*Article*

# Metaheuristics for a Flow Shop Scheduling Problem with Urgent Jobs and Limited Waiting Times

**BongJoo Jeong [1], Jun-Hee Han [2] and Ju-Yong Lee [3,*]**

1 Department of Business Administration, Kongju National University, Kongju-si 32588, Korea; jbj@kongju.ac.kr
2 Department of Industrial and Management Engineering, Dong-A University, Busan 49315, Korea; jheehan@dau.ac.kr
3 Division of Business Administration & Accounting, Kangwon National University, Chuncheon-si 24341, Korea
* Correspondence: jy.lee@kangwon.ac.kr; Tel.: +82-33-250-6150

**Abstract:** This study considers a scheduling problem for a flow shop with urgent jobs and limited waiting times. The urgent jobs and limited waiting times are major considerations for scheduling in semiconductor manufacturing systems. The objective function is to minimize a weighted sum of total tardiness of urgent jobs and the makespan of normal jobs. This problem is formulated in mixed integer programming (MIP). By using a commercial optimization solver, the MIP can be used to find an optimal solution. However, because this problem is proved to be NP-hard, solving to optimality requires a significantly long computation time for a practical size problem. Therefore, this study adopts metaheuristic algorithms to obtain a good solution quickly. To complete this, two metaheuristic algorithms (an iterated greedy algorithm and a simulated annealing algorithm) are proposed, and a series of computational experiments were performed to examine the effectiveness and efficiency of the proposed algorithms.

**Keywords:** scheduling; flow shop; urgent jobs; limited waiting times; metaheuristic

check for updates

**Citation:** Jeong, B.; Han, J.-H.; Lee, J.-Y. Metaheuristics for a Flow Shop Scheduling Problem with Urgent Jobs and Limited Waiting Times. *Algorithms* **2021**, *14*, 323. https://doi.org/10.3390/a14110323

Academic Editors: Yun-Chia Liang, Mehmet Fatih Tasgetiren, Quan-Ke Pan, Hsiang-Ling Chen and Frank Werner

Received: 5 October 2021
Accepted: 2 November 2021
Published: 3 November 2021

## 1. Introduction

This study deals with a scheduling problem in a flow shop with urgent jobs and limited waiting times. *Urgent jobs* refer to jobs that must be processed faster due to higher urgency than normal jobs. *Limited waiting time* is to limit the time that a job completed on the first machine waits to start the next process on the second machine. These two scheduling requirements are very common in semiconductor manufacturing processes.

In general, semiconductor manufacturing systems are operated in high-variety small-volume production and make-to-order policies. Nowadays, because the semiconductor foundry market is growing rapidly, customer order management and fast demand response capability are considered as very important competitive factors. To cope with this environment, jobs are classified into two (or more) groups depending on the urgencies of customer orders, i.e., a normal job and an urgent job. In this case, for normal jobs, a schedule is established with the objective of minimizing the maximum completion time (makespan) to raise throughput rate. However, urgent jobs are required to start processing as early as possible when they arrive at workstations. To accomplish this, each urgent job has a calculated due date, assuming that it is processed through all steps without waiting immediately upon arrival. That is, the due date on each step is equal to the earliest possible completion time. Based on this assumption, the objective of minimizing total tardiness can be considered as the scheduling measure for the urgent jobs. If normal and urgent jobs are required to be scheduled together on the same machine or shop, the scheduling measure is treated as a multi-objective because the makespan for normal jobs and total tardiness for urgent jobs are combined. Especially in semiconductor manufacturing, the urgent job is called *hot-lot* and managed with a high priority. The urgent jobs can be found in many make-to-order manufacturing systems besides semiconductors. In general, prototypes, samples, and lots to make up for the defective products are regarded as urgent jobs, and such jobs account for 30–50% of all jobs [1]. So, it is essential to make an effective and efficient schedule while considering the urgent jobs.

Semiconductor manufacturing has many chemical processes. To prevent quality problems of wafer lots caused by the characteristics of chemical processes, waiting time constraints are required between consecutive processes (or stages). For example, the waiting time constraints are set between operations in etch-oxidation-deposition-diffusion work areas [2]. If the chemical treatment is effective only within a certain period of time, the chemically treated wafer lot has to be entered into the next process before it exceeds the limited time. In the case that the first process of two consecutive processes is such a chemical process, the limited waiting time constraint is applied between the two processes. Moreover, wafers are naturally oxidized if wafers are left in the air for a certain period of time [3]. On the other hand, the waiting time constraint is also required to reduce the possibility of contamination that causes quality degradation. When the surface of a wafer is exposed to air for a long time before the next process, impurities such as particles may sit on the surface, which can cause a serious quality problem. To prevent this problem, the exposure time is limited by setting the waiting time constraint. A representative example with the waiting time constraint is a two-stage flow shop of clean-diffusion consecutive steps. In order to prevent the above problems, the waiting times between the clean and diffusion processes are limited by the constraint. If a wafer exceeds the constrained waiting time, the cleaning process is reworked, or if contamination is severe, the wafer is scrapped. The rework and scrap degrade productivity and cause losses of management. Therefore, the waiting time constraint should be considered for the flow shops where the above quality problems are likely to occur.

This study considers a scheduling problem for a flow shop with urgent jobs and a waiting time constraint. As mentioned, the urgent jobs want to be processed without causing tardiness. Due to the urgent jobs, normal jobs may be relatively delayed, but their waiting times must also be kept. Therefore, it is essential to develop an effective scheduling algorithm to efficiently operate such a manufacturing system. This scheduling problem can be proved to be NP-hard, because a two-machine flow shop total tardiness problem is NP-hard in the strong sense [4], and also a two-machine flow shop makespan problem with limited waiting time is NP-hard [5]. Since the considered problem is NP-hard, it is very tough to find an optimal solution. Hence, this study adopts metaheuristic algorithms to find a good (or near-optimal) solution. To the best of our knowledge, this study is the first attempt to consider the urgent jobs and waiting times simultaneously. In addition, the main contribution is that this work is expected to contribute to the improvement of productivity of semiconductor manufacturing systems.

The remainder of this paper is structured as follows. Section 2 introduces previous studies related to this study. Section 3 details the considered scheduling problem with assumptions and presents a mixed integer programming formulation. Section 4 describes the heuristic algorithms for the problem, and Section 5 reports the performance of the proposed algorithms through computational experiments. Finally, Section 6 discusses and concludes this paper with future research directions.

## 2. Previous Studies

This section introduces the previous studies related to the scheduling problem considered in this study.

First, the considered problem with urgent jobs can be classified as a multi-objective scheduling problem because two types of measures are combined, i.e., the makespan for normal jobs and total tardiness for urgent jobs. Recently, multi-objective scheduling problems are becoming popular as the manufacturing environments become more complex. Liang et al. [6] proposed a self-adaptive differential evolution algorithm for minimizing the maximum tardiness and makespan simultaneously in a flow shop with limited buffers. Anjana et al. [7] provided four metaheuristics for a flow shop that requires sequence-dependent setup times, with the objective of minimizing makespan and mean tardiness together, while Rifai et al. [8] considered a distributed reentrant permutation flow shop with sequence-dependent setup times for minimizing makespan, tardiness and production costs. Recently, multi-objective flow shop problems derived from energy consumption issues were considered in [9–11]. They suggested metaheuristic algorithms to minimize total energy consumption and makespan simultaneously.

However, the combined objective of this problem is different from a typical multi-objective. In this study, the objective functions for the two classes are different because jobs are classified into two classes depending on their urgencies. On the contrary, in the typical multi-objective problems, two or more objectives are applied in common to all jobs. Therefore, this problem can be regarded as a multi-agent problem that is the special case of multi-objective problems [12]. In multi-agent problems, there are two or more agents, and each agent has a different objective function for its own jobs. If the different classes with different objectives are defined as different agents, this problem can be regarded as a multi-agent problem.

Most studies on multi-agent scheduling considered problems for a single machine. Baker et al. [13] developed algorithms for the multi-agent problem of a single system, considering the maximum delay, total weighted completion times and makespan for objective function. Agnetis et al. [12] proposed algorithms for getting constrained optimal and pareto optimal solutions for the multi-agent on single machine problems and proved that the problem is NP-hard. Ng et al. [14] studied the problem of two-agents on a single machine with the objective functions that the total completion time of the first agent is minimized and the number of tardy jobs of the second agent cannot exceed a predetermined number. Leung et al. [15] proved that the same problem is NP-hard even if the number of tardy jobs for the second agent is limited to zero. Cheng et al. [16] proposed an algorithm for a two-agent problem on a single machine with the objective of minimizing the total weighted number of tardy jobs of each agent, and they also studied a multi-agent single machine problem with the max-form objective functions in [17]. Liu et al. [18] studied a single machine with a two-agent problem in which each job has a linear deteriorated processing time. Additionally, polynomial time solution algorithms for a single-machine with two agents were suggested in [19,20].

For a flow shop with multi agents, there have been only a few studies. Agnetis et al. [12] studied a two-agent problem on a flowshop for minimizing makespan of jobs in each agent. Lee et al. [21] developed a metaheuristic algorithm and a branch and bound algorithm to solve a two-agent problem in a two-machine flowshop. In addition, Lee et al. [22] studied a flowshop problem with two agents where the objective of each agent is to minimize the number of tardy jobs and the total tardiness, respectively. Mor et al. [23] studied three different flowshop problems with two agents and developed polynomial time solution algorithms for each problem. Fan and Cheng [24] proposed a linear programming-based approximation algorithm for a two-agent flowshop problem. Jeong and Shim [1] proposed a metaheuristic algorithm for a reentrant flowshop problem with two agents. Jeong et al. [25] presented a two-machine flowshop problem considering urgent jobs and developed metaheuristic algorithms and a branch and bound algorithm. Azerine et al. [26] considered a two-machine no-wait flow shop problem with two competing agents and proposed a branch and bound algorithm for small size problems and tabu search metaheuristics for large sized problems.

Now, a survey on flow shop problems with limited waiting times is provided. Most studies on flow shops with limited waiting times considered two-machine problems. For the objective of minimizing the makespan, the two-machine flow shop problem was proved to be NP-hard in [5]. To find an optimal solution for the problem, several branch and bound algorithms were developed by [5,27,28]. In addition, the reversibility property for the scheduling problem was proved and a constructive heuristic algorithm based on an insertion mechanism was developed in [2]. On the other hand, there are also studies with different scheduling measures. Hamdi and Loukil [29] developed a heuristic algorithm and a Lagrangian relaxation-based lower bound strategy for minimizing total tardiness, while Dhouib et al. [30] proposed simulated annealing algorithms to hierarchically minimize the number of tardy jobs and the makespan for a flow shop.

In addition, there have been studies considering variant problems with waiting time limits. Kim and Lee [31] developed a branch and bound algorithm to minimize the makespan in a three-machine flow shop with overlapping waiting time constraints. Furthermore, An et al. [3] suggested a branch and bound algorithm and heuristic algorithms for a two-machine flow shop with the waiting time constraints and sequence-dependent setup times to minimize the makespan, while Lee [32] provided a genetic algorithm to minimize total tardiness for the same flow shop. In addition, a case in which several jobs can skip the first stage was considered in [33], and mathematical properties for the problem were proposed. In addition, for a problem with a batch machine followed by a discrete machine, a hybrid membrane computing metaheuristic algorithm was developed in [34] to minimize the makespan. Additionally, there are recent studies considering flow shop problems with various scheduling requirements as well as limited waiting times [35–41].

## 3. Problem Description and an MIP Model

This section describes the considered problem in more detail with assumptions made in this study and provides a mixed-integer programming (MIP) formulation. There are $n$ independent jobs ($i = 1, \ldots, n$) to be scheduled in a two-stage flow shop with limited waiting times between the stages. The $n$ jobs belong to one of two classes, $A$ or $B$, according to their urgencies. Two classes $A$ and $B$ represent each class of urgent jobs and normal jobs, respectively. Accordingly, let $J_A$ and $J_B$ be the sets of jobs in classes $A$ and $B$, respectively. The objective function of this scheduling problem is the weighted sum of makespan for normal jobs and total tardiness for urgent jobs, which is to be minimized. For this problem, only permutation schedules are considered. In other words, jobs are processed on the two machines in the same order. Note that, although the permutation schedule does not guarantee optimality for the scheduling measures, semiconductor manufacturing systems process wafer lots in permutation schedules for ease of lot management and flexibility in material handling, and because of limited buffer spaces [3]. The following assumptions are also made this study:

1. no job can be preempted;
2. machines do not fail (no breakdown);
3. all normal jobs are available to be processed at time zero (beginning of the scheduling horizon);
4. release times and arriving times of urgent jobs are positive and given in advance;
5. the due date of an urgent job $i$ is set to the earliest possible completion time assuming that the urgent job $i$ is started immediately as arrived at the flow shop and processed without waiting, i.e., $d_i = r_i + p_{i1} + p_{i2}$.

Table 1 represents the notation for parameters and decision variables used to describe the MIP and proposed heuristic algorithms throughout this paper.

**Table 1.** Notation for parameters and decision variables.

| Symbol | Definition |
|---|---|
| $i$ | index for jobs ($i = 1, 2, \ldots, n$) |
| $k$ | index for machines ($k = 1, 2$) |
| $t$ | index for positions in a sequence |
| $[t]$ | index of the job in the $t$th position in a sequence |
| $p_{ik}$ | processing time of job $i$ on machine $k$ |
| $r_i$ | release time of job $i$ (=0 if job $i$ is a normal job) |
| $w_i$ | limited waiting time of job $i$ |
| $d_i$ | due date of job $i$ (=$r_i + p_i$ for $i \in J_A$, and a large number for $i \in J_B$) |
| $y_i$ | =1 if job $i$ is a normal job, and 0 otherwise |
| $x_{it}$ | =1 if job $i$ is placed in the $t$th position in a sequence, and 0 otherwise |
| $c_{tk}$ | completion time of the $t$th position on machine $k$ |
| $c_i$ | completion time of job $i$ on machine 2 |
| $\tau_i$ | tardiness of job $i$ |
| $Z$ | the makespan of normal jobs |
| $\alpha$ | weight for total tardiness of urgent jobs ($0 \le \alpha \le 1$) |
| $1 - \alpha$ | weight for the makespan of normal jobs |
| $M$ | a large number |
| $S$ | (partial or complete) schedule |
| $Si$ | a schedule in which job $i$ is added to the end of $S$ |
| $C(S)$ | completion time of $S$ |
| $U_A$ ($U_B$) | set of urgent (normal) jobs not scheduled yet |
| $obj(S)$ | objective function value of $S$ |

The following is the MIP for the scheduling problem considered in this study.

$$[\text{P}] \quad \text{minimize} \quad \alpha \sum\nolimits_i \tau_i + (1-\alpha)Z \tag{1}$$

$$\text{subject to} \quad \sum\nolimits_t x_{it} = 1, \forall i \tag{2}$$

$$\sum\nolimits_i x_{it} = 1, \forall t \tag{3}$$

$$c_{t1} \ge \sum\nolimits_i (r_i + p_{i1}) x_{it}, \forall t \tag{4}$$

$$c_{t1} \ge c_{(t-1)1} + \sum\nolimits_i p_{i1} \cdot x_{it}, \quad t \ge 2 \tag{5}$$

$$c_{t2} \ge c_{(t-1)2} + \sum\nolimits_i p_{i2} \cdot x_{it}, \quad t \ge 2 \tag{6}$$

$$c_{t2} \ge c_{t1} + \sum\nolimits_i p_{i2} \cdot x_{it}, \ \forall t \tag{7}$$

$$c_{t2} \le c_{t1} + \sum\nolimits_i (w_i + p_{i2}) x_{it}, \quad \forall t \tag{8}$$

$$c_{t2} \le c_i + (1 - x_{it})M, \quad \forall i, t \tag{9}$$

$$c_{t2} \ge c_i + (1 - x_{it})M, \quad \forall i, t \tag{10}$$

$$Z \ge c_i - (1 - y_i)M, \quad \forall i \tag{11}$$

$$\tau_i \ge c_i - d_i - M \cdot y_i, \quad \forall i \tag{12}$$

$$c_{tk}, c_i, Z, \tau_i \ge 0, \quad \forall i, t, k \tag{13}$$

$$x_{it} = \{0, 1\} \ \forall i, t \tag{14}$$

The objective function (1) is to minimize the weighted sum of total tardiness for urgent jobs and the makespan for normal jobs. Constraints (2) and (3) ensure that each job can be and should be assigned to only one position, and each position requires only one job. Constraint (4) ensures that jobs can be started after their release times. Constraints (5) amd (6) define the completion times of jobs assigned to $t$th position on machines 1 and 2, respectively. Constraints (7) and (8) ensure that jobs should keep their limited waiting times between machines 1 and 2. Constraints (9) and (10) define the completion time of each job on machine 2. Constraints (11) and (12) define the makespan of normal jobs and the tardiness of urgent jobs. Constraints (13) and (14) define the domain of decision variables.

## 4. Heuristic Algorithms

The MIP can be used to find an optimal solution using a commercial optimization solver. However, because the considered scheduling problem is NP-hard, the solver may require a significant computation time to obtain an optimal solution for large or practical-sized problems. Therefore, this study focuses on proposing heuristic algorithms that can find a good solution close to optimal or acceptable. Two metaheuristic algorithms (an iterated greedy algorithm and a simulated annealing algorithm) are adopted in this study.

Before introducing the metaheuristic algorithms, we provide the following equations to calculate completion times of jobs in a sequence obtained in procedures of the proposed heuristics. Note that, because this study considers only permutation schedules, the completion times are calculated sequentially one by one from the first to the last. Equations (15) and (16) calculate the completion times of normal jobs and urgent jobs on the first machine, respectively, while Equation (17) calculates the completion times on the second machine. In addition, Equation (18) calculates the tardiness of urgent jobs. Here, it is assumed that $c_{0,1} = c_{0,2} = 0$.

$$c_{t1} = \max\{c_{(t-1)1} + p_{[t]1}, c_{(t-1)2} - w_{[t]}\}, \text{ if the } t\text{th position job is a normal job} \tag{15}$$

$$c_{t1} = \max\{\max\{c_{(t-1)1}, r_{[t]}\} + p_{[t]1}, c_{(t-1)2} - w_{[t]}\}, \text{ if the } t\text{th position job is an urgent job} \tag{16}$$

$$c_{t2} = \max\{c_{t,1}, c_{(t-1)2}\} + p_{[t]2}, \text{ for } t = 1, \ldots, n \tag{17}$$

$$\tau_{[t]} = \max\{c_{t2} - d_{[t]}, 0\}, \text{ if the } t\text{th position job is an urgent job} \tag{18}$$

### *4.1. Initial Seed Sequence*

In general, the metaheuristic algorithm starts with an initial seed sequence, and the performance of the metaheuristic depends on the seed sequence. Thus, the method to obtain the seed is introduced. In this study, a two-step method is used. A feasible permutation is generated by a list-scheduling in the first step. After that, the permutation is improved by a constructive heuristic, called NEH algorithm [42], in the second step.

List scheduling is a common method in practice because it is intuitive and very easy to implement. This method is triggered when a machine becomes available, and a job with the highest priority is assigned to the machine. Defining the priority for the jobs is the only requirement of this method. Here, a modified earliest due date rule (MEDD) is used to focus on the urgent jobs with the objective of minimizing their tardiness. For an unscheduled job $i$ and a given partial schedule $S$, the modified due date ($d'_i$) is defined as $d'_i = \max\{d_i, C(Si)\}$ if $i \in U_A$, and $d'_i = C(Si)$ otherwise. Because the due dates of normal jobs are set to an infinite number, normal jobs may be placed on rear positions in the MEDD schedule. To avoid this schedule, the modified due dates of normal jobs are set to the completion times assuming that they are scheduled immediately after $S$.

In step 2, the NEH begins with the MEDD schedule and makes schedules in a constructive way. In each iteration of the NEH procedure, a partial schedule is created by inserting the job at the front position of the MEDD schedule into the best position of the current partial schedule, and the inserted job is removed from the MEDD schedule. After generating a complete schedule, a pairwise interchange method is applied to the complete schedule for improving it. A detailed procedure is given in Figure 1.

```
procedure NEH
    $S \leftarrow$ a sequence obtained by MEDD
    $S^* \leftarrow$ {the first job of $\pi$}
    for($t$=2; $t \leq n$; $t$++)
        $i \leftarrow t$-th job of $S$
        insert $i$ into the best position of $S^*$
    $t$=1, $t'$=2
    while($t \leq n$ and $t' \leq n$)
        $S' \leftarrow$ a new sequence by interchanging two jobs in $t$-th and $t'$-th jobs of $S^*$
        if($obj(S') < obj(S)$)
            $S^* \leftarrow S'$
        if($t' < n$) $t' \leftarrow t' + 1$
        else $t \leftarrow t + 1$ and $t' \leftarrow t + 1$
    return $S^*$
end procedure
```

**Figure 1.** Procedure of the proposed NEH algorithm.

## *4.2. Iterated Greedy Algorithm*

The iterated greedy algorithm (IG) is a metaheuristic based on a stochastic local search strategy, developed in [43]. The original IG was specially devised for the permutation flow shop scheduling problem. Because of its simplicity of the structure and excellent performance, IGs have been popularly used in much flow shop research. The procedure of IG iterates four phases of destruction, construction, local search, and acceptance, to find better solutions. In the destruction phase, the predetermined number ($d$) of jobs are removed from a current schedule. Then, in the construction phase, a neighborhood schedule is generated by inserting the removed jobs in a constructive way such as that of NEH.

1. In the proposed IG, $d$ jobs are removed randomly from a current solution ($S$) in the destruction phase. Then, two partial schedules are generated. One ($S_{\mathrm{D}}$) is a schedule consisting of $d$ jobs removed from $S$, the other is a partial schedule ($S_{\mathrm{R}}$) with ($n - d$) jobs. After that, in the construction phase, a complete schedule is constructed by inserting the jobs of $S_D$ into the best positions of $S_R$, such as the *for-loop* in Figure 1. After these two phases, the complete schedule is improved by a local search procedure. In this IG, the interchange method used in NEH is implemented for the improvement. Completing the local search, the new schedule ($S'$) is compared to the current schedule ($S$). If $S'$ is better than $S$, i.e., $obj(S') < obj(S)$, $S'$ replaces $S$. Otherwise, borrowing the concept of an acceptance in a simulated annealing algorithm, the replacement is accepted with a probability $exp(-\Delta/T)$, where $\Delta = obj(S') - obj(S)$, and $T$ is an adjustable parameter (called *temperature*) in the IG. However, unlike SA, $T$ is constant in IG. The temperature value $T$ is calculated as

$$T = \frac{\sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}}{10nm} \tag{19}$$

suggested in [44] for the permutation flow shop scheduling problem. In this study, $m$ = 2. These four phases are repeated until a termination condition is satisfied. Herein, the maximum computation time is used for the termination condition. The whole procedure of IG is summarized in Figure 2. In the procedure, let *random*() denote a function to return a random number greater than or equal to zero and less than one.

```
procedure IG
    $S^* \leftarrow S \leftarrow$ NEH
    while(elapsed time < maximum CPU time)
        // destruction phase
        $S_D \leftarrow \{d$ jobs randomly removed from $S\}$
        $S_R \leftarrow S \setminus S_D$

        //construction phase
        for($t$=1; $t \leq d$; $t$++)
            $i \leftarrow t$-th job of $S_D$
            insert $i$ into the best position of $S_R$
        $\hat{S} \leftarrow$ a schedule resulting from the construction phase

        //local search phase
        $t = 1$ and $t' = t + 1$
        while($t \leq n$ and $t' \leq n$)
            $S' \leftarrow$ a new sequence by interchanging two jobs in $t$-th and $t'$-th jobs of $\hat{S}$
            if($obj(S') < obj(\hat{S})$)
                $\hat{S} \leftarrow S'$
            if($t'<n$) $t' \leftarrow t' + 1$
            else $t \leftarrow t + 1$ and $t' \leftarrow t + 1$

        //acceptance phase
        if $obj(\hat{S}) < obj(S)$ then
            $S \leftarrow \hat{S}$
        else
            $\Delta = obj(\hat{S}) - obj(S)$
            if random() < exp($-\Delta/T$) then
                $S \leftarrow \hat{S}$
        if $obj(S) < obj(S^*)$ then
            $S^* \leftarrow S$
    return $S^*$
end procedure
```

**Figure 2.** Procedure of the proposed IG algorithm.

### *4.3. Simulated Annealing Algorithm*

Simulated annealing (SA) is one of the most popular metaheuristic algorithms for combinatorial optimization problems including operations scheduling. In addition, SA showed good performance in a recent paper related to a two-machine flowshop with urgent jobs [1]. This study also adopts SA for the solution methodology. Generally, in flow shop scheduling problems, SA attempts to improve a current schedule ($S$) by making a small change in $S$. To accomplish this, *insertion*, which moves a randomly selected job to a randomly selected position, is used to generate a neighborhood solution ($S'$). If $S'$ is better than $S$, $S'$ replaces $S$. On the other hand, if $S'$ is worse than $S$, $S$ is replaced with $S'$ with a probability $exp(-\Delta/T)$. The initial temperature is obtained by Equation (19), and the temperature is gradually decreased by multiplying a cooling ratio $\gamma$ if $S$ cannot be improved for $E$ iterations ($E$ is called an *epoch length*.). The algorithm is terminated when the maximum computation time is elapsed. The whole procedure is given in Figure 3.

```
procedure SA
    $S^* \leftarrow S \leftarrow$ NEH
    while(elapsed time < maximum CPU time)
        for($l$=1; $l \leq L$; $l$++)
            $S' \leftarrow$ a new sequence by doing an insertion operation once to $S$
            if $obj(S') < obj(S)$ then
                $S \leftarrow S'$
            else
                $\Delta = obj(S') - obj(S)$
                if $random() < \exp(-\Delta/T)$ then
                    $S \leftarrow S'$
            if $obj(S) < obj(S^*)$ then
                $S^* \leftarrow S$
        $T \leftarrow \gamma T$
    return $S^*$
end procedure
```

**Figure 3.** Procedure of the proposed SA algorithm.

## 5. Computational Experiments

This section reports and analyzes the performance of the proposed algorithms through computational experiments on randomly generated problem instances. All the tested algorithms were coded in Java programming language, and the experiments were conducted on a personal computer with a 2.6 GHz CPU.

To generate problem instances, (urgent and normal) jobs and limited waiting times were generated by the methods used in [25] and [28], respectively. Processing times were randomly generated from a discrete uniform distribution with a range of [1, 100], i.e., $U[1, 100]$. Let $h$ be the proportion of urgent jobs to all jobs. For urgent jobs, three levels (0.3, 0.5 and 0.7) were considered for $h$, and three levels (0.5, 0.7 and 0.9) were considered for the weight for total tardiness of urgent jobs ($\alpha$). The release times of urgent jobs were generated randomly from $U[0, LB]$, where $LB$ is a lower bound on the makespan of normal jobs. The lower bound is obtained by scheduling only normal jobs with Johnson's rule [45], assuming that their waiting times are infinite. Note that if all release times are zero or greater than $LB$, the problem can be solved more easily. In addition, limited waiting times were randomly generated from $U[0, 100]$.

Prior to evaluating the proposed algorithms, calibration for the suggested two metaheuristic algorithms were performed to achieve better performance, because performance of metaheuristic algorithms generally depends on their parameters. For the calibration experiments, three levels of $n$ = (50, 200, and 400) were considered, and three instances for each combination of ($n$, $h$, and $\alpha$) were generated. For the termination condition, the maximum computation time was set to $50n$ milliseconds. Because the IG and SA are stochastic algorithms, they can find a good or bad solution coincidentally. To avoid such a coincidental solution, both algorithms solved each instance three times independently, and the average objective value was used. In addition, to compare the objective values obtained by the algorithms, the relative deviation index (RDI) was used as a measure, defined as $(obj_x - obj_{min})/(obj_{max} - obj_{min})$ where $obj_x$, $obj_{min}$ and $obj_{max}$ are the objective function value from algorithm $x$, the minimum and the maximum, respectively.

The proposed IG has only one parameter which is the destruction size, $d$. To find the most appropriate value of $d$ for the considered problem, eight IGs with different $d$ = (4, 8, 10, 12, 14, 16, 18 and 20) were tested, respectively. Note that the original IG in [43] used $d$ = 4, which showed the best performance in the computational experiment. The results are summarized in Figure 4 which shows the average RDIs. As shown in the figure, IG with $d$ = 14 showed the lowest RDI, which means the best performance. Unlike the original IG, the RDI of IG with $d$ = 4 was the highest among the test results. In addition, the *bath-curve* was observed, meaning that both lower and higher values from $d$ = 14 deteriorate the performance and hence 14 is validated for the appropriate destruction size. Consequently, $d$ = 14 was used in the subsequent experiments.

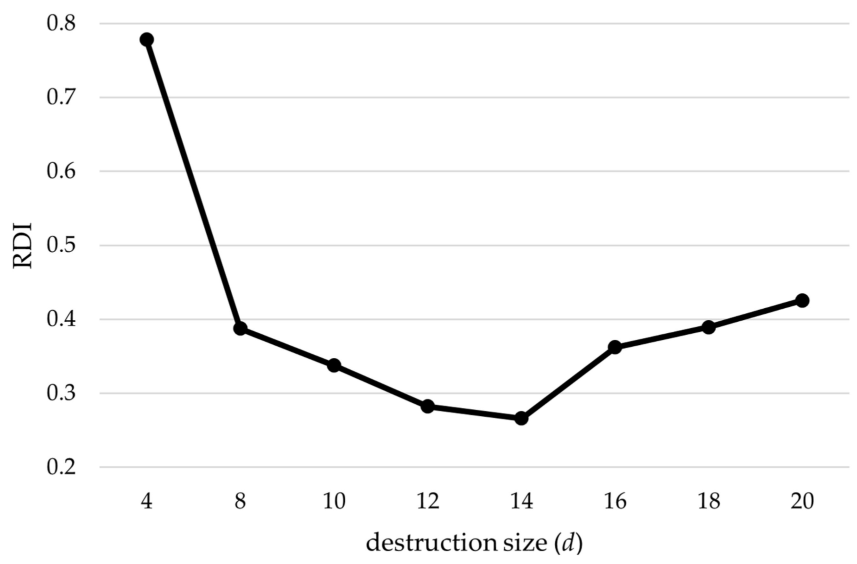

**Figure 4.** Calibration results of the proposed iterated greedy algorithm.

The proposed SA has two types of parameters: cooling ratio $\gamma$ and epoch length $E$. For the calibration, four levels of $\gamma$ and $E$ were considered, i.e., $\gamma$ = (0.99, 0.993, 0.996, and 0.999) and $E$ = (40, 80, 120, and 160). Hence, a total of 16 combinations were tested in this calibration. The results were summarized in Figure 5. Similar to the IG calibration, the bath-curves were observed for the cooling ratio and epoch length. Therefore, ($\gamma$ and $E$) = (0.996 and 80) were used in the subsequent experiments.

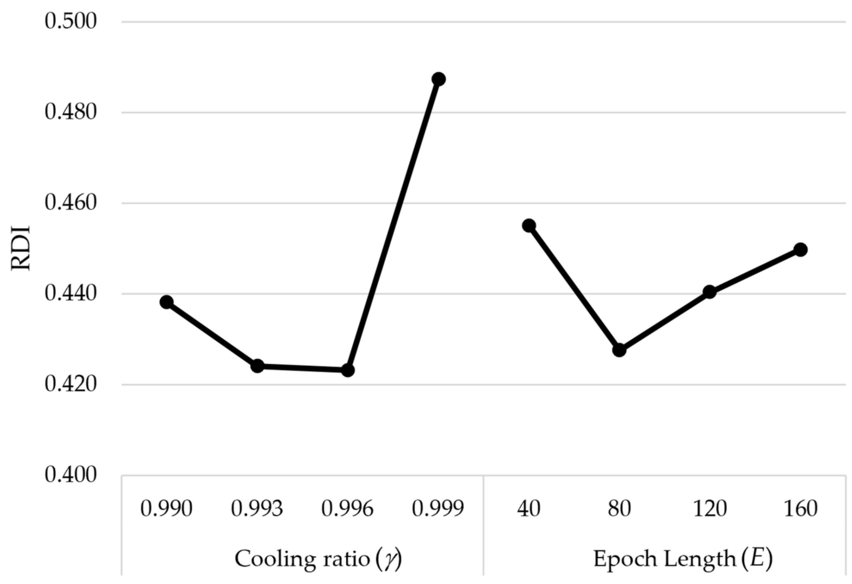

**Figure 5.** Calibration results of the proposed simulated annealing algorithm.

The performance of the proposed MIP was evaluated using a commercial optimization solver, CPLEX 12.10. For experiments, three levels of $n$ = (10, 20, and 30) were considered, and five instances were generated for each combination of ($n$, $h$, and $\alpha$). The maximum time limit for CPLEX was set to 3600 s to avoid an excessive computation time. The results were summarized in Table 2. From the table, CPLEX required a longer CPU time for the instances when $h$ was increased and $\alpha$ was decreased. The reason is that as urgent jobs become more important, they must be scheduled closer to their arrival time, which reduces the number of sequencing candidates to consider. In addition, most of the problems with more than 20 jobs were not terminated within the maximum time limit. Considering that problem size with 20 jobs is not large enough, these results demonstrate the necessity to develop heuristic algorithms that perform well.

**Table 2.** Performance of MIP using CPLEX.

| $n$ | $h$ | $\alpha$ | | |
|---|---|---|---|---|
| | | **0.5** | **0.7** | **0.9** |
| 10 | 0.3 | 0.93, (0) [1] | 0.85 (0) | 0.54 (0) |
| | 0.5 | 1.37 (0) | 1.23 (0) | 1.03 (0) |
| | 0.7 | 2.62 (0) | 2.48 (0) | 2.1 (0) |
| 20 | 0.3 | 2225.24 (1) | 659.25 (0) | 341.35 (0) |
| | 0.5 | 3600 (5) | 3145.93 (4) | 2751.78 (3) |
| | 0.7 | 3600 (5) | 3600 (5) | 3600 (5) |
| 30 | 0.3 | 3600 (5) | 3600 (5) | 3600 (5) |
| | 0.5 | 3600 (5) | 3600 (5) | 3600 (5) |
| | 0.7 | 3600 (5) | 3600 (5) | 3600 (5) |

[1] Average CPU time (number of instances that CPLEX did not terminate within 3600 s).

We compared the performance of proposed algorithms with that of CPLEX. For experiments, heuristic algorithms solved the same instances used to evaluate the MIP. Let IG($x$) and SA($x$) denote the IG and SA algorithm, respectively, with the makespan $nx$ milliseconds as a termination condition. Note that the destruction size for instances with $n$ = 10 was set to 5 which is a half of $n$ because using the determined value ($d$ = 14) was impossible. The results are summarized in Table 3, which shows the average percentage errors (APE) of solutions from the proposed heuristics to those of CPLEX and the number of instances (out of 45) for which the algorithm found solutions better than or equal to those from CPLEX. If CPLEX failed to solve the problem to optimality within the time limit, the best solution obtained within the time limit was compared. In the table, negative APE indicates that the solutions from the heuristic were better than those from CPLEX. As can be seen from the table, NEH could significantly improve the MEDD solutions. In addition, because CPLEX could not solve most problems with more than 20 problems, IG and SA provided better solutions than CPLEX in the 20–30 size problems. In addition, IG showed better performance than SA. That is, the greedy search is more effective than a random search in small sized problems.

**Table 3.** Performance of the heuristic algorithms (vs. CPLEX).

| **Algorithm** | $n$ **= 10** | **20** | **30** |
|---|---|---|---|
| MEDD | 1.048 (0) [1] | 1.371 (0) | 2.711 (0) |
| NEH | 0.096 (4) | 0.158 (0) | 0.266 (0) |
| IG(50) | 0.003 (21) | −0.001 (20) | −0.007 (29) |
| IG(100) | 0.003 (21) | −0.001 (20) | −0.008 (30) |
| SA(50) | 0.006 (19) | 0.002 (16) | −0.001 (27) |
| SA(100) | 0.006 (19) | 0.002 (16) | −0.001 (27) |

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

Additionally, to examine the performance of the proposed algorithms, IG and SA were compared with a tabu search (TS) based metaheuristic algorithm suggested in the latest study [26], which is the most similar one to our study. They considered a two-machine no-wait flow shop with two agents. The TS was also coded in Java language and ran on the same computer for comparison. The same parameters used in [26] were set, but the TS used the same initial sequences that were used in IG and SA, because the flow shop considered in [26] is different from this study. These comparison tests were conducted on medium and large size problems. We considered five instances for each combination of ($n$, $h$, and $\alpha$) where $n$ = 50, 100, 200, 300, 400, and 500. The objective values from each algorithm ($x$) were measured by the relative percentage deviation (RPD), defined as $(obj_x - obj_{\min})/obj_{\min}$. The results are summarized in Table 4, which shows the average RPD. Overall, the proposed IG and SA outperformed by far TS from [26], except for only one case of ($n$ = 50, $h$ = 0.5, $\alpha$ = 0.9).

Hence, we focused on the comparison between IG and SA. When the problem size is small and medium, IG found better solutions than SA, However, as the problem size increased, SA showed better performance than IG. This is probably because IG needs to check all insertion positions and this repeated procedure can be a significant computational burden for large size problems. In contrast, because SA generates a neighborhood solution with a simple insertion, the time to generate a new solution is almost the same for large size problems. According to the results, it can be said that IG is more efficient for a small size problem, whereas SA works better for a large size problem.

Based on the results, we plotted the interaction between algorithms and problem parameters ($n$, $h$ and $\alpha$) to check the effect on the performance of the algorithms. The interval plots are given in Figure 6, the 95% confidence interval for the mean RDIs. For the number of jobs $n$, as we stated, IG was better than SA with up to $n$ = 200. However, SA outperformed IG in problem instances over $n$ = 300. For the proportion ($h$) of urgent jobs to all jobs, SA showed better performance when $h$ = 0.3 and 0.5. However, when $h$ = 0.7, IG was superior to SA, and TS also showed good performance. In other words, three algorithms can be used in complement to one another when there are more urgent jobs than normal jobs. For the weight for total tardiness of urgent jobs ($\alpha$), SA and IG far outperformed TS, and SA performed a little better than IG.

Finally, to see statistically significant differences in the performance of IG and SA, Kruskal–Wallis (KW) tests, which are non-parametric methods, were performed with a commercial statistical analysis problem SPSS. For these tests, instances were divided into two groups according to the number of jobs. This is because IG worked well on small sized instances, whereas SA performed better on large sized instances. Thus, one was a small group ($n$ = 10, 20, 30, 50 and 100), and the other was a large group ($n$ = 200, 300, 400, and 500). Since objective values have different scales according to $n$, RPD values among IG(50), IG(100), SA(50) and SA(100) were set as the dependent variable for the tests. To perform the tests, a significance level was set to 0.05. In addition, to check the effectiveness of problem parameters ($h$ and $\alpha$) on the performance of the proposed algorithms, KW tests were performed for (algorithms $\times$ $h$ values) and (algorithms $\times$ $\alpha$ values) as well as algorithms.

For the small group, all $p$-values in the three KW tests were close to zero, i.e., less than the significance level. Detailed results were summarized in Table 5, which shows the results of pairwise comparisons from KW tests. The results confirmed a statistically significant difference between the performances of IG and SA. Thus, it can be said that IG is superior to SA for the small group. In addition, the performance of SA(50) and SA(100) was not different for the small group.

**Table 4.** Average RPD of IG, SA and TS (the best values are in bold).

| $n$ | $h$ | $\alpha$ | IG(50) | IG(100) | SA(50) | SA(100) | TS(50) | TS(100) |
|---|---|---|---|---|---|---|---|---|
| 50 | 0.3 | 0.5 | 0.0010 | **0.0003** | 0.0029 | 0.0026 | 0.0220 | 0.0201 |
| | | 0.7 | 0.0007 | **0.0000** | 0.0048 | 0.0045 | 0.0327 | 0.0304 |
| | | 0.9 | 0.0010 | **0.0007** | 0.0046 | 0.0041 | 0.0233 | 0.0195 |
| | 0.5 | 0.5 | 0.0011 | **0.0000** | 0.0071 | 0.0068 | 0.0242 | 0.0200 |
| | | 0.7 | 0.0050 | **0.0034** | 0.0128 | 0.0128 | 0.0113 | 0.0084 |
| | | 0.9 | 0.0067 | 0.0028 | 0.0174 | 0.0169 | 0.0047 | **0.0018** |
| | 0.7 | 0.5 | 0.0048 | **0.0029** | 0.0067 | 0.0061 | 0.0071 | 0.0042 |
| | | 0.7 | 0.0016 | **0.0005** | 0.0166 | 0.0163 | 0.0052 | 0.0028 |
| | | 0.9 | 0.0030 | **0.0011** | 0.0082 | 0.0075 | 0.0123 | 0.0094 |
| 100 | 0.3 | 0.5 | 0.0021 | **0.0010** | 0.0030 | 0.0023 | 0.0608 | 0.0574 |
| | | 0.7 | 0.0037 | **0.0027** | 0.0042 | 0.0035 | 0.0695 | 0.0561 |
| | | 0.9 | 0.0002 | **0.0000** | 0.0047 | 0.0038 | 0.0916 | 0.0608 |
| | 0.5 | 0.5 | 0.0062 | **0.0010** | 0.0204 | 0.0183 | 0.0856 | 0.0703 |
| | | 0.7 | 0.0150 | **0.0103** | 0.0158 | 0.0151 | 0.0777 | 0.0675 |
| | | 0.9 | 0.0325 | 0.0276 | 0.0335 | **0.0183** | 0.0989 | 0.0759 |
| | 0.7 | 0.5 | 0.0034 | **0.0020** | 0.0136 | 0.0119 | 0.0212 | 0.0171 |
| | | 0.7 | 0.0100 | **0.0060** | 0.0154 | 0.0128 | 0.0080 | 0.0070 |
| | | 0.9 | 0.0034 | **0.0007** | 0.0056 | 0.0048 | 0.0124 | 0.0095 |
| 200 | 0.3 | 0.5 | 0.0033 | **0.0009** | 0.0053 | 0.0035 | 0.1331 | 0.1238 |
| | | 0.7 | 0.0095 | 0.0020 | 0.0070 | **0.0010** | 0.2297 | 0.2080 |
| | | 0.9 | 0.0163 | **0.0022** | 0.0209 | 0.0046 | 0.4598 | 0.4036 |
| | 0.5 | 0.5 | 0.0329 | 0.0150 | 0.0233 | **0.0109** | 0.1951 | 0.1430 |
| | | 0.7 | 0.0362 | 0.0135 | 0.0106 | **0.0024** | 0.2090 | 0.1483 |
| | | 0.9 | 0.0313 | **0.0078** | 0.0273 | 0.0163 | 0.3087 | 0.2178 |
| | 0.7 | 0.5 | 0.0104 | **0.0003** | 0.0074 | 0.0060 | 0.0189 | 0.0119 |
| | | 0.7 | 0.0079 | **0.0003** | 0.0178 | 0.0143 | 0.0206 | 0.0115 |
| | | 0.9 | 0.0080 | **0.0026** | 0.0096 | 0.0065 | 0.0208 | 0.0102 |
| 300 | 0.3 | 0.5 | 0.0084 | **0.0012** | 0.0090 | 0.0014 | 0.1577 | 0.1462 |
| | | 0.7 | 0.0205 | 0.0058 | 0.0199 | **0.0011** | 0.2910 | 0.2773 |
| | | 0.9 | 0.0369 | 0.0061 | 0.0298 | **0.0030** | 0.6395 | 0.6095 |
| | 0.5 | 0.5 | 0.0358 | **0.0047** | 0.0247 | 0.0150 | 0.2161 | 0.1646 |
| | | 0.7 | 0.0661 | 0.0251 | 0.0253 | **0.0000** | 0.3805 | 0.2878 |
| | | 0.9 | 0.0682 | 0.0243 | 0.0374 | **0.0087** | 0.3840 | 0.2708 |
| | 0.7 | 0.5 | 0.0076 | 0.0041 | 0.0034 | **0.0018** | 0.0239 | 0.0104 |
| | | 0.7 | 0.0084 | **0.0027** | 0.0071 | 0.0035 | 0.0348 | 0.0164 |
| | | 0.9 | 0.0101 | **0.0005** | 0.0090 | 0.0045 | 0.0212 | 0.0113 |
| 400 | 0.3 | 0.5 | 0.0223 | 0.0102 | 0.0142 | **0.0000** | 0.2227 | 0.2022 |
| | | 0.7 | 0.0194 | 0.0028 | 0.0187 | **0.0013** | 0.2850 | 0.2630 |
| | | 0.9 | 0.0394 | 0.0142 | 0.0411 | **0.0044** | 0.7834 | 0.6784 |
| | 0.5 | 0.5 | 0.0617 | 0.0303 | 0.0333 | **0.0094** | 0.3610 | 0.3009 |
| | | 0.7 | 0.0655 | 0.0280 | 0.0375 | **0.0073** | 0.4807 | 0.3953 |
| | | 0.9 | 0.0849 | 0.0356 | 0.0345 | **0.0000** | 0.5155 | 0.4632 |
| | 0.7 | 0.5 | 0.0104 | **0.0008** | 0.0076 | 0.0041 | 0.0324 | 0.0181 |
| | | 0.7 | 0.0060 | **0.0000** | 0.0108 | 0.0073 | 0.0281 | 0.0124 |
| | | 0.9 | 0.0092 | 0.0045 | 0.0060 | **0.0027** | 0.0238 | 0.0108 |
| 500 | 0.3 | 0.5 | 0.0327 | 0.0068 | 0.0207 | **0.0011** | 0.2262 | 0.2019 |
| | | 0.7 | 0.0371 | 0.0143 | 0.0223 | **0.0000** | 0.3740 | 0.3344 |
| | | 0.9 | 0.0934 | 0.0478 | 0.0454 | **0.0002** | 0.8396 | 0.7805 |
| | 0.5 | 0.5 | 0.0671 | 0.0284 | 0.0118 | **0.0000** | 0.3808 | 0.3218 |
| | | 0.7 | 0.1466 | 0.0898 | 0.0212 | **0.0000** | 0.5184 | 0.4609 |
| | | 0.9 | 0.0778 | 0.0408 | 0.0214 | **0.0000** | 0.4614 | 0.3884 |
| | 0.7 | 0.5 | 0.0115 | 0.0064 | 0.0032 | **0.0005** | 0.0283 | 0.0153 |
| | | 0.7 | 0.0163 | 0.0094 | 0.0029 | **0.0000** | 0.0385 | 0.0234 |
| | | 0.9 | 0.0109 | **0.0021** | 0.0064 | 0.0033 | 0.0300 | 0.0164 |
| | Average | | 0.0247 | 0.0103 | 0.0158 | **0.0058** | 0.1860 | 0.1574 |

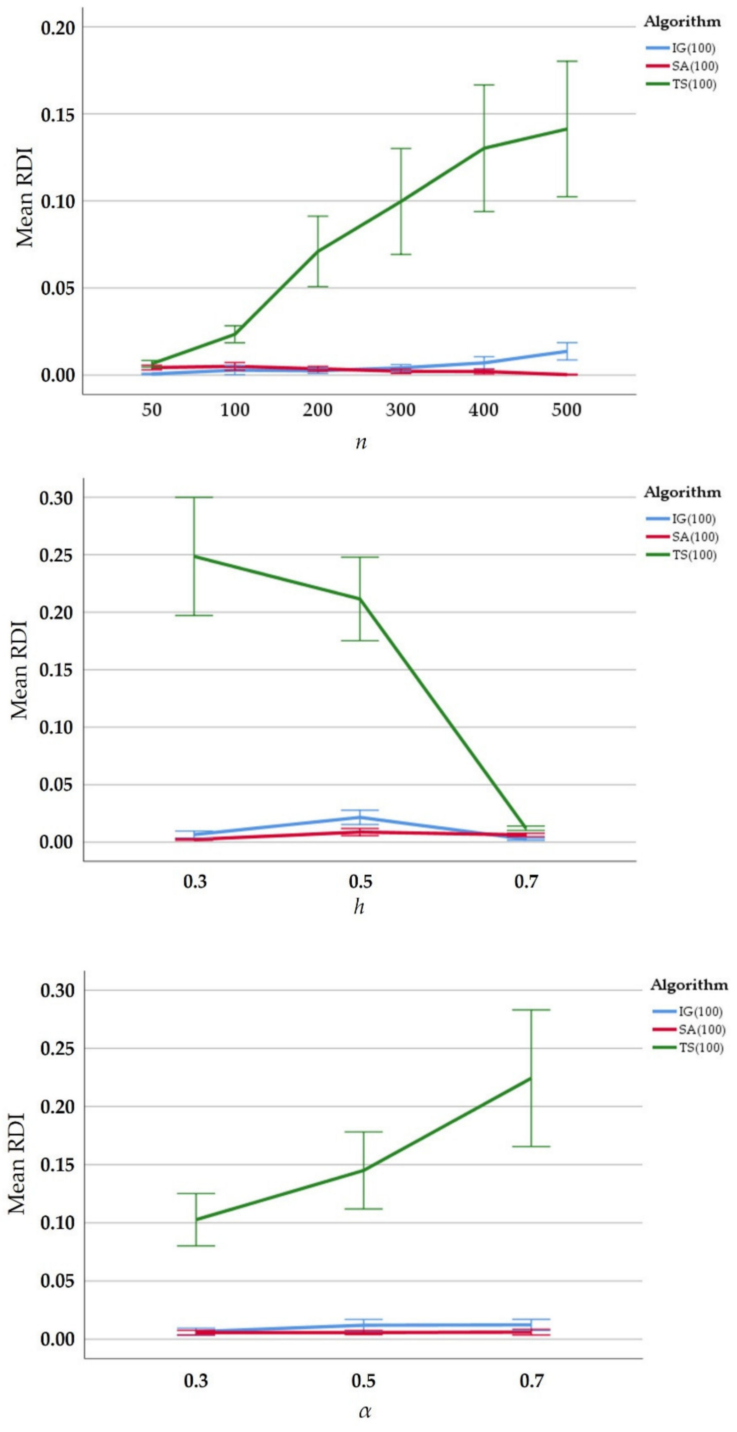

**Figure 6.** Mean plots for interaction between algorithms and problem parameters.

**Table 5.** Pairwise comparisons from Kruskal—Wallis tests for the small group.

| **Tests** | **Sample 1 − Sample 2** | **Test Statistic** | **Std. Error** | **Std. Test Statistic** | ***p*-Value** |
|---|---|---|---|---|---|
| Algorithms | SA(100) − SA(50) | −26.684 | 22.404 | −1.191 | 0.234 |
| | IG(50) − SA(100) | −111.542 | 22.404 | −4.979 | 0.000 |
| | IG(100) − IG(50) | −120.400 | 22.404 | −5.374 | 0.000 |
| | IG(50) − SA(50) | −138.227 | 22.404 | −6.170 | 0.000 |
| | IG(100) − SA(100) | −231.942 | 22.404 | −10.353 | 0.000 |
| | IG(100) − SA(50) | −258.627 | 22.404 | −11.544 | 0.000 |
| Algorithms × $h$ | SA(100) × 0.7 − SA(50) × 0.7 | −22.800 | 38.805 | −0.588 | 0.557 |
| | SA(100) × 0.3 − SA(50) × 0.3 | −23.280 | 38.805 | −0.600 | 0.549 |
| | SA(100) × 0.5 − SA(50) × 0.5 | −33.973 | 38.805 | −0.875 | 0.381 |
| | IG(50) × 0.5 − SA(100) × 0.5 | −63.053 | 38.805 | −1.625 | 0.104 |
| | IG(50) × 0.5 − SA(50) × 0.5 | −97.027 | 38.805 | −2.500 | 0.012 |
| | IG(100) × 0.5 − IG(50) × 0.5 | −113.413 | 38.805 | −2.923 | 0.003 |
| | IG(50) × 0.3 − SA(100) × 0.3 | −116.620 | 38.805 | −3.005 | 0.003 |
| | IG(100) × 0.3 − IG(50) × 0.3 | −119.587 | 38.805 | −3.082 | 0.002 |
| | IG(100) × 0.7 − IG(50) × 0.7 | −128.200 | 38.805 | −3.304 | 0.001 |
| | IG(100) × 0.7 − SA(100) × 0.7 | −283.153 | 38.805 | −7.297 | 0.000 |
| | IG(100) × 0.7 − SA(50) × 0.7 | −305.953 | 38.805 | −7.884 | 0.000 |
| | IG(100) × 0.3 − SA(100) × 0.3 | −236.207 | 38.805 | −6.087 | 0.000 |
| | IG(100) × 0.3 − SA(50) × 0.3 | −259.487 | 38.805 | −6.687 | 0.000 |
| | IG(100) × 0.5 − SA(100) × 0.5 | −176.467 | 38.805 | −4.548 | 0.000 |
| | IG(100) × 0.5 − SA(50) × 0.5 | −210.440 | 38.805 | −5.423 | 0.000 |
| | IG(50) × 0.7 − SA(100) × 0.7 | −154.953 | 38.805 | −3.993 | 0.000 |
| | IG(50) × 0.7 − SA(50) × 0.7 | −177.753 | 38.805 | −4.581 | 0.000 |
| | IG(50) × 0.3 − SA(50) × 0.3 | −139.900 | 38.805 | −3.605 | 0.000 |
| Algorithms · $\alpha$ | SA(100) × 0.7 − SA(50) × 0.7 | −18.453 | 38.805 | −0.476 | 0.634 |
| | SA(100) × 0.5 − SA(50) × 0.5 | −28.520 | 38.805 | −0.735 | 0.462 |
| | SA(100) × 0.9 − SA(50) × 0.9 | −33.080 | 38.805 | −0.852 | 0.394 |
| | IG(50) × 0.9 − SA(100) × 0.9 | −85.967 | 38.805 | −2.215 | 0.027 |
| | IG(100) × 0.9 − IG(50) × 0.9 | −115.920 | 38.805 | −2.987 | 0.003 |
| | IG(100) × 0.7 − IG(50) × 0.7 | −113.933 | 38.805 | −2.936 | 0.003 |
| | IG(50) × 0.5 − SA(100) × 0.5 | −115.040 | 38.805 | −2.965 | 0.003 |
| | IG(50) × 0.9 − SA(50) × 0.9 | −119.047 | 38.805 | −3.068 | 0.002 |
| | IG(100) × 0.5 − IG(50) × 0.5 | −131.347 | 38.805 | −3.385 | 0.001 |
| | IG(50) × 0.7 − SA(100) × 0.7 | −133.620 | 38.805 | −3.443 | 0.001 |
| | IG(100) × 0.5 − SA(100) × 0.5 | −246.387 | 38.805 | −6.349 | 0.000 |
| | IG(100) × 0.5 − SA(50) × 0.5 | −274.907 | 38.805 | −7.084 | 0.000 |
| | IG(100) × 0.9 − SA(100) × 0.9 | −201.887 | 38.805 | −5.203 | 0.000 |
| | IG(100) × 0.9 − SA(50) × 0.9 | −234.967 | 38.805 | −6.055 | 0.000 |
| | IG(100) × 0.7 − SA(100) × 0.7 | −247.553 | 38.805 | −6.379 | 0.000 |
| | IG(100) × 0.7 − SA(50) × 0.7 | −266.007 | 38.805 | −6.855 | 0.000 |
| | IG(50) × 0.7 − SA(50) × 0.7 | −152.073 | 38.805 | −3.919 | 0.000 |
| | IG(50) × 0.5 − SA(50) × 0.5 | −143.560 | 38.805 | −3.700 | 0.000 |

As in the KW tests for the small group, all $p$-values were close to zero and less than the significance level for the large group. Therefore, the results confirmed that there were statistically significant differences between algorithms. Table 6 shows the detailed test results. As stated earlier, $p$-values from pairwise comparisons between algorithms were less than 0.05, and hence it can be said that SA significantly outperformed IG. As shown in Figure 6, there were no significant differences between IG and SA when urgent jobs are more than normal jobs, i.e., $h = 0.7$, and the weights for two classes are equal, i.e., $\alpha = 0.05$. Therefore, IG can be an alternative for these cases.

**Table 6.** Pairwise comparisons from Kruskal—Wallis tests for the large group.

| **Tests** | **Sample 1 – Sample 2** | **Test Statistic** | **Std. Error** | **Std. Test Statistic** | ***p*-Value** |
|---|---|---|---|---|---|
| Algorithm | SA(50) − IG(50) | 62.786 | 21.752 | 2.886 | 0.004 |
| | SA(100) − IG(100) | 96.508 | 21.752 | 4.437 | 0.000 |
| | IG(100) − SA(50) | −145.267 | 21.752 | −6.678 | 0.000 |
| | SA(100) − SA(50) | −241.775 | 21.752 | −11.115 | 0.000 |
| | SA(100) − IG(50) | 304.561 | 21.752 | 14.001 | 0.000 |
| | IG(100) − IG(50) | −208.053 | 21.752 | −9.565 | 0.000 |
| Algorithms × $h$ | SA(50) × 0.3 − IG(50) × 0.3 | 16.433 | 37.676 | 0.436 | 0.663 |
| | SA(50) × 0.7 − IG(50) × 0.7 | 37.042 | 37.676 | 0.983 | 0.326 |
| | IG(100) × 0.7 − SA(100) × 0.7 | −51.792 | 37.676 | −1.375 | 0.169 |
| | IG(100) × 0.5 − SA(50) × 0.5 | −63.850 | 37.676 | −1.695 | 0.090 |
| | SA(100) × 0.3 − IG(100) × 0.3 | 90.183 | 37.676 | 2.394 | 0.017 |
| | SA(100) × 0.7 − SA(50) × 0.7 | −96.867 | 37.676 | −2.571 | 0.010 |
| | SA(100) × 0.7 − IG(50) × 0.7 | 133.908 | 37.676 | 3.554 | 0.000 |
| | SA(50) × 0.5 − IG(50) × 0.5 | 134.883 | 37.676 | 3.580 | 0.000 |
| | IG(100) × 0.7 − SA(50) × 0.7 | −148.658 | 37.676 | −3.946 | 0.000 |
| | IG(100) × 0.7 − IG(50) × 0.7 | −185.700 | 37.676 | −4.929 | 0.000 |
| | IG(100) × 0.5 − IG(50) × 0.5 | −198.733 | 37.676 | −5.275 | 0.000 |
| | IG(100) × 0.3 − SA(50) × 0.3 | −223.292 | 37.676 | −5.927 | 0.000 |
| | IG(100) × 0.3 − IG(50) × 0.3 | −239.725 | 37.676 | −6.363 | 0.000 |
| | SA(100) × 0.5 − IG(100) × 0.5 | 251.133 | 37.676 | 6.666 | 0.000 |
| | SA(100) × 0.3 − SA(50) × 0.3 | −313.475 | 37.676 | −8.320 | 0.000 |
| | SA(100) × 0.3 − IG(50) × 0.3 | 329.908 | 37.676 | 8.756 | 0.000 |
| | SA(100) × 0.5 − SA(50) × 0.5 | −314.983 | 37.676 | −8.360 | 0.000 |
| | SA(100) × 0.5 − IG(50) × 0.5 | 449.867 | 37.676 | 11.940 | 0.000 |
| Algorithms · $\alpha$ | SA(100) × 0.5 − IG(100) × 0.5 | 47.892 | 37.676 | 1.271 | 0.204 |
| | SA(50) × 0.9 − IG(50) × 0.9 | 50.367 | 37.676 | 1.337 | 0.181 |
| | SA(50) × 0.7 − IG(50) × 0.7 | 67.567 | 37.676 | 1.793 | 0.073 |
| | SA(50) × 0.5 − IG(50) × 0.5 | 70.425 | 37.676 | 1.869 | 0.062 |
| | SA(100) × 0.9 − IG(100) × 0.9 | 101.267 | 37.676 | 2.688 | 0.007 |
| | IG(100) × 0.7 − SA(50) × 0.7 | −114.108 | 37.676 | −3.029 | 0.002 |
| | SA(100) × 0.7 − IG(100) × 0.7 | 140.367 | 37.676 | 3.726 | 0.000 |
| | IG(100) × 0.5 − SA(50) × 0.5 | −146.667 | 37.676 | −3.893 | 0.000 |
| | IG(100) × 0.9 − SA(50) × 0.9 | −175.025 | 37.676 | −4.646 | 0.000 |
| | IG(100) × 0.7 − IG(50) × 0.7 | −181.675 | 37.676 | −4.822 | 0.000 |
| | SA(100) × 0.5 − SA(50) × 0.5 | −194.558 | 37.676 | −5.164 | 0.000 |
| | IG(100) × 0.5 − IG(50) × 0.5 | −217.092 | 37.676 | −5.762 | 0.000 |
| | IG(100) × 0.9 − IG(50) × 0.9 | −225.392 | 37.676 | −5.982 | 0.000 |
| | SA(100) × 0.7 − SA(50) × 0.7 | −254.475 | 37.676 | −6.754 | 0.000 |
| | SA(100) × 0.5 − IG(50) × 0.5 | 264.983 | 37.676 | 7.033 | 0.000 |
| | SA(100) × 0.9 − SA(50) × 0.9 | −276.292 | 37.676 | −7.333 | 0.000 |
| | SA(100) × 0.7 − IG(50) × 0.7 | 322.042 | 37.676 | 8.548 | 0.000 |
| | SA(100) × 0.9 − IG(50) × 0.9 | 326.658 | 37.676 | 8.670 | 0.000 |

## 6. Discussion and Conclusions

This study considered a scheduling problem for a two-machine flow shop with urgent jobs and limited waiting times. The objective of this problem is minimizing the weighted sum of total tardiness of urgent jobs and the makespan of normal jobs. This is the first study considering urgent jobs and limited waiting time constraints together. We proposed a mixed integer programming for this problem. However, because this problem is NP-hard, it is very difficult to solve actual size problems with the MIP. Therefore, we developed two metaheuristic algorithms (an iterated greedy algorithm and a simulated annealing algorithm). Through a series of computational experiments, we suggested the best parameters for IG and SA. In addition, the effectiveness of IG and SA were verified by comparison to MIP. In addition, as the results showed, IG was efficient for small size problems, but SA showed superiority in large size problems.

This problem can be extended in several directions in the future. For example, it is necessary to develop the lower bound to compare performance on large size problems. In addition, studies can investigate an optimal solution algorithm that is more efficient than MIP such as a branch and bound algorithm. If the proposed heuristic algorithms are appropriately modified, it is expected that this will become a standard for comparison of algorithms in future studies.

**Author Contributions:** Conceptualization, J.-Y.L. and B.J.; methodology, J.-Y.L. and B.J.; software, J.-Y.L.; validation, B.J. and J.-H.H.; formal analysis, B.J. and J.-H.H.; investigation, B.J. and J.-H.H.; resources, B.J.; data curation, B.J. and J.-H.H.; writing—original draft preparation, J.-Y.L.; writing—review and editing, J.-Y.L., B.J. and J.-H.H.; visualization, J.-Y.L., B.J. and J.-H.H.; supervision, J.-Y.L.; project administration, J.-Y.L.; funding acquisition, J.-Y.L. and J.-H.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** This study was supported by a 2021 Research Grant from Kangwon National University. In addition, this work was supported by the National Research Foundation of Korea (NRF) Grant Funded by the Korean Government (MSIT) (no. NRF-2020R1G1A1006268, NRF-2020R1G1A1099829).

**Data Availability Statement:** All data for the computational experiments were generated randomly and the method for generating data is written in Section 5 in the paper.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Jeong, B.; Shim, S.O. Heuristic algorithms for two-machine re-entrant flowshop scheduling problem with jobs of two classes. *J. Adv. Mech. Des. Syst. Manuf.* **2017**, *11*, 5. [CrossRef]
2. Wang, B.L.; Huang, K.; Li, T.K. Permutation flowshop scheduling with time lag constraints and makespan criterion. *Comput. Ind. Eng.* **2018**, *120*, 1–14. [CrossRef]
3. An, Y.J.; Kim, Y.D.; Choi, S.W. Minimizing makespan in a two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times. *Comput. Oper. Res.* **2016**, *71*, 127–136. [CrossRef]
4. Koulamas, C. The Total Tardiness Problem: Review and Extensions. *Oper. Res.* **1994**, *42*, 1025–1041. [CrossRef]
5. Yang, D.-L.; Maw-Sheng, C. A two-machine flowshop sequencing problem with limited waiting time constraints. *Comput. Ind. Eng.* **1995**, *28*, 8. [CrossRef]
6. Liang, J.; Wang, P.; Guo, L.; Qu, B.; Yue, C.; Yu, K.; Wang, Y. Multi-objective flow shop scheduling with limited buffers using hybrid self-adaptive differential evolution. *Memetic. Comput.* **2019**, *11*, 407–422. [CrossRef]
7. Anjana, V.; Sridharan, R.; Ram Kumar, P.N. Metaheuristics for solving a multi-objective flow shop scheduling problem with sequence-dependent setup times. *J. Sched.* **2020**, *23*, 49–69. [CrossRef]
8. Rifai, A.P.; Mara, S.T.W.; Sudiarso, A. Multi-objective distributed reentrant permutation flow shop scheduling with sequence-dependent setup time. *Expert Syst. Appl.* **2021**, *183*, 115339. [CrossRef]
9. Zhang, B.; Pan, Q.-k.; Gao, L.; Li, X.-y.; Meng, L.-l.; Peng, K.-k. A multiobjective evolutionary algorithm based on decomposition for hybrid flowshop green scheduling problem. *Comput. Ind. Eng.* **2019**, *136*, 325–344. [CrossRef]
10. Han, Y.; Li, J.; Sang, H.; Liu, Y.; Gao, K.; Pan, Q.-K. Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time. *Appl. Soft. Comput.* **2020**, *93*, 106343. [CrossRef]
11. Lu, C.; Gao, L.; Yi, J.; Li, X. Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in China. *IEEE Trans. Ind. Inform.* **2021**, *17*, 6687–6696. [CrossRef]
12. Agnetis, A.; Mirchandani, P.B.; Pacciarelli, D.; Pacifici, A. Scheduling problems with two competing agents. *Oper. Res.* **2004**, *52*, 229–242. [CrossRef]
13. Baker, K.R.; Cole Smith, J. A multiple-criterion model for machine scheduling. *J. Sched.* **2003**, *6*, 7–16. [CrossRef]
14. Ng, C.T.; Cheng, T.C.E.; Yuan, J.J. A note on the complexity of the problem of two-agent scheduling on a single machine. *J. Comb. Optim.* **2006**, *12*, 387–394. [CrossRef]
15. Leung, J.Y.-T.; Pinedo, M.; Wan, G. Competitive two-agent scheduling and its applications. *Oper. Res.* **2010**, *58*, 458–469. [CrossRef]
16. Cheng, T.C.E.; Ng, C.T.; Yuan, J.J. Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theor. Comput. Sci.* **2006**, *362*, 273–281. [CrossRef]
17. Cheng, T.C.E.; Ng, C.T.; Yuan, J.J. Multi-agent scheduling on a single machine with max-form criteria. *Eur. J. Oper. Res.* **2008**, *188*, 603–609. [CrossRef]
18. Liu, P.; Tang, L. *Two-Agent Scheduling with Linear Deteriorating Jobs on a Single Machine*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 642–650.
19. Lee, W.C.; Chung, Y.H.; Huang, Z.R. A single-machine bi-criterion scheduling problem with two agents. *Appl. Math. Comput.* **2013**, *219*, 10831–10841. [CrossRef]

20. Wan, L.; Yuan, J.J.; Wei, L.J. Pareto optimization scheduling with two competing agents to minimize the number of tardy jobs and the maximum cost. *Appl. Math. Comput.* **2016**, *273*, 912–923. [CrossRef]
21. Lee, W.C.; Chen, S.K.; Wu, C.C. Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem. *Expert Syst. Appl.* **2010**, *37*, 6594–6601. [CrossRef]
22. Lee, W.C.; Chen, S.K.; Chen, C.W.; Wu, C.C. A two-machine flowshop problem with two agents. *Comput. Oper. Res.* **2011**, *38*, 98–104. [CrossRef]
23. Mor, B.; Mosheiov, G. Polynomial time solutions for scheduling problems on a proportionate flowshop with two competing agents. *J. Oper. Res. Soc.* **2014**, *65*, 151–157. [CrossRef]
24. Fan, B.Q.; Cheng, T.C.E. Two-agent scheduling in a flowshop. *Eur. J. Oper. Res.* **2016**, *252*, 376–384. [CrossRef]
25. Jeong, B.; Kim, Y.D.; Shim, S.O. Algorithms for a two-machine flowshop problem with jobs of two classes. *Int. Trans. Oper. Res.* **2020**, *27*, 3123–3143. [CrossRef]
26. Azerine, A.; Boudhar, M.; Rebaine, D. A two-machine no-wait flow shop problem with two competing agents. *J. Comb. Optim.* **2021**, in press. [CrossRef]
27. Bouquard, J.-L.; Lenté, C. Two-machine flow shop scheduling problems with minimal and maximal delays. *4OR* **2006**, *4*, 15–28. [CrossRef]
28. Joo, B.J.; Kim, Y.D. A branch-and-bound algorithm for a two-machine flowshop scheduling problem with limited waiting time constraints. *J. Oper. Res. Soc.* **2009**, *60*, 572–582. [CrossRef]
29. Hamdi, I.; Loukil, T. Minimizing total tardiness in the permutation flowshop scheduling problem with minimal and maximal time lags. *Oper. Res. Ger.* **2015**, *15*, 95–114. [CrossRef]
30. Dhouib, E.; Teghem, J.; Loukil, T. Lexicographic optimization of a permutation flow shop scheduling problem with time lag constraints. *Int. Trans. Oper. Res.* **2013**, *20*, 213–232. [CrossRef]
31. Kim, H.J.; Lee, J.H. Three-machine flow shop scheduling with overlapping waiting time constraints. *Comput. Oper. Res.* **2019**, *101*, 93–102. [CrossRef]
32. Lee, J.Y. A genetic algorithm for a two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times. *Math. Probl. Eng.* **2020**, *2020*, 8833645. [CrossRef]
33. Yu, T.S.; Kim, H.J.; Lee, T.E. Minimization of waiting time variation in a generalized two-machine flowshop with waiting time constraints and skipping jobs. *IEEE T Semicond. Manuf.* **2017**, *30*, 155–165. [CrossRef]
34. Li, Y.; Dai, Z. A two-stage flow-shop scheduling problem with incompatible job families and limited waiting time. *Eng. Optim.* **2020**, *52*, 484–506. [CrossRef]
35. Hamdi, I.; Toumi, S. MILP models and valid inequalities for the two-machine permutation flowshop scheduling problem with minimal time lags. *J. Ind. Eng. Int.* **2019**, *15*, 223–229. [CrossRef]
36. Lima, A.; Borodin, V.; Dauzère-Pérès, S.; Vialletelle, P. Sampling-based release control of multiple lots in time constraint tunnels. *Comput. Ind.* **2019**, *110*, 3–11. [CrossRef]
37. Lima, A.; Borodin, V.; Dauzère-Pérès, S.; Vialletelle, P. A sampling-based approach for managing lot release in time constraint tunnels in semiconductor manufacturing. *Int. J. Prod. Res.* **2021**, *59*, 860–884. [CrossRef]
38. Maassena, K.; Perez-Gonzalez, P.; Gunther, L.C. Relationship between common objective functions, idle time and waiting time in permutation flow shop scheduling. *Comput. Oper. Res.* **2020**, *121*, 104965. [CrossRef]
39. Samarghandi, H. Minimizing the makespan in a flow shop environment under minimum and maximum time-lag constraints. *Comput. Ind. Eng.* **2019**, *136*, 614–634. [CrossRef]
40. Ye, S.; Zhao, N.; Li, K.D.; Lei, C.J. Efficient heuristic for solving non-permutation flow-shop scheduling problems with maximal and minimal time lags. *Comput. Ind. Eng.* **2017**, *113*, 160–184. [CrossRef]
41. Zhou, N.; Wu, M.; Zhou, J. Research on power battery formation production scheduling problem with limited waiting time constraints. In Proceedings of the 2018 10th International Conference on Communication Software and Networks, Chengdu, China, 6–9 July 2018; pp. 497–501.
42. Nawaz, M.; Enscore, E.E.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [CrossRef]
43. Ruiz, R.; Stutzle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **2007**, *177*, 2033–2049. [CrossRef]
44. Osman, I.H.; Potts, C.N. Simulated annealing for permutation flow-shop scheduling. *Omega* **1989**, *17*, 551–557. [CrossRef]
45. Johnson, S.M. Optimal two- and three-stage production schedules with setup times included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–68. [CrossRef]