mathematics

MDPI

Article

A Three-Stage ACO-Based Algorithm for Parallel Batch Loading and Scheduling Problem with Batch Deterioration and Rate-Modifying Activities

Jae Won Jang, Yong Jae Kim and Byung Soo Kim *

Department of Industrial and Management Engineering, Incheon National University, 119 Academy-ro, Yeonsu-gu, Incheon 22012, Korea; 8chris8@inu.ac.kr (J.W.J.); yongjae@inu.ac.kr (Y.J.K.)
* Correspondence: bskim@inu.ac.kr

Abstract: This paper addresses a batch loading and scheduling problem of minimizing the makespan on parallel batch processing machines. For batch loading, jobs with compatible families can be assigned to the same batch process even if they differ in size; however, batches can only be formed from jobs within the same family, and the batch production time is determined by the family. During the batch scheduling, the deterioration effects are continuously added to batches processed in each parallel machine so that the batch production times become deteriorated. The deteriorated processing time of batches can be recovered to the original processing times of batches by a maintenance or cleaning process of machines. In this problem, we sequentially determine the batching of jobs and the scheduling of batches. Due to the complexity of the problem, we proposed a three-stage ant colony optimization algorithm. The proposed algorithm found an optimal solution for small-sized problems and achieved near-optimal solutions and better performance than a genetic algorithm or a particle swarm optimization for large-sized problems.

Keywords: scheduling; batching; ant colony optimization; mixed linear integer programming; deterioration; rate-modifying activity

Citation: Jang, J.W.; Kim, Y.J.; Kim, B.S. A Three-Stage ACO-Based Algorithm for Parallel Batch Loading and Scheduling Problem with Batch Deterioration and Rate-Modifying Activities. *Mathematics* **2022**, *10*, 657. https://doi.org/10.3390/math10040657

Academic Editors: Alexander A. Lazarev, Frank Werner and Bertrand M. T. Lin

Received: 23 January 2022
Accepted: 16 February 2022
Published: 20 February 2022

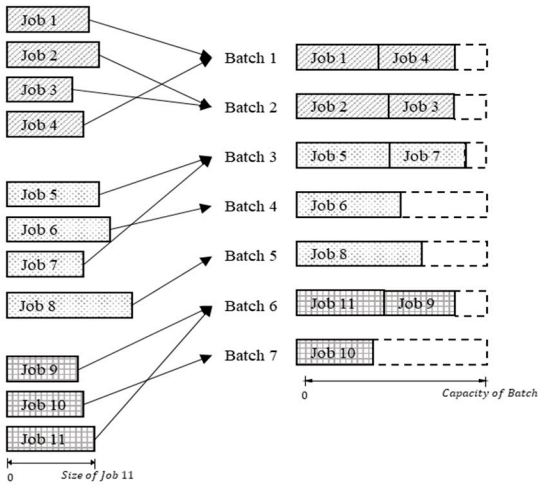
1. Introduction

Batch-processing machines (BPMs) have been applied to numerous manufacturing industries such as ceramics, steel, and integrated circuits industries to enhance the productivity of production. Due to this reason, several BPM scheduling problems have been studied in recent years. In general, the BPM sequentially processes batches, a group of jobs processed together in the same machine. The job is the smallest unit of an order requested by a customer. In this paper, we deal with the batch loading and scheduling problem (BLSP) at the diffusion operation in the semiconductor industry [1,2]. In addition to general batching, the concept of job family processing the same operation is adopted. Owing to the chemical nature of the diffusion operation in the semiconductor industry, only jobs with the same family can be assigned to the same batch. Thus, in this manufacturing environment, the batch production time is determined by the family type. In most batch scheduling studies, the batch production time is assumed to be constant. However, for a real-world scheduling problem, the batch production times increase due to the inclusion of activities such as the loading/unloading of jobs and alignment/calibration of tools. The increased batch production times can be recovered to the original production times of each batch by a maintenance or cleaning process of machines. The recovering process is called rate-modifying activity (RMA) [3]; multiple RMAs are considered in the schedule. The RMA time is assumed to be constant, and the RMA can be scheduled between batches. In this paper, the deterioration of batch production time linearly depends on the consecutive batch production runs without the RMA. Thus, it can be formulated as a linear function of the interval between the starting time of the first batch after the previous RMA and

the completion time of the last batch before the recent RMA. The interval between RMAs including periods before the first and after the last RMAs is called a bucket, which is defined in Joo and Kim [4].

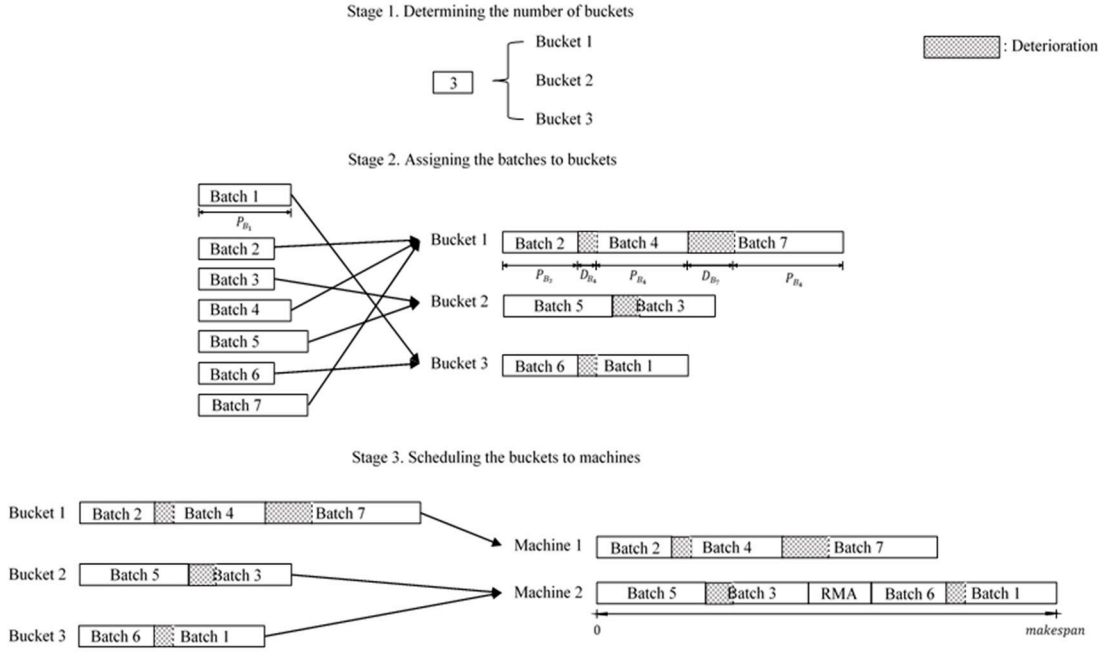
In this paper, we address BLSP with incompatible job families in parallel BPMs subject to time-dependent batch deterioration and RMAs applied between batches. The BLSP can be decomposed into the two sub-problems of batch loading and batch scheduling [5–7]. In the example of a batch loading problem shown in Figure 1a, 11 jobs belonging to three families are shown. All of the jobs must be formed in batches, which can only be assigned to batches within the same family. For example, batch 1 is formed from jobs 1 and 6, which have the same family type. The sum of their job sizes does not exceed the batch capacity. In Figure 1a, job 1–4, 5–8, and 9–11 belong to family 1, 2, and 3, respectively. The size of job is (0.43, 0.49, 0.34, 0.40, 0.49, 0.54, 0.43, 0.66, 0.37, 0.40, 0.46). The machine capacity is 1. The sum of job sizes in a batch does not exceed the batch capacity. For example, the sum of job sizes of job 1 and 4 is 0.83 ($S_1 + S_4 = 0.43 + 0.40 = 0.83 \leq 1$). After batches have been formed from all of the jobs, the batches must be scheduled to machines, which is referred to as the batch scheduling procedure.

Here, the problem with batch deterioration and RMAs during batch scheduling is considered. An illustrative example of the batch scheduling problem is shown in Figure 1b. As the batch production time including the deterioration increases, the assigning of RMAs between batches should be considered. Since the problem becomes complex, the batch scheduling problem is decomposed into three stages using buckets. The bucket is the set of batches between RMAs. The first stage is to determine the number of buckets. The second stage is to assign batches to each bucket. The last stage is to schedule the buckets to machines. In Figure 1b, the number of buckets is set to 3. The processing time for family is (44, 36, 52). Since batch 1–2, 3–5, and 6–7 consist of jobs for family 1, 2, and 3, respectively, the processing time of batch 1–2, 3–5, and 6–7 is 44, 36, and 52, respectively. The number of machines is 2. The deterioration rate is 0.25 and the processing time of RMA is 30. As a batch is processed, time is added equally to the difference between the last RMA completion time and the batch start time multiplied by the deterioration rate. For example, when batch 4 is processed in bucket 1, since the processing time of batch 2 has elapsed without RMA, 11 ($= 44 \times 0.25$) is added to the original processing time of batch 4. On the other hand, batch 6 of bucket 3 is not affected by deterioration because there is an RMA immediately before it. Once the batches have been assigned and sequenced in three buckets, the buckets are scheduled to the corresponding machines. As a result, the makespan can be calculated.



(**a**)

Figure 1. *Cont.*



(**b**)

Figure 1. Batch loading and scheduling problem. (**a**) Batch loading procedure. (**b**) Batch scheduling procedure.

The BLSP with no deterioration was addressed by Jia et al. [2], who proposed ant colony optimization (ACO) and the multi-fit (MF) algorithm for solving the batch loading and batch scheduling problems, respectively. We proposed the same algorithm for the batch loading problem, but the MF algorithm cannot be used in our problem because batch deterioration and RMAs impose complexity on the batch scheduling problem. Therefore, we decompose the batch scheduling problem into the three stages mentioned above. In the first stage, the number of buckets is determined between the lower and upper bounds. The fixed number of buckets determines the positions of the RMAs and helps to reduce the search space. In the second stage, a rule-based ACO is applied to assign batches to the buckets defined in the preceding stage. In the last stage, the buckets are scheduled to machines using a dispatching rule.

The remainder of this paper is organized as follows. A previous related work is reviewed in Section 2. Section 3 presents our mixed-integer programming (MIP) model. In Sections 4–6, the overall batch scheduling algorithm is proposed, with the MF algorithm applying the RMA scheduling rule developed in Section 4, the ACO-based three-stage algorithm proposed in Section 5, and the genetic algorithm (GA) and particle swarm optimization (PSO) approaches proposed in Section 6. Section 7 presents our experimental results and, finally, the conclusions are presented in Section 8.

2. Literature Review

To enhance productivity, many BPM studies have attracted the attention of various production areas such as ceramics, steel, and integrated circuit industries. For reviewing the related studies on BPM, we divided BPM studies into seven categories: manufacturing system, production methods, family constraint, job sizes, deterioration constant, recovering process, method, and objective function. The initial studies on BPMs assumed the size of jobs in a batch to be identical. The batch scheduling problem was first introduced by Ikura and Gimple [8]. They proposed a polynomial-time algorithm for minimizing the makespan under an assumption of identical job sizes. In this problem, the total job size in each batch must not exceed the batch capacity. For solving the BLSP with identical job sizes, Lee et al. [9] proposed dynamic programming algorithms to minimize the maximum

tardiness and the number of tardy jobs. Cheng et al. [10] proposed a novel ACO for minimizing the makespan. Subsequent studies have extended this approach to single BPMs with non-identical job sizes. Uzsoy [11] proposed the use of heuristics and a B&B algorithm to minimize the total weighted completion time. Two heuristics for solving this problem were proposed by Dupont and Ghazvini [5]. Later studies extended this to the problem of parallel BPMs with non-identical job sizes. Jia et al. [12] considered parallel BPMs with arbitrary capacities carrying out jobs of arbitrary sizes with dynamic arrival times. They proposed the use of two meta-heuristics based on ACO to minimize the makespan. Ozturk et al. [13] considered the problem of parallel BPMs with identical job sizes and release dates. They proposed a branch and bound (B&B) algorithm to minimize the makespan. Zhou et al. [6] considered parallel BPMs with arbitrary release times and non-identical job sizes. They presented different heuristics for solving batch loading and batch scheduling problems. The batch loading problem is tractable under the application of GA, ACO, and greedy randomized adaptive search procedure algorithms. Jia et al. [14] proposed a fuzzy ACO for minimizing the makespans for parallel batch processing machines with jobs having non-identical sizes and fuzzy processing times. Parallel BPM studies have also been extended to unrelated parallel BPMs. Arroyo and Leung [15,16] considered unrelated parallel batch machines with different capacities processing jobs with arbitrary sizes and non-zero ready times. They proposed several heuristics based on rules applying best-fit, first-fit, and meta-heuristic-based iterated greedy algorithms. Arroyo et al. [17] considered the use of unrelated parallel BPMs to minimize the total job flow time. They proposed a simple and effective iterated greedy algorithm and compared them to three meta-heuristic algorithms. Gao and Yuan [18] considered unbounded parallel batch scheduling problems involving jobs with agreeable release dates and processing times. They demonstrated that such problems are binary NP-hard. Zhou et al. [19] considered unrelated parallel BPM processing jobs with non-identical sizes and arbitrary release times. They proposed using a GA with a random key to schedule batches to the machines in such cases.

The studies described above focused on problems in which all jobs were compatible. However, the issue of incompatible jobs has occurred in actual situations. For instance, incompatibilities arise from a non-empty intersection on intervals and two-dimensional volumes for individual jobs [20,21]. Uzsoy [1] considered the problem of single BPM with incompatible families. He developed B&B and polynomial-time dynamic programming algorithms to minimize the makespan, lateness, and total weighted completion time. Azizoglu and Webster [22] considered a BPM with incompatible families, identical processing times, arbitrary job weights, and arbitrary job sizes. They proposed a B&B algorithm that can solve this problem with less than 25 jobs in a reasonable time. Dupont and Dhaenens-Flipo [7] considered a BPM with non-identical job sizes for minimizing the makespan. To solve problems with large numbers of jobs, they added two rules to a B&B algorithm to minimize the makespan. Yao et al. [23] considered single BPMs with incompatible job families and dynamic arrivals. They proposed a decomposed B&B algorithm for minimizing the total batch completion time and makespan. Jolai [24] considered BPMs with identical job sizes and incompatible families and applied a polynomial-time dynamic programming algorithm to minimize the number of tardy jobs. Parallel BPMs with incompatible families have also been extensively examined. Balasubramanian et al. [25] considered the problem of identical parallel BPMs with incompatible families. They proposed an approach in which the batch processing time is determined based on the family and developed two GAs for scheduling jobs to batches and machines, respectively. Li et al. [26] further considered the problem of incompatible families and proposed a method for determining the batch processing time based on the length of the longest job. For parallel BPMs with incompatible job families and release times, constraint programming and apparent tardiness cost approaches were presented [27,28].

In previous studies on the BLSP, the processing time was assumed to be constant and known in advance. In several real-world industry situations, however, the processing time increases due to the deterioration phenomenon. The concept of a deteriorating job was

introduced by Gupta and Gupta [29], who considered a single BPM governed by a linear deterioration function. The literature on deterioration follows two general tracks: sequence- and time-dependent deterioration. The case of parallel machine scheduling under sequence-dependent deterioration was addressed by Ding et al. [30], who proposed an ejection chain algorithm for minimizing the completion-time-based criteria. The case of time-dependent deterioration was first introduced by Browne and Yechiali [31]. Soleimani et al. [32] proposed cat swarm optimization on unrelated parallel machine scheduling problems with time-dependent deterioration. In general, the degree of deterioration increases with the length of the job process. To recover the inefficient execution of processes under deterioration, it is necessary to carry out the recovering processes called RMAs by Lee and Leon [3]. RMAs return a machine to its initial state. Joo and Kim [33] considered a single BPM with time-dependent deterioration to which RMAs are applied and proposed hybrid meta-heuristic algorithms to minimize the makespan. Woo et al. [34] considered unrelated parallel BLSPs with time-dependent deterioration and RMAs, proposing a GA with a random key to solve the problem. Abdullah and Süer [35] considered the decision-making on the selection of a manufacturing strategy between the classical assembly line and seru according to the skill levels of the operator. Liu et al. [36] proposed exact solutions and heuristic algorithms to minimize the makespan and balance the worker's workload of divisional and rotating seru in the seru production system. Gai et al. [37] presented an accurate dimensionality reduction algorithm to minimize makespan. It is also compared to the greedy algorithm to verify it.

As a problem becomes more complex, it becomes increasingly important to decompose it. The meta-heuristic approaches developed in previous studies generally decompose a BLSP into sub-problems. Dupont and Dhaenens-Flipo [7] considered unrelated parallel BPMs with non-identically sized jobs. They developed an approach for decomposing the BLSP into two sub-problems of forming and scheduling batches to improve performance. Determining which algorithm to apply to each sub-problem following division is another important challenge. Dupont and Ghazvini [5] assessed GA, ACO, and greedy adaptive search heuristic algorithms as methods for addressing the batch scheduling problem and compared their performances on batch loading situations fixed using the same algorithm. Similarly, Zhou et al. [19] proposed a GA with a random key for solving the batch scheduling problem. In this paper, the BLSP for processes with time-dependent batch deterioration for which RMAs are applied between batches is considered. To the best of our knowledge, this is the first paper to address this problem.

A list of recent studies on BLSPs and scheduling problems under deterioration is provided in Table 1**.** The recent studies are categorized by manufacturing system, production methods, family constraint, job sizes, deterioration constant, recovery process, method, and objective function. To the best of our knowledge, none of the studies dealt with the BLSP simultaneously considering parallel BPMs, incompatible job families, non-identical job sizes, time-dependent deterioration, and RMAs. Among the research, Jia et al. [2] proposed the ACO and MF algorithm for addressing, respectively, the batch loading and batch scheduling problems for parallel BPMs with incompatible families and arbitrary job sizes. For our problem, we adopted the ACO by Jia et al. [2] to solve the batch loading problem. However, the MF algorithm cannot provide the near-optimal makespan during the batch scheduling because time-dependent deterioration and RMAs increase the complexity of the problem. Therefore, in this paper, we decompose the batch scheduling problem into three stages and propose an ACO-based three-stage algorithm for solving the scheduling problem.

Table 1. The comparisons between recent studies.

	Mfg. System	Production Methods	Family Constraint	Job sizes	Deterioration Constraint	Recovery Process	Method	Objective
Woo et al. [34]	Parallel	Single	No family	N/A	Time-dependent deterioration	RMA	GA	Makespan
Ding et al. [30]	Parallel	Single	No family	N/A	Sequence-dependent deterioration	No recovery	ECA	Completion time
Soleimani et al. [32]	Parallel	Single	No family	N/A	Time-dependent deterioration	No recovery	GA, CSO, IABC	Mean weighted tardiness
Ozturk et al. [13]	Parallel	Batch	No family	Identical	No deterioration	No recovery	B&B	Makespan
Jia et al. [14]	Parallel	Batch	No family	Non-identical	No deterioration	No recovery	ACO	Makespan
Jia et al. [12]	Parallel	Batch	No family	Non-identical	No deterioration	No recovery	ACO	Makespan
Arroyo et al. [17]	Parallel	Batch	No family	Non-identical	No deterioration	No recovery	Iterated greedy	Total flow time
Li et al. [26]	Parallel	Batch	Incompatible family	Non-identical	No deterioration	No recovery	LB	Lateness
Jia et al. [2]	Parallel	Batch	Incompatible family	Non-identical	No deterioration	No recovery	ACO	Makespan
This paper	**Parallel**	**Batch**	**Incompatible family**	**Non-identical**	**Time-dependent deterioration**	**RMA**	**ACO**	**Makespan**

3. Mixed-Integer Programming Model

The BLSP is the problem of scheduling jobs to machines. In this problem, we have to make two decisions. We need to determine which jobs to assign in which batches and then schedule the assigned batches and RMA to machines. This is called batch loading problem and batch scheduling problem. In this section, the BLSP for time-dependent deterioration with rate-modifying activities and incompatible families for minimizing the makespan is formulated using MIP. The following parameters and decision variables are used in the mathematical formulation:

Parameters	
J	set of jobs
F	set of families
B	set of batches
K	set of buckets
M	set of machines
J^f	set of jobs not belongings to family $f \in F$
F_j	family type of job $j \in J$
F_b	family type of batch $b \in B$
S_j	size of job $j \in J$
P_b	processing time of batch $b \in B$
Q	processing time of RMA
DR	deterioration rate
L	large number
SC	size of machine
Decision variables	
X_{jb}	Equals 1 if job $j \in J$ is assigned in batch $b \in B$
Y_{bkm}	Equals 1 if batch $b \in B$ is assigned to bucket $k \in K$ from machine $m \in M$
Z_{abkm}	Equals 1 if batch $a \in B$ precedes batch $b \in B$ in bucket $k \in K$ from machine $m \in M$
Dependent variables	
MS	Makespan
C_k	Completion time of bucket $k \in K$
C_m	Completion time of machine $m \in M$
T_b	Time gap between starting time of batch $b \in B$ and completion time of recent RMA

Based on parameters and decision variables, the MIP is formulated as follows:

$$Minimize\ MS$$

Subject to

$$\sum_{b \in B} X_{jb} = 1\ ,\ \forall j \in J \tag{1}$$

$$X_{jb} = 0,\ \forall b \in B, \forall j \in J^{F_b}, \tag{2}$$

$$\sum_{j \in J} S_j \cdot X_{jb} \leq SC,\ \forall b \in B \tag{3}$$

$$\sum_{j \in J} X_{jb} \leq L \cdot \sum_{k \in K} \sum_{m \in M} Y_{bkm}\ ,\ \forall b \in B \tag{4}$$

$$\sum_{k \in K} \sum_{m \in M} Y_{bkm} \leq 1,\ \forall b \in B \tag{5}$$

$$\sum_{k \in K} \sum_{m \in M} Z_{bbkm} \leq 1,\ \forall b \in B \tag{6}$$

$$\sum_{a \in B} Z_{abkm} = Y_{bkm},\ \forall b \in B, \forall k \in K, \forall m \in M \tag{7}$$

$$\sum_{\substack{b \in B \\ b \neq a}} Z_{abkm} \leq Y_{akm},\ \forall b \in B,\ \forall a \in B, \forall k \in K, \forall m \in M \tag{8}$$

$$\sum_{b \in B} Z_{bbkm} \leq 1,\ \forall k \in K, \forall m \in M \tag{9}$$

$$T_a \cdot (1 + D) + P_a \leq T_b + L \cdot \left(1 - \sum_{k \in K} \sum_{m \in M} Z_{abkm}\right),\ \forall a, b \in B, a \neq b \tag{10}$$

$$T_b \cdot (1 + D) + P_b \leq C_k + L \cdot (1 - Y_{bkm}),\ \forall b \in B, \forall k \in K, \forall m \in M \tag{11}$$

$$\sum_{k \in K} C_k + Q \cdot (\sum_{k \in K} \sum_{b \in B} Z_{bbkm}) - Q \leq MS,\ \forall m \in M \tag{12}$$

The first decision we have to make is which job should be assigned to which batch by Constraint (1), (2) and (3). One job can be assigned to only one batch, and to be assigned to the same batch, the type of family must be the same and the size must not exceed the capacity. Constraints (1) and (2) confirm that each job is assigned to only one batch and that each batch comprises jobs within the same family. Constraint (3) ensures that the total job sizes assigned to each batch do not exceed the capacity of the batch. After all, once batches have been assigned, the assigned batches should be scheduled to the machine considering the RMA by constraint (4) to constraint (9). There are two ways to consider RMAs, i.e., considering scheduling between all batches, and assuming that RMAs are scheduled only between buckets. The bucket means a set of batches between buckets; hence the RMA does not exist between batches in the same bucket. In the batch scheduling problem, we determine which batch should be assigned to which bucket and the order of the batches in the bucket by constraint (4) to constraint (9). Constraints (4) and (5) ensure that each assigned batch is sequenced in one bucket. Constraints (6) and (8) ensure that there cannot be two first-sequence batches in a given bucket. Constraint (7) ensures that, except for the first batch in each bucket, any batches assigned to a bucket must be immediately preceded. Similarly, constraint (8) ensures that if a batch is assigned to a bucket, at most one batch can be performed immediately afterward. When all batches are scheduled, the makespan can be calculated by constraint (10) to constraint (12). Constraint (10) ensures the precedence relation among assigned batches in a bucket and calculates the starting time for each batch. Constraint (11) calculates the completion time of each bucket by computing the maximum time needed to complete each batch in the bucket. Constraint (12) determines the completion times of the machines needed to calculate the makespan.

4. Multi-Fit Batch Scheduling Algorithm with the RMA Scheduling Rule

The MF algorithm has demonstrated good performance in solving the batch scheduling problem with no deterioration [2]. Unlike previous approaches, however, the scheduling of RMAs must be considered in our problem. One of the ways to do so is to consider the scheduling between all batches. Whenever a batch is scheduled one by one, scheduling the RMA should be considered. To determine whether an RMA should be scheduled, an RMA scheduling rule is used. The RMA scheduling rule is that an RMA is scheduled if the deterioration is longer than the RMA processing time. The MF algorithm using this RMA scheduling rule is proposed in Algorithm 1 and the initial lower and upper bounds on completion time, LB_{ct} and UB_{ct}, can be calculated as

$$LB_{ct} = max\left\{\max_{b \in B} P_b, \left\lfloor \sum_{b \in B} \frac{P_b}{M} \right\rfloor\right\}, \tag{13}$$

$$UB_{ct} = max\left\{\max_{b \in B} P_b, \left\lceil \sum_{b \in B} \frac{P_b + Q}{M} \right\rceil\right\}. \tag{14}$$

Algorithm 1. Multi-fit algorithm with the RMA scheduling rule for the batch scheduling problem

Input: The set of assigned batches B.
Output: The makespan.
Begin:
Sort B in non-increasing order of processing times and obtain a batch set B'.
Compute LB_{ct} and UB_{ct}, respectively.
Let iteration $h = 1$.
While $(h < 8)$
  $A = (UB_{ct} + LB_{ct})/2$
  Let batch index $b = 1$.
  While $(b \leq |B'|)$
    Select a batch with a sequence b in B'.
    Schedule the batch to the machine according to the first-fit rule if the completion time does not exceed A.
    If (Completion time of machine when the batch is scheduled > Completion time of machine when additionally scheduled RMA precedes the batch) **then**
      Schedule the RMA to precede the batch.
    $b = b + 1$.
  End While
  If (All batches in B' are scheduled) **then**
    $UB_{ct} = A$.
  Else
    $LB_{ct} = A$.
  $h = h + 1$.
End While
Output the makespan of the global best solution.

Because the positioning of the RMAs and batches has a significant influence on the makespan, the MF algorithm does not perform well in solving this problem; therefore, we divide the batch scheduling problem into three stages and propose an ACO-based three-stage batch scheduling algorithm for solving it.

5. Ant Colony Optimization-Based Three-Stage Algorithm for Batch Scheduling Problem

The batch loading part of the BLSP referred to the solution of Jia et al. [2], and this section deals with the batch scheduling part. Determining the positioning of RMAs between all batches requires lots of calculations. If the number of buckets is pre-determined, however, the RMA positioning will be fixed between buckets. To reduce the computational complexity, we divide the batch scheduling problem into three stages, in which (1) the number of buckets is determined, (2) batches are assigned to the respective buckets, and (3) the buckets are dispatched to the machines.

5.1. Determining the Number of Buckets

In stage 1, the number of buckets is determined. Determining it fixes the positions of the RMAs between the buckets and it can range from zero to $|B| - 1$ in each machine. To reduce the range of this number, we calculate LB_k and UB_k, the lower and upper bounds, respectively, for the number of buckets during batch scheduling with the dispatching rule. The dispatching rule is the rule that selects the bucket with the shortest tentative completion time. Tentative completion time in each machine means the completion time if the current batch is assigned to the machine. The dispatching rule helps to search for the solution with good quality. A detailed explanation of the dispatching rule is given in Algorithm 2.

Algorithm 2. The dispatching rule

Input: The processing time of batch P_b.
The current completion time of buckets C_k.
Output: Selected bucket k', which must be assigned by batch b
Begin: Let bucket index $k = 1$
While $(k \leq |K|)$
Calculate the tentative completion time using the RMA rule
$k = k + 1$
End While
Select the bucket k' with the smallest tentative completion time

To calculate the LB_k and UB_k in subsequent algorithm, we must calculate the minimum and the maximum numbers of batches that can be scheduled on one machine. We assume that the batch processing time can be determined between P_{min} and P_{max} during batch scheduling with the dispatching rule, then the minimum number of batches on a machine is calculated in Algorithm 3. P_{min} and P_{max} are the longest and shortest processing time of the given batches, respectively. In determining the minimum number of batches that can be scheduled to the first machine, the first machine should be assigned a minimum number of batches and the rest of the machines should be assigned as many batches as possible. To assign a small number of batches to the machine under the dispatching rule, the completion time of the machine should be tentatively set to a large value. Since the dispatching rule schedules a batch on a machine with a small tentatively completion time, the allocated batches in the first machine must be given the longest processing time as P_{max}. The lower bound on the number of batches input to the first machine is calculated using the dispatching rule in Algorithm 3.

Algorithm 3. Calculating the lower bound on the number of batches input to a machine

Input: P_{min}; P_{max}; The number of batches; The number of machines.
Output: Lower bound on the number of batches input to a machine.
Begin: Let batch index $b = 1$
While$(b \leq |B|)$
Select a machine m' using the dispatching rule.
If$(m' = 1)$ **then**
Assign a batch assuming the processing time is P_{max}.
Else
Assign a batch assuming the processing time is P_{min}.
$b = b + 1$.
End While
Output the number of batches in machine 1

Before calculating the LB_k using Algorithm 4, the batch deteriorations must be represented in order. A time-dependent batch deterioration can be represented as the product of DR and P_i. In this section, P_k and D_k represent the processing time and deterioration of a batch in sequence k, respectively. D_k and D_{k-1} can be represented by the processing time and the deterioration of previous batches as

$$D_k = DR \times \left(\sum_{i=1}^{k-1} P_i + \sum_{i=1}^{k-1} D_i \right), \tag{15}$$

$$D_{k-1} = DR \times \left(\sum_{i=1}^{k-2} P_i + \sum_{i=1}^{k-2} D_i \right). \tag{16}$$

By subtracting D_{k-1} from D_k, D_k can be given in terms of the processing time and the deterioration of the preceding batch as

$$D_k - D_{k-1} = DR \times (P_{k-1} + D_{k-1}), \tag{17}$$

and then D_k can be represented as

$$D_k = (1 + DR) \times D_{k-1} + DR \times P_{k-1}. \tag{18}$$

Using the above equations, the situation in which an RMA must be scheduled is defined through Proposition 1.

Proposition 1. *If the deterioration portion of a batch exceeds the processing time of an RMA, scheduling an RMA before the batch will reduce the completion time of the bucket.*

Proof. Let $\widetilde{B}$ be a set of scheduled batches with no RMAs, which comprises $B_1, B_2 \ldots, B_n$. Let $\widetilde{B'}$ be a set of the same scheduled batches with one RMA, comprising $B'_1, B'_2 \ldots, B'_{l-1}$, RMA, $B'_l, \ldots, B'_n$. Deterioration of B'_i can be represented as D'_i. Except for the RMA in $\widetilde{B'}$, the processing time of batches in the same sequence as $\widetilde{B}$ are equal. Thus, the sums of the deteriorations up to $l-1$ will be equal in both $\widetilde{B}$ and $\widetilde{B'}$ as follows:

$$\sum_{i=1}^{l-1} D_i = \sum_{i=1}^{l-1} D'_i. \tag{19}$$

As we assume that D_l is greater than Q, the sum of D'_l and Q is less than D_l:

$$D_l > D'_l + Q. \tag{20}$$

From Equation (19), the deteriorations of D_{l+1} and D'_{l+1} can be obtained as, respectively,

$$D_{l+1} = (1 + DR) \times D_l + DR \times P_l, \tag{21}$$

$$D'_{l+1} = (1 + DR) \times D'_l + DR \times P_l \tag{22}$$

Because the processing time of each batch is the same and $D'_l = 0$, D'_{l+1} is larger than D_{l+1}, repeating the above reasoning, D'_{l+2} is larger than D_{l+2}, D'_{l+3} is larger than D_{l+3}, $\ldots$, and D'_n is larger than D_n. Adding these deteriorations, we obtain

$$\sum_{i=l+1}^{n} D_i > \sum_{i=l+1}^{n} D'_i. \tag{23}$$

From Equations (20), (21) and (24), $D_1 + D_2 + \ldots + D_l + \ldots + D_n$ is larger than $D'_1 + D'_2 + \ldots + Q + D'_l + \ldots + D'_n$. Hence, the deterioration period of $\widetilde{B}$ is longer than that of $\widetilde{B'}$. □

The above proof gives the condition under which scheduling an RMA reduces the makespan. This can be used to find the minimum number of RMAs. If we set the processing time of each scheduled batch as P_{min}, then we can use Proposition 1 to schedule the RMAs starting from the first batch. The lower bound on the number of buckets can then be calculated based on the number of RMAs scheduled per machine using Algorithm 4.

Using these algorithms and propositions, we can obtain the LB_k for a machine. In a similar manner, UB_k can be calculated. Assuming a batch processing time of P_{min} for the first machine and P_{max} for the remaining machines in Algorithm 3, the upper bound on the number of batches per machine can be applied in Algorithm 4.

Algorithm 4. Calculating the lower bound on the number of buckets (LB_k)

Input: P_{max}; the lower bound on the number of batches in a machine;
Output: the lower bound on the number of RMAs required by a machine
Let batch index $b = 1$
Begin: Let RMA index $r = 0$
While $(b \leq$ *The lower bound on the number of batches in a machine*$)$
Calculate a tentative deterioration proportional to the gap between the preceding RMA and C_{b-1}
If (Tentative deterioration $>$ Processing time of RMA) **then**
Schedule RMA after C_{b-1}.
$r = r + 1$.
Else
$C_b = (DR + 1) \times C_{b-1} + P_{max}$
$b = b + 1$.
End While
Output $(r + 1) \times |M|$

5.2. Assigning Batches to Buckets

In stage 2, an ACO algorithm is used to assign batches to each bucket and then a rule is applied to determine the sequencing of the batches. Unlike the problem of assigning batches, the individual buckets do not have size capacities, and the objective function is the makespan. In this case, the batch deterioration has a significant influence on the makespan. Based on the number of buckets determined from stage 1, the batches must be scheduled using a load balancing among buckets.

To assign batches, an ACO algorithm called the min-max ant system (MMAS) is used. MMAS is a constructive meta-heuristic algorithm that builds solutions sequentially, i.e., to solve the batch assignment problem, the batches are assigned one by one. During the batch assignment, we make a probabilistic choice based on pheromone trails and heuristic information until no further batches are available. After assigning the batches, the minimum completion time of each bucket can be calculated using Proposition 2 (Section 5.2.5).

5.2.1. Pheromone Trails

The desirability of unscheduled batches can be calculated using a pheromone trail that gives the relationship between assigned batches. There is a pheromone trail value between all batches and a higher pheromone trail value indicates a higher probability that two batches will be allocated to the same bucket. Defining $\tau_{il}(t)$ as the pheromone trail between B_i and B_l in iteration t, $\tau_{il}(1)$ is initialized as

$$\tau_{il}(1) = ((1 - \rho) \times LB_\tau)^{-1}, \tag{24}$$

where ρ means evaporation rate and LB_τ is the lower bound given by

$$LB_\tau = \sum_{b \epsilon B} \frac{P_b}{|M|} \tag{25}$$

The desirability of a given batch is calculated using the pheromone trails between the given batch and batches, $B_k(t)$, that have already been assigned to the selected bucket in iteration t. The desirability of assigning batch i into the current bucket k during iteration t can be represented as

$$\theta_{ik}(t) = \sum_{B_l \epsilon B_k(t)} \frac{\tau_{il}(t)}{|B_k(t)|}. \tag{26}$$

5.2.2. Heuristic Information

The load balance of each bucket is important because the deterioration increases rapidly when many batches are allocated per bucket. The heuristic information between batch i and bucket k during iteration t can be represented by

$$\eta_{ik}(t) = \frac{1}{|\max\{c_k(t)\} - c_{k+i}(t)| + p_{max}}, \tag{27}$$

where $c_{k+i}(h)$ is the completion time of bucket k calculated after assigning batch i to it during iteration t.

5.2.3. Forming the Buckets

The pheromone trails and heuristic information are used together to calculate the probability that each batch can be formed. The probability that batch i can be assigned to bucket k during iteration t is expressed as $P_{ik}(t)$, which represents the desirability of the batches assigned to bucket k as follows:

$$P_{ik}(t) = \begin{cases} \frac{\theta_{ik}(t) \times \eta_{ik}(t)^{\beta}}{\sum_{B_l \epsilon U_k} \theta_{ik}(t) \times \eta_{ik}(t)^{\beta}}, & if\ B_i \epsilon U_k \\ 0, & otherwise \end{cases}, \tag{28}$$

where β represents the relative importance of heuristic information and U_k represents the set of unscheduled batches.

5.2.4. Updating the Pheromone Trails

In MMAS, the use of pheromone trails leads to solutions based on experience. Thus, updating the pheromone trails has a significant effect on the performance of the algorithm. Global- and iteration-best solutions are used to update the pheromone trails. Representing the frequency with which jobs i and l are placed in the same batch as m_{il}, the pheromone trail updating process can be defined as follows:

$$\tau_{il}(t+1) = (1-\rho) \times \tau_{il}(t) + m_{il} \times \Delta\tau_{il}(t), \tag{29}$$

$$\Delta\tau_{il}(t) = \frac{Q}{Makespan_{iteration\ best}(t)}. \tag{30}$$

The solutions corresponding to pheromone trails that are too extreme cannot be found easily. Under MMAS, pheromone trail lower and upper bounds are defined as τ_{min} and τ_{max}, respectively, and when the trails are updated, they are modulated as follows:

$$\tau_{il}(t+1) = \begin{cases} \tau_{min}, & \tau_{il}(t+1) < \tau_{min} \\ \tau_{il}(t+1), & \tau_{min} \le \tau_{il}(t+1) \le \tau_{max} \\ \tau_{max}, & \tau_{il}(t+1) > \tau_{max} \end{cases}, \tag{31}$$

$$\tau_{max} = \left((1-\rho) \times Makespan_{global\ best}(t)\right)^{-1}, \tag{32}$$

$$\tau_{min} = \frac{\tau_{max} \times \left(1 - \sqrt[|B|]{0.05}\right)}{(\frac{|B|}{2} - 1) \times \sqrt[|B|]{0.05}}. \tag{33}$$

5.2.5. Rule-Based Batch Sequencing

After assigning the batches to the buckets, the completion time of each bucket can be minimized using Proposition 2.

Proposition 2. *Let B be the set of batches in one bucket. Then, scheduling B in ascending order of processing time minimizes the completion time of each bucket.*

Proof. As the starting time of the first batch is zero, we have

$$D_1 = 0. \tag{34}$$

By applying the previous deterioration to Equation (16), we obtain

$$\begin{aligned} D_2 &= DR \times P_1 + (1 + DR) \times D_1 \\ &= DR \times P_1, \end{aligned} \tag{35}$$

$$\begin{aligned} D_3 &= DR \times P_2 + (1 + DR) \times D_2 \\ &= DR \times P_2 + (1 + DR) \times DR \times P_1, \end{aligned} \tag{36}$$

$$\begin{aligned} D_4 &= DR \times P_3 + (1 + DR) \times D_3 \\ &= DR \times P_3 + (1 + DR) \times DR \times P_2 + (1 + DR)^2 \times DR \times P_1, \end{aligned} \tag{37}$$

$$\begin{aligned} D_n &= DR \times P_{n-1} + (1 + DR) \times DR \times P_{n-2} + \ldots + (1 + DR)^{n-2} \times DR \times P_1 \\ &= \sum_{i=1}^{n-1} (1 + DR)^{n-1-i} \times DR \times P_i. \end{aligned} \tag{38}$$

By using the generalized formula above, D_n, the sum of the deteriorations, can be represented as

$$\begin{aligned} \sum_{i=1}^{n} D_i &= D_2 + D_3 + D_4 + \ldots + D_n \\ &= [DR \times P_1] + [DR \times P_2 + (1 + DR) \times DR \times P_1] + \ldots \\ &\quad + \sum_{i=1}^{n-1} (1 + DR)^{n-1-i} \times DR \times P_i \\ &= \left[1 + (1 + DR) + (1 + DR)^2 + \ldots + (1 + DR)^{n-2}\right] \times DR \times P_1 \\ &\quad + \left[1 + (1 + DR) + (1 + DR)^2 + \ldots + (1 + DR)^{n-3}\right] \times DR \times P_2 \\ &\quad + \ldots + [1 + (1 + DR)] \times DR \times P_{n-2} + DR \times P_{n-1} \\ &= \sum_{i=0}^{n-2} (1 + DR)^i \times DR \times P_1 + \sum_{i=0}^{n-3} (1 + DR)^i \times DR \times P_2 + \ldots \\ &\quad + \sum_{i=0}^{1} (1 + DR)^i \times DR \times P_{n-2} + DR \times P_{n-1} \end{aligned} \tag{39}$$

In Equation (39), P_k, the processing time of a batch in sequence k, is multiplied by $\sum_{i=0}^{n-1-k}(1 + DR)^i$. $\sum_{i=0}^{n-1-k}(1 + DR)^i$ increases as the sequence become slower. As the batch we have to schedule is fixed, the total deterioration can be minimized by scheduling the batches with longer processing times in faster sequences. □

Using the above proof, we can obtain the shortest completion for each bucket. After all, once the shortest completion time of buckets is calculated, buckets must be scheduled to the machines.

5.3. Scheduling Buckets to Machines

In stage 3, buckets are scheduled to machines using the dispatching rule. Tentative timings are used to determine the completion time at which the current batch is scheduled using the RMA rule as Algorithm 2 (Section 5.1). To balance the load among the machines, therefore, the current batch should be scheduled to the machine with the shortest tentative completion time.

5.4. Overall Algorithm

Using the algorithms defined for the three stages in the preceding sections, the ACO-based three-stage algorithm can be defined as Algorithm 5.

Algorithm 5. ACO-based three-stage algorithm for batch scheduling problem

Input: The set of assigned batches.
Output: The makespan
Compute the LB_K and UB_K, respectively.
Compute the LB_τ and $\tau_{il}(1)$, respectively.
Begin: Initialize pheromone trails.
Let index of iteration $t = 1$
While$(t \leq t_{max})$
Let index of iteration $a = 1$
While $(a \leq a_{max})$
Select the number of buckets between LB_K and UB_K.
While$(U_B \neq \varnothing)$
Select bucket k using the dispatching rule.
If (Bucket k has no assigned batches) **then**
Select a batch b randomly.
Assigned the batch b to bucket k.
Else
Select a batch b using pheromone trail and heuristic information.
Assign the batch b to bucket k.
End While
Minimize the completion time of the buckets by sequencing the batches.
Dispatch the buckets to machines.
Calculate the makespan.
Update the iteration and global best solutions
$a = a + 1$
End While
Compute τ_{max} and τ_{min}.
Update $\Delta\tau_{il}(h)$ and $\tau_{il}(h)$.
$t = t + 1$
End While
Output the makespan of the global best solution

The algorithm divides the batch scheduling problem into three-stage. In stage 1, LB_K and UB_K are used for fast convergence. To demonstrate that LB_K and UB_K affect the performance of the algorithm, an ACO with no bound (ACO_NB) is proposed.

6. Genetic Algorithm- and Particle Swarm Optimization-Based Batch Scheduling Algorithms

In this section, two population-based meta-heuristic algorithms for solving the batch scheduling problem—a GA-based three-stage algorithm and a PSO-based three-stage algorithm—are proposed. Unlike constructively generated ACO solutions, these two algorithms search solutions to develop an encoded solution that is represented using a random key. The two proposed population-based meta-heuristic algorithms (Algorithms 6 and 7) are defined as follows.

Algorithm 6. GA-based three-stage algorithm for batch scheduling problem

Input: The set of assigned batches from the batch loading problem
Output: The makespan
Begin:
Compute the LB_K and UB_K respectively.
Set the initial decoded solution.
Let index of generation $t = 1$
While $(t \leq t_{max})$
  Let index of chromosome $c = 1$
  While $(c \leq c_{max})$
    Select the number of buckets between LB_K and UB_K.
    Let random value $r_1 = Unif(0, 1)$.
    Let random value $r_2 = Unif(0, 1)$.
    Select two parent chromosomes, c, and $c + 1$, in sequence.
    If (r_1 < Crossover rate) **then**
      Do one-point crossover operations.
    If (r_2 < Mutation rate) **then**
      Do swap mutation operation.
    Decode the offspring chromosome c and calculate the makespan.
    $c = c + 1$
  End While
  Reproduce the next generation parents from current-generation offspring
  Update the iteration and global best solutions
  $t = t + 1$
End While
Output the makespan of the global best solution

Algorithm 7. PSO-based three-stage algorithm for batch scheduling problem

Input: The set of assigned batches from the batch loading problem
Output: The makespan
Begin:
Compute the LB_K and UB_K, respectively.
Set the initial solution.
Let index of iteration $t = 1$
While $(t \leq t_{max})$
  Let index of particle $p = 1$
  While $(p \leq p_{max})$
    Select the number of buckets between LB_K and UB_K
    Update the factor velocity
    Update the factor position
    Decode the particle p.
    $p = p + 1$
  End While
  Decode the particles.
  Update the iteration and global best solutions;
  $t = t + 1$
End While
Output the makespan of global best solution

7. Computational Experiments

To evaluate the performance of the ACO-based three-stage batch scheduling algorithm, extensive computational experiments were conducted using the solutions obtained by an ACO-based batch loading algorithm presented in Jia et al. [2]. In the first experiment, the absolute differences between the optimal solutions produced by CPLEX and the solutions produced by the ACO, ACO_NB, GA, PSO, and MF algorithms were compared for small-sized problems. In the second experiment, ACO and ACO_NB were compared to validate the LB_k and UB_k used in the proposed algorithm. In the third experiment, the relative differences between the solutions obtained by ACO and other meta-heuristic algorithms,

GA and PSO, were compared for large-sized problems. All meta-heuristic algorithms are implemented in C#.

7.1. Problem Parameter Settings

In the experiments, BLSPs corresponding to the batch scheduling problem with varying complexity were randomly generated. The complexity of a batch scheduling problem with batch deterioration and RMAs is highly dependent on $|M|$, $|K|$, $|B|$, and $|F|$ [24–26]. Therefore, four problem parameters, $|M|$, $|K|/|M|$, $|B|/|K|$, and $|F|$, were used to generate problems with different complexities, where $|K|/|M|$ is the average number of buckets per machine and $|B|/|K|$ is the average number of batches per bucket. The machines and families were generated as $|M|$ and $|F|$, respectively, and the batches were generated using $|M| \times |K|/|M| \times |B|/|K|$. To assign $|B|/|K|$ batches to each bucket, Q was generated as $\frac{Q_{min}+Q_{max}}{2}$. As Q decreases, RMAs will occur more frequently and the number of batches per bucket will be reduced. To assign $|B|/|K|$ batches to each bucket, Q should be adjusted so that it falls between Q_{min} and Q_{max}, where those values mean the minimum and maximum sums of deteriorations that occur when there are $|B|/|K|$ batches, respectively. The sums can be calculated by assuming the processing time of batches as P_{min} and P_{max}, respectively. P_f is randomly generated in $[P_{min}, P_{max}]$ with P_{min} and P_{max} fixed at 40 and 60, respectively. D and $\overline{S_j}$ are fixed at 0.2 and 0.4, respectively. Using the four problem parameters defined above, two groups of problems (small-sized problems with fewer than 10 batches and large-sized problems with more than 10 batches) were randomly generated.

7.2. Algorithm Parameter Settings

To find the major parameters that affect the performance of algorithms, the Taguchi method was applied in this section. The method can be used to conduct experiments more rapidly than a full factorial experiment because the method can carry out experiments using fewer scenarios. Using the Taguchi method, five control factors for the ACO-based batch scheduling algorithm were analyzed. The levels of each control factor are listed in Table 2. The first factor, A, is the number of ants (Ant_{max}). When a problem becomes more complicated, more ants are needed to solve it. Therefore, we set Ant_{max} to increase according to the complexity of the problem; as the complexity was proportional to $|B|$, Ant_{max} is varied over the range $\{1.0 \times |B|, 1.5 \times |B|, 2.0 \times |B|, 2.5 \times |B|, 3.0 \times |B|\}$. The second factor, B, is the evaporation rate (ρ), which is used to calculate the initial pheromone trails and update later trails. Because the evaporation must be a real value between [0, 1] and values lower than 0.5 are relatively insignificant to the parameter settings, ρ is varied over $\{0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9\}$. The third factor, C, is the iteration limit (G), which indicates how many iterations are needed to reset a pheromone trail in the ACO. If there is no change in the global solution over G iterations, the trail must be reset and, therefore, smaller values of G mean that trails must be more often reset. In this paper, G is varied over $\{10,\ 30,\ 50,\ 70,\ 90\}$. The fourth factor, D, is the updating parameter (δ), which is used to update the pheromone trails. In a previous paper [2], δ performs better when it is based on LB_τ rather than a constant parameter. Therefore, δ is varied over $\{0.5 \times LB_\tau, 1.0 \times LB_\tau, 1.5 \times LB_\tau,\ 2.0 \times LB_\tau,\ 2.5 \times LB_\tau\}$. The final factor, E, is the relative importance of heuristic information (β), which is used in calculating desirability. β is varied over {4, 6, 8, 10, 12}.

Table 2. Factors levels for MMAS based three-stage batch scheduling algorithm.

Factor 1	Factor 2	Factor 3	Factor 4	Factor 5
Number of Ants	**Evaporation Rate**	**Iteration Limit**	**Updating Parameter**	**Relative Importance of Heuristic Information**
Ant_{max} **(A)**	ρ **(B)**	**G (C)**	δ **(D)**	β **(E)**
A (1): $1.0 \times \|B\|$	B (1): 0.5	C (1): 10	D (1): $0.5 \times LB_\tau$	E (1): 4
A (2): $1.5 \times \|B\|$	B (2): 0.6	C (2): 30	D (2): $1.0 \times LB_\tau$	E (2): 6
A (3): $2.0 \times \|B\|$	B (3): 0.7	C (3): 50	D (3): $1.5 \times LB_\tau$	E (3): 8
A (4): $2.5 \times \|B\|$	B (4): 0.8	C (4): 70	D (4): $2.0 \times LB_\tau$	E (4): 10
A (5): $3.0 \times \|B\|$	B (5): 0.9	C (5): 90	D (5): $2.5 \times LB_\tau$	E (5): 12

Using the control factors described above, an experiment involving 25 scenarios was conducted. The number of scenarios was chosen so that the best-fit design for five factors with five levels. Table 3 shows $L_{25}(5^5)$ orthogonal array. In each scenario, 12 test problems were solved three times each for a total of 900 runs for the experiment. Under the Taguchi method, variation in performance is measured using the mean signal-to-noise (S/N) ratio. Generally, the S/N ratio is expressed as $-10\log(objective\ function)^2$, but this formula was not directly applicable to this case because the experimental tests had varying objective functions and sizes. Therefore, we adopted the relative percentage deviation (RPD) of the objective function as follows:

$$RPD(\%) = \frac{OBJ_{sol} - OBJ_{best}}{OBJ_{best}} \times 100, \tag{40}$$

where OBJ_{sol} is the objective function given by an algorithm and OBJ_{best} is the objective function of the best solution. Using the RPD, the S/N ratio can be computed as follows:

$$S/N\ ratio_k = -10\log\left(\frac{1}{12}\sum_{i=1}^{4}\sum_{j=1}^{3} RPD_{ijk}^2\right) \qquad \forall k \in 1, 2, \ldots, 2, D \qquad h\ the\ 12525. \tag{41}$$

Table 3. Factor levels of array L_{25}.

Scenario No.	Factor Levels				
	A	***B***	***C***	***D***	***E***
1	A (1)	B (1)	C (1)	D (1)	E (1)
2	A (1)	B (2)	C (2)	D (2)	E (2)
3	A (1)	B (3)	C (3)	D (3)	E (3)
4	A (1)	B (4)	C (4)	D (4)	E (4)
5	A (1)	B (5)	C (5)	D (5)	E (5)
6	A (2)	B (1)	C (2)	D (3)	E (4)
7	A (2)	B (2)	C (3)	D (4)	E (5)
8	A (2)	B (3)	C (4)	D (5)	E (1)
9	A (2)	B (4)	C (5)	D (1)	E (2)
10	A (2)	B (5)	C (1)	D (2)	E (3)
11	A (3)	B (1)	C (3)	D (5)	E (2)
12	A (3)	B (2)	C (4)	D (1)	E (3)
13	A (3)	B (3)	C (5)	D (2)	E (4)
14	A (3)	B (4)	C (1)	D (3)	E (5)
15	A (3)	B (5)	C (2)	D (4)	E (1)
16	A (4)	B (1)	C (4)	D (2)	E (5)
17	A (4)	B (2)	C (5)	D (3)	E (1)
18	A (4)	B (3)	C (1)	D (4)	E (2)
19	A (4)	B (4)	C (2)	D (5)	E (3)

Table 3. *Cont.*

Scenario No.	Factor Levels				
	A	***B***	***C***	***D***	***E***
20	*A* (4)	*B* (5)	*C* (3)	*D* (1)	*E* (4)
21	*A* (5)	*B* (1)	*C* (5)	*D* (4)	*E* (3)
22	*A* (5)	*B* (2)	*C* (1)	*D* (5)	*E* (4)
23	*A* (5)	*B* (3)	*C* (2)	*D* (1)	*E* (5)
24	*A* (5)	*B* (4)	*C* (3)	*D* (2)	*E* (1)
25	*A* (5)	*B* (5)	*C* (4)	*D* (3)	*E* (2)

To analyze statistically significant factors, analysis of variance (ANOVA) was performed using the S/N ratio data. The factor D, which had the smallest sum squares (SS), was selected as the error term. After pooling this factor, ANOVA was performed, the results of which are listed in Table 4. Factors B and E exceeded the significance level, indicating that they had a large effect on the algorithm performance. The levels of these factors with the highest S/N ratios—*B* (5) and *E* (5)—were identified based on an examination of the S/N results in Figure 2. The factors and levels *A* (4), *C* (3), and *D* (1) had the least effect on the performance, as reflected by the *RPD* values shown in Figure 3. From these results, we identified an optimal combination of *A* (4), *B* (5), *C* (3), *D* (1), *and E* (5).

Table 4. Analysis of variance for the S/N ratio.

Factor	*SS*	*df*	*V*	F_0	*p*-Value
A	0.2593	4	0.0648	2.6456	0.1844
B	1.4578	4	0.3644	14.8721	0.0114
C	0.1809	4	0.0452	1.8459	0.2836
D(Error)	0.0980		0.0245		
E	0.2144	4	0.5360	21.8756	0.0055
Total	4.1404	20			

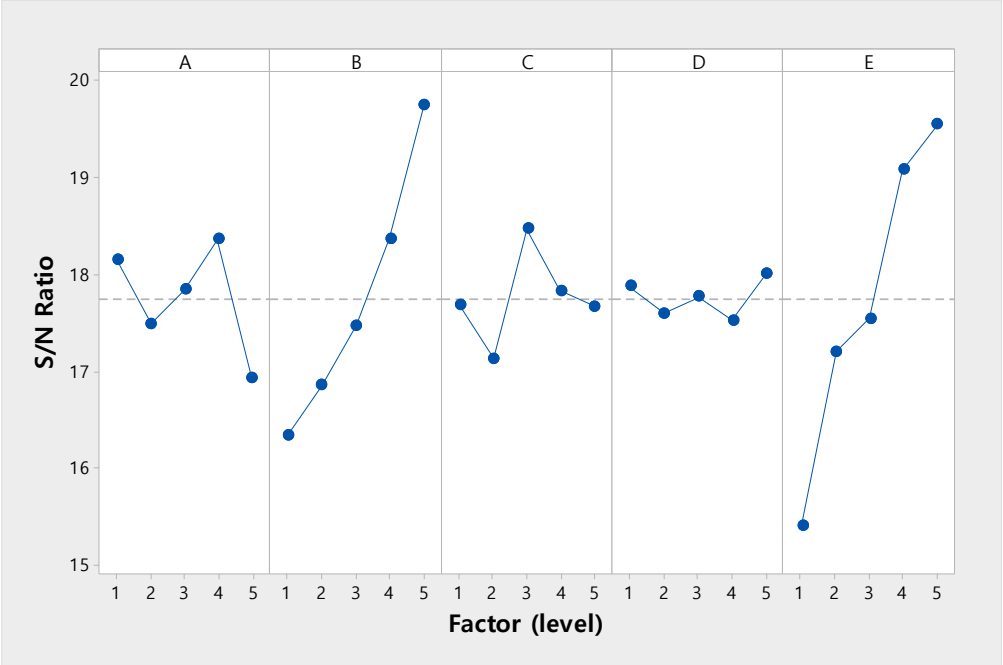


Figure 2. The mean S/N ratio plot for each level of the factor.

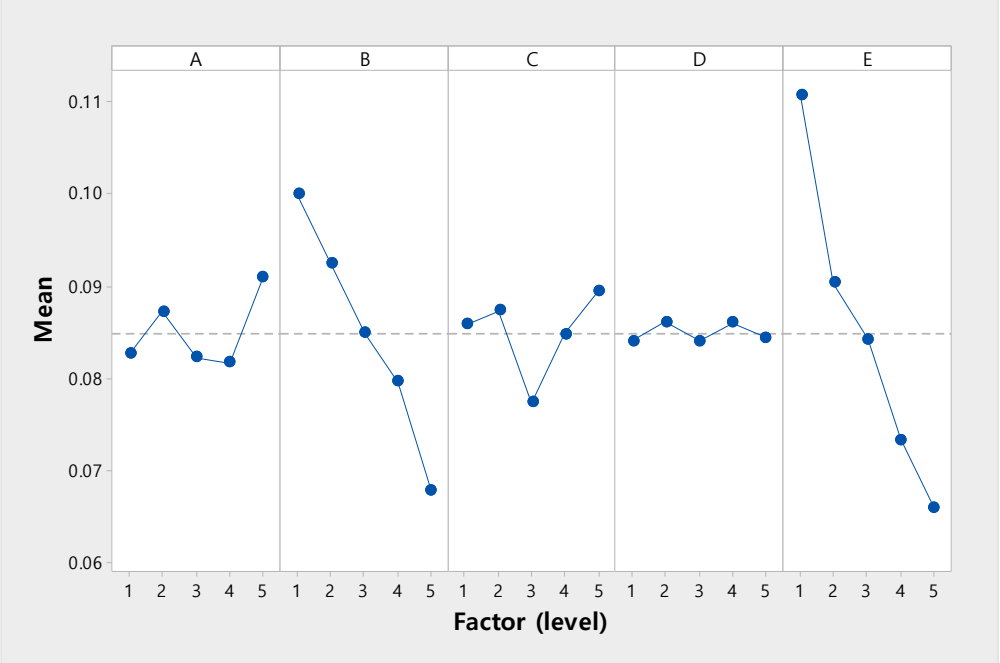


Figure 3. The mean RPD plot for each level of the factors.

The GA and PSO parameters were also set using the Taguchi method. For the GA, the parameters population size $= 3 \times |B|$, crossover rate $= 0.9$, and mutation rate $= 0.2$ were set. For the PSO, the parameters population size $= 3 \times |B|$, inertia weight $= 0.9$, social cognitive 1$= 1.0$, and social cognitive 2 $= 0.5$ were set.

7.3. Experimental Results

Three experiments were conducted to validate the ACO-based three-stage batch scheduling algorithm. All of the experiments were carried out on a PC with an Intel Core i7-7700 CPU running at 3.6 GHz with 16 GB of memory. In each experiment, the batch scheduling problem with the same problem instance was solved using the same given solution obtained from the ACO-based batch loading algorithm. All the algorithms were run until iteration reached 1500 or performance did not increase for 100 iterations.

In the first experiment, the ACO, ACO_NB, GA, PSO, and MF algorithms were validated by comparing their solutions with optimal solutions produced by ILOG CPLEX. The experiment was conducted on eight small-sized problems with 30 replications, with the results summarized in Table 5. To compare the performance of the respective algorithms, the RPD values were calculated using the optimal solution obtained from CPLEX. In the table, "N/A" denotes cases in which CPLEX was unable to obtain the optimal solution before 7200s. The problems with more than eight batches could not be solved by ILOG CPLEX. For problems with fewer than eight batches, the average RPD values of ACO, ACO_NB, GA, and PSO were zero whereas the average for the MF algorithm was 2.16. The result indicates that the meta-heuristic algorithms provided optimal solutions to small-sized problems. However, the MF algorithm performed poorly even on small-sized problems.

In the second experiment, the relative performances of ACO and ACO_NB in solving large-sized problems were compared to validate the LB_k and UB_k values obtained in stage 1. The experiment was conducted on 36 large-sized problems with 30 replications. The results are summarized in Table 6. To validate the results, a paired t-test $(\alpha = 0.05)$ was conducted to statistically evaluate the differences in performance between ACO and ACO_NB. The results of the paired t-tests are listed in Table 7. ACO significantly outperformed ACO_NB because it is seen from the table that all of the p-values are less than 0.05. The differences between the results obtained by ACO and ACO_NB are dependent upon the calculated values of LB_k and UB_k, respectively. To demonstrate that the use of LB_k and UB_k quickly

converge $|K|$ and the makespan, the performance improvements obtained at different levels of $|F|$ are shown in Figure 4, in which the average makespan and $|K|$ are presented for five iterations of ACO and ACO_NB, respectively. The results demonstrate that the makespan converges faster under ACO than under ACO_NB because the $|K|$ from ACO rapidly converges to an optimal value between LB_k and UB_k.

Table 5. The test results of small-sized problems.

$\|M\|$	$\frac{\|K\|}{\|M\|}$	$\frac{\|B\|}{\|K\|}$	CPLEX		ACO		ACO_NB		GA		PSO		MF
			Opt.	Time	RPD	Time	RPD	Time	RPD	Time	RPD	Time	RPD
1	1	1	65	0.02	0.00	0.01	0.00	0.01	0.00	0.02	0.00	0.01	0.00
1	1	2	217.2	0.24	0.00	0.01	0.00	0.01	0.00	0.02	0.00	0.01	0.73
1	2	1	149	0.14	0.00	0.01	0.00	0.01	0.00	0.02	0.00	0.01	0.00
1	2	2	423	5.93	0.00	0.01	0.00	0.01	0.00	0.02	0.00	0.01	3.30
2	1	1	136	0.12	0.00	0.01	0.00	0.01	0.00	0.03	0.00	0.01	0.73
2	1	2	246	8.85	0.00	0.01	0.00	0.01	0.00	0.03	0.00	0.01	9.75
2	2	1	219	9.412	0.00	0.01	0.00	0.01	0.00	0.03	0.00	0.01	2.73
2	2	2	N/A	7200+	N/A		N/A		N/A		N/A		N/A
Avg.					0.00	0.01	0.00	0.01	0.00	0.02	0.00	0.01	2.16

Table 6. The test results of large-sized problems.

$\|F\|$	$\|M\|$	$\frac{\|K\|}{\|M\|}$	$\frac{\|B\|}{\|K\|}$	Best	ACO		ACO_NB		GA		PSO	
					RPD	Time	RPD	Time	RPD	Time	RPD	Time
4	3	3	3	964.68	0.08	0.50	0.08	0.48	0.59	0.22	2.16	0.15
4	3	3	4	1458.448	0.00	1.09	0.00	1.10	0.96	0.42	1.10	0.28
4	3	4	3	1402.04	0.29	1.19	0.29	1.17	0.07	0.55	1.55	0.39
4	3	4	4	1776.96	1.36	2.62	1.39	3.37	0.15	1.03	1.89	0.56
4	4	3	3	859.28	0.00	0.65	0.00	0.69	0.43	0.38	0.47	0.27
4	4	3	4	1498.52	0.22	3.49	0.30	3.23	0.48	1.21	2.73	0.60
4	4	4	3	1218.04	0.00	1.40	0.00	1.48	1.54	0.53	1.28	0.54
4	4	4	4	1806.64	0.51	7.96	0.53	6.81	0.14	2.34	1.80	1.28
4	5	3	3	953.6	0.00	1.00	0.00	1.14	0.39	0.72	0.39	0.46
4	5	3	4	1383.896	0.00	3.50	0.00	3.19	1.08	1.01	1.07	0.92
4	5	4	3	1324.6	0.64	3.72	0.67	3.74	0.10	1.97	1.96	0.94
4	5	4	4	1924.68	0.00	7.12	0.00	6.75	0.73	2.62	0.82	1.76
5	3	3	3	944	0.00	0.28	0.00	0.31	0.79	0.22	0.91	0.14
5	3	3	4	1347.904	0.08	1.28	0.09	1.30	0.32	0.53	2.29	0.25
5	3	4	3	1168.44	0.12	1.12	0.14	1.08	0.27	0.54	1.58	0.29
5	3	4	4	1922.64	0.00	3.32	0.01	2.91	0.86	1.17	1.03	0.58
5	4	3	3	966	0.00	0.64	0.00	0.78	0.56	0.43	0.59	0.27
5	4	3	4	1583.16	0.61	3.52	0.60	3.26	0.27	1.31	1.27	0.53
5	4	4	3	1206.48	0.00	1.86	0.00	1.94	1.42	0.56	1.13	0.55
5	4	4	4	1733.96	0.00	5.08	0.00	4.52	1.54	1.68	1.67	1.03
5	5	3	3	977.6	0.73	1.80	0.78	2.18	0.22	0.89	3.51	0.52
5	5	3	4	1360.248	0.06	5.22	0.10	5.39	1.05	2.51	2.31	0.97
5	5	4	3	1152	0.00	2.76	0.00	2.98	1.06	0.89	0.73	0.90
5	5	4	4	1827.6	0.11	15.07	0.16	14.06	0.10	4.65	1.64	2.09
6	3	3	3	990.92	0.72	0.55	0.73	0.57	0.36	0.31	2.20	0.16
6	3	3	4	1449.64	0.06	1.47	0.07	1.52	0.31	0.66	0.92	0.27
6	3	4	3	1300.6	0.00	0.62	0.00	0.71	0.45	0.39	0.52	0.28
6	3	4	4	1932.56	0.02	3.25	0.02	2.99	0.80	1.43	0.95	0.53
6	4	3	3	915.04	0.08	1.14	0.11	0.95	0.25	0.54	1.72	0.26
6	4	3	4	1382.528	0.26	3.27	0.27	3.73	1.48	1.35	2.98	0.52
6	4	4	3	1286.6	0.00	1.52	0.00	1.67	0.35	0.93	0.41	0.66
6	4	4	4	1959.84	0.06	8.84	0.08	7.50	1.33	3.27	2.65	1.33
6	5	3	3	962.8	0.00	1.35	0.01	1.66	0.64	0.73	0.67	0.47
6	5	3	4	1437.832	0.32	5.16	0.38	5.81	1.74	1.83	2.33	0.90
6	5	4	3	1306.64	0.06	4.24	0.06	4.35	0.21	1.90	1.19	1.02
6	5	4	4	1919.76	0.08	13.48	0.09	12.55	0.40	4.64	1.88	1.91
Avg.					0.18	3.36	0.19	3.27	0.65	1.29	1.51	0.68

Table 7. Paired *t*-test between ACO_NB-ACO.

.	*N*	Mean	St. Dev.	St. e. Mean	Lower	*t*-Value	*p*-Value
Paired *t*-test for ACO_NB-ACO					0.16	3.75	0.001
ACO_NB	36	1386.2	337.0	56.2			
ACO	36	1385.8	336.8	56.1			
Difference	36	0.35	0.56	0.09			

In the third experiment, the results produced by ACO, GA, and PSO in solving large-sized problems were relatively compared. The experiment was conducted on the same large-sized problems used in the second experiment, and the average RPD values obtained by ACO, GA, and PSO were 0.18, 0.65, and1.51 in Table 6, respectively. These results indicate that ACO obtained better RPD values than GA and PSO. However, it took more computing time for it to obtain the best feasible solutions because, despite the fact that the algorithm generally converges quickly to the best solution, ACO requires a considerable amount of computing time to execute the proposed MMAS algorithm within one pheromone search iteration. In our batch scheduling problem, the calculation of desirability under the MMAS algorithm exponentially increases the algorithm running time when $|B|/|K|$ increases.

To determine whether the application of GA or PSO could result in an improved solution, an additional experiment in which the computing times for GA and PSO were extended until the ACO solution converged was conducted. Using the input data from the large-sized problem, GA and PSO were retested, with the results summarized in Table 8. The average RPD values produced by ACO, GA, and PSO were 0.33, 0.48, 1.55, respectively, indicating that applying GA and PSO could not improve the ACO solution quality even if they were provided with additional computing time. To statistically validate the results in Table 8, *t*-tests were conducted to verify the differences in performance between ACO and the other meta-heuristics. As the same instance of the problem was used in each case, a paired *t*-test $(\alpha = 0.05)$ was conducted. The results of paired *t*-testing between PSO-ACO and GA-ACO are shown in Table 9. In both cases, the *p*-values are less than 0.05, suggesting that ACO performed significantly better than either PSO or GA.

Table 8. The mean RPD of large-sized problems with an equal running time.

$\|F\|$	ACO	GA	PSO
4	0.35	0.51	1.36
5	0.40	0.42	1.66
6	0.23	0.51	1.63
Average	0.33	0.48	1.55

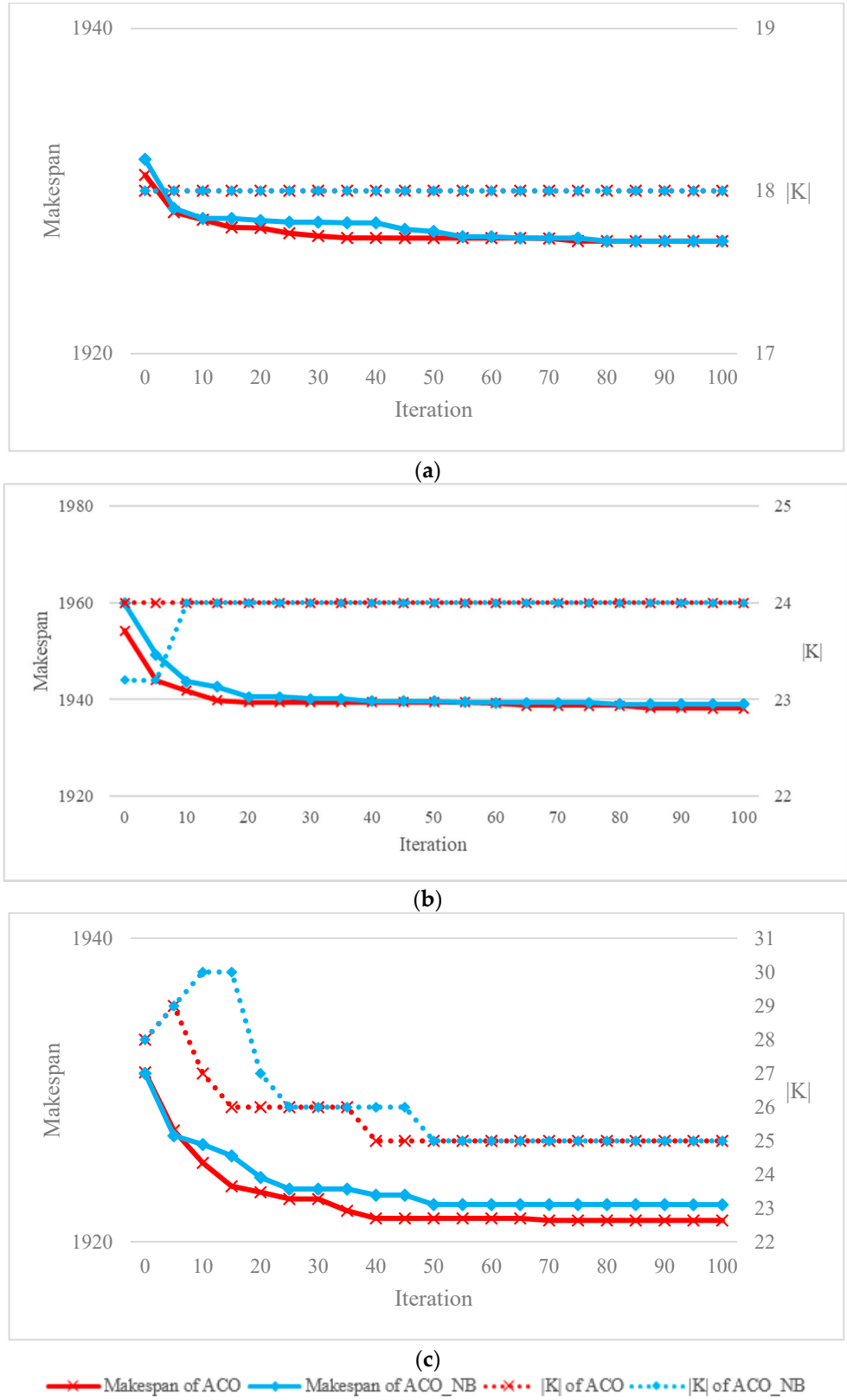


Figure 4. Convergence of ACO and ACO_NB in each level for $|F|$. (**a**) Convergence of algorithms with the instance as $|F| = 4$. (**b**) Convergence of algorithms with the instance as $|F| = 5$. (**c**) Convergence of algorithms with the instance as $|F| = 6$.

Table 9. Paired *t*-test between PSO-ACO and GA-ACO.

	N	Mean	St. Dev.	St. e. Mean	Lower	*t*-Value	*p*-Value
Paired *t*-test for PSO-ACO					15.577	18.04	0.000
PSO	180	1403.6	339.0	25.3			
ACO	180	1386.1	332.6	24.8			
Difference	180	17.490	13.009	0.970			
Paired *t*-test for GA-ACO					0.209	2.21	0.029
GA	180	1388.1	332.5	24.8			
ACO	180	1386.1	332.6	24.8			
Difference	180	1.987	12.090	0.901			

8. Conclusions

In this paper, we considered a BLSP for parallel BPMs with batch deterioration and applied RMAs. In the proposed BLSP, the time-dependent deterioration occurring during batch processing and the need to schedule RMAs between all batches increases the complexity of the batch scheduling problem. As an MF algorithm from a previous paper could not find an optimal solution to even small-sized versions of this problem, we solved the batch scheduling problem by dividing it into three stages, determining the number of buckets, assigning the batches to buckets, and scheduling the buckets to machines. Determining the number of buckets fixes the position of the RMAs and reduces the complexity of the batch scheduling problem. The lower and upper bounds of the number of buckets, LB_k and UB_k, respectively, are calculated to improve the solution performance and increase the speed of convergence. To schedule batches into a fixed number of buckets, an ACO is used to assign batches and a derived rule is used to sequence the batches. In solving the batch assignment problem, the ACO outperformed both GA and PSO. Finally, a dispatching rule is used to schedule buckets to the machines. Using this three-stage ACO-based batch scheduling algorithm, the proposed method finds optimal solutions for small-sized problems and provides better-quality solutions for large-size problems than can be obtained using other meta-heuristics.

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

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (grant number NRF-2019R1F1A1056119).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Uzsoy, R. Scheduling batch processing machines with incompatible job families. *Int. J. Prod. Res.* **1995**, *33*, 2685–2708. [CrossRef]
2. Jia, Z.; Wang, C.; Leung, J.Y.T. An ACO algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families. *Appl. Soft Comput.* **2016**, *38*, 395–404. [CrossRef]
3. Lee, C.Y.; Leon, V.J. Machine scheduling with a rate-modifying activity. *Eur. J. Oper. Res.* **2001**, *128*, 119–128. [CrossRef]
4. Joo, C.M.; Kim, B.S. Genetic algorithms for single machine scheduling with time-dependent deterioration and rate-modifying activities. *Expert Syst. Appl.* **2013**, *40*, 3036–3043. [CrossRef]

5. Dupont, L.; Ghazvini, F.J. Minimizing makespan on a single batch processing machine with non-identical job sizes. *J. Eur. Des Systèmes Autom.* **1998**, *32*, 431–440.
6. Zhou, S.; Chen, H.; Li, X. Distance matrix based heuristics to minimize makespan of parallel batch processing machines with arbitrary job sizes and release times. *Appl. Soft Comput. J.* **2017**, *52*, 630–641. [CrossRef]
7. Dupont, L.; Dhaenens-Flipo, C. Minimizing the makespan on a batch machine with non-identical job sizes: An exact procedure. *Comput. Oper. Res.* **2002**, *29*, 807–819. [CrossRef]
8. Ikura, Y.; Gimple, M. Efficient scheduling algorithms for a single batch processing machine. *Oper. Res. Lett.* **1986**, *5*, 61–65. [CrossRef]
9. Lee, C.Y.; Uzsoy, R.; Martin-Vega, L.A. Efficient algorithms for scheduling semiconductor burn-in operations. *Oper. Res.* **1992**, *40*, 764–775. [CrossRef]
10. Cheng, B.; Wang, Q.; Yang, S.; Hu, X. An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes. *Appl. Soft Comput. J.* **2013**, *13*, 765–772. [CrossRef]
11. Uzsoy, R. Scheduling a single batch processing machine with non-identical job sizes. *Int. J. Prod. Res.* **1994**, *32*, 1615–1635. [CrossRef]
12. Jia, Z.; Li, X.; Leung, J.Y.T. Minimizing makespan for arbitrary size jobs with release times on P-batch machines with arbitrary capacities. *Future Gener. Comput. Syst.* **2017**, *67*, 22–34. [CrossRef]
13. Ozturk, O.; Begen, M.A.; Zaric, G.S. A branch and bound algorithm for scheduling unit size jobs on parallel batching machines to minimize makespan. *Int. J. Prod. Res.* **2017**, *55*, 1815–1831. [CrossRef]
14. Jia, Z.; Yan, J.; Leung, J.Y.T.; Li, K.; Chen, H. Ant colony optimization algorithm for scheduling jobs with fuzzy processing time on parallel batch machines with different capacities. *Appl. Soft Comput. J.* **2019**, *75*, 548–561. [CrossRef]
15. Arroyo, J.E.C.; Leung, J.Y.T. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. *Comput. Oper. Res.* **2017**, *78*, 117–128. [CrossRef]
16. Arroyo, J.E.C.; Leung, J.Y.T. An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. *Comput. Ind. Eng.* **2017**, *105*, 84–100. [CrossRef]
17. Arroyo, J.E.C.; Leung, J.Y.T.; Tavares, R.G. An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times. *Eng. Appl. Artif. Intell.* **2019**, *77*, 239–254. [CrossRef]
18. Gao, Y.; Yuan, J. Unbounded parallel-batch scheduling under agreeable release and processing to minimize total weighted number of tardy jobs. *J. Comb. Optim.* **2019**, *38*, 698–711. [CrossRef]
19. Zhou, S.; Xie, J.; Du, N.; Pang, Y. A random-keys genetic algorithm for scheduling unrelated parallel batch processing machines with different capacities and arbitrary job sizes. *Appl. Math. Comput.* **2018**, *334*, 254–268. [CrossRef]
20. Li, X.; Zhang, K. Single batch processing machine scheduling with two-dimensional bin packing constraints. *Int. J. Prod. Econ.* **2018**, *196*, 113–121. [CrossRef]
21. Fu, R.; Tian, J.; Li, S.; Yuan, J. An optimal online algorithm for the parallel-batch scheduling with job processing time compatibilities. *J. Comb. Optim.* **2017**, *34*, 1187–1197. [CrossRef]
22. Azizoglu, M.; Webster, S. Scheduling a batch processing machine with incompatible job families. *Comput. Ind. Eng.* **2001**, *39*, 325–335. [CrossRef]
23. Yao, S.; Jiang, Z.; Li, N. A branch and bound algorithm for minimizing total completion time on a single batch machine with incompatible job families and dynamic arrivals. *Comput. Oper. Res.* **2012**, *39*, 939–951. [CrossRef]
24. Jolai, F. Minimizing number of tardy jobs on a batch processing machine with incompatible job families. *Eur. J. Oper. Res.* **2005**, *162*, 184–190. [CrossRef]
25. Balasubramanian, H.; Mönch, L.; Fowler, J.; Pfund, M. Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *Int. J. Prod. Res.* **2004**, *42*, 1621–1638. [CrossRef]
26. Li, X.L.; Li, Y.P.; Huang, Y.L. Heuristics and lower bound for minimizing maximum lateness on a batch processing machine with incompatible job families. *Comput. Oper. Res.* **2019**, *106*, 91–101. [CrossRef]
27. Ham, A.; Fowler, J.W.; Cakici, E. Constraint programming approach for scheduling jobs with release times, non-identical sizes, and incompatible families on parallel batching machines. *IEEE Trans. Semicond. Manuf.* **2017**, *30*, 500–507. [CrossRef]
28. Vimala Rani, M.; Mathirajan, M. Performance evaluation of ATC based greedy heuristic algorithms in scheduling diffusion furnace in wafer fabrication. *J. Inf. Optim. Sci.* **2016**, *37*, 717–762. [CrossRef]
29. Gupta, J.N.D.; Gupta, S.K. Single facility scheduling with nonlinear processing times. *Comput. Industiral Eng.* **1988**, *14*, 387–393. [CrossRef]
30. Ding, J.; Shen, L.; Lü, Z.; Peng, B. Parallel machine scheduling with completion-time-based criteria and sequence-dependent deterioration. *Comput. Oper. Res.* **2019**, *103*, 35–45. [CrossRef]
31. Browne, S.; Yechiali, U. Scheduling deteriorating jobs on a single processor. *Oper. Res.* **1990**, *38*, 495–498. [CrossRef]
32. Soleimani, H.; Ghaderi, H.; Tsai, P.W.; Zarbakhshnia, N.; Maleki, M. Scheduling of unrelated parallel machines considering sequence-related setup time, start time-dependent deterioration, position-dependent learning and power consumption minimization. *J. Clean. Prod.* **2020**, *249*, 119428. [CrossRef]
33. Joo, C.M.; Kim, B.S. Machine scheduling of time-dependent deteriorating jobs with determining the optimal number of rate modifying activities and the position of the activities. *J. Adv. Mech. Des. Syst. Manuf.* **2015**, *9*, JAMDSM0007. [CrossRef]

34. Woo, Y.B.; Jung, S.W.; Kim, B.S. A rule-based genetic algorithm with an improvement heuristic for unrelated parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities. *Comput. Ind. Eng.* **2017**, *109*, 179–190. [CrossRef]
35. Abdullah, M.; Süer, G.A. Consideration of skills in assembly lines and seru production systems. *Asian J. Manag. Sci. Appl.* **2019**, *4*, 99–123. [CrossRef]
36. Gai, Y.; Yin, Y.; Tang, J.; Liu, S. Minimizing makespan of a production batch within concurrent systems: Seru production perspective. *J. Manag. Sci. Eng.* **2020**, in press. [CrossRef]
37. Liu, F.; Niu, B.; Xing, M.; Wu, L.; Feng, Y. Optimal cross-trained worker assignment for a hybrid seru production system to minimize makespan and workload imbalance. *Comput. Ind. Eng.* **2021**, *160*, 107552. [CrossRef]