}$, the setup time required for performing an operation of type $O_{i,j}$ on $M_k$ is $\sigma_{i',j',i,j}$. Moreover, at the start of the scheduling, each $M_k$ has been set up for setup type $Z_k$, indicating the initial setup status of $M_k$.

- In this paper, the following assumptions are introduced. At the start of the scheduling, jobs and machines are waiting to be processed and being idle, respectively. An operation must be performed on only one machine at a time, and a machine can process only one operation at a time. Once an operation starts to perform on a machine, it must be finished without interruption. Finally, the moving time of each job is not considered.
- The two performance measures considered in this paper are the average loss time of machines and the average waiting time of jobs, called $ALT$ and $AWT$, respectively. Specifically, $ALT$ is defined as below:

$$ALT = \frac{\sum_{k=1}^{N_M} f_k - p_{tot}}{N_M} \tag{3}$$

where $f_k$ is the finish time of the last operation on $M_k$. We remark that $ALT$ is equivalent to the average sum of idle and setup time since the $f_k$ is computed by summing the processing, setup, and idle time incurred on $M_k$. On the other hand, $AWT$ is defined as below:

$$AWT = \frac{\sum_{l=1}^{N_J} \tau_l - p_{tot} - \sigma_{tot}}{N_J} \tag{4}$$

where $\tau_l$ indicates the completion time of $J_l$, and $\sigma_{tot}$ is the total sum of setup time incurred in a schedule. Finally, the objective function of the scheduling problem, denoted as $F_w$, is to minimize the weighted sum of $ALT$ and $AWT$, which is defined as follows:

$$F_w = w \cdot ALT + (1 - w) \cdot AWT \tag{5}$$

where $w$ indicates the relative importance between $ALT$ and $AWT$, which is determined by practitioners at the beginning of the scheduling.

## IV. PROPOSED METHOD

### A. OVERVIEW

In this section, we describe the overall framework of the proposed scheduling method. The framework consists of two phases, which are case generation and repair, as depicted in Fig. 1. In the case generation phase, case scheduling problems are solved by using the solver to build the case database. After each case problem has been solved, a case is stored in the case database. A case $c$ is defined as a quintuple of $h$, $\mathcal{P}$, $S_O$, $ALT$, and $AWT$, where $h$, $\mathcal{P}$ and $S_O$ indicate an index of a case scheduling problem, the production requirements of a case scheduling problem, and the operation type sequence of a schedule, respectively.

At the beginning of the case reasoning phase, dozens of schedules are retrieved from the case database. The retrieved schedules cannot be deployed for solving a test scheduling problem since the production requirements, the number of machines, the initial setup status, and $w$ are different from those of the case problems. After the repair algorithm is executed with the retrieved schedules as inputs, the schedules are finally obtained by solving test schedule problems.

### B. CASE GENERATION

Generally, the case generation phase plays a critical role in a CBR-based method since the quality of the case database is related to the performance of a final solution. The case database $\mathcal{C}$ is defined as follows.

$$\mathcal{C} = \{c_e | e = 1, \ldots, |\mathcal{C}|\} \tag{6}$$

where $c_e$ and $|\mathcal{C}|$ indicate the $e^{th}$ case and the number of cases in $\mathcal{C}$, respectively. Since the case generation is conducted in an offline manner, it is possible to spend a large enough amount of time for building the case database. Based on the remarks above, we adopt TSGA [22] and GASO [21] to obtain high-quality schedules by solving case scheduling problems. The reason for choosing these two GAs is that, unlike other previous studies, they are capable of representing a solution that distinguishes between job types and jobs.

The production requirements, available machines, the initial setup status, and $w$ are varied during the case generation phase in order to cope with their variabilities of an unseen scheduling problem. Meanwhile, the other configurations of a scheduling problem such as job types, operation types, and alternative machines are fixed during the case generation and reasoning. The assumption behind this is that these configurations are rarely changed in the semiconductor packaging facilities [32].

### C. CASE REASONING FOR SCHEDULING

Algorithm 1 presents a case retrieval procedure for selecting a number of cases each of which belongs to $\mathcal{C}$. In line 1, $h$ for each case is inserted in $H$ that indicates the set of case indexes in $\mathcal{C}$. Afterwards, each case in $\mathcal{C}$ is sorted from lowest to highest based on $F_w$ (line 2). Let $\delta$ be the difference between the number of retrieved cases, denoted as $N_R$, and the size of $H$ (line 3). During lines 4–14, $N_R$ cases are retrieved from $\mathcal{C}$ and then stored in $\mathcal{R}$ under the condition that the case indexes of the retrieved ones do not overlap as much as possible. The rationale behind is that $F_w$ results yielded by repairing the cases retrieved from the same index are highly likely to be similar to each other. Lines 10–13 aim to additionally retrieve a case if $\delta$ is larger than 0. We remark

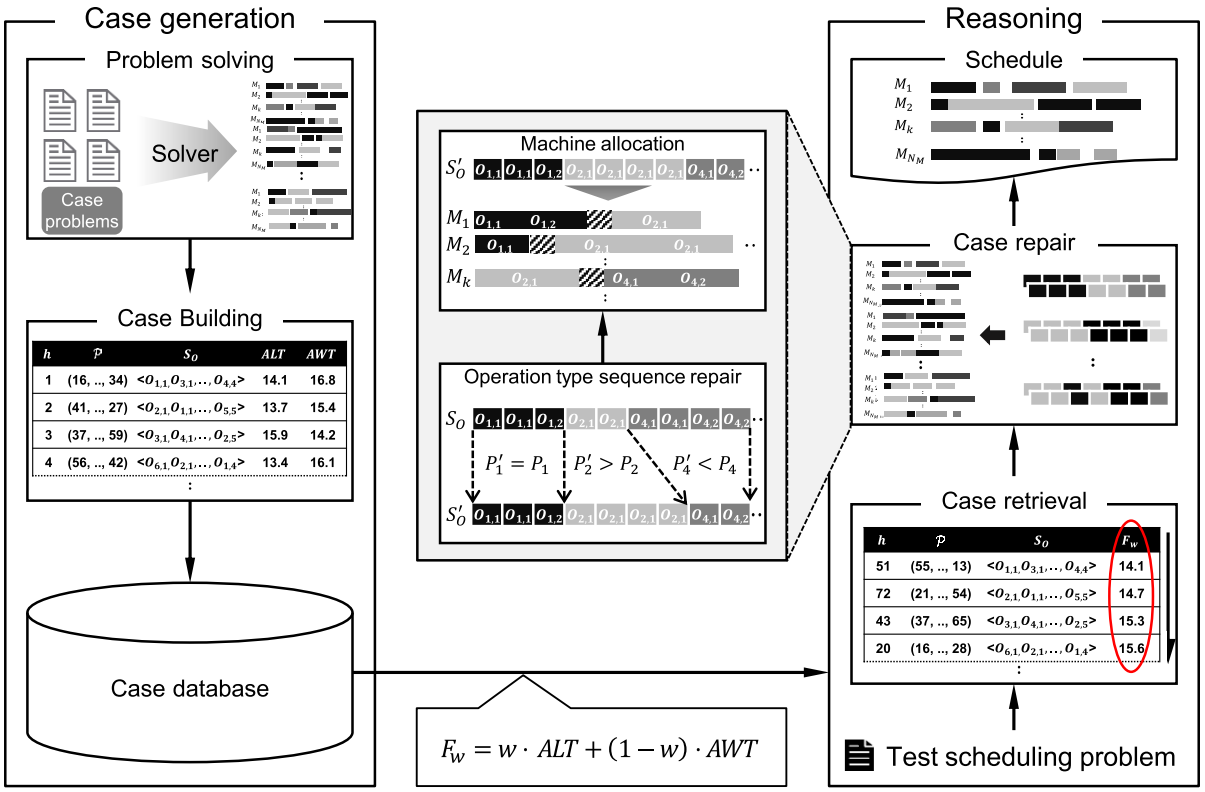


**FIGURE 1.** Proposed scheduling framework.

**Algorithm 1** Case Retrieval Procedure

**Input:** $\mathcal{C}$, $N_R$, and $w$
**Output:** $\mathcal{R}$

```
Initialize H from C
Sort c ∈ C in ascending order based on F_w
δ = N_R − |H|
for c ∈ C do
    while |R| < N_R do
        Obtain c = (h, P, AWT, ALT, S_O)
        if h ∈ H then
            R ← R ∪ {c}
            H ← H \ {h}
        else if δ > 0 then
            R ← R ∪ {c}
            δ ← δ − 1
        end if
    end while
end for
return R
```

**Algorithm 2** Repair Procedure of $S_O$

**Input:** $\mathcal{P}'$ and a case $c \in \mathcal{R}$
**Output:** $S'_O$

```
Set S'_O to be an empty sequence
Obtain S_O and P from c
Set d(O_{i,j}) = P'_i − P_i, ∀i = 1, ..., N_JT, ∀j = 1, ..., N(O_i)
for u = 1, ..., L do
    Obtain O_{i,j} = S_O[L − u + 1]
    if d(O_{i,j}) < 0 then
        d(O_{i,j}) ← d(O_{i,j}) + 1
    else
        Insert O_{i,j} into the first position of S'_O
        while d(O_{i,j}) > 0 do
            Insert O_{i,j} into the first position of S'_O
            d(O_{i,j}) ← d(O_{i,j}) − 1
        end while
    end if
end for
return S'_O
```

that the computation time for conducting the case retrieval procedure can be reduced since Algorithm 1 can be executed in an off-line manner for a variety of $w$s and large enough $N_R$.

After building $\mathcal{R}$, the operation type sequence of a test scheduling problem, denoted as $S'_O$ is obtained by conducting the case repair procedure. A case repair procedure is described in Algorithm 2 whose inputs are the production requirements of a test scheduling problem, denoted as $\mathcal{P}'$, and a case $c$ retrieved from $\mathcal{R}$.

**Algorithm 3** Machine Allocation Procedure by Using $S'_O$

**Input:** a test scheduling problem and $S'_O$
**Output:** Schedule
1: Set $\mathcal{Q}_{i,j}$ to be an empty sequence $\forall i, j$
2: Set $f_k = 0, \forall k$
3: **for** $u = 1, \ldots, L'$ **do**
4: Obtain $O_{i,j} = S'_O[u]$
5: Set $v_* = \infty$
6: **for** $k = 1, \ldots, N_M$ **do**
7: Obtain $O_{i',j'} = \sigma(M_k)$
8: **if** $j = 1$ **then**
9: Compute $v_k = \sigma_{i',j',i,j} + p_{i,j}$
10: **else**
11: Compute $v_k = \sigma_{i',j',i,j} + p_{i,j} + max(f_k, \mathcal{Q}_{i,j}[1])$
12: **end if**
13: **if** $v_* > v_k$ **then**
14: Set $O_* = O_{i,j}$, $M_* = M_k$, and $v_* = v_k$
15: **end if**
16: **end for**
17: Assign an operation of type $O_*$ to $M_*$
18: Set $f_k = v_*$
19: **if** $j \neq 1$ **then**
20: Delete $\mathcal{Q}_{i,j}[1]$
21: **end if**
22: Append $v_*$ to $\mathcal{Q}_{i,j+1}$
23: **end for**
24: **return** Schedule

**Algorithm 4** Overall Procedure for Solving a Test Scheduling Problem Given $\mathcal{R}$

**Input:** $\mathcal{R}$, $N_R$, $w$, and a test scheduling problem
**Output:** Schedule
1: Obtain $\mathcal{P}'$ from a test scheduling problem
2: **for** $c \in \mathcal{R}$ **do**
3: Obtain $S'_O$ by executing Algorithm 2
4: Generate a schedule using $S'_O$ by executing Algorithm 3
5: **end for**
6: Obtain a schedule whose $F_w$ is lowest among $N_R$ schedules
7: **return** Schedule

In line 1, $S'_O$ is set to be an empty sequence. After $S_O$ and $\mathcal{P}$ are obtained from $c$ in line 2, $d(O_{i,j})$ is set to be the difference of the production requirements between test and case scheduling problems (line 3). During lines 4–15, $S'_O$ is built by using $S_O$, $d(O_{i,j})$, and $S_O$, where the lengths of $S_O$ and $S'_O$ are denoted as $L$ and $L'$, respectively. Note that $L$ is equivalent to $\sum_{i=1}^{N_{JT}} P_i \times N(O_i)$. First, the last element of $S_O$, called $O_{i,j}$, is retrieved, where $L$ is the length of $S_O$ (line 5). If $P'_i$ is smaller than $P_i$, $d(O_{i,j})$ is increased by the value of 1 without inserting $O_{i,j}$ into $S'_O$ (lines 6 and 7). Otherwise, $O_{i,j}$ is inserted in front of $S'_O$ (line 9). Moreover, if $d(O_{i,j})$ is greater than 0, $O_{i,j}$ is additionally inserted while $d(O_{i,j})$ is decreased by the value of 1 (lines 11 and 12). This continues until $d(O_{i,j})$ becomes the value of 0 (line 10). The computational complexity of Algorithm 2 is equal to $O(d_{max}L)$, where $d_{max}$ is $\max_{i,j} d(O_{i,j})$.

Algorithm 3 presents a machine allocation procedure for obtaining a schedule given $S'_O$, which is motivated by [41]. However, the main parts of Algorithm 3 is newly designed by modifying the algorithm in [41] since the previous method can not be employed when obtaining a schedule consisting of multiple operations with the same operation type.

We denote a sequence that contains the completion time of an operation of type $O_{i,j}$ as $\mathcal{Q}_{i,j}$. In lines 1 and 2, $f_k$ and $\mathcal{Q}_{i,j}$ are set to be 0 and an empty sequence, respectively. After the $u^{th}$ element of $S'_O$ is obtained (line 4), the algorithm initializes $v_*$ that indicates the minimum completion time of an operation of the $u^{th}$ operation type in $S'_O$. Then, an operation of type $O_{i,j}$ is assigned to one of $N_M$ machines by executing lines 6–17.

In lines 6 and 7, after $M_k$ is selected, the setup status of $M_k$ is obtained. Lines 7–11 aim to calculate the completion time $v_k$ when an operation of type $O_{i,j}$ is allocated to $M_k$ by dividing the case whether $O_{i,j}$ is the first operation type of $JT_i$ or not. If $v_k$ is less than $v_*$, $O_*$, $M_*$, and $v_*$ are set to be $O_{i,j}$, $M_k$, and $v_k$, respectively (lines 13–15). After an operation of type $O_*$ is allocated on $M_*$ (line 17), $f_k$ is set to be $v_*$ (line 18). Finally, after the first element of $\mathcal{Q}_{i,j}$ is deleted when $j$ is not equal to the value of 1 (lines 19–21), $v_*$ is added to the last position of $\mathcal{Q}_{i,j}$ (line 22). By repeating lines 4–22 for each element in $S'_O$, all operations for a test scheduling problem are allocated to one of the machines. The computational complexity of Algorithm 3 is equal to $O(N_M L')$.

We summarize the overall procedure for solving a test scheduling problem, which is described in Algorithm 4. After $\mathcal{P}'$ is obtained from a test scheduling problem (line 1), lines 2–5 are repeated for each case that belongs to $\mathcal{R}$. Specifically, $N_R$ schedules are generated by performing Algorithms 2 and 3 to a test scheduling problem. Finally, the best schedule whose $F_w$ is lowest among $N_R$ schedules is obtained (line 6). We note that the computational complexity of Algorithm 4 is $O(N_R \cdot (N_M L' + d_{max}L))$.

## V. EXPERIMENTAL RESULTS

### A. DATASET

We prepared 6 datasets that simulated the die attach and wire bonding stages of the real-world semiconductor packaging facilities, which are well-known bottleneck stages. Table 1 shows the datasets used in the experiments. It can be observed that $N_{JT}$ and $N_O$ on D4 to D6 are larger than those of D1 to D3, respectively.

For each dataset, we generated 70 case scheduling problems by randomly varying the initial setup status. Specifically, the production requirements of job types and $N_M$ for each scheduling problem were perturbed by 10% to 30% on the basis of the datasets provided in the supplementary materials of [27]. Moreover, to validate the performances

**TABLE 1. Datasets used for performance comparison.**

| Dataset | The average number of operations | $N_{JT}$ | $N_O$ | The average number of machines | Perturbation on $\mathcal{P}$ and $N_M$ |
|---|---|---|---|---|---|
| D1 | 2,360 | 12 | 64 | 140 | $\pm$ 10% |
| D2 | 2,360 | 12 | 64 | 140 | $\pm$ 20% |
| D3 | 2,360 | 12 | 64 | 140 | $\pm$ 30% |
| D4 | 2,400 | 15 | 100 | 140 | $\pm$ 10% |
| D5 | 2,400 | 15 | 100 | 140 | $\pm$ 20% |
| D6 | 2,400 | 15 | 100 | 140 | $\pm$ 30% |

of the proposed method, we randomly generated 30 test scheduling problems in the same manner as when building the case scheduling problems. In order to demonstrate the effectiveness of the proposed method with respect to the change in $w$, the performances of the proposed method and the other baseline methods were investigated by varying $w$ to be 0, 0.25, 0.5, 0.75, and 1 for each dataset.

As mentioned in Section IV-B, a separate case database was built for each dataset by solving case scheduling problems using TSGA and GASO, resulting in the construction of six different case databases. In order to avoid building too many cases for a single case scheduling problem, a case was stored in the case database only when $F_w$ decreased while the two genetic algorithms were running. During the case generation and the performance comparisons, the parameters of TSGA and GASO were set to the best ones presented in [22] and [21], respectively. All the experiments were conducted using Python on a Core i7 3.6 GHz PC with 8-GB memory.

### B. PERFORMANCE COMPARISON

Fig. 2 depicts $F_w$ (in hours) curves when $w$ = 0, 0.5, and 1, where Figs. 2(a) and 2(b) show the results for D3 and D6, respectively, which have the highest perturbation rate. The $x$ and $y$ axes in the plots shown in Fig. 2 indicate the number of retrieved cases and the average $F_w$ of the scheduling problems, respectively.

As shown in Fig. 2, $F_w$ declined at different rates according to for each dataset. Specifically, $F_w$ tends to be more quickly decreased when the number of retrieved cases is less than 5. This implies that the performance of the proposed method might not be guaranteed when a small number of cases were retrieved to solve test scheduling problems. On the other hand, the performance changes were negligible when the number of retrieved cases is larger than 30. Therefore, the number of retrieved cases was set to be 30 during the performance comparisons with the other methods considered in order to reduce the computation time of the proposed method.

To investigate the effect of the case database size on the performance of the proposed method, we visualized the average $F_w$ (in hours) results on D1–D3 and D4–D6 in Figs. 3 (a)–(c) and Figs. 3 (d)–(f), respectively. For each curve in Fig. 3, the number of retrieved cases was set to be 30. Since the changes in $F_w$ were negligible when $|\mathcal{C}|$ exceeds 3,000, Fig. 3 only presented the curves up to 3,000 cases. It was observed that $F_w$ tends to decrease until $|\mathcal{C}|$ reaches a certain threshold. Once beyond this threshold, significant changes in $F_w$ were not found, and even slight increases in $F_w$ occurred as $|\mathcal{C}|$ grows. This may be attributed to the fact that a newly generated case is more likely to be subsumed by or conflicted with the others as the size of case database becomes larger [42].

In the experiments, TSGA, GASO, and 8 rule-based methods were compared with the proposed method. It is worth noting that the execution time of TSGA and GASO was set to one hour, since a schedule is built on an hourly basis in the real-world semiconductor packaging facilities [32]. Furthermore, the rule-based methods considered are shortest setup unit (SSU), shortest sum of processing time and setup unit (SPTSSU), most operations remaining (MOR), least operations remaining (LOR), most work remaining (MWR), least work remaining (LWR), shortest processing time (SPT), and longest processing time (LPT), which was presented in [27].

Table 2 presents $F_w$ results obtained from the proposed method as well as its percent improvement rates over TSGA, GASO, and the rule-based methods considered in terms of $F_w$. The negative value indicates that the $F_w$ yielded by our method is larger than that of the other methods.

It was observed that $F_w$ values of the proposed method, compared to those of TSGA, become larger in D3 and D6 than in D1 and D4, respectively. This is because the performance of the proposed method might achieve the better results in terms of $F_w$ as the perturbation rate decreases. Compared to TSGA and GASO, the performance of the proposed method tends to become better as $w$ decreases. This might be attributed to the fact that TSGA and GASO do not relatively effective to reduce *AWT* since they were respectively devised to minimize the makespan and maximize the utilization of machines, which is related to reduce *ALT* [21].

Nevertheless, the proposed method outperformed the other methods considered on D1, and yielded the comparable performance to TSGA. Based on the observations, it can be said that $F_w$ obtained from the proposed method was comparable to those of TSGA and GASO when the perturbation level is low. Meanwhile, in order to obtain the satisfactory schedules by employing the proposed approach in a real-world

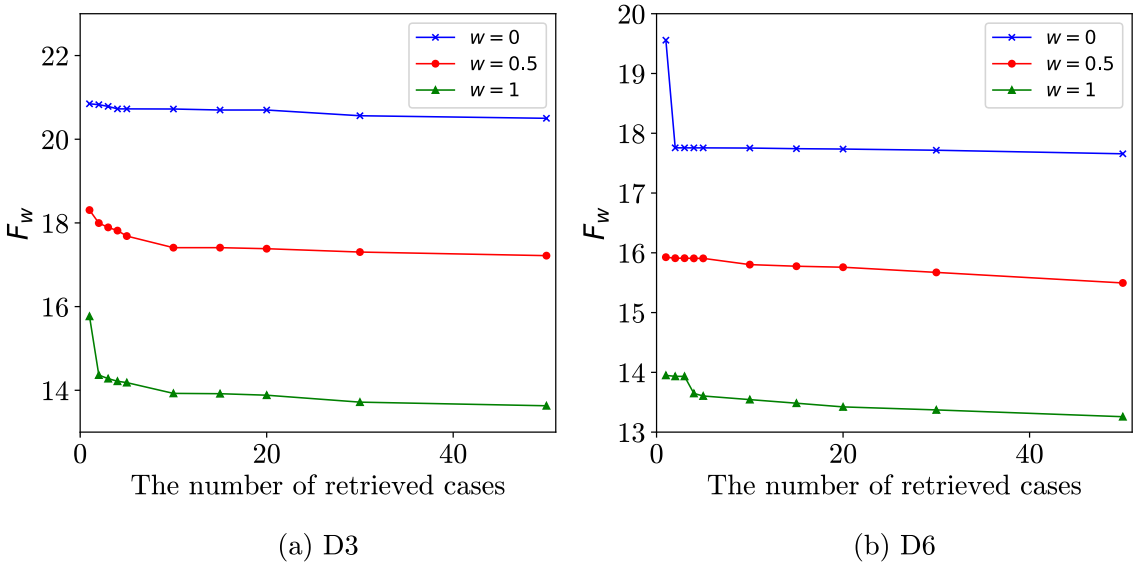


**FIGURE 2.** $F_W$ results with respect to the number of retrieval cases on D3 and D6.

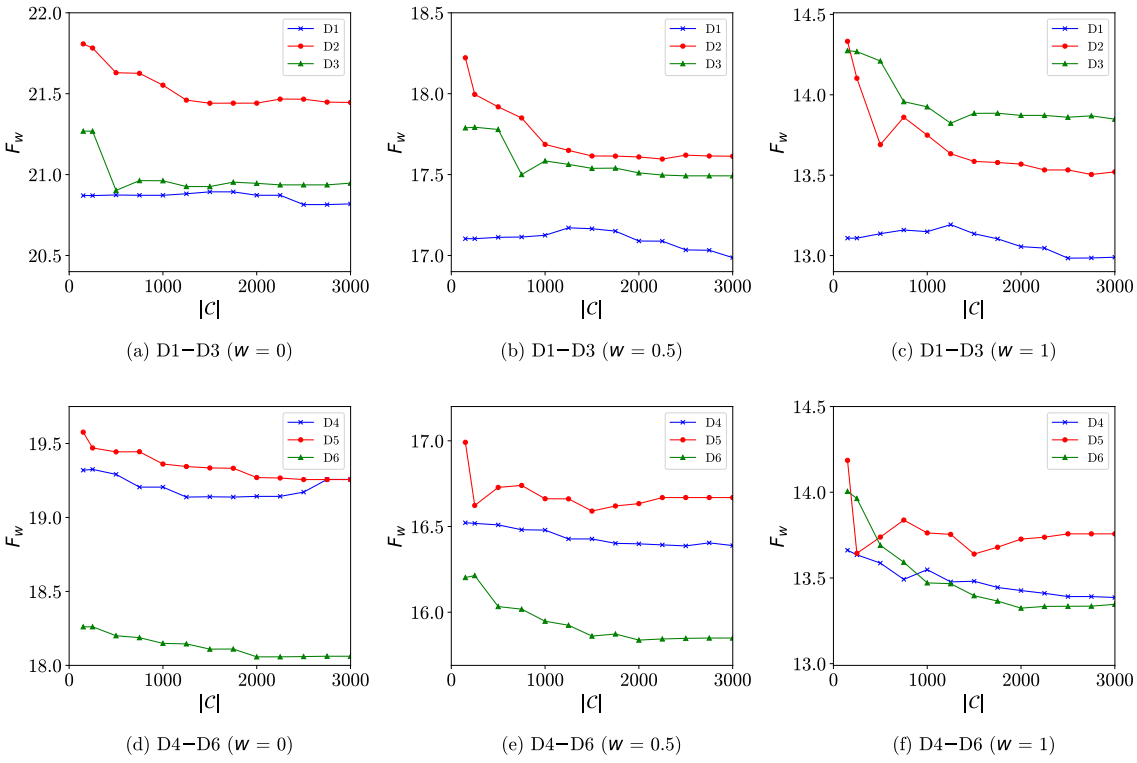


**FIGURE 3.** $F_W$ results according to the size of the case database.

semiconductor packaging system, it is of great importance to build a case database that minimizes the differences between case and test scheduling problems.

The performance of SPTSSU was better than that of the other rules in terms of $F_w$ on D1 to D3. On the other hand, on D4 to D6, $F_0$ and $F_1$ vaalues obtained by SPTSSU were larger than those of LOR and MWR, respectively. This reveals that the best rule is determined depending on the characteristics of scheduling problems such as the number of machines, $N_J$, and $N_O$. However, across all datasets, the $F_w$ values of the proposed method were smaller than those of the rules, demonstrating the superiority of the proposed method in comparison with the rule-based methods considered in terms of $F_w$.

**TABLE 2.** $F_W$ results (in hours) of ours with its percent improvements over TSGA, GASO, and 8 rule-based methods.

| Dataset | $w$ | Ours | TSGA | GASO | SSU | SPTSSU | MOR | LOR | MWR | LWR | SPT | LPT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D1 | 0 | 20.8 | 1.3% | 4.7% | 17.8% | 3.8% | 51.2% | 7.3% | 52.1% | 8.3% | 27.6% | 36.3% |
| | 0.25 | 19.0 | 1.1% | 3.4% | 26.4% | 13.9% | 50.3% | 24.3% | 51.1% | 21.8% | 35.6% | 42.8% |
| | 0.5 | 17.0 | 0.8% | 3.4% | 35.0% | 24.1% | 49.4% | 38.5% | 50.0% | 34.1% | 43.6% | 49.4% |
| | 0.75 | 15.0 | 0.8% | 3.2% | 43.6% | 34.1% | 48.3% | 50.4% | 48.7% | 45.1% | 51.4% | 56.0% |
| | 1 | 12.9 | 0.9% | 4.2% | 52.1% | 44.1% | 47.1% | 60.6% | 47.1% | 55.2% | 59.0% | 62.5% |
| D2 | 0 | 21.4 | 1.4% | 4.4% | 17.6% | 3.0% | 51.0% | 5.5% | 52.0% | 7.1% | 25.8% | 36.9% |
| | 0.25 | 19.5 | 0.9% | 4.2% | 25.7% | 13.6% | 49.8% | 22.6% | 50.7% | 20.6% | 34.0% | 43.0% |
| | 0.5 | 17.6 | 0.4% | 3.6% | 33.9% | 23.9% | 48.5% | 36.8% | 49.3% | 32.7% | 42.1% | 49.2% |
| | 0.75 | 15.5 | 0.1% | 3.4% | 42.2% | 34.1% | 47.0% | 48.7% | 47.6% | 43.8% | 50.0% | 55.5% |
| | 1 | 13.5 | -0.1% | 4.8% | 50.5% | 44.1% | 45.2% | 59.1% | 45.5% | 53.9% | 57.7% | 61.9% |
| D3 | 0 | 20.9 | 0.4% | 3.7% | 17.3% | 3.1% | 50.5% | 5.8% | 51.6% | 6.9% | 25.4% | 33.9% |
| | 0.25 | 19.3 | -0.8% | 2.0% | 25.3% | 12.4% | 48.9% | 22.4% | 49.9% | 19.7% | 33.9% | 40.4% |
| | 0.5 | 17.5 | -1.9% | 0.6% | 33.6% | 21.9% | 47.3% | 36.4% | 48.1% | 31.5% | 42.2% | 47.0% |
| | 0.75 | 15.6 | -3.0% | -0.7% | 41.7% | 31.4% | 45.4% | 48.1% | 45.9% | 42.2% | 50.1% | 53.5% |
| | 1 | 13.7 | -3.9% | -1.9% | 49.8% | 40.8% | 43.1% | 58.1% | 43.2% | 52.0% | 57.7% | 60.0% |
| D4 | 0 | 19.2 | 0.2% | 4.3% | 21.8% | 21.6% | 56.4% | 17.8% | 57.2% | 19.0% | 38.7% | 34.3% |
| | 0.25 | 17.9 | 0.0% | 2.7% | 33.9% | 29.6% | 54.8% | 35.5% | 55.6% | 30.1% | 45.2% | 44.9% |
| | 0.5 | 16.4 | -0.3% | 2.6% | 44.4% | 37.5% | 53.1% | 48.8% | 53.8% | 40.0% | 51.6% | 53.8% |
| | 0.75 | 14.9 | -0.6% | 2.3% | 53.3% | 45.0% | 51.0% | 59.0% | 51.6% | 48.8% | 57.6% | 61.5% |
| | 1 | 13.4 | -1.1% | 1.4% | 61.0% | 52.0% | 48.1% | 67.0% | 48.6% | 56.6% | 63.2% | 67.9% |
| D5 | 0 | 19.3 | 0.2% | 4.2% | 21.9% | 21.1% | 56.3% | 17.1% | 57.2% | 18.6% | 38.2% | 34.6% |
| | 0.25 | 18.1 | -1.2% | 2.1% | 32.8% | 28.4% | 54.3% | 34.4% | 55.1% | 28.9% | 44.2% | 44.4% |
| | 0.5 | 16.7 | -2.0% | 1.1% | 42.8% | 36.0% | 52.3% | 47.7% | 53.0% | 38.7% | 50.4% | 53.1% |
| | 0.75 | 15.2 | -2.9% | -0.1% | 51.5% | 43.3% | 49.9% | 57.9% | 50.5% | 47.4% | 56.3% | 60.6% |
| | 1 | 13.8 | -3.9% | -1.9% | 59.1% | 50.3% | 46.7% | 66.1% | 47.2% | 55.2% | 61.9% | 67.0% |
| D6 | 0 | 18.1 | 0.3% | 2.9% | 23.3% | 21.9% | 56.7% | 19.0% | 57.5% | 20.6% | 39.2% | 35.6% |
| | 0.25 | 17.0 | -1.1% | 2.1% | 34.4% | 29.2% | 54.6% | 35.7% | 55.4% | 30.6% | 45.2% | 45.2% |
| | 0.5 | 15.9 | -2.7% | 0.3% | 43.9% | 36.3% | 52.2% | 48.1% | 53.0% | 39.6% | 50.9% | 53.3% |
| | 0.75 | 14.6 | -3.9% | -0.6% | 52.4% | 43.4% | 49.5% | 57.9% | 50.2% | 47.8% | 56.5% | 60.4% |
| | 1 | 13.4 | -5.8% | -3.1% | 59.5% | 49.8% | 45.7% | 65.5% | 46.4% | 55.0% | 61.6% | 66.4% |

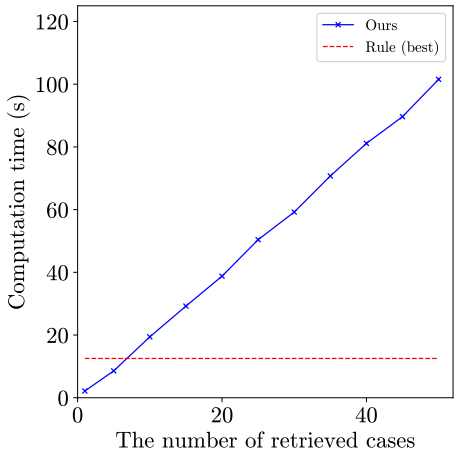


**FIGURE 4.** Computation time (in seconds) of the proposed method and the best rule on D6.

To assess the computational efficiency of the proposed method, the computation time (in seconds) required to solve a test scheduling problem was calculated for D6, which has the largest values for $N_O$ and the perturbation rate among all the datasets. Fig. 4 represents the average computation time of the proposed method and the best rule whose average computation time is the lowest among the rules. In the plot, the $x$ and $y$ coordinates of each point refer to the number of the retrieved cases and the average computation time for solving a test scheduling problem, respectively.

As depicted in Fig. 4, the computation time of the best rule was similar to that of the proposed method when retrieved cases were between 5 and 10. Furthermore, there is a strong tendency for the computation time of the proposed method to increase linearly with the number of retrieved cases. This is because the computational complexity becomes $O(N_R \cdot (N_M L' + d_{max} L))$, as described in Section IV-B. Therefore, if practitioners have sufficient time to generate a schedule, they can select a relatively large value for $N_R$. However, if time is limited, they should choose a smaller value for $N_R$. Based on the observation, the proposed approach seems to be practicable for contemporary semiconductor manufacturers in terms of the computation time since a scheduling decision is required to made within an hour to accommodate the variations in production requirements, the available machines, and $w$ [32].

## VI. CONCLUSION

In this paper, we proposed a CBR-based framework for scheduling semiconductor packaging facilities. Specifically, the proposed framework attempts to utilize cases belonging to the case database that is built by solving a number of case scheduling problems in advance. Furthermore, efficient

operation type repair and machine allocation algorithms are devised to address the variabilities in initial setup status, available machines, production requirements, as well as the relative importance between waiting time of jobs and loss time of machines.

To demonstrate the effectiveness and robustness of the proposed framework, the experiments are conducted for large-scale datasets that simulate the real-world semiconductor packaging facilities. The results showed that the proposed approach outperformed the existing rule-based methods in terms of the weighted sum of the average waiting time of jobs and loss time of machines while the quality of the schedule remains almost intact compared to the metaheuristics considered. Furthermore, the computation time taken by the proposed method is similar to that of the rule-based methods.

Despite these achievements, there are still some rooms for further research. We plan to improve the consistency of the proposed CBR framework. To attain this goal, the maintenance strategy of the pre-generated case database and a similarity measure for retrieving more relevant cases will be developed. Moreover, real-world constraints such as the stochastic processing time and dynamic job arrivals will be considered in future work.

## REFERENCES

[1] J. Lin, Y.-Y. Li, and H.-B. Song, "Semiconductor final testing scheduling using Q-learning based hyper-heuristic," *Exp. Syst. Appl.*, vol. 187, Jan. 2022, Art. no. 115978.

[2] J. Huh, I. Park, S. Lim, B. Paeng, J. Park, and K. Kim, "Learning to dispatch operations with intentional delay for re-entrant multiple-chip product assembly lines," *Sustainability*, vol. 10, no. 11, p. 4123, Nov. 2018.

[3] M. Fu, R. Askin, J. Fowler, M. Haghnevis, N. Keng, J. S. Pettinato, and M. Zhang, "Batch production scheduling for semiconductor back-end operations," *IEEE Trans. Semicond. Manuf.*, vol. 24, no. 2, pp. 249–260, May 2011.

[4] G. Weigert, A. Klemmt, and S. Horn, "Design and validation of heuristic algorithms for simulation-based scheduling of a semiconductor backend facility," *Int. J. Prod. Res.*, vol. 47, no. 8, pp. 2165–2184, Apr. 2009.

[5] C. Zhang, J. F. Bard, and R. Chacon, "Controlling work in process during semiconductor assembly and test operations," *Int. J. Prod. Res.*, vol. 55, no. 24, pp. 7251–7275, Dec. 2017.

[6] S. Yao, Z. Jiang, N. Li, H. Zhang, and N. Geng, "A multi-objective dynamic scheduling approach using multiple attribute decision making in semiconductor manufacturing," *Int. J. Prod. Econ.*, vol. 130, no. 1, pp. 125–133, Mar. 2011.

[7] D. Lei, "A Pareto archive particle swarm optimization for multi-objective job shop scheduling," *Comput. Ind. Eng.*, vol. 54, no. 4, pp. 960–971, May 2008.

[8] S. Chen, Q.-K. Pan, L. Gao, and H.-Y. Sang, "A population-based iterated greedy algorithm to minimize total flowtime for the distributed blocking flowshop scheduling problem," *Eng. Appl. Artif. Intell.*, vol. 104, Sep. 2021, Art. no. 104375.

[9] M. Li, D. Lei, and J. Cai, "Two-level imperialist competitive algorithm for energy-efficient hybrid flow shop scheduling problem with relative importance of objectives," *Swarm Evol. Comput.*, vol. 49, pp. 34–43, Sep. 2019.

[10] J. Liu, F. Qiao, and W. Kong, "Scenario-based multi-objective robust scheduling for a semiconductor production line," *Int. J. Prod. Res.*, vol. 57, no. 21, pp. 6807–6826, Nov. 2019.

[11] I. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 3, pp. 1420–1431, Jul. 2020.

[12] E. Hernández-Nieves, G. Hernández, A. B. Gil-González, S. Rodríguez-González, and J. M. Corchado, "CEBRA: A CasE-based reasoning application to recommend banking products," *Eng. Appl. Artif. Intell.*, vol. 104, Sep. 2021, Art. no. 104327.

[13] Y. Gao, X. Chen, Y. Chen, Y. Sun, X. Niu, and Y. Yang, "A secure cryptocurrency scheme based on post-quantum blockchain," *IEEE Access*, vol. 6, pp. 27205–27213, 2018.

[14] S. Jia, D. J. Morrice, and J. F. Bard, "A performance analysis of dispatch rules for semiconductor assembly & test operations," *J. Simul.*, vol. 13, no. 3, pp. 163–180, Jul. 2019.

[15] H. Zhang, Z. Jiang, and C. Guo, "Simulation-based optimization of dispatching rules for semiconductor wafer fabrication system scheduling by the response surface methodology," *Int. J. Adv. Manuf. Technol.*, vol. 41, nos. 1–2, pp. 110–121, Mar. 2009.

[16] Y. Ni, Y. Li, J. Yao, and J. Li, "Development of an integrated real time dispatching system: A case study at a semiconductor assembly and test factory," *J. Manuf. Technol. Manage.*, vol. 25, no. 7, pp. 980–997, Aug. 2014.

[17] N. B. Ho and J. C. Tay, "Evolving dispatching rules for solving the flexible job-shop problem," in *Proc. IEEE Congr. Evol. Comput.*, vol. 3, Sep. 2005, pp. 2848–2855.

[18] F. Zhang, Y. Mei, and M. Zhang, "Can stochastic dispatching rules evolved by genetic programming hyper-heuristics help in dynamic flexible job shop scheduling?" in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 41–48.

[19] Q. Luo, Q. Deng, G. Gong, L. Zhang, W. Han, and K. Li, "An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers," *Exp. Syst. Appl.*, vol. 160, Dec. 2020, Art. no. 113721.

[20] Z. Zhu and X. Zhou, "A multi-objective multi-micro-swarm leadership hierarchy-based optimizer for uncertain flexible job shop scheduling problem with job precedence constraints," *Exp. Syst. Appl.*, vol. 182, Nov. 2021, Art. no. 115214.

[21] B.-S. Chung, J. Lim, I.-B. Park, J. Park, M. Seo, and J. Seo, "Setup change scheduling for semiconductor packaging facilities using a genetic algorithm with an operator recommender," *IEEE Trans. Semicond. Manuf.*, vol. 27, no. 3, pp. 377–387, Aug. 2014.

[22] F. M. Defersha and D. Rooyani, "An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time," *Comput. Ind. Eng.*, vol. 147, Sep. 2020, Art. no. 106605.

[23] Y. Wang and Q. Zhu, "A hybrid genetic algorithm for flexible job shop scheduling problem with sequence-dependent setup times and job lag times," *IEEE Access*, vol. 9, pp. 104864–104873, 2021.

[24] J. Fan, C. Zhang, Q. Liu, W. Shen, and L. Gao, "An improved genetic algorithm for flexible job shop scheduling problem considering reconfigurable machine tools with limited auxiliary modules," *J. Manuf. Syst.*, vol. 62, pp. 650–667, Jan. 2022.

[25] K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, and L. Tang, "A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem," *Exp. Syst. Appl.*, vol. 205, Nov. 2022, Art. no. 117796.

[26] R. Li, W. Gong, and C. Lu, "A reinforcement learning based RMOEA/D for bi-objective fuzzy flexible job shop scheduling," *Exp. Syst. Appl.*, vol. 203, Oct. 2022, Art. no. 117380.

[27] I. Park and J. Park, "Scalable scheduling of semiconductor packaging facilities using deep reinforcement learning," *IEEE Trans. Cybern.*, vol. 53, no. 6, pp. 3518–3531, Jun. 2021.

[28] S. H. Oh, Y. I. Cho, and J. H. Woo, "Distributional reinforcement learning with the independent learners for flexible job shop scheduling problem with high variability," *J. Comput. Design Eng.*, vol. 9, no. 4, pp. 1157–1174, Jul. 2022.

[29] J. Kolodner, *Case-Based Reasoning*. Burlington, MA, USA: Morgan Kaufmann, 2014.

[30] L. Fei and Y. Feng, "A novel retrieval strategy for case-based reasoning based on attitudinal choquet integral," *Eng. Appl. Artif. Intell.*, vol. 94, Sep. 2020, Art. no. 103791.

[31] P. Veerakamolmal and S. M. Gupta, "A case-based reasoning approach for automating disassembly process planning," *J. Intell. Manuf.*, vol. 13, no. 1, pp. 47–60, 2002.

[32] J. Lim, M. Chae, Y. Yang, I. Park, J. Lee, and J. Park, "Fast scheduling of semiconductor manufacturing facilities using case-based reasoning," *IEEE Trans. Semicond. Manuf.*, vol. 29, no. 1, pp. 22–32, Feb. 2016.

[33] J. Huh, M.-J. Chae, J. Park, and K. Kim, "A case-based reasoning approach to fast optimization of travel routes for large-scale AS/RSs," *J. Intell. Manuf.*, vol. 30, no. 4, pp. 1765–1778, Apr. 2019.
[34] M. R. Khosravani, S. Nasiri, and K. Weinberg, "Application of case-based reasoning in a fault detection system on production of drippers," *Appl. Soft Comput.*, vol. 75, pp. 227–232, Feb. 2019.
[35] M. T. Rezvan, A. Zeinal Hamadani, and A. Shalbafzadeh, "Case-based reasoning for classification in the mixed data sets employing the compound distance methods," *Eng. Appl. Artif. Intell.*, vol. 26, no. 9, pp. 2001–2009, Oct. 2013.
[36] S. Simić, D. Milutinović, S. Sekulić, D. Simić, S. D. Simić, and J. Dorđević, "A hybrid case-based reasoning approach to detecting the optimal solution in nurse scheduling problem," *Log. J. IGPL*, vol. 28, no. 2, pp. 226–238, Oct. 2018.
[37] J. I. Guerrero, G. Miró-Amarante, and A. Martín, "Decision support system in health care building design based on case-based reasoning and reinforcement learning," *Exp. Syst. Appl.*, vol. 187, Jan. 2022, Art. no. 116037.
[38] C.-G. Chang, D.-W. Wang, K.-Y. Hu, and B.-L. Zheng, "Two stage case-based reasoning application research on steel-making dynamic scheduling," in *Proc. Int. Conf. Mach. Learn. Cybern.*, Aug. 2004, pp. 2116–2121.
[39] P.-C. Chang, J.-C. Hsieh, and C.-H. Liu, "A case-injected genetic algorithm for single machine scheduling problems with release time," *Int. J. Prod. Econ.*, vol. 103, no. 2, pp. 551–564, Oct. 2006.
[40] I. Pereira and A. Madureira, "Meta-heuristics tunning using CBR for dynamic scheduling," in *Proc. IEEE 9th Int. Conf. Cyberntic Intell. Syst.*, Sep. 2010, pp. 1–6.
[41] K. F. Guimaraes and M. A. Fernandes, "An approach for flexible job-shop scheduling with separable sequence-dependent setup time," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 5, Oct. 2006, pp. 3727–3731.
[42] S. K. Pal and S. C. Shiu, *Foundations of Soft Case-Based Reasoning*. Hoboken, NJ, USA: Wiley, 2004.

**IN-BEOM PARK** received the B.S. and Ph.D. degrees in industrial engineering from Seoul National University, South Korea, in 2012 and 2020, respectively. He is currently an Assistant Professor with the Department of Industrial and Management Engineering, Myongji University, South Korea. His research interests include scheduling manufacturing systems and solving real-world sequential decision making problems.

**JAESEOK HUH** received the B.S. and Ph.D. degrees in industrial engineering from Seoul National University (SNU), South Korea, in 2013 and 2019, respectively. He is currently an Assistant Professor with the Department of Business Administration, Tech University of Korea, South Korea. His research interests include semiconductor manufacturing system dispatching/scheduling, machine learning, smart factory, and deep reinforcement learning.

**JONGHUN PARK** received the Ph.D. degree in industrial and systems engineering with a minor in computer science from the Georgia Institute of Technology, Atlanta, in 2000. He is currently a Professor with the Department of Industrial Engineering, Seoul National University (SNU), South Korea. Before joining SNU, he was an Assistant Professor with the School of Information Sciences and Technology, Pennsylvania State University, University Park, and an Assistant Professor with the Department of Industrial Engineering, KAIST, Daejeon. His research interests include generative artificial intelligence and deep learning applications.