Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 8833645, 13 pages
https://doi.org/10.1155/2020/8833645

Research Article

A Genetic Algorithm for a Two-Machine Flowshop with a Limited Waiting Time Constraint and Sequence-Dependent Setup Times

Ju-Yong Lee

Division of Business Administration & Accounting, Kangwon National University, Chuncheon, Republic of Korea

Correspondence should be addressed to Ju-Yong Lee; jy.lee@kangwon.ac.kr

Received 27 August 2020; Revised 18 October 2020; Accepted 26 October 2020; Published 24 November 2020

Academic Editor: Dr. Dilbag Singh

This study considers a two-machine flowshop with a limited waiting time constraint between the two machines and sequence-dependent setup times on the second machine. These characteristics are motivated from semiconductor manufacturing systems. The objective of this scheduling problem is to minimize the total tardiness. In this study, a mixed-integer linear programming formulation was provided to define the problem mathematically and used to find optimal solutions using a mathematical programming solver, CPLEX. As CPLEX required a significantly long computation time because this problem is known to be NP-complete, a genetic algorithm was proposed to solve the problem within a short computation time. Computational experiments were performed to evaluate the performance of the proposed algorithm and the suggested GA outperformed the other heuristics considered in the study.

1. Introduction

The two-machine flowshop scheduling problem with a limited waiting time constraint and sequence-dependent setup times investigated in this study is motivated by semiconductor manufacturing systems. In a two-machine flowshop, all jobs are processed in the same order of machine 1 and then machine 2, which is called permutation schedule. Owing to the limited waiting time constraint, jobs must be started on the second machine within a limited waiting time after the completion of those jobs on the first machine. Additionally, sequence-dependent setup times are incurred between two consecutive jobs on the second machine. The objective function of this scheduling problem is a minimization of the total tardiness. Thus, this problem can be defined by $F2|\max - \text{wait}$ and $s_{ij}|\sum_i T_i$ in the three-field notation suggested by [1], where max-wait and s_{ij} refer to a limited waiting time constraint and sequence-dependent setup times, respectively. T_i is the tardiness of job i defined as $T_i = \max\{C_i - d_i, 0\}$, where C_i and d_i are the completion time and due date of job i, respectively. This scheduling problem is NP-complete because the typical two-machine flowshop tardiness problem is NP-complete [2].

The considered limited waiting time constraint is common in semiconductor wafer fabrication, in which there are many chemical processing processes. One example is a flowshop consisting of cleaning and diffusion processes [3]. Wafers that have completed the cleaning process may have problems that increase the likelihood of contamination and decrease quality if the surface is exposed to air for a long time before the next process (the diffusion process). To prevent this problem, the limited waiting time constraint is set between cleaning and diffusion processes. Additionally, if the chemical treatment effect is valid only when the treated wafer is processed on the next machine within a certain time after the treatment, the waiting time would be limited. If a wafer violates this time limit, the first operation must be reworked or discarded in case of high contamination. Failure to comply with the time constraints usually occurs when production volume is high, reducing productivity and causing management losses.

Setup time refers to the time required for preparation before processing jobs. If the setup time for a succeeding job varies depending on the preceding job, it is called the sequence-dependent setup time. In other words, the setup time depends on the sequence of the jobs. In semiconductor

manufacturing systems, wafers for different types of products may require different chemicals or operating temperatures even though they are processed in the same machine. Therefore, job scheduling is important to schedule jobs to minimize the setup times when the sequence-dependent setup times are considered.

A two-machine flowshop scheduling problem with sequence-dependent setup times (SDSTs) to minimize the makespan is very similar to the typical travel salesman problem (TSP) [4], and dynamic programming methods [5, 6] and branch and bound algorithms [7, 8] have been proposed to obtain optimal solutions for the problem. However, since finding optimal solutions of the flowshop problem with SDST requires considerably long computation times, recent studies have focused on development of metaheuristic algorithms: genetic algorithms [9–11], variable neighborhood search algorithm [12], migrating bird optimization algorithm [13], discrete artificial bee colony optimization [14], iterated greedy algorithm [15–17], and local search based heuristic algorithm [18].

A two-machine flowshop scheduling problem with a limited waiting time (LWT) constraint to minimize the makespan has been proven to be NP-hard [19]. For this problem, [19, 20] suggest branch and bound algorithms to find optimal solutions, and [21] proposes several dominance properties for optimal solutions. The study in [22] considers a situation where some jobs skip the first machine in a two-machine flowshop and suggests an approximation algorithm to minimize the waiting time variation of jobs. The study in [23] considers a three-machine flowshop with overlapping waiting time constraints and proposes a branch and bound algorithm to minimize makespan. The study in [24, 25] consider flowshop problems with time lag constraints and develop a simulated annealing algorithm and a constructive heuristic algorithm with and insertion mechanism, respectively. On the other hand, for a flowshop problem with LWT for the objective of minimizing total tardiness, a heuristic algorithm and a Lagrangian relaxation-based lower bound strategy [26] have been developed.

Only a few studies have considered SDSTs and LWT together, although both are common and important in semiconductor manufacturing systems. The study in [3] suggests a branch and bound algorithm for the objective of minimizing makespan. On the other hand, no-wait flowshop problems with SDSTs have been studied recently. The study in [27] proposes a branch and bound algorithm for no-wait flowshop, and [28] proposes a pairwise iterated greedy algorithm, while [29] suggests an insertion neighborhood search algorithm to minimize total flowtime. Note that this study is the first attempt to consider the two-machine flowshop with SDST and LWT for the objective of minimizing total tardiness.

In this study, a mixed-integer programming formulation for the considered scheduling problem is provided to describe the problem clearly and to obtain optimal schedule by CPLEX. Additionally, heuristic algorithms are developed to obtain good (or near-optimal) schedules in a short computation time rather than an optimal schedule because the problem is NP-complete. Three types of heuristics are proposed: list scheduling methods, a constructive heuristic, and a genetic algorithm. To demonstrate the performance of the proposed heuristic algorithms, computational experiments were performed on randomly generated instances.

In the remainder of this paper, Section 2 describes the problem considered in this study in more detail including several assumptions and presents a mixed-integer linear programming formulation. Section 3 proposes heuristic algorithms. Section 4 describes the computational experiments and shows the results. Finally, Section 5 concludes the paper with a short summary.

2. Problem Description

In this section, the considered scheduling problem is described in more detail, and a mixed-integer linear programming formulation is presented. There are n jobs to be processed in the two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times on the second machine. Here, only permutation schedules are considered; that is, job sequences on the two machines are the same. Although permutation schedules are not dominant in the problem, permutation schedules are common in real situations because of the simplicity of work management, flexibility in material handling, and limitation in buffer space. Note that permutation schedules are dominant in a typical two-machine flowshop with regular measures, including total tardiness. Other assumptions include the following:

(i) All jobs are ready to start at time zero

(ii) Information on the jobs to be scheduled is given; that is, processing time, due date, limited waiting time, and sequence-dependent setup time are given in advance

(iii) Preemption is not allowed

(iv) Each machine can process only one job at a time, and a job can be processed on only one machine at a time

In this paper, symbols in Table 1 are used to describe the mathematical formulation and proposed algorithms.

Completion times and tardiness of the jobs in a given sequence are computed as follows:

$$c_{[1]1} = p_{[1]1}, \tag{1}$$

$$c_{[1]2} = p_{[1]1} + p_{[1]2}, \tag{2}$$

$$c_{[r]1} = \max\left\{c_{[r-1]1} + p_{[r]1},\ c_{[r-1]2} + s_{[r-1][r]} - w_{[r]}\right\}, \quad \text{for } r \geq 2, \tag{3}$$

$$c_{[r]2} = \max\left\{c_{[r]1},\ c_{[r-1]2} + s_{[r-1][r]}\right\} + p_{[r]2}, \quad \text{for } r \geq 2, \tag{4}$$

$$T_{[r]} = \max\left\{0, c_{[r]2} - d_{[r]}\right\}, \quad \text{for all } i. \tag{5}$$

The mixed-integer linear programming (MILP) formulation for the problems is given:

Table 1: Notations for the description of the proposed algorithms.

Symbol	Definition
J	Set of jobs, $J = \{1, \ldots, n\}$
i, j	Job indices
k	Machine index $(k = 1, 2)$
p_{ik}	Processing time of job i on machine k
d_i	Due date of job i
w_i	Limited waiting time of job i
s_{ij}	Sequence-dependent setup time between jobs i and j
$[r]$	Index of the job at the r^{th} position in a schedule
x_{ir}	1 if job i is assigned to the rth position in a schedule and 0 otherwise
y_{ijr}	1 if jobs i and j are assigned consecutively to positions r and $r+1$ and 0 otherwise
$b_{[r]k}$	Start time of the r^{th} job on machine k
$c_{[r]k}$	Completion time of the r^{th} job on machine k
$T_{[r]}$	Tardiness of the r^{th} job
Σ	Partial schedule
σ_i	Partial schedule obtained with σ followed by job i
U	Set of unscheduled jobs
$C_k(\sigma)$	Completion time of partial schedule σ on machine k

$$[P] \text{ minimize } \sum_r T_{[r]}, \tag{6}$$

subject to

$$\sum_r x_{ir} = 1, \quad \forall i, \tag{7}$$

$$\sum_i x_{ir} = 1, \quad \forall r, \tag{8}$$

$$b_{[r]1} + \sum_i p_{i1} x_{ir} \le b_{[r+1]1}, \quad r \le n-1, \tag{9}$$

$$b_{[r]2} + \sum_i p_{i2} x_{ir} + \sum_i \sum_j s_{ij} y_{ijr} \le b_{[r+1]2}, \quad r \le n-1, \tag{10}$$

$$b_{[r]1} + \sum_i p_{i1} x_r \le b_{[r]2}, \quad \forall r, \tag{11}$$

$$b_{[r]1} + \sum_i (p_{i1} + w_i) x_{ir} \ge b_{[r]2}, \quad \forall r, \tag{12}$$

$$x_{ir} + x_{j(r+1)} - 1 \le y_{ijr}, \quad \forall i \ne j, r = 1, \ldots, n-1, \tag{13}$$

$$y_{ijr} \le x_{ir}, \quad \forall i \ne j, r = 1, \ldots, r, \ldots, n-1, \tag{14}$$

$$y_{ijr} \le x_{j(r+1)}, \quad \forall i \ne j, r = 1, \ldots, r, \ldots, n-1 \tag{15}$$

$$b_{[r]2} + \sum_i p_{i2} x_{ir} \le c_{[r]2}, \quad \forall r, \tag{16}$$

$$c_{[r]2} - \sum_i d_i x_{ir} \le T_{[r]}, \quad \forall r, \tag{17}$$

$$b_{[r]1}, b_{[r]2}, c_{[r]2} \ge 0, \quad \forall r, \tag{18}$$

$$x_{ir}, y_{ijr} \in \{0, 1\}, \quad \forall i, j, r. \tag{19}$$

Objective function (6) is to minimize the total tardiness. Constraints (7) and (8) ensure that each job is placed at only one position in the sequence, and only one job can be placed at each position. Constraints (9)–(12) define the start time of each job in the considered flowshop. Constraints (13)–(15) define the relationship between x_{ir} and y_{ijr} for sequence-dependent setup times. Constraints (16) and (17) compute the completion time and tardiness of jobs. Constraints (18) and (19) define the domain of the decision variables.

3. Heuristic Algorithms

In this section, three types of heuristic algorithms are proposed to solve the considered problem. The algorithms are list scheduling rules, a modified NEH algorithm (MNEH), and a genetic algorithm (GA). As only permutation schedules are considered in this study, feasible permutation schedules are obtained by these heuristic algorithms, and completion times and tardiness of each jobs are computed with equations (1)–(5) provided in Section 2.

3.1. List Scheduling Rules. List scheduling rules are widely used in practice (e.g., in semiconductor manufacturing systems) because they are intuitive and easy to develop and apply. Four list scheduling rules are used as described below, and schedules are obtained by sorting jobs in a nondecreasing order of the values according to the methods:

(i) Earliest due date (EDD): d_i

(ii) Modified due date (MDD): $d_i' = \max\{d_i, C_2(\sigma i)\}$, for $i \varepsilon U$

(iii) Slack: $\max(d_i - C_2(\sigma i), 0)$, for $i \varepsilon U$

(iv) Greedy: $\max\{C_2(\sigma i) - d_i, 0\}$, for $i \varepsilon U$

3.2. Modified NEH Algorithm. The heuristic algorithm proposed in this subsection is modified from the constructive NEH algorithm [30], which is known to work well

in flowshop scheduling problems and has been modified by many researchers for various scheduling problems. The procedure of MNEH is summarized as follows.

3.2.1. Procedure: MNEH

(i) *Step 0.* Obtain a seed sequence from the list scheduling method. Let σ be this sequence and set $\sigma^* = \varnothing, r = 1$. Go to Step 1.

(ii) *Step 1.* Select the rth job in σ. Go to Step 2.

(iii) *Step 2.* For $(i = 1; i \leq (|\sigma^*| + 1); i++)$, insert the selected job into the ith position of σ^* and compute the total tardiness.

(iv) *Step 3.* Among the $(|\sigma^*| + 1)$ partial schedules obtained in Step 2, select a partial schedule that results in minimum total tardiness. Let σ^* be the selected partial schedule. Go to Step 4.

(v) *Step 4.* $r \longleftarrow r + 1$. If $r < n$, go to Step 1; otherwise, go to Step 5.

(vi) *Step 5.* Set $i = 1$ and $j = 2$. Go to Step 6.

(vii) *Step 6.* Generate a new sequence by interchanging two jobs in the ith and jth positions in σ^*. If the total tardiness of the new sequence is less than that of σ^*, then replace σ^* with the new sequence and go to Step 5; otherwise, go to Step 7.

(viii) *Step 7.* Let $j \leftarrow j + 1$. If $j \leq n$, go to Step 6; otherwise, let $i \leftarrow i + 1$ and $j \leftarrow i + 1$. If $i < n$, go to Step 6; otherwise, terminate. (σ^*) is the final solution of this heuristic.

3.3. Genetic Algorithm. The GA was first introduced [31] as an optimization method that mimics the evolution of the natural world, and it is one of the most popular methods in the field of optimization because of its powerful search mechanism and ability to solve problems. The effectiveness of a GA depends on the procedure design, operators, and parameters. Thus, the following subsections describe the design of the proposed GA in this study.

3.3.1. Solution Representation. In the proposed GA, solutions are expressed in the chromosome structure, and the performance of the algorithm may vary depending on how the chromosome is expressed. In this study, the sequence of jobs is expressed in chromosomal structure because only the permutation schedule is considered.

3.3.2. Initial Population. To generate an initial population, list scheduling methods and the MNEH algorithm are used. That is, four solutions are generated from the list scheduling methods and used to generate four further solutions by MNEH. However, Steps 5–7 of MNEH are not applied here because these steps require a long computation time in large-sized problems. The remainder of the initial population is generated randomly.

3.3.3. Fitness Evaluation. Fitness is the evaluated objective value of a solution; that is, the evaluation value represents how good a solution is. Here, the total tardiness, which is aimed to be minimized by the objective function of the considered problem, is used for fitness evaluation. Then, the lower the total tardiness, the better the solution.

3.3.4. Selection. Selection is to choose a set of parents from the population to generate the offspring. In general, selection is based on the quality of the solutions and randomness. In the proposed GA, the two most widely used schemes in the literature (*tournament* and *roulette*) are used for selection.

In the tournament method, four chromosomes are selected randomly from the population, and pairwise comparisons are performed to choose two chromosomes as parents. That is, the better of the first two chromosomes and the better of the last two chromosomes are selected as the parents.

For the roulette methods, the inverse of the objective function value for all chromosomes (i) in the population is computed; that is, $f_i = 1/\sum_r (T_r + 1)$. Note that 1 is added to the denominator to avoid a case of dividing by zero because the objective function value (total tardiness T_r) can be zero.

The selection probability of each chromosome in the population is then obtained by $\text{prob}_i = f_i/\sum f_i$. Based on these selection probabilities of chromosomes, two chromosomes to be parents are selected randomly.

3.3.5. Crossover. The purpose of the crossover operation is to generate better offspring by exchanging the information of the selected parents. In the proposed GA, nine types of crossover are considered: one-point order crossover (OX1), two-point order crossover (OX2), order-based crossover (OBX), position-based crossover (PBX), partially matched crossover (PMX), similar job crossover (SJX), two-point similar job crossover (SJX2), similar block crossover (SBX), and two-point similar block crossover (SBX2).

For a description of these crossovers, let $parent_1$ and $parent_2$ be the selected parents to generate *offspring*.

(i) Procedure: OX1

Step 0. Let $point_1$ be a randomly selected cutoff point from $parent_1$.
Step 1. Offspring inherits the front part (from the first to $point_1$) of $parent_1$.
Step 2. Delete jobs that are already placed in *offspring* from $parent_2$.
Step 3. Place the jobs in $parent_2$ into the unplaced positions of *offspring* in the order in which they appear in $parent_2$.

(ii) Procedure: OX2

Step 0. Let $point_1$ and $point_2$ be two randomly selected cutoff points from $parent_1$.
Step 1. Offspring inherits the middle part (between $point_1$ and $point_2$) of $parent_1$.
Step 2. Delete jobs that are already placed in *offspring* from $parent_2$.

Step 3. Place the jobs in $parent_2$ into the unplaced positions of *offspring* from left to right in the order in which they appear in $parent_2$.

(iii) Procedure: OBX

Step 0. Let set_J be a set of jobs from $parent_1$ at random.
Step 1. Let set_P be a set of positions corresponding jobs in set_J from $parent_2$.
Step 2. Place the jobs in set_J into the positions in set_P of offspring from left to right in the order in which the jobs appear in $parent_1$.
Step 3. Delete jobs that are already placed in *offspring* from $parent_2$.
Step 4. Place the jobs in $parent_2$ into the unplaced positions of *offspring* from left to right in the order in which they appear in $parent_2$.

(iv) Procedure: PBX

Step 0. Select a set of positions from $parent_1$ at random.
Step 1. Offspring inherits the jobs placed in the positions of the set from $parent_1$.
Step 2. Delete jobs that are already placed in *offspring* from $parent_2$.
Step 3. Place the jobs in $parent_2$ into the unplaced positions of *offspring* from left to right in the order in which they appear in $parent_2$.

(v) Procedure: PMX

Step 0. Let $point_1$ and $point_2$ be two randomly selected cutoff points from $parent_1$.
Step 1. Offspring inherits the middle part (between $point_1$ and $point_2$) of $parent_1$.
Step 2. For each unplaced position in *offspring*, the job in the same position in $parent_2$ is placed.
Step 3. Delete jobs that are already placed in *offspring* from $parent_2$.
Step 4. If duplicate jobs occur before $point_1$ and after $point_2$, these jobs are exchanged with the remaining jobs in $parent_2$, considering the order of $parent_2$.

(vi) Procedure: SJX

Step 0. Offspring inherits jobs in the same position in the two parents.
Step 1. Let $point_1$ be a randomly selected cutoff point from $parent_1$.
Step 2. Offspring inherits the front part (from the first to the $point_1$) of $parent_1$.
Step 3. Delete jobs that are already placed in *offspring* from $parent_2$.
Step 4. Place the jobs in $parent_2$ into the unplaced positions of *offspring* in the order in which they appear in $parent_2$.

In SJX2, Steps 1 and 2 of SJX are modified by considering two cutoff points, as in OX2. Additionally, SBX and SBX2 are very similar to SJX and SJX2, respectively. In SBX and SBX2, a block means two consecutive jobs, and the only difference is that an offspring inherits the blocks in the same position in the two parents in Step 0. Thus, the detailed procedures of SJX2, SBX, and SBX2 are omitted.

3.3.6. Mutation. A mutation is an operation to escape from the local optimum and increase the variability and diversity in the population. This operation partially modifies a chromosome to generate *offspring* with a new chromosome. In the proposed GA, three types of mutation schemes are considered: interchange, insertion, and swap.

(i) Interchange: two randomly selected jobs are interchanged.

(ii) Insertion: a randomly selected job is inserted into a randomly selected position.

(iii) Swap: a randomly selected job is swapped with the next job.

3.3.7. Improvement (Local Search). Local search methods are commonly used in genetic algorithms as well as other (meta)heuristics to improve the solutions. A local search strategy based on insertion is also used in the proposed GA. This local search is triggered when the current best objective function value is not improved for a predetermined number of consecutive generations. When the local search is triggered, insertion is applied $3n$ times to the best solution in the population, where n is the number of jobs. If the new solution is better than the current best solution, the current solution is replaced with the new solution. Note that because excessive local search attempts may consume a long computation time, the number of interchanges is limited to $3n$.

3.3.8. Restart. The goal of a restart procedure is to avoid premature convergence in the population and improve the solution quality obtained by GA. The restart procedure in the proposed GA is based on that of Ruiz and Maroto [32]. This restart procedure is triggered when the current best objective function value is not improved for a predetermined number of consecutive generations. The restart procedure is summarized below.

(i) Procedure: restart

Step 0. Sort the chromosomes in the population in a nondecreasing order of their fitness value.
Step 1. Keep the first 20% of chromosomes and remove the remaining 80% of the chromosomes from the population.
Step 2. For the 20% of the population size, chromosomes are selected randomly from the first 20% of chromosomes, and the selected chromosomes are mutated once with an insertion operation.
Step 3. For the 20% of the population size, chromosomes are selected randomly from the first 20% of chromosomes, and the selected chromosomes are mutated once with an interchange operation.

Step 4. The remaining 40% of chromosomes are generated randomly.

3.3.9. Termination Criterion. The termination criterion applied to the proposed GA is the maximum CPU time. That is, when the maximum CPU time from the start of the GA elapses, the GA procedure stops.

3.3.10. Whole Procedure of the Proposed GA. The procedure of the proposed GA is summarized below, using the operators described above.

(i) Procedure: proposed GA

(ii) *Step 0.* Let POP, P_{size}, p_c, and p_m be the population, population size, crossover probability, and mutation probability, respectively.

(iii) *Step 1.* Generate an initial population with P_{size} chromosomes. Let POP be the initial population.

(iv) *Step 2.* Set $i = 1$. While $i < P_{\text{size}}$, do the following.

(v) Perform a selection operation to select two parents from POP. Let $parent_1$ and $parent_2$ be the selected parents.

(vi) Generate a random number rand in [0, 1]. If rand $< p_c$, perform a crossover operation to generate $offspring_1$ and $offspring_2$. Otherwise, $offspring_1$ and $offspring_2$ inherit the information from $parent_1$ and $parent_2$, respectively.

(vii) Generate a random number rand in [0, 1]. If rand $< p_m$, perform a mutation operation for $offspring_1$ and $offspring_2$.

(viii) Evaluate the fitness value of $offspring_1$ and $offspring_2$.

(ix) If $offsping_1$ is better than the worst chromosome in POP, replace the worst chromosome with $offsping_1$. Repeat the same procedure for $offspring_2$.

(x) Let $i = i + 2$.

(xi) *Step 3.* Check the local search condition; if it is met, trigger a local search for the best chromosome in POP.

(xii) *Step 4.* Check the restart condition; if it is met, trigger the restart procedure for POP. Evaluate the fitness values of the chromosomes in POP.

(xiii) *Step 5.* Check the termination condition; if it is met, terminate and the best solution in POP is the final solution. Otherwise, go to Step 2.

4. Computational Experiments

4.1. Test Instance and Environment. Computational experiments were performed with randomly generated problems to evaluate the performance of the proposed algorithms. All algorithms were coded in Java, and the experiments were performed on a PC with a 3.2 GHz Intel Core i7-8700 CPU.

To generate problem instances, job information (processing times, sequence-dependent setup times SDST, and limited waiting times LWT) was generated according to the method presented in [3]. The processing times were generated from $U(1, 50)$, where $U(a, b)$ denotes the discrete uniform distribution with a range $[a, b]$. SDST and LWT were generated from $U(0, 50)$ and $U(1, 100)$, respectively. The due dates of the jobs were generated from $U(X(1 - T - R/2), X(1 - T + R/2))$, where X is the lower bound on the makespan, and T and R are the tardiness factor and range of due dates parameters, respectively. Here, to obtain the values of X, we assumed that SDSTs for each job i were set to $s_{ij} = \min(s_{ij} | \forall j)$ and LWTs were infinite; then X is obtained by Johnson's algorithm [33] that provides the minimum makespan in the typical two-machine flowshop. For the distribution of due dates, three levels for each of T and R were used; that is, $T = (0.2, 0.4, \text{and } 0.6)$ and $R = (0.2, 0.5, \text{and } 0.8)$.

4.2. GA Calibration. The performance of GA significantly depends on its operators and parameters. Hence, a calibration experiment is needed to select the best operators and parameters. The proposed GA has eight operators and parameters, which are listed below, and all possible combinations were evaluated to obtain the best among them.

(i) Selection: two levels (tournament and roulette)

(ii) Crossover: nine levels (OX1, OX2, OBX, PBX, PMX, SJX, SJX2, SBX, and SBX2)

(iii) Mutation: three levels (interchange, insertion, and swap)

(iv) Population size (P_{size}): three levels $(30, 50, 100)$

(v) Crossover probability (p_c): six levels $(0.2, 0.35, 0.5, 0.65, 0.8, 0.95)$

(vi) Mutation probability (p_m): four levels $(0.2, 0.4, 0.6, 0.8)$

(vii) Local search (P_l): three levels $(5, 10, 20)$

(viii) Restart (P_r): four levels $(25, 50, 75, 100)$

For these eight types of operators and parameters, 46,656 different combinations are possible. To simplify the experiment, this test is divided into two experiments. The first experiment was to select the best combination for selection, crossover, and mutation, and the second experiment was to find the best combination for the other parameters.

For the evaluation and selection for the GA operators in the first experiment, a full factorial design of selection, crossover, and mutation is considered; hence, 54 different algorithms are tested. The remaining parameters were fixed to $(P_{\text{size}}, p_c, p_m, P_l, P_r) = (50, 0.5, 0.4, 10, 50)$. Additionally, four levels were considered for the number of jobs ($n = 20, 50, 100$ and 200), and five instances for each combination (T, R, and n) were generated. That is, 180 problem instances were generated. For a termination criterion, the maximum CPU time was set to $30n$ milliseconds (ms). To compare different GAs for each problem instance, the relative deviation index (RDI) was considered as a measure; RDI is defined as $\text{RDUI} = (\text{Alg}_{\text{sol}} - \text{Min}_{\text{sol}})/(\text{Max}_{\text{sol}} - \text{Min}_{\text{sol}})$, where Alg_{sol} is the objective function value from the

current algorithm, and Min_{sol} and Max_{sol} are the minimum and maximum values among the objective function values obtained from the algorithms in this comparison, respectively. Particularly when $\text{Max}_{\text{sol}} = \text{Min}_{\text{sol}}$, RDI will be 0 for all the algorithms.

To see the effect of GA operators on the performance, analysis of variance (ANOVA) was performed using SPSS and the ANOVA table is shown in Table 2. As can be seen in the table, the performance of the proposed GA was significantly affected by the operators with a significance level of 0.05. According to the *F*-values, three operators were significantly effective on the RDI. Figure 1 illustrates the main effect plot to find the best operators with lower RDI values. As shown in the figure, roulette, OX2, and insertion showed better performance. This is possibly because the change of fore and rear parts in sequences can lead to searching for better sequences in this scheduling problem with sequence-dependent setup times. For mutation, insertion is observed to be the best, which has been regarded in various flowshop scheduling studies as the best mutation operator [32]. Therefore, roulette, OX2, and insertion were used in the following experiment.

The second experiment was performed to find the best combination of parameters of the proposed GA. Considering all levels of $(P_{\text{size}}, p_c, p_m, P_l, P_r)$, 864 different combinations were evaluated using the same approach as that for the operators. The ANOVA results are summarized in Table 3, which shows that p_c, p_m, and P_r are the most significant parameters that affect the GA performance. As shown in Figure 2, which illustrates the main effect plot of GA parameters, RDI had the lowest value at each of $(P_{\text{size}}, p_c, p_m, P_l, P_r) = (50, 0.35, 0.6, 5, 75)$, respectively. The *bath-curves* are observed for all parameters except P_l. That is, both higher and lower values from the selected values yielded worse results, validating the chosen range of values for these parameters in the experiments. For P_l, frequent local searches were more effective when the best solution in the population was not improved. Notably, the probability of mutation is larger than that of crossover. This is probably because a small change by a mutation may lead to a better sequence rather than crossover. That is, because limited waiting time constraints and sequence-dependent setup times were considered in this study, the solution may be improved by small changes in a job sequence. Thus, using the selected parameters, the proposed GA attempts to find better solutions by keeping the sequences in good solutions and making a few changes. In summary, the proposed GA uses roulette, OX2, and insertion for the operators and $P_{\text{size}} = 50$, $p_c = 0.35$, $p_m = 0.6$, $P_l = 5$, and $P_r = 75$ for the parameters.

4.3. Performance Evaluation. First, the performance of the proposed MILP formulation was evaluated using CPLEX 12.9. To avoid excessive computation times for CPLEX, the maximum CPU time limit was set to 3,600 seconds (s) for each instance. For the evaluation of MILP, four levels for the number of jobs ($n = 5, 10, 15,$ and 20) were considered, and five instances for each combination of $(T, R,$ and $n)s$ were generated. That is, 36 combinations for $(T, R,$ and $n)$ were considered, and 180 problem instances are generated. The results are summarized in Table 4, which shows the average computation time (ACPUT) and the number of instances (NI) that failed to find an optimal solution within the computation time limit (3,600 s). As can be seen in the table, CPLEX obtained optimal solutions for all problems up to $n = 10$ but could not find optimal solutions for eight problems with $n = 15$. Furthermore, for $n = 20$, no problems were solved to optimality within the computation time limit.

In general, due dates significantly affect due date-related measures (e.g., total tardiness, which is the objective function of the considered scheduling problem in this study). Hence, the impact of the due dates on the performance of CPLEX was analyzed. The computation times of CPLEX with respect to the due date parameters (T, R) are summarized in Table 5, when $n = 15$. As can be seen from the table, the performance of CPLEX was significantly affected by the values of T and R. Particularly, when $T = (0.2 \text{ and } 0.4)$, the instances took a significantly long computation time to be solved. However, when $T = 0.6$, the instances were solved within shorter computation times. That is, CPLEX can solve problems quickly when the due dates of jobs are tight. Regarding the range of due dates R, problems in which due dates of jobs are widely distributed were solved in a shorter computation time.

Next, the performances of the proposed list scheduling rules and MNEH algorithms were evaluated by comparison to solutions obtained from CPLEX. Here, let MNEH_x be an MNEH algorithm that starts with an initial solution obtained by a list scheduling rule x. Note that, for the instances not solved within the time limit, the solutions terminated at the time limit were compared. Table 6 presents the average gaps between the heuristics and CPLEX, $(\text{Heu}_{\text{sol}} - \text{CPLEX}_{\text{sol}})/\text{CPLEX}_{\text{sol}}$, and the number of instances where heuristics found solutions which were equal to or better than those of CPLEX. As can be observed in the table, MEDD and Greedy demonstrated good performance among the list scheduling rules, and $\text{MNEH}_{\text{MEDD}}$ was the best performing heuristic algorithm. However, the average gaps from CPLEX were too large. These results explain why a metaheuristic algorithm, GA, was needed to find good solutions within a short computation time in this study.

To evaluate the performance of the suggested GA, a performance evaluation by comparing it with other algorithms was required. However, to the best of our knowledge, there are no algorithms for the same problem considered in this study. Thus, the evaluation was performed by comparing the proposed GA to CPLEX in small-sized problem instances ($n = 5$, 10, and 15) and the proposed heuristic algorithms in medium- and large-sized problem instances ($n = 20$, 50, 100, and 200). Through these comparisons, the effectiveness of the solutions from the proposed GA was evaluated. In these experiments, the maximum computation times of GA were set to four levels $(50n, 100n, 500n,$ and $1,000n)$ms. Let GA_x be the genetic algorithm in which the maximum computation time is set to $x \cdot n$ ms. The effectiveness of GA solutions compared to those from CPLEX is shown in Table 7. As can be seen from the table, the suggested GA found optimal solutions for

Table 2: ANOVA for the effect of GA operators on the performance.

Sources	Sum. squares	DF	MS	F	p-value
Corrected model	200.087	53	3.775	66.464	≤0.001
Intercept	2110.730	1	2110.730	37160.094	≤0.001
Selection	9.658	1	9.658	170.026	≤0.001
Crossover	35.657	8	4.457	78.468	≤0.001
Mutation	116.139	2	58.070	1022.336	≤0.001
Selection × crossover	27.605	8	3.451	60.749	≤0.001
Selection × mutation	0.588	2	0.294	5.175	0.006
Crossover × mutation	7.298	16	0.456	8.030	≤0.001
Selection × crossover × mutation	3.143	16	0.196	3.458	≤0.001
Error	549.038	9666	0.057	–	–
Total	2859.855	9720	–	–	–
Corrected total	749.125	9719	–	–	–

Significance level = 0.05.

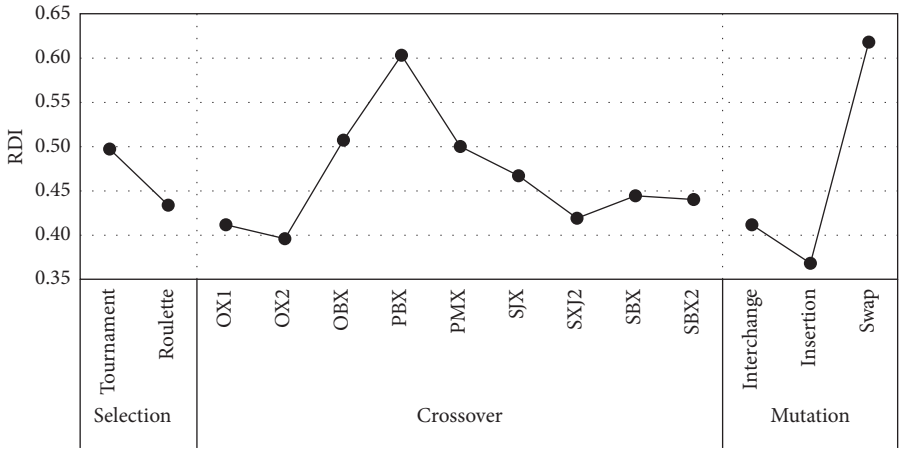

Figure 1: Main effect plot of GA operators.

almost all problem instances. Particularly, GA outperformed CPLEX for $n = 20$; a minus value means that the objective value of solutions from GA is less than that from CPLEX. This is probably because CPLEX fails to obtain optimal solutions within the maximum CPU time for $n = 20$.

For the medium and large problem instances, the GA solutions were compared with those from the MNEH algorithms. The results of this test are summarized in Table 8, which shows RDI values and the number of instances where a heuristic finds the best solutions among the heuristic solutions. The table illustrates that GAs outperformed MNEH algorithms, although GAs took longer CPU times. Additionally, in the case where GA_{50} and $MNEH_{Greedy}$ had similar computation times (approximately 10 s), GA_{50} still showed better performance. Overall, although GA requires a longer CPU time, GA solves the problems better, and the CPU time of GA_{50} is also reasonably short.

Additionally, the effectiveness of the proposed GA was evaluated through comparison with a simulated annealing (SA) algorithm, which is also a widely used metaheuristic for various optimization problems. SA attempts to improve an initial solution repeatedly by making small alterations until further improvements cannot be made by such alterations. The SA algorithm can avoid entrapment in a local optimum by allowing occasional uphill moves that deteriorate the objective function value. In this experiment, a simple SA algorithm is devised for comparison with the proposed GA. In this SA algorithm, a neighborhood solution (σ') of a solution (σ) is generated using an insertion method, which is reported as the best mutation in the previous experiment. If $\Delta = T(\sigma') - T(\sigma) < 0$, where $T(x)$ is the total tardiness of solution x, σ is replaced with σ'. Otherwise, if $\Delta \geq 0$, the replacement is done with probability $e^{-\Delta/\tau}$, where τ is a parameter called the temperature. This replacement procedure is repeated L times at a temperature, where L is called the epoch length, which is set to n^2. The temperature is initially set to $\tau_0 = 100$ and decreases gradually by a cooling function, which is set as $\tau_{x+1} = \tau_x - \alpha$, where α is a parameter that is set to 0.025 in this algorithm. The algorithm is terminated when the temperature decreases to zero or the computation time reaches the maximum computation time. The procedure of this SA algorithm is summarized below.

4.3.1. Procedure: SA

(i) *Step 0.* Set initial values for parameters used in the algorithm ($\tau_0 = 100, L = n^2$ and $\alpha = 0.025$). Let $\tau = \tau_0$ and $l = 0$.

Table 3: ANOVA for the effect of GA parameters on the performance.

Sources	Sum. squares	DF	MS	F	p-value
Corrected model	157.103	863	0.182	3.329	≤0.001
Intercept	29023.089	1	29023.089	530673.217	≤0.001
P_{size}	1.629	2	0.815	14.897	≤0.001
p_c	23.043	5	4.609	84.267	≤0.001
p_m	40.992	3	13.664	249.839	≤0.001
P_r	23.989	3	7.996	146.206	≤0.001
P_l	0.197	2	0.098	1.800	0.165
$P_{\text{size}} \times p_c$	1.070	10	0.107	1.957	0.034
$P_{\text{size}} \times p_m$	1.585	6	0.264	4.831	≤0.001
$P_{\text{size}} \times P_r$	1.922	6	0.320	5.859	≤0.001
$P_{\text{size}} \times P_l$	0.528	4	0.132	2.413	0.047
$p_c \times p_m$	12.065	15	0.804	14.707	≤0.001
$p_c \times P_r$	12.324	15	0.822	15.023	≤0.001
$p_c \times P_l$	0.752	10	0.075	1.375	0.185
$p_m \times P_r$	3.827	9	0.425	7.774	≤0.001
$p_m \times P_l$	0.355	6	0.059	1.083	0.370
$P_r \times P_l$	2.804	6	0.467	8.544	≤0.001
$P_{\text{size}} \times p_c \times p_m$	1.502	30	0.050	0.916	0.599
$P_{\text{size}} \times p_c \times P_r$	1.334	30	0.044	0.813	0.754
$P_{\text{size}} \times p_c \times P_l$	0.845	20	0.042	0.773	0.750
$P_{\text{size}} \times p_m \times P_r$	0.280	18	0.016	0.285	0.999
$P_{\text{size}} \times p_m \times P_l$	0.454	12	0.038	0.692	0.761
$P_{\text{size}} \times P_r \times P_l$	5.064	12	0.422	7.716	≤0.001
$p_c \times p_m \times P_r$	4.008	45	0.089	1.628	0.005
$p_c \times p_m \times P_l$	1.068	30	0.036	0.651	0.928
$p_c \times P_r \times P_l$	0.914	30	0.030	0.557	0.976
$p_m \times P_r \times P_l$	0.676	18	0.038	0.687	0.828
$P_{\text{size}} \times p_c \times p_m * P_r$	2.371	90	0.026	0.482	1.000
$P_{\text{size}} \times p_c * p_m \times P_l$	1.741	60	0.029	0.531	0.999
$P_{\text{size}} \times p_c \times P_r \times P_l$	1.798	60	0.030	0.548	0.998
$P_{\text{size}} \times p_m \times P_r \times P_l$	1.024	36	0.028	0.520	0.992
$p_c \times p_m \times P_r \times P_l$	2.236	90	0.025	0.454	1.000
$P_{\text{size}} \times p_c \times p_m \times P_r \times P_l$	4.704	180	0.026	0.478	1.000
Error	8458.303	154656	0.055	—	—
Total	37638.495	155520	—	—	—
Corrected total	8615.406	155519	—	—	—

Significance level = 0.05.

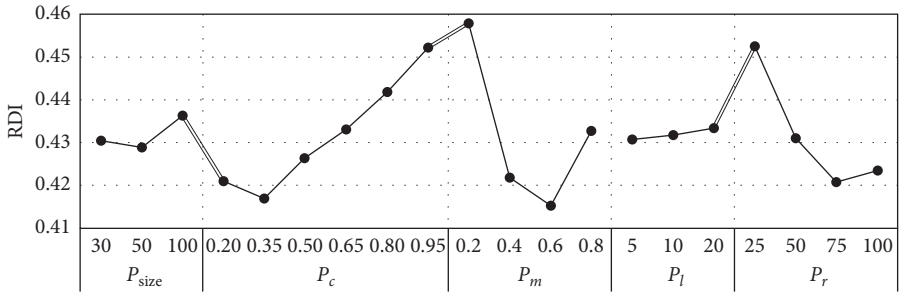

Figure 2: Main effect plots of GA parameters.

Table 4: Performance of CPLEX.

n	ACPUT(s)	NI
5	0.08	0
10	3.22	0
15	1419.53	8
20	3600	45

(ii) *Step 1.* Set an initial solution σ as the best solution among those obtained with MNEH_{EDD}, $\text{MNEH}_{\text{MEDD}}$, $\text{MNEH}_{\text{Slack}}$, and $\text{MNEH}_{\text{Greedy}}$. Let $\sigma^* \longleftarrow \sigma$.

(iii) *Step 2.* If $\tau = 0$, terminate and return σ^* as the final solution. Otherwise, let $l = 0$.

Table 5: Effect of due date parameters on the performance of CPLEX.

T	$R = 0.2$	0.5	0.8	Average
0.2	1435.14	1580.84	2313.12	1776.37
0.4	2172.74	2473.35	880.95	1842.35
0.6	905.23	255.19	759.17	639.86
Average	1504.37	1436.46	1317.75	1419.53

Table 6: Performance of the heuristic algorithms (versus CPLEX).

Algorithm	$n = 5$	10	15	20	Avg, sum
EDD	0.97, 0[a]	2.07, 0	10.58, 0	4.52, 0	4.53, 0
MEDD	0.30, 9	1.1, 0	4.69, 0	2.03, 0	2.03, 9
Slack	0.88, 3	1.48, 0	6.17, 0	2.73, 0	2.81, 3
Greedy	0.44, 9	1.02, 0	4.39, 0	2.25, 0	2.02, 9
$MNEH_{EDD}$	0.06, 29	0.17, 9	0.71, 0	0.25, 0	0.3, 40
$MNEH_{MEDD}$	0.05, 31	0.16, 7	0.66, 0	0.26, 2	0.28, 40
$MNEH_{Slack}$	0.04, 32	0.21, 7	0.71, 0	0.3, 1	0.31, 40
$MNEH_{Greedy}$	0.08, 29	0.28, 1	1.52, 0	0.38, 0	0.56, 31

[a]Average error gap, number of instances (out of 45 instances) for which the algorithm found solutions better than or equal to those from CPLEX.

Table 7: Evaluation of GA compared to CPLEX.

n	GA_{50}	GA_{100}	GA_{500}	GA_{1000}
5	0.00, 45[a]	0.00, 45	0.00, 45	0.00, 45
10	0.00, 44	0.00, 45	0.00, 45	0.00, 45
15	0.01, 35	0, 39	0, 42	−0.01, 43
20	−0.14, 38	−0.15, 39	−0.17, 41	−0.18, 43

[a]Average error gap, number of instances (out of 45 instances) for which the algorithm found solutions better than or equal to those from CPLEX.

Table 8: Comparison between the proposed heuristic algorithms.

Algorithm	RDI				CPUT			
	$n = 20$	50	100	200	20	50	100	200
$MNEH_{EDD}$	0.73, 0[a]	0.67, 1	0.57, 1	0.55, 0	<0.01	0.03	0.38	4.63
$MNEH_{MEDD}$	0.72, 0	0.64, 0	0.6, 0	0.96, 0	<0.01	0.04	0.4	6.12
$MNEH_{Slack}$	0.86, 0	0.93, 1	0.97, 0	0.66, 0	<0.01	0.04	0.44	6.12
$MNEH_{Greedy}$	0.10, 15	0.7, 1	0.66, 0	0.23, 4	<0.01	0.04	0.59	10.19
GA_{50}	0.07, 23	0.13, 6	0.19, 4	0.17, 4	1	2.5	5	10
GA_{100}	0.05, 27	0.1, 9	0.14, 5	0.15, 4	2	5	10	20
GA_{500}	0.02, 38	0.02, 33	0.05, 13	0.08, 5	10	25	50	100
GA_{1000}	0.00, 43	0.00, 45	0, 45	0.00, 45	20	50	100	200

[a]Average RDI, number of instances (out of 45 instances) for which the algorithm found solutions better than or equal to those from CPLEX.

(iv) *Step 3.* Generate a neighborhood solution σ' of σ by an insertion method.

(v) *Step 4.* Compute $\Delta = T(\sigma') - T(\sigma)$. If $\Delta < 0$, let $\sigma \leftarrow \sigma'$ and $\sigma^* \leftarrow \sigma$. Otherwise, let $\sigma \leftarrow \sigma'$ with probability $e^{-\Delta/\tau}$.

(vi) *Step 5.* Let $l = l + 1$. If $l < L$, go to Step 3. Otherwise, $\tau = \tau - \alpha$ and go to Step 2.

For fair comparison, the same maximum completion time is set for SA. Let SA_{500} and SA_{1000} be the simulated annealing algorithms in which the maximum CPU times are set to $500n$ and $1000n$, respectively. Here, GA_{500} and GA_{1000} are compared with SA_{500} and SA_{1000}, respectively.

The results of comparisons with the SA algorithms are summarized in Table 9, which shows the average rate of Alg_{so}/max_{sol}, where Alg_{sol} is the objective function value from the current algorithm, and $Max_{sol} = \max\{GA_x, SA_x\}$, where $x = 500$ and 1000. Note that because the objective value can be zero, the maximum values are used as the denominator rather than the minimum value to avoid dividing by zero; and if Max_{sol} is zero, then the rate will be zero. According to this measurement, algorithms with low

Table 9: Comparison between GA and SA.

n	GA_{500}	SA_{500}	GA_{1000}	SA_{1000}
20	0.966	0.983	0.966	0.987
50	0.907	0.908	0.926	0.903
100	0.818	0.967	0.828	0.968
200	0.844	0.934	0.834	0.926

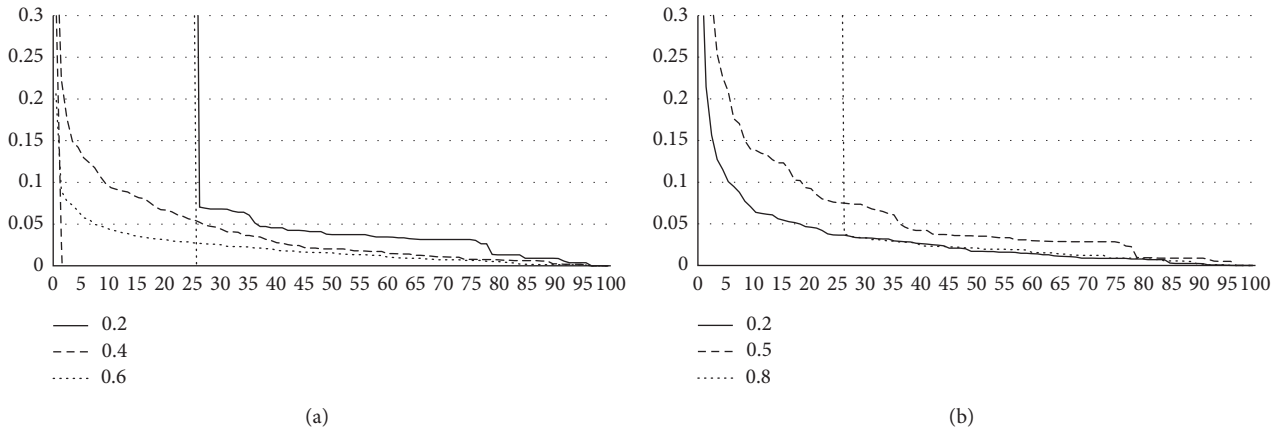

Figure 3: Convergence rate of GA solutions according to the due date parameters, T and R. (a) Convergence rate according to T values. (b) Convergence rate according to R values.

rate value are better algorithms. As can be seen from Table 9, the proposed GA outperformed a simple SA except for $n = 50$. Additionally, the outperformance is clearer in large-sized problems. Although the devised SA for this test was simple, the effectiveness of the proposed GA is validated from the results.

Finally, to check the effects of due date parameters (T and R) on the performance of GA, convergence rate charts are proposed in Figure 3. For T values, instances generated with large T values converged more quickly in the early generations. That is, when the due dates of jobs are tight, GA can find good solutions in early generations. This result is like that of CPLEX. On the other hand, instances with $R = 0.5$ converged at later generations. That is, when the range of due dates is moderate, GA requires more computation time to find good solutions. According to the results, instances with a moderate range of due dates are more complicated than those of other cases.

5. Conclusion

In this study, we considered a two-machine flowshop scheduling problem with limited waiting time constraints and sequence-dependent setup times, which is very important in semiconductor manufacturing systems. A mixed-integer programming formulation is provided to describe the considered problem clearly and used to obtain optimal solutions with CPLEX. However, CPLEX could not obtain optimal solutions when $n = 20$ because the considered problem is NP-complete. Thus, to obtain good solutions in a short computation time, three types of heuristic algorithms were proposed: list scheduling methods, a constructive heuristic algorithm, and a genetic algorithm (GA). To find the best combination of the GA, 46,656 different combinations were evaluated, and the suggested GA outperformed the other heuristics. Additionally, GA found optimal solutions for all instances with $n = 5$, 10, and 15 and showed better performance compared to CPLEX when $n = 20$. Thus, it can be said that GA works well to obtain acceptable solutions within a reasonably short computation time.

Although the limited waiting time constraint and sequence-dependent setup times are very important characteristics in semiconductor manufacturing systems, there are only a few studies considering both characteristics together. Thus, we need to develop more effective and efficient methodologies including many types of metaheuristics. On the other hand, since it is very difficult to find optimal solutions for the considered problem, it is necessary to develop effective lower bounding strategies to evaluate solutions obtained by heuristic algorithms. Also, different waiting time constraints like overlapping waiting time constraints in [23] can be considered in the future research. Finally, since the problem considered in this study is a simple type of flowshop, this problem can be extended to more practical flowshops like hybrid flowshops in which there are parallel machines in each stage.

Data Availability

All data for the computational experiments are generated randomly and the method for generating data is written in Section 4.1 in the paper.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) Grant Funded by the Korea Government (MSIT) (no. NRF-2020R1G1A1006268).

References

[1] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[2] C. Koulamas, "The total tardiness problem: review and extensions," *Operations Research*, vol. 42, no. 6, pp. 1025–1041, 1994.

[3] Y.-J. An, Y.-D. Kim, and S.-W. Choi, "Minimizing makespan in a two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times," *Computers & Operations Research*, vol. 71, pp. 127–136, 2016.

[4] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *Journal of the ACM*, vol. 9, no. 1, pp. 61–63, 1962.

[5] R. Bellman, A. O. Esogbue, and I. Nabeshima, *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, New York, NY, USA, 1982.

[6] B. D. Corwin and A. O. Esogbue, "Two machine flow shop scheduling problems with sequence dependent setup times: a dynamic programming approach," *Naval Research Logistics Quarterly*, vol. 21, no. 3, pp. 515–524, 1974.

[7] J. N. D. Gupta, "A search algorithm for the generalized flowshop scheduling problem," *Computers & Operations Research*, vol. 2, no. 2, pp. 83–90, 1975.

[8] J. N. D. Gupta and W. P. Darrow, "The two-machine sequence dependent flowshop scheduling problem," *European Journal of Operational Research*, vol. 24, no. 3, pp. 439–446, 1986.

[9] R. Ruiz, C. Maroto, and J. Alcaraz, "Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics," *European Journal of Operational Research*, vol. 165, no. 1, pp. 34–54, 2005.

[10] X. Li and Y. Zhang, "Adaptive hybrid algorithms for the sequence-dependent setup time permutation flow shop scheduling problem," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 3, pp. 578–595, 2012.

[11] K. Peng, L. Wen, R. Li, L. Gao, and X. Li, "An effective hybrid algorithm for permutation flow shop scheduling problem with setup time," *Procedia CIRP*, vol. 72, pp. 1288–1292, 2018.

[12] R. Vanchipura, R. Sridharan, and A. S. Babu, "Improvement of constructive heuristics using variable neighbourhood descent for scheduling a flow shop with sequence dependent setup time," *Journal of Manufacturing Systems*, vol. 33, no. 1, pp. 65–75, 2014.

[13] A. Sioud and C. Gagne, "Enhanced migrating birds optimization algorithm for the permutation flow shop problem with sequence dependent setup times," *European Journal of Operational Research*, vol. 264, no. 1, pp. 66–73, 2018.

[14] Q.-K. Pan, L. Gao, X.-Y. Li, and K.-Z. Gao, "Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times," *Applied Mathematics and Computation*, vol. 303, no. 15, pp. 89–112, 2017.

[15] J.-P. Huang, Q.-K. Pan, and L. Gao, "An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times," *Swarm and Evolutionary Computation*, vol. 59, Article ID 100742, 2020.

[16] F. B. Ozsoydan and M. Sağir, "Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flowshop scheduling problem with sequence dependent setup times: a case study at a manufacturing plant," *Computers and Operations Research*, vol. 125, Article ID 105044, 2021.

[17] R. Ramezanian, M. M. Vali-Siar, and M. Jalalian, "Green permutation flowshop scheduling problem with sequence-dependent setup times: a case study," *International Journal of Production Research*, vol. 57, no. 10, pp. 3311–3333, 2019.

[18] Y. Wang and X. Li, "A hybrid local search algorithm for the sequence dependent setup times flowshop scheduling problem with makespan criterion," *Sustainability*, vol. 9, no. 12, p. 2318, 2017.

[19] D.-L. Yang and M.-S. Chern, "A two-machine flowshop sequencing problem with limited waiting time constraints," *Computers & Industrial Engineering*, vol. 28, no. 1, pp. 63–70, 1995.

[20] J.-L. Bouquard and C. Lenté, "Two-machine flow shop scheduling problems with minimal and maximal delays," *4OR*, vol. 4, no. 1, pp. 15–28, 2006.

[21] B.-J. Joo and Y.-D. Kim, "A branch-and-bound algorithm for a two-machine flowshop scheduling problem with limited waiting time constraints," *Journal of the Operational Research Society*, vol. 60, no. 4, pp. 572–582, 2009.

[22] T.-S. Yu, H.-J. Kim, and T.-E. Lee, "Minimization of waiting time variation in a generalized two-machine flowshop with waiting time constraints and skipping jobs," *IEEE Transactions on Semiconductor Manufacturing*, vol. 30, no. 2, pp. 155–165, 2017.

[23] H.-J. Kim and J.-H. Lee, "Three-machine flow shop scheduling with overlapping waiting time constraints," *Computers and Operations Research*, vol. 101, pp. 93–102, 2019.

[24] E. Dhouib, J. Teghem, and T. Loukil, "Lexicographic optimization of a permutation flow shop scheduling problem with time lag constraints," *International Transactions in Operational Research*, vol. 20, no. 2, pp. 213–232, 2013.

[25] B. Wang, K. Huang, and T. Li, "Permutation flowshop scheduling with time lag constraints and makespan criterion," *Computers & Industrial Engineering*, vol. 120, pp. 1–14, 2018.

[26] I. Hamdi and T. Loukil, "Minimizing total tardiness in the permutation flowshop scheduling problem with minimal and maximal time lags," *Operational Research*, vol. 15, no. 1, pp. 95–114, 2015.

[27] S. Wang, X. Wang, and L. Yu, "Two-stage no-wait hybrid flow-shop scheduling with sequence-dependent setup times," *International Journal of Systems Science: Operations & Logistics*, vol. 7, no. 3, pp. 291–307, 2020.

[28] C.-Y. Cheng, K.-C. Ying, S.-F. Li, and Y.-C. Hsieh, "Minimizing makespan in mixed no-wait flowshops with sequence-dependent setup times," *Computers and Industrial Engineering*, vol. 130, pp. 338–347, 2019.

[29] F. L. Rossi and M. S. Nagano, "Heuristics for the mixed no-idle flowshop with sequence-dependent setup times and total flowtime criterion," *Expert Systems with Applications*, vol. 125, no. 1, pp. 40–54, 2019.

[30] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[31] J. H. Holland, *Adaptation in Natural and Artificial System*, University of Michigan Press, Ann Arbor, MI, USA, 1975.
[32] R. Ruiz and C. Maroto, "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility," *European Journal of Operational Research*, vol. 169, no. 3, pp. 781–800, 2006.
[33] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954.