applied sciences

MDPI

Article

Heuristics for a Two-Stage Assembly-Type Flow Shop with Limited Waiting Time Constraints

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

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

Featured Application: Manufacturing.

Abstract: This study investigates a two-stage assembly-type flow shop with limited waiting time constraints for minimizing the makespan. The first stage consists of m machines fabricating m types of components, whereas the second stage has a single machine to assemble the components into the final product. In the flow shop, the assembly operations in the second stage should start within the limited waiting times after those components complete in the first stage. For this problem, a mixed-integer programming formulation is provided, and this formulation is used to find an optimal solution using a commercial optimization solver CPLEX. As this problem is proved to be NP-hard, various heuristic algorithms (priority rule-based list scheduling, constructive heuristic, and metaheuristic) are proposed to solve a large-scale problem within a short computation time. To evaluate the proposed algorithms, a series of computational experiments, including the calibration of the metaheuristics, were performed on randomly generated problem instances, and the results showed outperformance of the proposed iterated greedy algorithm and simulated annealing algorithm in small- and large-sized problems, respectively.

Keywords: scheduling; two-stage assembly-type flow shop; limited waiting times; makespan; heuristic

Citation: Han, J.-H.; Lee, J.-Y. Heuristics for a Two-Stage Assembly-Type Flow Shop with Limited Waiting Time Constraints. *Appl. Sci.* **2021**, *11*, 11240. https://doi.org/10.3390/app112311240

Academic Editor: Vincent A. Cicirello

Received: 27 October 2021
Accepted: 23 November 2021
Published: 26 November 2021

1. Introduction

Competition in manufacturing industries is intensifying around the world, and these industries are facing changes, such as the diversification of customer requirements, shorter product life cycles, and the transition to a multiproduct small-volume production system. In addition, a new area, such as smart manufacturing, is emerging with the 4th Industrial Revolution. To enhance manufacturing competitiveness and improve productivity, several manufacturers are focusing on reducing the production lead time and minimizing inventory. To achieve these goals, production management and scheduling techniques have emerged as core competencies [1]. Numerous scheduling problems have been studied for various types of products that require assembly operations, considering the production of components and subsequent assembly processes together [2]. For example, Lee et al. [3] introduced a scheduling study of a manufacturing system that produces and assembles the body and vehicle of a fire engine, and Potts et al. [4] dealt with a scheduling problem in make-to-order manufacturing systems, such as PC assembly. In addition, there are scheduling problems with assembly operations arising from various manufacturing systems (refrigerators [5], clothing [6], food [7], and semiconductors [8–11]).

This study deals with a scheduling problem in a two-stage assembly-type flow shop. The first stage is the component production process in which each component is independently fabricated on a dedicated machine. The components made in the first phase are moved to the second stage and assembled into the final product. The assembly operation in

the second stage can start after all necessary components have completed in the first stage. The completion time of each product is defined as the time when the assembly operation completes in the second stage. To enhance the flow shop productivity, this study aims to minimize the makespan, which is equal to the completion time of the last scheduled job.

This study constrains the waiting times between stages 1 and 2. This implies that components completed in the first stage must be entered into the second stage for the assembly operation within the limited waiting times. Therefore, it is necessary to complete the operations of the first stage considering the time when the assembly operation can start in the second stage. Figure 1a,b illustrates example schedules with two jobs in the assembly flow shop with and without limited waiting times, respectively. The starts in the first stage in Figure 1a are delayed observing the limited waiting times w_{21} and w_{22}, whereas the second job in Figure 1b can start without any delay. In general, waiting times are limited to prevent quality deterioration owing to increased waiting times. There are various cases considering the waiting time constraints in manufacturing industries, such as semiconductors, batteries, food, steelmaking, and biotechnology [12–14].

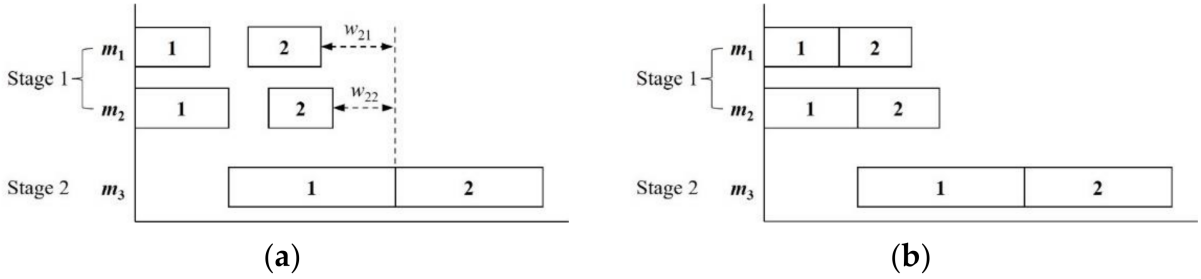

Figure 1. Example schedules with two jobs: (**a**) with waiting time constraints; (**b**) without waiting time constraints.

Since Lee et al. [3] introduced a two-stage assembly-type flow shop problem, there have been many studies on the scheduling problems, and two review papers (Komaki et al. [2] and Framinan et al. [15]) were recently published. Many studies have sought to minimize the makespan. The makespan minimization problem was demonstrated to be strongly NP-hard in [3], even though only two machines existed in the first stage. Lee et al. [3] also developed three heuristics based on Johnson's algorithm [16] and suggested a branch-and-bound (B&B) algorithm. Potts et al. [4] showed that an optimal solution for the problem with more than two machines in the first stage is always in permutation schedules where the sequences on all machines in the flow shop are the same. Additionally, Hariri and Potts [17] and Haouari and Daouas [18] provided B&B algorithms, while Sun et al. [19] proposed heuristic algorithms. On the other hand, Koulamas and Kyparisis [20] considered the extended problem in which there are three stages and they developed heuristic algorithms. Additionally, Sung and Juhn [21] proposed a B&B algorithm for the problem of outsourcing one type of component subject to the job-dependent lead time in the first stage. Wu et al. [22] addressed two scenario-dependent jobs processing times and proposed a B&B algorithm and metaheuristics to minimize the robust makespan, and Wu et al. [23] considered a processing time-based learning effect and proposed a B&B algorithm and several metaheuristics. On the contrary, for other objectives, Lee and Bang [24] proposed a B&B algorithm to minimize total tardiness, whereas Lee [25,26] developed B&B algorithms to minimize total completion times and total weighted tardiness, respectively. Additionally, there have been various studies on the assembly-type flow shop problems [27–32].

Although typical assembly-type flow shop scheduling problems have been studied by many researchers, only a few studies have considered waiting time constraints or time lags. According to recent review papers, it is necessary to study scheduling problems with waiting time constraints, but there are only a few studies that considered no-wait constraints. Mozdgir et al. [33] introduced a no-wait two-stage assembly flow shop and proposed three metaheuristic algorithms: a genetic algorithm, differential evolution algorithm, and population-based variable neighborhood algorithm. In addition, Ji et al. [34]

provided a hybrid particle swarm optimization algorithm, and Li et al. [35] proposed an iterated local search method. Shao et al. [36] suggested an iterated local search method and a variable neighborhood search. Most of the studies related to the no-wait assembly flow shop considered metaheuristic algorithms [37–40] as solution methodologies because these problems are NP-hard.

The considered scheduling problem is defined as $AF(m,1) \mid max\text{-}wait \mid C_{max}$ in three-field notation suggested by Graham et al. [41]. $AF(m,1)$ represents an assembly-type flow shop where there are m machines to fabricate components in stage 1 and a single assembly machine in stage 2. *Max-wait* denotes the limited waiting constraints between the two stages. C_{max} represents the makespan, which is the objective function of the scheduling problem. Assuming that there is only one machine in the first stage, this scheduling problem becomes a two-machine flow shop problem with limited waiting time constraints ($F2 \mid max\text{-}wait \mid C_{max}$), which has been proven to be NP-hard in [42]. Therefore, this problem is also NP-hard.

To the best of the authors' knowledge, no assembly flow shop study has considered the limited waiting time constraints that are generalized from no-wait. Thus, as its main contribution, this study is the first attempt to consider the assembly flow shop with limited waiting times to fill part of the existing research gap. Additionally, this paper proposes various solution methodologies, including the mathematical formulation, priority rule-based list scheduling, constructive heuristic, and three metaheuristic algorithms. Furthermore, we suggest a method that converts a sequence obtained by the heuristics into a complete schedule with waiting time constraints.

This study aims to minimize the makespan in a two-stage assembly flow shop with limited waiting time constraints. We provide a mixed-integer programming (MIP) to define the considered problem clearly and use a commercial optimization solver, CPLEX 12.10, to solve the MIP. However, because this problem is NP-hard, there is a limitation in obtaining an optimal solution despite using the solver. Thus, we propose various heuristic algorithms, such as priority rule-based list scheduling, constructive heuristic, and metaheuristic to solve the large-sized problem within a short computation time. To evaluate the proposed algorithms, we performed a series of computational experiments, including the calibration of the metaheuristics, on randomly generated problem instances and reported the results.

The remainder of this paper is organized as follows. Section 2 provides the assumptions made in this study and notation for a clear description and provides a mathematical model, while Section 3 proposes heuristic algorithms. Section 4 reports the computational tests to evaluate the performance of the heuristic algorithms as well as the mathematical model. Finally, Section 5 concludes this study and presents future research.

2. Problem Description

This section describes the scheduling problems considered in more detail with assumptions and mathematically defines the problem through an MIP formulation. In this study, there are n jobs to be scheduled, and it is assumed that information regarding the jobs is provided in advance. The first stage consists of m dedicated machines that fabricate m different components, whereas there is a single assembly machine to assemble the m components into a final product in the second stage. The assembly operations of all jobs must start within limited waiting times given to the jobs after completing the fabrication of their components. The objective function of the scheduling problem is to minimize the makespan, which is equal to the completion time of the last job. The other assumptions made in this study are as follows:

- All jobs can be started at time zero;
- Preemption is not allowed;
- Not all machines can process multiple jobs simultaneously. In other words, machines can process only one job at a time;
- The distances and delivery times between the two stages are ignored.

In this problem, there are $(n!)^{(m+1)}$ alternative sequences, because $n!$ sequences are possible for each machine. However, there is an optimal solution among $n!$ permutation schedules in which the operating sequences of the jobs are the same on the machines. This can be proven according to the proof by Tozkapan et al. [43] that permutation schedules are dominant for the two-stage assembly flow shop to minimize the total completion time, and this proof is sufficient to show that permutation schedules are also dominant for any regular performance measure, including makespan. Although this study considers the limited waiting time constraints, the left-shift schedules in which some operations begin earlier without delaying any other operations are dominant, because a delayed start increases the waiting time. Furthermore, in practical production lines, permutation schedules are commonly utilized to ensure simplicity of managing work, flexibility of material handling, and constrained buffer space. Therefore, this study only considers permutation schedules; that is, all sequences on the machines in the considered flow shop are the same. Table 1 provides the notation for a clear description of the algorithms proposed in this study.

Table 1. Notation for the description of the algorithms proposed in this study.

Symbol	Definition
n	number of jobs
i, j	job indices
m	number of machines in the first stage
k	machine index ($k = 1, \ldots, m, m+1$); if $k = 1, \ldots, m$, it represents the first stage machines, otherwise, the second stage machine
p_{ik}	processing time of job i on machine k
w_{ik}	waiting time of job i between machine k and machine $(k+1)$
r	position index
$[r]$	index of the job in rth position of a permutation schedule
x_{ir}	equals to 1 if job i is scheduled in rth position, 0 otherwise
$s_{[r]k}$	starting time of the rth job on machine k
$c_{[r]k}$	completion time of the rth job on machine k

The following is the MIP formulation for the considered scheduling problem.

$$\text{Minimize} \quad c_{[n](m+1)} \tag{1}$$

$$\text{subject to} \quad s_{[r]k} + \sum_{i=1}^{n} p_{ik} x_{ir} \leq s_{[r+1]k} \quad r \leq n-1 \text{ and } k \leq m \tag{2}$$

$$s_{[r]k} + \sum_{i=1}^{n} p_{ik} x_{ir} \leq s_{[r](m+1)} \quad \forall r \text{ and } k \leq m \tag{3}$$

$$s_{[r]k} + \sum_{i=1}^{n} (p_{ik} + w_{ik}) x_{ir} \geq s_{[r](m+1)} \quad \forall r \text{ and } k \leq m \tag{4}$$

$$s_{[r](m+1)} + \sum_{i=1}^{n} p_{i(m+1)} x_{ir} = c_{[r](m+1)} \quad \forall r \tag{5}$$

$$\sum_{r=1}^{n} x_{ir} = 1 \quad \forall i \tag{6}$$

$$\sum_{i=1}^{n} x_{ir} = 1 \quad \forall r \tag{7}$$

$$s_{[r]k}, c_{[r](m+1)} \geq 0 \quad \forall r, k \tag{8}$$

$$x_{ir} \in \{0, 1\} \quad \forall i, r \tag{9}$$

The objective Function (1) is the minimization of the makespan, which is equal to $c_{[n](m+1)}$, the completion time of the last scheduled job $[n]$ in the second-stage machine, that is, machine $(m+1)$. Constraints (2) and (3) define the starting times on the same machine and between the two stages, respectively. Constraint (4) ensures that jobs must maintain the limited waiting times between both stages. Constraint (5) computes the completion times of the jobs on the machine $(m+1)$. Constraints (6) and (7) consider only the permutation schedules. That is, each job can only be sequenced once, without being partitioned. Constraints (8) and (9) define the domain of the decision variables.

3. Heuristic Algorithms

The MIP formulation can obtain an optimal solution for the considered problem using a commercial solver. However, because this problem is NP-hard, solvers may require a

considerable amount of computational time to achieve an optimal solution for large or practical size problems. Therefore, this study focuses on heuristic algorithms to find a good solution within a short computation time.

This section proposes three types of heuristic algorithms: priority rule-based scheduling methods, a constructive algorithm, and three metaheuristic algorithms. In the heuristic algorithm procedures, we calculate the completion times of jobs using the following equations: the first two Equations (10) and (11) are for the first job (r = 1), whereas Equations (12) and (13) are for the second to the last job (r = 2, . . . , n).

$$c_{[1](m+1)} = MAX\left(p_{[1]k} \forall k \leq m\right) + p_{[1](m+1)} \tag{10}$$

$$c_{[1]k} = MAX\left(p_{[1]k}, c_{[1](m+1)} - w_{[1]k}\right) \text{ for } k = 1, \ldots, m \tag{11}$$

$$c_{[r](m+1)} = MAX\left(MAX\left(c_{[r-1]k} + p_{[r]k} \forall k \leq m\right), c_{[r-1](m+1)}\right) + p_{[r](m+1)} \quad \text{for } r = 2, \ldots, n \tag{12}$$

$$c_{[r]k} = MAX\left(c_{[r-1]k} + p_{[r]k}, c_{[r](m+1)} - p_{[r](m+1)} - w_{[r]k}\right) \text{ for } r = 2, \ldots, n \tag{13}$$

3.1. Priority Rule-Based Scheduling

Priority rule-based list scheduling is a commonly used method in many manufacturing systems because it is intuitive and easy to implement. Such a method sorts jobs according to priority values and then assigns the sorted jobs to the machines when the machines become available. In general, the order of jobs can be generated from a priority rule based on the characteristics of the considered problem. As this study considers only permutation schedules, the priority rule-based scheduling can find feasible schedules by ordering jobs in ascending order of the priority values. The following six priority rules generate six feasible schedules, and the completion times of the jobs in a given schedule are calculated using Equations (10)–(13).

- LS1: $MAX(p_{ik} \; \forall k \leq m)$;
- LS2: $p_{i(m+1)}$;
- LS3: $MAX(p_{ik} \; \forall k)$;
- LS4: $MAX(p_{ik} \; \forall k \leq m) + p_{i(m+1)}$;
- LS5: $\sum_{k=1}^{m} p_{ik}/m + p_{i(m+1)}$;
- LS6: $MIN\left(MAX(p_{ik} \; \forall k \leq m), p_{i(m+1)}\right)$.

The proposed priority rules are modified from the shortest processing time rule. In the considered flow shop, the start of the second stage operation is affected by the longest processing time of the first stage, not the shortest. This is because the second stage operation can start after all the operations in the first stage complete. Thus, the longest processing times in the first stage are considered. LS1 considers the first stage only, LS2 focuses on the second stage, and LS3–6 consider both. Particularly in LS5, it considers the average processing time, not the longest, for the first stage. LS6 is inspired by Johnson's rule [16] which provides an optimal solution for a classical two-machine flow shop makespan problem.

3.2. Modified NEH Algorithm (MNEH)

This subsection provides a constructive heuristic algorithm based on the NEH algorithm developed by Nawaz et al. [44]. This NEH algorithm is well known as an effective and efficient heuristic for flow shop scheduling problems, and thus many researchers have used the NEH for various flow shop problems. The original procedure begins with an initial seed sequence obtained by sorting jobs in descending order of total processing times (i.e., longest processing time order), and then it creates a complete schedule in an insertion-based constructive manner. At each iteration, a job in front of the seed sequence

is inserted into the best position in the current partial schedule and then it is removed from the seed. These iterations are repeated until obtaining a complete schedule.

In this study, the modified version uses the best priority rule to obtain an initial seed sequence, instead of the longest processing time order. The best rule will be demonstrated in the computational experiments (Section 4). Additionally, after obtaining a partial schedule with the insertion method at each iteration, an interchange method is applied to improve the partial schedule. That is, after job (i) is inserted into the current best position at each iteration, partial schedules obtained by interchanging job i and other jobs are evaluated, and then the minimum makespan schedule is moved to the next iteration. Figure 2 shows the procedure for this algorithm.

```
procedure Modified NEH
    π ← sequence obtained by the priority rule
    π* ← {the first job of π}
    for(r = 2; r ≤ n; r++)
        i ← rth job of π
        Insert i into the best position of π*, achieving the minimum makespan
        Find the best sequence by interchanging i and other jobs of π*
    return π*
end procedure
```

Figure 2. Modified NEH algorithm procedure.

3.3. Genetic Algorithm

The genetic algorithm (GA) is an evolutionary population-based optimization algorithm inspired by genetics and Darwin's theory of evolution, and it is based on the survival of the fittest or natural selection [45]. GA is one of the most popular metaheuristics in the field of optimization. GA involves many operators and parameters, and the performance depends on the procedure design, operators, and parameters. The following subsections describe the design of the proposed GA.

3.3.1. Solution Representation

In the GA, solutions are expressed in a chromosome structure, and the performance of the algorithm depends on the chromosome structure. Because this study considers only permutation schedules, the chromosome of a solution is represented in a permutation.

3.3.2. Initial Population

The GA is a population-based heuristic algorithm. Priority rule-based scheduling and MNEH algorithm are used to generate an initial population. That is, six priority rules generate six solutions, and the MNEH algorithm generates six solutions with each of the six solutions by the rules, i.e., a total of 12 solutions. The remainder of the initial population is randomly generated.

3.3.3. Solution Representation

Fitness represents how good a solution is. As this study has the objective of minimizing the makespan, the fitness of a solution is equal to the makespan obtained by Equations (10)–(13). A lower makespan indicates better fitness.

3.3.4. Selection

The selection chooses parents randomly from the current population to generate offspring. The selection is based on the fitness values of chromosomes to generate offspring that inherit good genes. In other words, good chromosomes with low fitness values are likely to be selected as parents. This study considers two commonly used methods (tournament and roulette) in the literature for selection. In the tournament method, four solutions are selected randomly from the population, and two winner solutions from the semi-finals

are selected. That is, a better of the first two solutions and a better of the last two solutions are paired. For the roulette methods, the inverse (f_π) of the objective function value ($C_{max}(\pi)$) for each solution (π) in the population is computed, that is, $f_\pi = 1/C_{max}(\pi)$ and let $F = \sum f_\pi$. The selection probability of each solution in the population is then obtained by $prob_\pi = f_\pi/F$. Based on these selection probabilities of solutions, two solutions are selected randomly as parents. Note that the proposed GA uses a better one of two methods after computational calibration (in Section 4).

3.3.5. Crossover

The crossover operation aims to generate better offspring by exchanging the information of the selected parents. In this study, six types of crossover operation are considered: one-point order crossover (OX1), two-point order crossover (OX2), similar job crossover (SJX), two-point similar job crossover (SJX2), similar block crossover (SBX), and two-point similar block crossover (SBX2). Like the selection operation, the proposed GA uses the best of these six methods. For a description of these crossovers, let $parent_1$ and $parent_2$ be the selected parents to generate *offspring*.

- Procedure: OX1

  Step 0. Let $point_1$ be a cutoff point randomly selected from $parent_1$.
  Step 1. Give the front part (from the first to $point_1$) of $parent_1$ to *offspring*.
  Step 2. Remove the jobs, that are given to *offspring* in *Step* 1, from $parent_2$.
  Step 3. Sequence jobs remaining in $parent_2$ into the unoccupied positions of *offspring* from left to right in the order that they appear in $parent_2$.

- Procedure: OX2

  Step 0. Let $point_1$ and $point_2$ be cutoff points randomly selected from $parent_1$, respectively.
  Step 1. Give the middle part (between $point_1$ and $point_2$) of $parent_1$ to *offspring*.
  Step 2. Remove the jobs, that are given to offspring in *Step* 1, from $parent_2$.
  Step 3. Sequence jobs remaining in $parent_2$ into the unoccupied positions of *offspring* from left to right in the order that they appear in $parent_2$.

- Procedure: SJX

  Step 0. Give the jobs in the same position in both parents to *offspring*.
  Step 1. Let $point_1$ be a cutoff point randomly selected from $parent_1$.
  Step 2. Give the front part (from the first to $point_1$) of $parent_1$ to *offspring*.
  Step 3. Remove the jobs, that are already given to *offspring*, from $parent_2$.
  Step 4. Sequence jobs remaining in $parent_2$ into the unoccupied positions of *offspring* from left to right in the order that they appear in $parent_2$.

SJX2 modified from SJX selects two cutoff points and gives the middle part between the two points of $parent_1$ to *offspring*, as in OX2. Additionally, SBX and SBX2 are very similar to SJX and SJX2, respectively. The only difference is that, in *Step* 0 of SJX (and SJX2), two adjacent jobs are defined as a block and not an individual job but blocks in the same position in both parents are given to *offspring*. The detailed procedures of SJX2, SBX, and SBX2 are omitted because the rest of the steps are the same.

3.3.6. Mutation

The mutation operation increases the diversity of chromosomes in the population and helps to escape the local optimum. This operation partially alters a solution to generate new *offspring* with a new chromosome. In this study, two types of mutation schemes are considered: *insertion* and *interchange*. The insertion with probability p_{IS} inserts a randomly selected job into a randomly selected position, whereas the interchange with probability $(1 - p_{IS})$ exchanges the positions of two randomly selected jobs.

3.3.7. Improvement (Local Search)

Usually, Gas and other metaheuristics adopt local search methods to improve solutions. The proposed GA also uses a local search strategy of the insertion and interchange used

in the mutation. The local search starts when the current best solution does not improve during a predetermined number of consecutive generations, and insertion with probability p_{IS} or interchange with probability $(1 - p_{IS})$ is applied n times to the current best solution in the population, where n is the number of jobs. If the new solution is better than the current best solution, the new solution replaces the current best solution. To avoid an excessively long computation time, we limit the number of local searches to the current best solution to n times.

3.3.8. Restart

As the procedure of generating the next generation in GA iterates, the population can converge prematurely and get stuck in a local optimum. To avoid premature convergence and enhance the quality of solutions in the population, we use a restart procedure suggested by Ruiz and Maroto [46]. This restart procedure starts when the current best solution does not improve during a predetermined number of consecutive generations. The following is the restart procedure.

- Procedure: Restart

  Step 0. The chromosomes in the current population are sorted in ascending order of their fitness values.
  Step 1. Keep the top 20% of chromosomes and remove the remaining 80% from the population.
  Step 2. 20% of the population size are newly generated by randomly selecting from the kept chromosomes and then mutating once based on the insertion.
  Step 3. 20% of the population size are newly generated by randomly selecting from the kept chromosomes and then mutating once based on the interchange.
  Step 4. 40% of the population size are generated randomly.

3.3.9. Termination Criterion

The GA procedure is terminated at the maximum computation (CPU). In other words, the GA procedure stops when the maximum CPU time elapses.

3.3.10. Entire Procedure of the Proposed GA

The entire procedure of the proposed GA is in Figure 3, in which POP, P_{size}, p_c, p_m, and p_{IS} denote the population, population size, crossover probability, mutation probability, and insertion probability for a mutation operation, respectively. In addition, P_l and P_r denote the parameters that trigger the local search and restart procedures, respectively, whereas g_l and g_r indicate the number of consecutive generations that the current best solution does not improve. Additionally, *random*() represents a function that generates a random number from a uniform distribution with a range (0, 1).

3.4. Iterated Greedy Algorithm

The iterated greedy (IG) algorithm is a metaheuristic developed by Ruiz and Stutzle [47] for the permutation flow shop problem to minimize the makespan. IG is an extension of the NEH algorithm. The main strategy of IG involves the insertion method of NEH stochastically to find better solutions. Not only NEH but also IG was developed originally and particularly for flow shop scheduling problems, and IG has been verified to work well in many research studies [48–51]. The advantage of IG is that the procedure is simple, and there are few parameters that require calibration.

IG is composed of four phases: destruction, construction, local search, and acceptance, and these four phases are executed iteratively. IG also starts with an initial seed sequence. In the destruction phase, d jobs randomly selected from the incumbent solution are removed, where d is a control parameter called destruction size. Then, the construction phase generates a new complete solution by inserting the removed jobs in the constructive way of the NEH. After that, a local search is performed to improve the newly generated schedule.

Finally, the acceptance phase determines whether to accept the new schedule, as in the uphill movement of a simulated annealing algorithm.

```
procedure GA
    Generate an initial population with priority rules, MNEH, and random methods.
    POP ← the current population
    π* ← the best solution in POP
    g_l = g_r = 0

    while(procedure running time < maximum CPU time)
        for(i = 1; i ≤ P_size; i = i + 2)
            parent_1 and parent_2 ← Selection
            if random() < p_c then
                offspring_1 and offspring_2 ← Crossover(parent_1, parent_2)
            else
                offspring_1 ← parent_1 and offspring_2 ← parent_2
            end if
            if random() < p_m then
                Mutation(offspring_1, offspring_2)
            end if
            evaluate the fitness value(offspring_1, offspring_2)
            for(j = 1; j ≤ 2; j++)
                π_w ← the worst solution in POP
                if C_max(offspring_j) < C_max(π_w) then
                    offspring_j replaces π_w
                end if
            end for
        end for
        π_b ← the best solution in POP
        if C_max(π_b) < C_max(π*) then
            π* ← π_b and g_l = g_r = 0
        else
            g_l ← g_l + 1 and g_r ← g_r + 1
        end if
        if g_l == P_l then
            π_b ← the best solution in POP
            Local Search(π_b, p_LS) and g_l = 0
            if C_max(π_b) < C_max(π*) then
                π* ← π_b
                g_r = 0
            end if
        end if
        if g_r == P_r then
            Restart and g_r = 0
            π_b ← the best solution in POP
            if C_max(π_b) < C_max(π*) then
                π* ← π_b
                g_l = 0
            end if
        end if
    end while
    return π*
end procedure
```

Figure 3. Proposed genetic algorithm procedure.

In the IG, an initial (incumbent) solution (π) is obtained from the MNEH (in Section 3.2), and the destruction phase is applied to the incumbent solution. As stated briefly above, in the destruction phase, a given number of d jobs are removed from π, generating two partial schedules. The first one (π_R) consists of the d jobs removed from π. The second one (π_D) contains the $(n - d)$ remaining jobs, i.e., the original one without the removed jobs. Then, in the construction phase, jobs in π_R are inserted into the best position of π_D as in NEH. After the construction phase, a new complete solution (π') with size n is created. Then, the local search procedure which is introduced in Section 3.3.7 of GA is executed to improve the new

solution. For the new solution, the acceptance phase determines whether to replace the current solution with a new one for the next iteration. If $C_{max}(\pi') < C_{max}(\pi)$, the transition to π' is accepted. Otherwise, the transition is accepted with probability

$$exp^{(-\Delta/\tau)}, \tag{14}$$

where Δ represents the difference between the objective function values of the two solutions and is defined as $\Delta = C_{max}(\pi') - C_{max}(\pi)$, and τ is an adjustable parameter (called temperature) in the IG. The temperature value is obtained using Equation (15)

$$\tau = \frac{\sum_{i=1}^{n}\sum_{k=1}^{m+1} p_{ik}}{10n(m+1)}, \tag{15}$$

suggested by Osman and Potts [52] for the permutation flow shop scheduling problem. Note that, in IG, the temperature is constant instead of decreasing as in the simulated annealing. The proposed IG repeats the destruction, construction, local search, and acceptance until the maximum computation time elapses. Figure 4 shows the entire IG procedure.

```
procedure IG
    π* ← π ← NEH
    while(procedure running time < maximum CPU time)
        π_D, π_R ← Destruction(π, d)
        π' ← Construction(π_D, π_R)
        π' ← Local Search(π')
        if C_max(π') < C_max(π) then
            π ← π'
        else
            Δ = C_max(π') − C_max(π)
            if random() < exp(− Δ/τ) then
                π ← π'
            end if
        end if
        if C_max(π) < C_max(π*) then
            π* ← π
        end if
    end while
    return π*
end procedure

procedure Destruction(π, d)
    π_R ← ∅
    π_D ← π
    for(i = 1; i ≤ d; i++)
        j ← a randomly selected job in π_D
        Remove j from π_D and      add to π_R
    end for
    return π_R and π_D
end procedure

procedure Construction(π_D, π_R)
    for(i = 1; i ≤ d; i++)
        j ← the ith job in π_R
        Insert j into the best position of π_D, achieving the minimum makespan
    end for
    return π_D
end procedure
```

Figure 4. Proposed iterated greedy algorithm procedures.

3.5. Simulated Annealing Algorithm

Simulated annealing (SA) mimics the physical annealing process, and it is a meta-heuristic optimization algorithm that works dynamically and iteratively to search for a better solution [53]. SA has been commonly used to solve various combinatorial optimization problems including scheduling problems. SA also begins with an initial seed sequence and attempts to generate a better solution by repeating small alterations to the current solution. However, such attempts may lead to being stuck in a local optimum. To escape the local optimum, SA occasionally allows uphill movements that accept worse solutions deteriorating the objective function value (makespan). Uphill movements are allowed with an acceptance probability of Equation (14).

In this study, SA also begins with an initial seed sequence obtained from MNEH and uses the insertion with probability p_{IS} and interchange with probability $(1 - p_{IS})$ to generate a neighborhood solution (π') of a current solution (π). If $C_{max}(\pi') < C_{max}(\pi)$, the transition to π' is accepted. Otherwise, the transition is accepted with Equation (14), and the initial τ is obtained using Equation (15). SA repeats these generation and transition procedures L times at the current temperature, where L is an epoch length as a parameter. After repeating L times, the temperature decreases gradually by a cooling function as $\tau \leftarrow \alpha\tau$, where $0 < \alpha < 1$, and this SA procedure terminates when the maximum CPU time elapses. Figure 5 shows the procedure of the SA.

```
procedure SA
    π* ← π ← NEH
    while(procedure running time < maximum CPU time)
        for(l = 1; l ≤ L; l++)
            π′ ← π
            (pos1, pos2) ← two positions randomly selected from π
            if random() < pIS then
                Insert the job at pos1 into pos2 in π′
            else
                Interchange the jobs at pos1 and pos2 in π′
            end if
            if Cmax(π′) < Cmax(π) then
                π ← π′
            else
                Δ = Cmax(π′) − Cmax(π)
                if random() < e^(−Δ/τ) then
                    π ← π′
                end if
            end if
            if Cmax(π) < Cmax(π*) then
                π* ← π
            end if
        end for
        τ ← ατ
    end while
    return π*
end procedure
```

Figure 5. Proposed simulated annealing algorithm procedure.

4. Computational Experiments

This section evaluates the proposed algorithms on randomly generated problem instances. All the algorithms were coded in Java, and the tests were conducted on a personal computer with Intel Core i7-8700 CPU (3.2 GHz) and 16 GB RAM. To generate problem instances, we used the same method in [25]. In the first stage, there are m dedicated

machines, and three levels of m were used, that is, m = (2, 5, 10), whereas there is a single machine in the second stage. Thus, there are a total of (m + 1) machines in the tests. The processing times were generated from a uniform distribution. Table 2 shows the discrete uniform distribution $U(a, b)$ with a range between a and b. In Set A, the workloads of the two stages are equal. On the contrary, in Sets B and C, the second and first stages become bottlenecks, respectively. Limited waiting times were also generated from $U(1, 100)$.

Table 2. Processing time distribution.

Set	First Stage	Second Stage
A	$U(1, 100)$	$U(1, 100)$
B	$U(1, 80)$	$U(20, 100)$
C	$U(20, 100)$	$U(1, 80)$

The first experiment evaluated the mixed integer programming formulation (MIP) through a commercial optimization solver CPLEX 12.10. In the experiment, we considered five levels of jobs (n = 10, 20, 30, 40, and 50) and generated five instances for each of the (m, n) combinations. In addition, the computation time limit (T_L) for each instance was set to 1800 s to avoid excessive computation times. A summary of the results is presented in Table 3, including the average computation times and the number of instances not terminated until the time limit T_L. As can be observed from the table, the ACPUT in Set A increased exponentially as (m, n) increased. For m = (5 and 10), CPLEX found an optimal solution only for instances with n = 10 within 1800 s. On the contrary, CPLEX solved instances of Sets B and C relatively faster than those in Set A. One drawback of CPLEX in this test is its wide range of computation times. For (m = 10, n = 20) in all three sets, the minimum CPU time was only a few seconds, whereas the maximum time was 1800 s. If CPLEX shows a significant range of CPU times, the estimates for solving times are inaccurate and even impossible; hence, using CPLEX may be impractical.

Table 3. Performance of MIP using CPLEX.

m	n	CPUT [1]			NI [2]		
		Set A	Set B	Set C	Set A	Set B	Set C
2	10	0.1 [0, 0.2]	0.1 [0, 0.1]	0 [0, 0]	0	0	0
	20	58.9 [0.1, 172.2]	0.1 [0.1, 0.1]	0.1 [0.1, 0.1]	0	0	0
	30	716.2 [0.3, T_L]	0.6 [0.1, 2.2]	0.3 [0.2, 0.4]	1	0	0
	40	754.7 [6.2, T_L]	0.4 [0.2, 0.5]	0.4 [0.2, 0.5]	2	0	0
	50	T_L	0.8 [0.5, 1.3]	0.7 [0.4, 0.9]	5	0	0
5	10	1.2 [0, 2.7]	0.4 [0, 1.3]	0.1 [0, 0.1]	0	0	0
	20	T_L	98.2 [0.2, 489.9]	2.8 [0.1, 12.9]	5	0	0
	30	T_L	360.6 [0.4, T_L]	1.2 [0.6, 2.5]	5	1	0
	40	T_L	15.6 [0.5, 70.9]	177.8 [0.9, 873.8]	5	0	0
	50	T_L	344.6 [1.6, 1630.4]	207.8 [2.6, 998.8]	5	0	0
10	10	6.3 [2.5, 13]	0.8 [0.1, 2.3]	1.4 [0.1, 2.4]	0	0	0
	20	1440.7 [3.4, T_L]	1075.8 [1.8, T_L]	1080.4 [0.6, T_L]	4	2	3
	30	T_L	1086.3 [5.6, T_L]	527.5 [3.3, T_L]	5	3	1
	40	T_L	1419.2 [54.8, T_L]	1455.2 [76, T_L]	5	3	4
	50	T_L	805.5 [26.2, T_L]	1756.3 [1637.6, T_L]	5	2	3

[1] Average [min, max] computation time, where T_L = 1800 s. [2] Number of instances not terminated within 1800 s.

Next, we tested the priority rule-based list scheduling methods to determine the best method for offering an initial seed sequence for MNEH. The same instances with n = 10, 20, 30, 40, and 50 were used in these tests. To compare the list scheduling methods, the relative deviation index (RDI) was used as a measure, defined as $(obj_{\#} - obj_{best})/(obj_{worst} - obj_{best})$ for each instance where $obj_{\#}$ represents the makespan using priority rule #, and obj_{best} and

obj_{worst} represent the best and worst, respectively. According to the definition of RDI, the lower the RDI, the better the performance.

A summary of the results is in Table 4 which provides the average RDI and number of instances (out of 75) that LS# found the best solutions. As observed from the table, LS1 outperformed the other rules. This indicates that the first stage on which LS1 focuses is more important than the second stage, probably because the decision on sequencing jobs on the parallel machines in the first stage must consider the limited waiting time constraints. However, in the second stage, there is a single assembly machine, and the sequence is equivalent to the sequence in the first stage because this study considers only permutation schedules. Thus, if a shop manager wants to focus on either stage, the result recommends the first stage. In summary, LS1 showed the best performance among the proposed list scheduling rules, therefore LS1 will offer an initial seed sequence for MNEH in the subsequence experiments.

Table 4. Comparison of the list scheduling methods (the best values are in bold).

Set	LS1	LS2	LS3	LS4	LS4	LS6
A	**0.286** (**29**) [1]	0.886 (0)	0.242 (29)	0.527 (12)	0.466 (11)	0.621 (2)
B	**0.165** (**42**)	0.906 (0)	0.480 (9)	0.459 (11)	0.570 (5)	0.436 (9)
C	**0.195** (**40**)	0.900 (0)	0.488 (7)	0.513 (13)	0.518 (8)	0.456 (13)
Overall	**0.215** (**111**)	0.897 (0)	0.403 (45)	0.500 (36)	0.518 (24)	0.504 (24)

[1] Average RDI, number of instances (out of 75) that the method found the best solutions.

To check the efficiency of the interchange method in the proposed MNEH, comparison tests were performed on the instances with up to $n = 50$. In this test, initial seed sequences were obtained using LS1. Table 5 and Figure 6 show the results that show the effectiveness of the interchange method in MNEH. The table reports the percentage change defined as $100 \times (obj_{MNEH} - obj_{NEH})/obj_{NEH}$. Note that the negative percentage change indicates that MNEH found a better solution than NEH. As revealed in the table, MNEH worked better than NEH, which indicates that the simple interchange method can improve the solution quality. In addition, the improvement was more apparent when the problem size increased. The same result was obtained in Set A than in Sets B and C as shown in Figure 6. Thus, it can be concluded that the interchange method used in MNEH worked well. In addition, although the interchange procedure was added, MNEH provided a solution for all instances in less than a second.

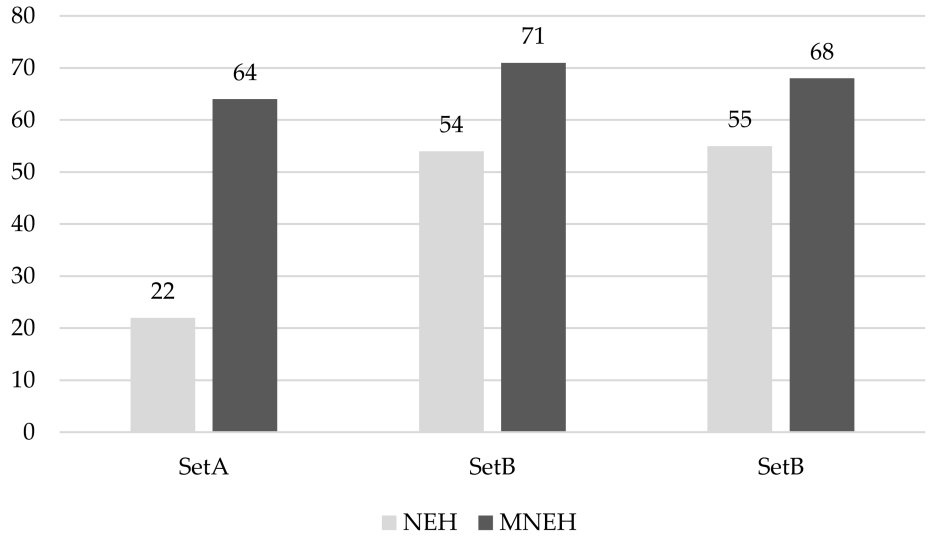

Figure 6. Number of instances that NEH solved better than or equivalent to MNEH and vice versa.

Table 5. Improvement by the interchange method in the proposed MNEH algorithm.

m	*n*	Set A	Set B	Set C
2	10	0.181 [1]	0.000	0.000
	20	−1.091	0.000	0.048
	30	0.155	0.011	−0.024
	40	−0.720	0.000	0.000
	50	−1.638	0.000	0.000
5	10	−0.935	0.000	0.000
	20	−1.418	0.000	−0.368
	30	0.053	−0.353	−0.178
	40	−0.988	−0.017	−0.369
	50	−0.909	−0.140	0.116
10	10	−1.473	0.076	−0.260
	20	−1.141	−0.779	−0.835
	30	−0.993	−0.204	−0.353
	40	−1.224	−0.451	−0.145
	50	−0.599	−0.050	−0.311
Overall		−0.849	−0.127	−0.179

[1] Percentage change, i.e., $100 \times (obj_{MNEH} - obj_{NEH})/obj_{NEH}$.

To achieve the best performance of the proposed metaheuristic algorithms (GA, IG, and SA), calibration experiments were performed. In general, calibration before using a metaheuristic is essential because the performance depends significantly on the control parameters. In other to calibrate the metaheuristics, we generated three instances for each combination of (m, n), with three levels of machines (m = 2, 5, and 10) and four levels of jobs (n = 50, 100, 300, and 500), on Set A. The algorithms solved each instance independently three times, and the average value resulting from the three runs was used to compute the RDI for comparison. For the termination condition, we set the maximum CPU time defined as $t_{max} = n(m + 1)(tf)/2$ ms, where tf denotes the time factor and tf = 60 in this calibration.

The proposed GA has eight operators and parameters which are listed below.

- Selection: two levels (tournament and roulette).
- Crossover: six levels (OX1, OX2, SJX, SJX2, SBX, and SBX2).
- Mutation (p_{IS}): five levels (0, 0.25, 0.5, 0.75, and 1).
- Population size (P_{size}): five levels (20, 30, 40, 50, and 60).
- Crossover probability (p_c): five levels (0.0, 0.1, 0.2, 0.3, and 0.4).
- Mutation probability (p_m): five levels (0.6, 0.7, 0.8, 0.9, and 1.0).
- Local search (P_l): five levels (30, 40, 50, 60, and 70).
- Restart (P_r): five levels (50, 60, 70, 80, and 90).

For the operators and parameters, 468,750 combinations were possible. To simplify this calibration, we performed two separate 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 first experiment, a full factorial design for selection, crossover, and mutation was considered; hence, 60 different algorithms were tested. The remaining parameters were fixed as (P_{size}, p_c, p_m, P_l, P_r) = (50, 1, 1, ∞, ∞). These values were intended to make the parameters meaningless and to focus on the effects of selection, crossover, and mutation on the solutions. Figure 7 illustrates the main effect plot to determine the best operators with lower RDI values. As shown in the figure, (tournament, OX1 and 0.5) showed better performance. This was likely because many changes by crossover operation did not lead to better sequences in this problem with limited waiting time constraints. For mutation, the use of insertion and interchange with the same probability provided good results. Therefore, tournament, OX1 and p_{IS} = 0.5 were used in subsequent experiments.

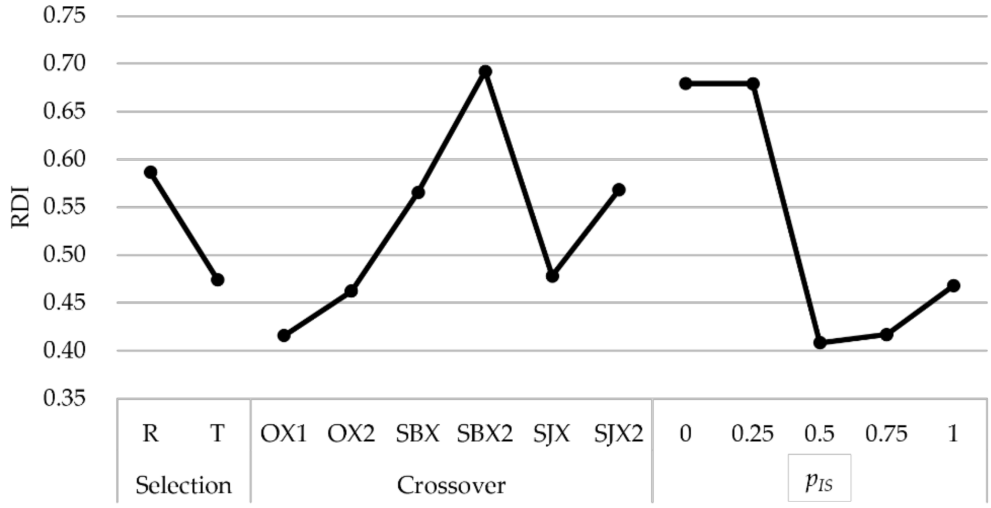

Figure 7. GA calibration for selection, crossover, and mutation.

The second experiment examined the remaining parameters (P_{size}, p_c, p_m, P_l, P_r). The full factorial design of the parameters could make 3125 combinations. However, because testing these combinations requires significant time, we adopted the Taguchi L_{25} orthogonal array design instead of a full factorial design. The L_{25} design is provided in Table 6; that is, the 25 combinations provided in the table were tested. Note that the Taguchi design developed by Genichi Taguchi can achieve parametric research and optimization while reducing the number of experiments or numerical tests [54]. Figure 8 shows that the lowest RDIs for the parameters were achieved at (P_{size}, p_c, p_m, P_l and P_r) = (30, 0.2, 1, 60 and 60). Remarkably, the probabilities of crossover and mutation were opposite. With these probabilities, the GA occasionally executes crossover, but it always performs mutation operations. Considering the calibration results, i.e., OX1 and p_c = 0.2, the mutual interchange between two solutions may not help search for better solutions to this problem. In contrast, the mutation was more useful for improving the solution. The values of P_l and P_r were the same as 60. This indicates that the local search and restart procedures start when the best solution did not improve during 60 generations.

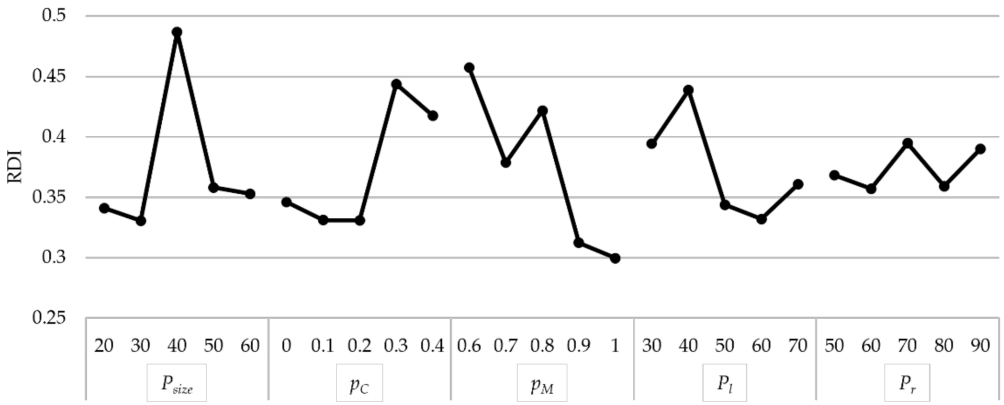

Figure 8. GA calibration for P_{size}, p_C, p_M, P_l, and P_r.

Table 6. Orthogonal array for GA calibration.

Experiment	P_{size}	p_C	p_M	P_l	P_r
1	20	0	0.6	50	30
2	20	0.1	0.7	60	40
3	20	0.2	0.8	70	50
4	20	0.3	0.9	80	60
5	20	0.4	1	90	70
6	30	0	0.7	70	60
7	30	0.1	0.8	80	70
8	30	0.2	0.9	90	30
9	30	0.3	1	50	40
10	30	0.4	0.6	60	50
11	40	0	0.8	90	40
12	40	0.1	0.9	50	50
13	40	0.2	1	60	60
14	40	0.3	0.6	70	70
15	40	0.4	0.7	80	30
16	50	0	0.9	60	70
17	50	0.1	1	70	30
18	50	0.2	0.6	80	40
19	50	0.3	0.7	90	50
20	50	0.4	0.8	50	60
21	60	0	1	80	50
22	60	0.1	0.6	90	60
23	60	0.2	0.7	50	70
24	60	0.3	0.8	60	30
25	60	0.4	0.9	70	40

The suggested IG has two parameters: d and p_{IS}. This calibration considered five levels of each parameter, that is, d = (6, 8, 10, 12, and 14) and p_{IS} = (0, 0.25, 0.5, 0.75, and 1). Thus, we tested 25 combinations. A summary of the results is presented in Figure 9. As shown in the figure, RDI had the lowest at d = 10 and the difference in RDIs between d values was very clear. This value differed significantly from the original value of 4 which was experimentally found in [47]. It is likely because computing power improved compared to in the past. Note that the destruction size d is determined by considering the trade-off between the solution quality and computation time. That is, a large d can provide good solutions but requires a large amount of computation. Therefore, in a limited CPU time, if the d value is too large, IG may provide a poor solution. The effect of the p_{IS} was not significant. This may be because IG is based on the insertion mechanism. Consequently, the IG uses d = 10 and p_{IS} = 0.75.

For calibration of the proposed SA, we considered five levels of each of the three parameters (α, L, and p_{IS}). In this experiment, the epoch length L was defined as $L = n \times l$, where l is a control parameter. The design of this calibration was also based on the Taguchi L_{25} orthogonal array (Table 7). Figure 10 shows the calibration results. Three parameters (α, l, and p_{IS}) had the lowest RDI values at (0.995, 15, and 0.25), respectively. In contrast to GA and IG, the results showed bath-curves for all the parameters. In other words, both higher and lower values from the best values resulted in worse outcomes. This is probably because SA generates a neighborhood solution using only a local search of insertion and interchange without any other methods. From the value of p_{IS} = 0.25, the interchange appears to be more effective in SA. Therefore, the SA used α = 0.995, l = 15, and p_{IS} = 0.25 in subsequence tests.

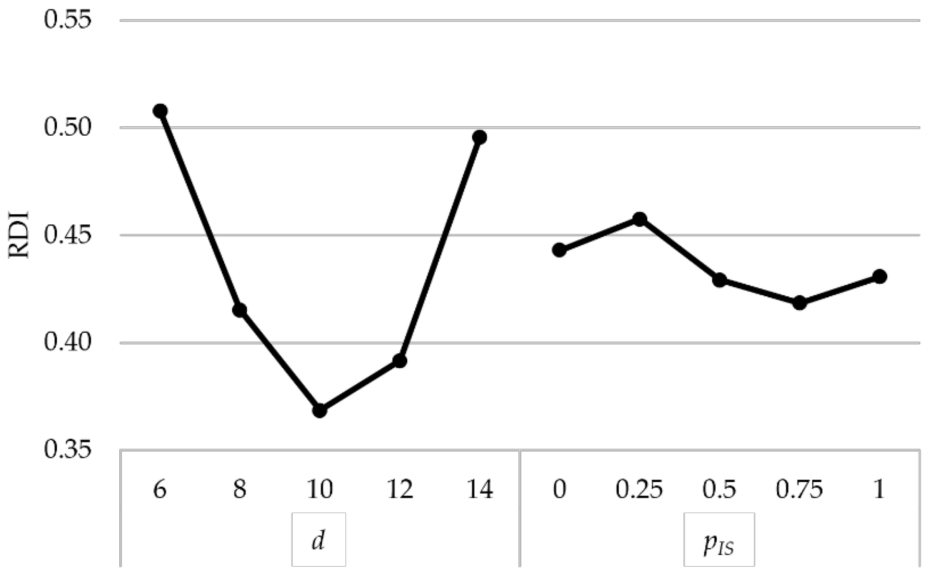

Figure 9. IG calibration.

Table 7. Orthogonal array for SA calibration.

Experiment	α	l	p_{IS}
1	0.9	1	0
2	0.9	15	0.25
3	0.9	30	0.5
4	0.9	45	0.75
5	0.9	60	1
6	0.95	1	0.25
7	0.95	15	0.5
8	0.95	30	0.75
9	0.95	45	1
10	0.95	60	0
11	0.99	1	0.5
12	0.99	15	0.75
13	0.99	30	1
14	0.99	45	0
15	0.99	60	0.25
16	0.995	1	0.75
17	0.995	15	1
18	0.995	30	0
19	0.995	45	0.25
20	0.995	60	0.5
21	0.999	1	1
22	0.999	15	0
23	0.999	30	0.25
24	0.999	45	0.5
25	0.999	60	0.75

After calibrating the proposed metaheuristics, we examined the solutions obtained from the heuristics by comparing them with those from CPLEX. Here, we considered four levels of n = 20, 30, 40, and 50 because the destruction size d of IG was determined to be 10 in the calibration. Metaheuristics independently solved each instance five times, and the average values of the five independent results were used. In addition, for the termination condition t_{max} $n(m + 1)(tf)/2$ ms, three levels of tf = (30, 60, and 90) were considered, and let A_{tf} be the algorithm A using tf = (30, 60, and 90). Table 8 summarizes

the comparison results, which shows the average percentage gaps of heuristic solutions from a solution using CPLEX, and Figure 11 represents the number of instances for which the heuristic solutions are better than or equal to the CPLEX solutions. The negative percentage gap indicates that the heuristic solutions are better than those of the CPLEX. Overall, the IG algorithm exhibited the best performance. For 179 out of 180 instances, IG found solutions better than or equal to those of CPLEX. Both algorithms were better than CPLEX, and SA outperformed GA. For the three levels of *tf*, only a few changes were observed in all cases. This indicates that the proposed metaheuristics can find a solution close to an optimal solution within a short computation time for instances of up to $n = 50$. These experimental results verified the effectiveness and efficiency of the proposed metaheuristics. Meanwhile, MNEH showed percentage gaps of approximately 2% in Set A and less than 1% in Sets B and C. Although MNEH was overwhelmed by the metaheuristics, its performance is acceptable, bearing in mind that MNEH is a constructive heuristic.

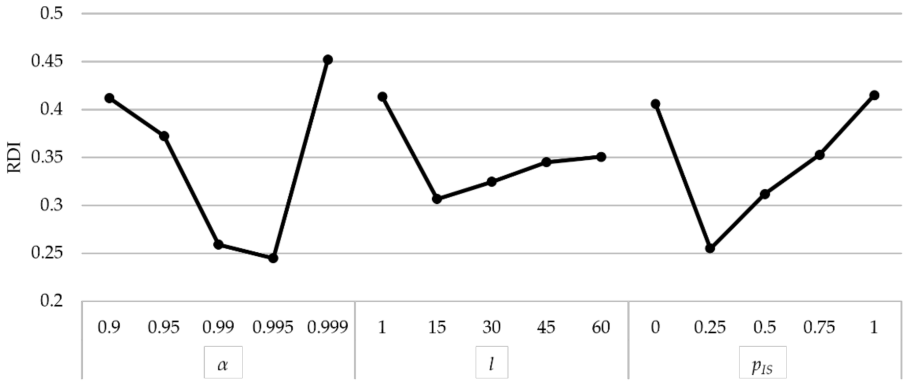

Figure 10. SA calibration.

Table 8. Average percentage gaps of heuristic solutions from a solution using CPLEX.

Heuristic	Set A	Set B	Set C
MNEH	2.110 [1]	0.408	0.356
GA_{30}	−1.299	−0.032	−0.121
GA_{60}	−1.362	−0.035	−0.128
GA_{90}	−1.393	−0.036	−0.130
IG_{30}	−1.869	−0.073	−0.175
IG_{60}	−1.944	−0.075	−0.178
IG_{90}	**−1.976**	**−0.079**	**−0.180**
SA_{30}	−1.260	−0.065	−0.126
SA_{60}	−1.262	−0.065	−0.126
SA_{90}	−1.262	−0.065	−0.126

[1] Average percentage gap, $100 \times (obj_{Heuristic} - obj_{CPLEX})/obj_{CPLEX}$.

We also examined the performance of the heuristics on large instances with $n = 100$, 200, 300, 400, and 500. A summary of the comparison results is in Figure 12 which shows the RDI values of the heuristic algorithms and Figure 13 which represents the number of instances that the heuristic found the best solution. Overall, SA outperformed the other algorithms, and even SA_{30} was better than GA_{90} and IG_{90}. These results were remarkably different from those of the previous test with small instances of up to $n = 50$. This is probably because of the trade-off relationship between solution quality and computation time. In IG, the greedy reinsertion method checks all possible positions to find the best one, and thus, this mechanism is very effective in small instances. However, this technique becomes computationally burdensome when n increases. In contrast, the SA procedure, which uses only a local search method with insertion and interchange, is relatively simple. That is, SA generates a new solution by slightly changing the current solution, and the

time to create a new solution rarely increases even if n increases. Therefore, in terms of the trade-off relation, it can be said that SA is much more effective and efficient. On the contrary, for the instances with m = (2 and 5) of Sets B and C, there was no difference in the performance of all algorithms, including MNEH. However, for instances with m = 10, SA overwhelmed the other algorithms, and IG performed better than GA. A summary of the experimental results thus far reveals that attempts to improve the solution by changing the sequence slightly are desirable in this scheduling problem.

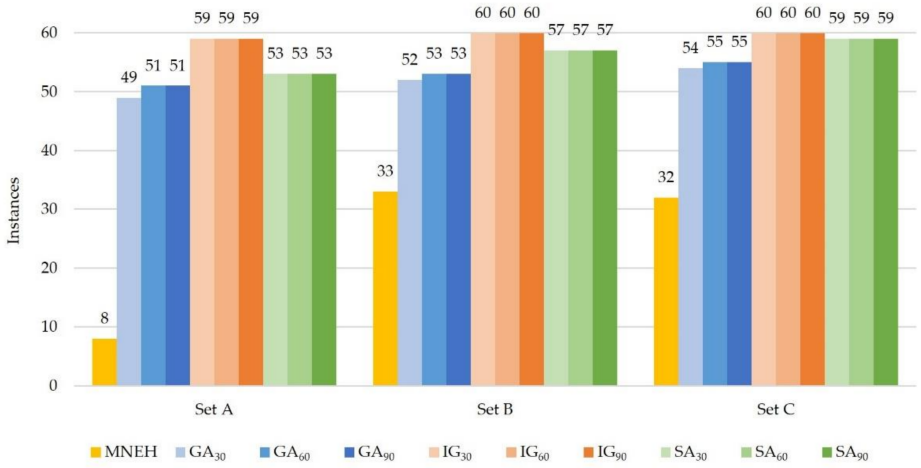

Figure 11. Number of instances that heuristic solutions are better than or equal to CPLEX solutions.

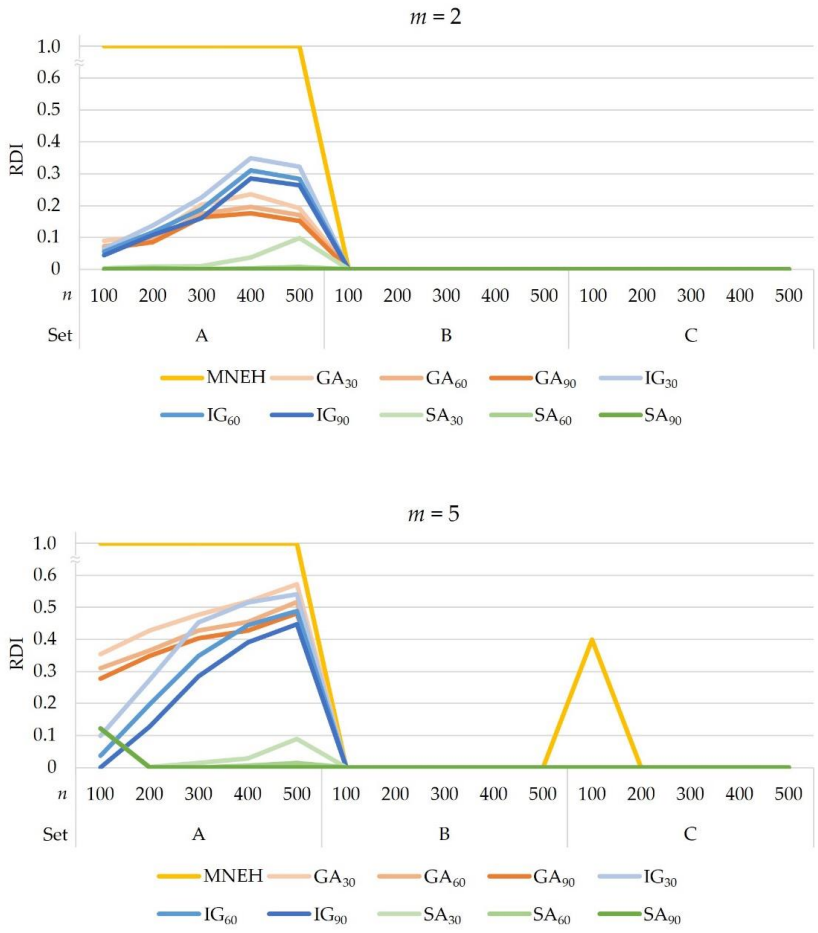

Figure 12. *Cont.*

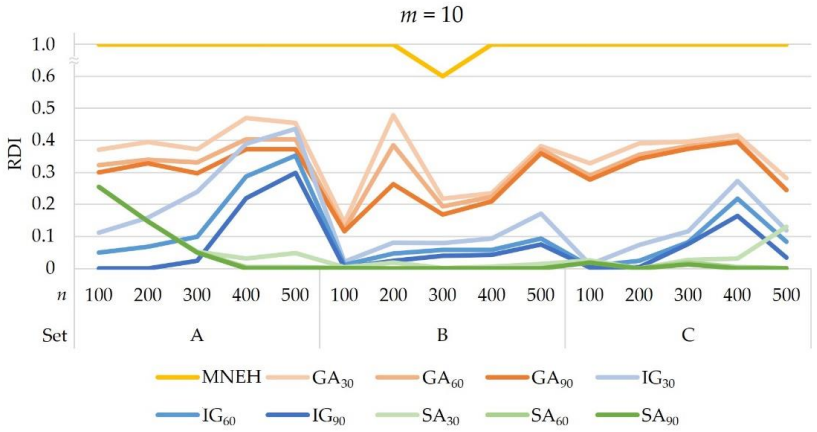

Figure 12. RDIs of the heuristic algorithms on large instances.

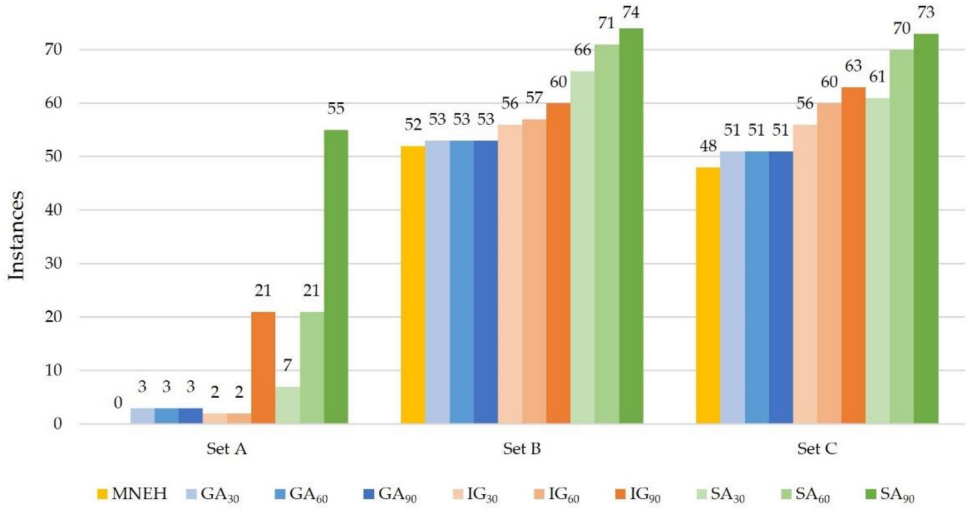

Figure 13. Number of instances that the heuristic found the best solution.

In addition, we performed statistical tests to see statistically significant differences in the performance of the proposed metaheuristics. This test used RDI values obtained from GA_{90}, IG_{90}, and SA_{90} on large-size instances of n = 100, 200, 300, 400, and 500. Figure 14 shows the 95% confidence interval for the mean RDIs of the algorithms. As shown in Figure 14, SA_{90} performed significantly better in terms of the mean RDI. We also performed Kruskal–Wallis tests which is a non-parametric method. Table 9 shows the results of pairwise comparisons from the tests. For the tests in Set A, SA_{90} significantly outperformed GA_{90} and IG_{90}. However, there is no significant difference between SA and IG in Sets B and C. This is likely because the RDIs for m = 2 and 5 were zero.

The last analysis shows the effect of the waiting time constraints on solutions. For this analysis, we generated 10 instances of (m, n) = (5, 300) in Set A. The waiting times were generated randomly with a range of [1, 100], and these values were treated by multiplying the ratios (0.5, 1.0, 1.5, 2.0, 2.5, and 3.0). We analyzed the results obtained by solving each instance independently five times using SA_{90} which showed the best performance. As shown in Figure 15, the makespan decreased as the waiting time ratio increased. This is because tight limited waiting times caused delayed starts; hence, the makespan increased accordingly. The decline of the makespan slowed when the ratio exceeded 2. That is, if the limited waiting times are twice the processing times, the effect of the waiting time limit on the solution is reduced. These results suggest the need for manufacturing technologies and systems that can achieve product quality while setting enough limited waiting times. It may also provide a guideline for considering the trade-off between production efficiency

and product quality. Meanwhile, the range of the objective values increased as the ratio increased. This may be due to the loose waiting time limits that increased the time range in which jobs could start. In other words, tight waiting time limits reduced the solution space for start times in a problem.

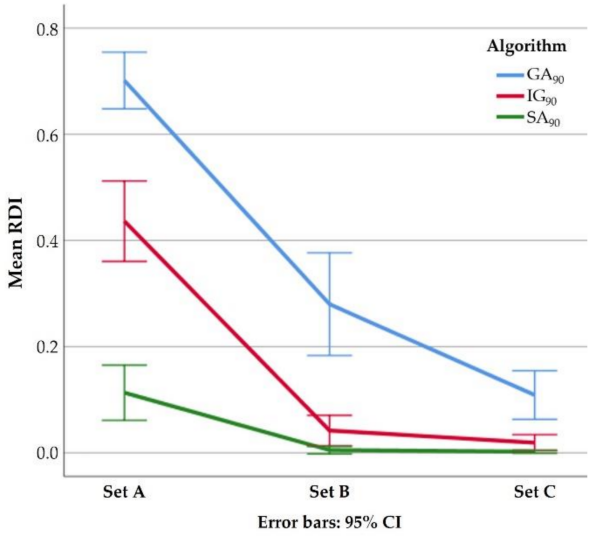

Figure 14. 95% CIs for the mean RDI of algorithms.

Table 9. Pairwise comparisons from Kruskal–Wallis tests.

Set	Sample 1 -Sample 2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj. Sig. [1]
A	SA_{90}-IG_{90}	53.800	10.398	5.174	0.000	0.000
	SA_{90}-GA_{90}	103.840	10.398	9.987	0.000	0.000
	IG_{90}-GA_{90}	50.040	10.398	4.813	0.000	0.000
B	SA_{90}-IG_{90}	13.480	6.937	1.943	0.052	0.156
	SA_{90}-GA_{90}	35.480	6.937	5.114	0.000	0.000
	IG_{90}-GA_{90}	22.000	6.937	3.171	0.002	0.005
C	SA_{90}-IG_{90}	13.987	6.937	2.016	0.044	0.131
	SA_{90}-GA_{90}	34.853	6.937	5.024	0.000	0.000
	IG_{90}-GA_{90}	20.867	6.937	3.008	0.003	0.008

[1] Significance values adjusted by the Bonferroni correction for multiple tests.

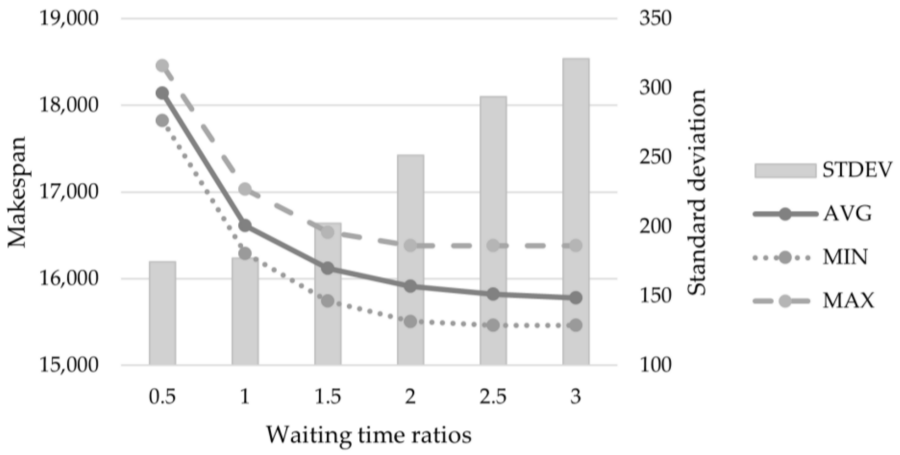

Figure 15. Effect of limited waiting time on solutions.

5. Discussion and Conclusions

This study investigated a scheduling problem in a two-stage assembly-type flow shop with limited waiting time constraints to minimize the makespan. For this problem, a mixed-integer programming formulation was provided to mathematically describe and find an optimal solution using CPLEX. If the limited waiting times are treated as zero time, we can also use the MIP for solving a no-wait constrained problem. This problem is NP-hard, which requires an excessive computational time to solve the problem optimally. Therefore, this study focused on heuristic algorithms to quickly find good solutions. Various heuristic algorithms were provided, including priority rule-based list scheduling methods, a modified NEH algorithm, a genetic algorithm, an iterated greedy algorithm, and a simulated annealing algorithm. In addition, this study suggested equations to calculate the completion times of jobs in a given sequence. Computational experiments demonstrated that IG and SA in small- and large-sized problems, respectively, were effective and efficient within a short computation time.

This study had some limitations. First, this study considered only two stages and a single machine in the second stage. Next, the suggested problem assumed that all jobs are ready to process at time zero and considered the given and fixed information of jobs. In addition, because finding exact solutions for large problems requires a significant computation time or may be impossible, the solutions obtained by the heuristics were examined by comparing them with each other. These limitations necessitate further studies.

Future studies may need to consider more general models, such as multiple stages and machines in each stage. In addition, it may be necessary to consider a problem in which jobs arrive at the first stage dynamically or a problem with stochastic processing times. Furthermore, the multi-objective of the throughput of the system and the due date-based measure may also be considered in future studies. Additionally, one may develop an exact algorithm or effective lower bounds to evaluate the solutions obtained by the heuristic algorithms. Finally, optimal solution properties may be necessary to reduce the solution space to be explored.

Author Contributions: Conceptualization, J.-H.H. and J.-Y.L.; methodology, J.-Y.L.; software, J.-Y.L.; validation, J.-H.H.; formal analysis, J.-Y.L.; investigation, J.-H.H. and J.-Y.L.; resources, J.-Y.L.; data curation, J.-Y.L.; writing—original draft preparation, J.-Y.L.; writing—review and editing, J.-H.H. and J.-Y.L.; visualization, J.-H.H. and J.-Y.L.; 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 the National Research Foundation of Korea (NRF) Grant Funded by the Korean Government (MSIT) (no. NRF-2020R1G1A1006268, NRF-2020R1G1A1099829). The APC was funded by NRF-2020R1G1A1006268.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data for the computational experiments are generated randomly and the method for generating data is written in Section 4 in the paper.

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

References

1. Lee, J.Y.; Chin, M.S. Research trends of scheduling techniques for domestic major industries. *J. Soc. Korea Ind. Syst. Eng.* **2018**, *41*, 59–69. [CrossRef]
2. Komaki, G.M.; Sheikh, S.; Malakooti, B. Flow shop scheduling problems with assembly operations: A review and new trends. *Int. J. Prod. Res.* **2019**, *57*, 2926–2955. [CrossRef]
3. Lee, C.-Y.; Cheng, T.C.E.; Lin, B.M.T. Minimizing the makespan in the 3-machine assembly scheduling problem. *Manag. Sci.* **1993**, *39*, 616–625. [CrossRef]
4. Potts, C.N.; Sevast'janov, S.V.; Strusevich, V.A.; Van-Wassenhove, L.N.; Zwaneveld, C.M. The two-stage assembly scheduling problem: Complexity and approximation. *Oper. Res.* **1995**, *43*, 346–355. [CrossRef]

5. Lagodimos, A.G.; Mihiotis, A.N.; Kosmidis, V.C. Scheduling a multi-stage fabrication shop for efficient subsequent assembly operations. *Int. J. Prod. Econ.* **2004**, *90*, 345–359. [CrossRef]
6. Yokoyama, M. Scheduling for two-stage production system with setup and assembly operations. *Comput. Oper. Res.* **2004**, *31*, 2063–2078. [CrossRef]
7. Blocher, J.D.; Chhajed, D. Minimizing Customer Order Lead-Time in a Two-Stage Assembly Supply Chain. *Ann. Oper. Res.* **2008**, *161*, 25–52. [CrossRef]
8. Bard, J.F.; Jia, S.; Chacon, R.; Stuber, J. Integrating optimisation and simulation approaches for daily scheduling of assembly and test operations. *Int. J. Prod. Res.* **2015**, *53*, 2617–2632. [CrossRef]
9. Gholami-Zanjani, S.M.; Hakimifar, M.; Nazemi, N.; Jolai, F. Robust and fuzzy optimisation models for a flow shop scheduling problem with sequence dependent setup times: A real case study on a PCB assembly company. *Int. J. Comput. Integ. Manuf.* **2017**, *30*, 552–563. [CrossRef]
10. Hayrinen, T.; Johnsson, M.; Johtela, T.; Smed, J.; Nevalainen, O. Scheduling algorithms for computer-aided line balancing in printed circuit board assembly. *Prod. Plann. Control* **2000**, *11*, 497–510. [CrossRef]
11. Jin, Z.H.; Ohno, K.; Ito, T.; Elmaghraby, S.E. Scheduling hybrid flowshops in printed circuit board assembly lines. *Prod. Oper. Manag.* **2002**, *11*, 216–230. [CrossRef]
12. Ju, F.; Li, J.S.; Horst, J.A. Transient Analysis of Serial Production Lines With Perishable Products: Bernoulli Reliability Model. *IEEE Trans. Autom. Control.* **2017**, *62*, 694–707. [CrossRef]
13. Wang, J.W.; Hu, Y.; Li, J.S. Transient analysis to design buffer capacity in dairy filling and packing production lines. *J. Food. Eng.* **2010**, *98*, 1–12. [CrossRef]
14. 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; 2018; pp. 497–501.
15. Framinan, J.M.; Perez-Gonzalez, P.; Fernandez-Viagas, V. Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *Eur. J. Oper. Res.* **2019**, *273*, 401–417. [CrossRef]
16. Johnson, S.M. Optimal two- and three-stage production schedules with setup times included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–68. [CrossRef]
17. Hariri, A.M.A.; Potts, C.N. A branch and bound algorithm for the two-stage assembly scheduling problem. *Eur. J. Oper. Res.* **1997**, *103*, 547–556. [CrossRef]
18. Haouari, M.; Daouas, T. Optimal scheduling of the 3-machine assembly-type flow shop. *RAIRO Rech. Oper.* **1999**, *33*, 439–445. [CrossRef]
19. Sun, X.; Morizawa, K.; Nagasawa, H. Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling. *Eur. J. Oper. Res.* **2003**, *146*, 498–516. [CrossRef]
20. Koulamas, C.; Kyparisis, G.J. The three stage assembly flowshop scheduling problem. *Comput. Oper. Res.* **2001**, *28*, 689–704. [CrossRef]
21. Sung, C.S.; Juhn, J. Makespan minimization for a 2-stage assembly scheduling problem subject to component available time constraint. *Int. J. Prod. Econ.* **2009**, *119*, 392–401. [CrossRef]
22. Wu, C.-C.; Gupta, J.N.D.; Cheng, S.-R.; Lin, B.M.T.; Yip, S.-H.; Lin, W.-C. Robust scheduling for a two-stage assembly shop with scenario-dependent processing times. *Int. J. Prod. Res.* **2021**, *59*, 5372–5387. [CrossRef]
23. Wu, C.-C.; Zhang, X.; Azzouz, A.; Shen, W.-L.; Cheng, S.-R.; Hsu, P.-H.; Lin, W.-C. Metaheuristics for two-stage flow-shop assembly problem with a truncation learning function. *Eng. Optimiz.* **2021**, *53*, 843–866. [CrossRef]
24. Lee, J.Y.; Bang, J.Y. A two-stage assembly-type flowshop scheduling problem for minimizing total tardiness. *Math. Probl. Eng.* **2016**, *2016*. [CrossRef]
25. Lee, I.S. Minimizing total completion time in the assembly scheduling problem. *Comput. Ind. Eng.* **2018**, *122*, 211–218. [CrossRef]
26. Lee, I.S. A scheduling problem to minimize total weighted tardiness in the two-stage assembly flowshop. *Math. Probl. Eng.* **2020**, *2020*. [CrossRef]
27. Azzouz, A.; Pan, P.A.; Hsu, P.H.; Lin, W.C.; Liu, S.C.; Ben Said, L.; Wu, C.C. A two-stage three-machine assembly scheduling problem with a truncation position-based learning effect. *Soft Comput.* **2020**, *24*, 10515–10533. [CrossRef]
28. Wu, C.C.; Wang, D.J.; Cheng, S.R.; Chung, I.H.; Lin, W.C. A two-stage three-machine assembly scheduling problem with a position-based learning effect. *Int. J. Prod. Res.* **2018**, *56*, 3064–3079. [CrossRef]
29. Wu, C.C.; Bai, D.Y.; Azzouz, A.; Chung, I.H.; Cheng, S.R.; Jhwueng, D.C.; Lin, W.C.; Ben Said, L. A branch-and-bound algorithm and four metaheuristics for minimizing total completion time for a two-stage assembly flow-shop scheduling problem with learning consideration. *Eng. Optimiz.* **2020**, *52*, 1009–1036. [CrossRef]
30. Talens, C.; Fernandez-Viagas, V.; Perez-Gonzalez, P.; Framinan, J.M. New efficient constructive heuristics for the two-stage multi-machine assembly scheduling problem. *Comput. Ind. Eng.* **2020**, *140*, 106223. [CrossRef]
31. Wu, C.C.; Azzouz, A.; Chung, I.H.; Lin, W.C.; Ben Said, L. A two-stage three-machine assembly scheduling problem with deterioration effect. *Int. J. Prod. Res.* **2019**, *57*, 6634–6647. [CrossRef]
32. Luo, J.C.; Liu, Z.Q.; Xing, K.Y. Hybrid branch and bound algorithms for the two-stage assembly scheduling problem with separated setup times. *Int. J. Prod. Res.* **2019**, *57*, 1398–1412. [CrossRef]

33. Mozdgir, A.; Ghomi, S.M.T.F.; Jolai, F.; Navaei, J. Three meta-heuristics to solve the no-wait two-stage assembly flow-shop scheduling problem. *Sci. Iran.* **2013**, *20*, 2275–2283.
34. Ji, M.; Yang, Y.; Duan, W.; Wang, S.; Liu, B. Scheduling of No-Wait Stochastic Distributed Assembly Flowshop by Hybrid PSO. In Proceedings of the IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, 24–29 July 2016; pp. 2649–2654.
35. Li, P.; Yang, Y.; Du, X.; Qu, X.; Wang, K.; Liu, B. Iterated Local Search for Distributed Multiple Assembly No-Wait Flow-shop Scheduling. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation, Donostia, Spain, 5–8 June 2017; pp. 1565–1571.
36. Shao, W.S.; Pi, D.C.; Shao, Z.S. Local search methods for a distributed assembly no-idle flow shop scheduling problem. *IEEE Syst. J.* **2019**, *13*, 1945–1956. [CrossRef]
37. Zhao, F.Q.; Liu, H.; Zhang, Y.; Ma, W.M.; Zhang, C. A discrete water wave optimization algorithm for no-wait flow shop scheduling problem. *Expert Syst. Appl.* **2018**, *91*, 347–363. [CrossRef]
38. Zhao, F.Q.; Qin, S.; Zhang, Y.; Ma, W.M.; Zhang, C.; Song, H.B. A hybrid biogeography-based optimization with variable neighborhood search mechanism for no-wait flow shop scheduling problem. *Expert Syst. Appl.* **2019**, *126*, 321–339. [CrossRef]
39. Zhao, F.Q.; Zhang, L.X.; Cao, J.; Tang, J.X. A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Comput. Ind. Eng.* **2021**, *153*. [CrossRef]
40. Zhao, F.Q.; Zhang, L.X.; Liu, H.; Zhang, Y.; Ma, W.M.; Zhang, C.; Song, H.B. An improved water wave optimization algorithm with the single wave mechanism for the no-wait flow-shop scheduling problem. *Eng. Optimiz.* **2019**, *51*, 1727–1742. [CrossRef]
41. Graham, R.L.L.; Lawler, E.L.; Lenstra, J.K.; Rinnooy-Kan, A.H.G. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.* **1979**, *5*, 287–326. [CrossRef]
42. Yang, D.-L.; Maw-Sheng, C. A two-machine flowshop sequencing problem with limited waiting time constraints. *Comput. Ind. Eng.* **1995**, *28*, 8. [CrossRef]
43. Tozkapan, A.; Kirca, O.; Chung, C.-S. A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem. *Comput. Oper. Res.* **2003**, *30*, 309–320. [CrossRef]
44. 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]
45. Sadeghi, A.; Doumari, S.A.; Dehghani, M.; Montazeri, Z.; Trojovsky, P.; Ashtiani, H.J. A new "Good and Bad Groups-Based Optimizer" for solving various optimization problems. *Appl. Sci.* **2021**, *11*, 4382. [CrossRef]
46. Ruiz, R.; Maroto, C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *Eur. J. Oper. Res.* **2006**, *169*, 781–800. [CrossRef]
47. 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]
48. Fernandez-Viagas, V.; Valente, J.M.S.; Framinan, J.M. Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. *Expert Syst. Appl.* **2018**, *94*, 58–69. [CrossRef]
49. Karabulut, K. A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Comput. Ind. Eng.* **2016**, *98*, 300–307. [CrossRef]
50. Mao, J.Y.; Pan, Q.K.; Miao, Z.H.; Gao, L. An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Syst. Appl.* **2021**, *169*. [CrossRef]
51. Ribas, I.; Companys, R.; Tort-Martorell, X. An iterated greedy algorithm for the parallel blocking flow shop scheduling problem and sequence-dependent setup times. *Expert Syst. Appl.* **2021**, *184*, 115535. [CrossRef]
52. Osman, I.H.; Potts, C.N. Simulated annealing for permutation flow-shop scheduling. *Omega* **1989**, *17*, 551–557. [CrossRef]
53. Abuajwa, O.; Bin Roslee, M.; Yusoff, Z.B. Simulated annealing for resource allocation in downlink NOMA systems in 5G networks. *Appl. Sci.* **2021**, *11*, 4592. [CrossRef]
54. Thao, P.B.; Truyen, D.C.; Phu, N.M. CFD analysis and taguchi-based optimization of the thermohydraulic performance of a solar air heater duct baffled on a back plate. *Appl. Sci.* **2021**, *11*, 4645. [CrossRef]