International Journal of Industrial Engineering, 32(1), 16-33, 2025

JOINTLY OPTIMIZING PARALLEL BATCH PROCESSING SCHEDULING IN A SEMICONDUCTOR MANUFACTURING ENVIRONMENT

Rui Xu[1, 2, *], Xing Fan[1, 2], Changqing Pang[1], and Jinxue Xu[3]

[1]School of Business
Hohai University
Nanjing, China

[2]Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation
Southern University of Science and Technology
Shenzhen, China
[*]Corresponding author's e-mail: rxu@hhu.edu.cn

[3]School of Computer Science and Technology
University of Science and Technology of China
Hefei, China

The rapid growth of the semiconductor industry has led to high water and energy consumption and substantial greenhouse gas emissions. Achieving sustainability in the semiconductor industry has become an exceedingly important issue. This paper investigates a complex batch processing scheduling problem in the final testing phase of semiconductor manufacturing, where chambers and chips are modeled as batch processing machines and jobs. Machines can process multiple jobs simultaneously, with each job defined by its processing time, release time, and size. A mixed-integer linear programming model is presented, along with a constructive-based metaheuristic, the ACS-PBPMs algorithm, to optimize batch formation and scheduling decisions jointly. The algorithm uses an effective candidate list strategy to address constraints and incorporates a local search phase based on solution characteristics. Experimental results on diverse problem instances show that the ACS-PBPMs algorithm outperforms CPLEX and competitive algorithms in computational efficiency and solution quality.

Keywords: Semiconductor Industry, Batch Processing, Parallel Machines, Ant Colony Optimization, Jointly Optimization

(Received on November 6, 2024; Accepted on January 1, 2025)

1. INTRODUCTION

The semiconductor industry is a cornerstone of modern technology, driving technological innovations across critical sectors. Its importance extends beyond economic impact to strategic relevance, enabling core technologies such as artificial intelligence, 5G communications, and the Internet of Things (IoT), which are fundamental to advancements in smart cities and automation (Rodrigues *et al.*, 2022). However, the rapid growth of this industry has led to significant challenges, including high water and energy consumption and considerable greenhouse gas emissions (Wang *et al.*, 2023). Efficient scheduling methods are essential to mitigate bottlenecks and enhance operational efficiency in semiconductor manufacturing.

The semiconductor manufacturing process is divided into four main stages: wafer fabrication, probing, assembly, and final testing, each characterized by diverse processing times. Some operations may be completed within 15 minutes, while others require over 12 hours. Batch processing is particularly common, often involving extended operation times that create bottlenecks at non-batching machines, leading to a non-linear flow of products through the factory. As a result, effective batch processing is crucial for improving overall production efficiency (Wu *et al*., 2023; Zhou *et al*., 2021).

One prominent example of batch processing in semiconductor manufacturing is the testing of chips in burn-in ovens during the final testing phase. These ovens, which can handle multiple chips at the same time as long as their combined size does not exceed the oven's capacity, are considered Batch Processing Machines (BPMs). Similar scheduling challenges occur in other manufacturing steps, such as processes that involve heating, coating, or material changes.

Given the critical role of batch processing in semiconductor manufacturing, the effective scheduling of BPMs under dynamic and heterogeneous conditions has emerged as a key challenge for achieving sustainable and efficient production

systems. Based on the sequencing methods of jobs within a batch, there are three types of batch processing problems (Beldar *et al.*, 2024). In the semiconductor manufacturing industry, p-batch is more important and commonly used. This paper focuses on investigating a significant p-batch-based batch processing problem within the context of semiconductor manufacturing. A critical aspect of batch processing problems is managing non-identical job sizes within a given machine capacity, which limits the total job size to not exceed the machine's maximum capacity. Batch processing problems become more complex when considering parallel machines, especially when those jobs arrive dynamically, meaning they have varying release times and sizes. For example, in semiconductor manufacturing, some jobs may be delayed due to material shortages or machine maintenance, leading to dynamic arrival times that complicate scheduling decisions.

Ant colony optimization (ACO) algorithms have been proven effective for various scheduling and routing problems, often outperforming traditional heuristic approaches such as simulated annealing (SA) and genetic algorithms (GA) in terms of solution quality and robustness (Nguyen and Sheen, 2023). However, unlike SA and GA, which primarily focus on exploring solution spaces globally, ACO offers a construction-based framework that is particularly suitable for integrating batch formation and sequencing decisions. Furthermore, as a population-based metaheuristic algorithm, ACO also has better adaptability than individual-based metaheuristic algorithms since it can maintain solution diversity during the evolutionary process and is not sensitive to the initial solution. However, directly using the ACO algorithm to generate an integrated solution containing both batch formation and batch sequencing for complex environments is difficult due to the interrelated nature of these two decisions, as well as dynamic arrivals and varying job size constraints. Therefore, it is necessary to effectively integrate both decisions during solution construction while balancing machine utilization and job waiting times to address these challenges.

The main contributions are summarized as follows.

(1) We established a mixed integer linear programming model (MILP) for the parallel batch processing problem arising from semiconductor manufacturing.
(2) We developed a hybrid ant colony optimization (ACO) algorithm that incorporates characteristics of candidate lists and local search to guide the search and enhance performance.
(3) We verified the performance of the proposed algorithm using CPLEX and benchmark algorithms, demonstrating superior performance, especially in large search spaces.

The remainder of this paper is structured as follows: Section 2 provides a brief review of existing work related to the problem. In Section 3, the mathematical model and a constructive heuristic for the problem are defined. Section 4 introduces the proposed ACO approach, incorporating local search and a candidate list strategy based on problem-specific constraints. Section 5 evaluates and compares the performance of the proposed approaches with CPLEX and other benchmark algorithms. Finally, Section 6 concludes the paper.

2. LITERATURE REVIEW

In past research, the BPM scheduling problem has attracted widespread research interest. Most of these studies revolve around single-machine problems and parallel-machine problems, focusing on minimizing job times and using homogeneous machines. Our literature review focuses on single-machine and parallel-machine cases, with a focus on features such as non-identical job sizes and dynamic job arrivals.

In the beginning, researchers started with the simplest model scheduling for a single BPM with limited constraints (Lee *et al*., 1992; Uzsoy, 1994; Uzsoy and Yang, 1997). To prevent the system from being blocked due to the unavailability of a single BPM, most studies have extended the BPM problem to parallel machine environments. Chang *et al.* (2004) proposed a simulated annealing (SA) algorithm for minimizing makespan, C_{max}, on PBPMs with non-identical job sizes and compared the results with CPLEX software. Zhang *et al.* (2022a) study a single-batch-processing machine (SBPM) scheduling problem by considering a just-in-time criterion and modeling this scheduling problem using a mixed-integer linear model. Nguyen and Sheen (2023) consider a parallel batch processing problem to minimize the makespan under constraints of arbitrary lot sizes, start time windows and incompatible families. Gahm *et al.* (2022) dealt with parallel serial-batch processing machine scheduling by a mixed-integer program and several tailor-made construction heuristics. Park *et al.* (2024) demonstrate the utilization of the Stochastic Utility Evaluation (SUE) function approach to the performance of batch process systems using multiple criteria. Wang *et al.* (2024) consider EFFSP with BPM at a middle stage in the hot & cold casting process, and a feedback-based artificial bee colony (FABC) algorithm is proposed to simultaneously minimize the makespan, total tardiness, and total energy consumption. Muter (2020) tackles both single and parallel batch processing machine scheduling problems and proposes the first exact algorithm based on two-level decomposition for parallel machine problems using and column-and-cut generation algorithm. Zhang *et al.* (2022) and Zhou *et al.* (2022) both studied parallel batch processing in last-mile logistics considering two-dimensional rectangular packing constraints.

In the real world of semiconductor manufacturing, several characteristics, such as non-identical job size and arrival time, are common and have been considered in the recent literature. Uzsoy (1994) was the first to present complexity results for C_{max} criteria. He also provided several heuristics and a branch-and-bound algorithm. Jia *et al*. (2019) studied the problem of scheduling on parallel batch processing machines with non-identical sizes and fuzzy processing times to minimize the makespan. And after constructing a mathematical model of the problem, we propose a fuzzy ant colony optimization (FACO) algorithm. Bektur (2022) considers machine factory-dependent setup times, non-identical factories, position-based learning effects on processing times and setup times, and factory eligibility constraints for the DFSS problem. Xiao *et al*. (2024) investigate an unrelated parallel batch processing machine scheduling problem. A set of jobs with non-identical sizes and arbitrary ready times are scheduled on unrelated parallel batch processing machines with different capacities to minimize the makespan. Zhou *et al*. (2020) consider energy-efficient scheduling of a single batch processing machine with non-identical job sizes and release times under a time-of-use (TOU) electric tariff, a pricing scheme where electricity costs vary based on the time of consumption, aiming to simultaneously minimize total electricity cost (a criterion of environmental and energy sustainability) and makespan (a criterion of productivity).

Recently, Fowler and Mönch (2022) surveyed the literature on PBPMs with compatible settings and incompatible family settings, considered makespan, flow time, and due date-related measures, and discussed scheduling approaches for single machines, parallel machines, and other environments such as flow shops and job shops. Durasevic and Jakobovic (2023) provide a comprehensive literature review on the application of heuristic and metaheuristic methods to solve UPMSP. Despite the extensive research on BPM scheduling problems, the challenges posed by varying job release times and heterogeneous job sizes remain insufficiently addressed, especially in the context of PBPMs. Furthermore, limited attention has been paid to integrating batch formation and sequencing decisions, which are critical for achieving optimal solutions in real-world scenarios.

3. PROBLEM DESCRIPTION AND HEURISTIC

3.1 Problem description

A summary of the notations used in the problem description and mathematical formulation is provided in Table 1. Each variable and parameter are explicitly defined to ensure clarity and to facilitate understanding of the model.

Table 1. List of parameters and variables

Indices	
j	Job index, where $j \in \{1,2,\ldots,n\}$
b	Batch index, where $b \in \{1,2,\ldots,n\}$
m	Machine index, where $m \in \{1,2,\ldots,k\}$
Parameters	
B	Maximum capacity of a machine m
p_j	Processing time of job j
r_j	Release time of job j
s_j	Size of job j
Variables	
x_{jbm}	Binary variable: 1 if job j is assigned to batch b and scheduled on machine m, 0 otherwise.
R_{bm}	Release time of batch b scheduled on machine m
S_{bm}	Starting time of batch b scheduled on machine m
P_{bm}	Processing time of batch b scheduled on machine m, determined by the longest processing time among jobs in the batch.
C_{bm}	Completion time of batch b scheduled on machine m

The scheduling challenge involving multiple identical PBPMs, characterized by non-identical job sizes and job arrivals, aims to minimize makespan. This batch-processing problem is formally denoted as $P_m|r_i, s_i, p\text{-}batch|C_{max}$. We have n jobs that need to be allocated into several batches and scheduled across m identical PBPMs. Each job i is defined by its release time r_j, processing time p_j and size s_j. We operate under the assumption that estimates for the processing times, sizes, and arrival times of the jobs are available and known in advance based on deterministic data. In practice, these estimates can be derived from engineering data and real-time tracking through Shop Floor Information Systems that monitor Work in Process

inventory and machine statuses. When multiple jobs are selected for batch processing, the time required for the batch is determined by the longest processing time among the jobs included. Additionally, the cumulative size of all jobs within a batch must not exceed the processing capacity of the machine handling it.

3.2 Model Formulation

Given the aforementioned assumptions and variables, we develop the following mixed integer linear programming (MILP) model for the scheduling problem as follows.

$$minimize C_{max} \tag{1}$$

$$s.t. \sum_{m=1}^{k} \sum_{b=1}^{n} x_{jbm} = 1, j = 1, \dots, n \tag{2}$$

$$\sum_{j=1}^{n} s_j x_{jbm} \leq B, b = 1, \dots, n; m = 1, \dots, k \tag{3}$$

$$P_{bm} \geq p_j x_{jbm}, j = 1, \dots, n; b = 1, \dots, n; m = 1, \dots, k \tag{4}$$

$$S_{bm} \geq r_j x_{jbm}, j = 1, \dots, n; b = 1, \dots, n; m = 1, \dots, k \tag{5}$$

$$S_{bm} \geq S_{(b-1)m} + P_{(b-1)m}, b = 2, \dots, n; m = 1, \dots, k \tag{6}$$

$$C_{bm} \geq S_{bm} + P_{bm}, b = 1, \dots, n; m = 1, \dots, k \tag{7}$$

$$C_{bm} \geq 0, b = 1, \dots, n; m = 1, \dots, k \tag{8}$$

$$C_{max} \geq C_{bm}, b = 1, \dots, n; m = 1, \dots, k \tag{9}$$

$$x_{jbm} \in \{0,1\}, j = 1, \dots, n; b = 1, \dots, n; m = 1, \dots, k \tag{10}$$

Equation (1) defines the objective function aimed at minimizing the makespan. Equation (2) ensures that each job is assigned to exactly one batch on a given parallel machine. Equation (3) establishes that the total size of all jobs within each batch processed on a machine does not exceed its capacity. Equation (4) calculates the processing time for batch b on machine m, determined by the longest processing time of the jobs included in the batch. Equations (5) and (6) specify the starting time of batch b on machine m, which is based on either the release time of the job with the latest release time in the batch or the completion time of the preceding batch. Equations (7) and (8) compute the completion time of batch b on machine m, which is the sum of the starting time and the processing time. Equation (9) identifies the objective, represented as the maximum completion time across all batches. Finally, Equation (10) outlines the decision variable. Since the number of batches processed on a machine is not predetermined prior to reaching a final solution, it is initially set to equal n, with the processing time for batch b designated as zero if the batch is empty.

3.3 Heuristic algorithm

As previously mentioned, addressing the target problem requires making two crucial decisions: batch formation and batch sequencing. In this paper, we develop two constructive heuristics called the Batch Earliest (BE) heuristic and the Free Machine First-Waste and Idle Space (FMF-WIS) heuristic for the problem. The Batch Earliest (BE) heuristic generates a solution by applying two separate dispatching rules. First, it uses the BFLPT rule for batch formation, then schedules the formed batches onto machines using the ERT-LPT rule, treating batch formation and scheduling as independent steps. In contrast, the Free Machine First-Waste and Idle Space (FMF-WIS) heuristic integrates these decisions within a single process. It begins by selecting the first available machine and dynamically forming batches based on the current context. Once a batch is formed, it is immediately assigned, and its processing sequence is established, ensuring an interactive and iterative approach that continues until all jobs are batched and scheduled efficiently.

Batch scheduling for parallel machines involves balancing multiple dimensions, such as job sizes, release times, and processing times. Selecting an appropriate heuristic to guide batch formation is critical to achieving efficient schedules. The FMF-WIS heuristic was selected because it captures both machine capacity and scheduling constraints in a single metric Waste and Idle Space (WIS), making it particularly suited to the dynamic and multi-dimensional nature of PBPM scheduling problems.

Thus, our goal is to group jobs in a way that reduces WIS as much as possible during batch formation. Specifically, a candidate job u reducing the most value of $WIS(\pi_m)$ on machine m of schedule π will be a better choice. The reduced value $\Lambda WIS(\pi_m^u)$ after assigning job u can be obtained by Equation (11).

$$\Lambda WIS(\pi_m^u) = s_u \cdot p_u - B \cdot (C_{m'} - C_m) \tag{11}$$

where $C_{m'}$ denote the new value of C_m after job u has been assigned into the current schedule π_m.

The detailed FMF-WIS algorithm is outlined as follows:

Step 1: Select a free machine m (ties are broken arbitrarily when multiple machines are free) and create an empty batch b on this machine.

Step 2: Compile jobs with release times less than or equal to the machine's ready time into a candidate set, referred to as S. If set S is empty, assign the earliest job to batch b. If S is not empty, assign the job with the longest processing time from set S to batch b.

Step 3: Calculate the $\Lambda WIS(\pi_m^u)$ value and index the jobs in non-increasing order of their $\Lambda WIS(\pi_m^u)$. Assign the first job that satisfies the machine capacity constraint to the current batch b. Repeat this step until no more jobs can fit in the current batch.

Step 4: Continue to repeat the above steps until all jobs have been assigned to batches on machines.

To demonstrate how to obtain a feasible solution using the BE and FMF-WIS heuristics, we present a 10-job problem instance (see Table 2) with two machines, each having a capacity of 10. Tables 3 and 4 illustrate the progression of the BE heuristic in obtaining a feasible solution for this example, while Table 5 shows the evolution of the FMF-WIS heuristic. Column Unassigned Jobs contains all unscheduled jobs at the beginning of each iteration. Column Candidate Jobs presents the candidate jobs of the current open batch b for FMF-WIS. Column b presents the index of the current batch for each iteration. Column J presents the job set assigned in the batch b. Columns P_{bm} and R_{bm} present the processing time and release time of each batch. Column RC_{bm} presents the residual capacity of the batch b after job set J is located. Column M presents the machine where the batch b is located. Columns C_{max} $(M1)$ and C_{max} $(M2)$ present when each machine will become available to process the next batch, which is the completion time of the current batch b.

Table 2. Data for 10-job of problem instance

j	1	2	3	4	5	6	7	8	9	10
p_j	9	3	4	1	9	5	4	2	3	6
r_j	8	1	15	8	7	14	9	1	1	12
s_j	2	1	4	4	2	7	2	7	6	1

Table 3. Evolution of the BE heuristic (the first BFLPT rule)

Unassigned Jobs	b	J	P_{bm}	R_{bm}	RC_{bm}
1,2,3,4,5,6,7,8,9,10	1	1	9	8	8
2,3,4,5,6,7,8,9,10	1	1,5	9	8	6
2,3,4,6,7,8,9,10	1	1,5,10	9	12	5
2,3,4,6,7,8,9	2	6	5	14	3
2,3,4,7,8,9	1	1,5,10,3	9	15	1
2,4,7,8,9	2	6,7	5	14	1
2,4,8,9	1	1,5,10,3,2	9	15	0
4,8,9	3	9	3	1	4
4,8	3	9,4	3	8	0
8	4	8	2	1	3

Table 4. Evolution of the BE heuristic (the second ERT-LPT rule)

b	M	$C_{max}(M1)$	$C_{max}(M2)$
4	1	3	0
3	2	3	11
2	1	19	11
1	2	19	24

Table 5. Evolution of the FMF-WIS heuristic

Unassigned Jobs	Candidate Jobs	b	J	P_{bm}	R_{bm}	RC_{bm}	M	$C_{max}(M1)$	$C_{max}(M2)$
1,2,3,4,5,6,7,8,9,10	2,8,9	1	2	3	1	9	1	4	0
1,3,4,5,6,7,8,9,10	8,9	1	2,9	3	1	3	1	4	0
1,3,4,5,6,7,8,10	8	2	8	2	1	3	2	4	3
1,3,4,5,6,7,10	5	3	5	9	7	8	2	4	16
1,3,4,6,7,10	1,4,7,10	3	5,1	9	8	6	2	4	17
3,4,6,7,10	4,7,10	3	5,1,4	9	8	2	2	4	17
3,6,7,10	7,10	3	5,1,4,7	9	9	0	2	4	18
3,6,10	10	4	10	6	12	9	1	18	18
3,6	3,6	4	10,6	6	14	2	1	20	18
3	3	5	3	4	15	6	2	20	22

In contrast, the FMF-WIS heuristic constructs a final solution directly, eliminating the need for supplementary rules. It integrates both decisions in a sequential and interactive manner, allowing for a more cohesive approach to scheduling.

In the example, the optimal solution obtained by CPLEX is 21. The figures show that the BE heuristic yields a solution comprising four batches, resulting in an objective completion time C_{max} of 24. In contrast, the FMF-WIS heuristic constructs a solution with five batches, achieving a C_{max} of 22, which is closer to the optimal solution. This improvement over BE can be attributed to FMF-WIS's ability to effectively share information during the decision-making process. Additionally, WIS serves as a reliable indicator, incorporating all three key factors mentioned earlier.

4. SOLUTION APPROACH

The ACO framework is a novel nature-inspired metaheuristic introduced by Marco Dorigo in the early 1990s and has gained immense popularity in recent years (Liu *et al.*, 2022; Saemi *et al*., 2022). For a more comprehensive review of ACO applications, please refer to the works of Saber et al. (2023a) and Blum (2024).

Since ACO is a construction-based metaheuristic, it not only provides the constructive advantages of FMF-WIS but also addresses its myopic limitations. In this paper, we adopt the ACS framework, which is recognized as one of the most effective and widely used variations (Blum, 2024). To enhance the effectiveness and efficiency of the proposed ACS algorithm (referred to as ACS-PBPMs) for the specific problem at hand, we incorporate two distinctive features based on problem-specific knowledge:

1. We have developed an efficient candidate list strategy that considers two critical constraints: machine capacity and release time. This strategy directs the search toward promising areas within the solution space, effectively balancing increased machine utilization with reduced job waiting times.
2. A local search procedure is designed based on the characteristics of the solution and is executed after an ant has completed its solution construction. This enhancement significantly boosts the algorithm's performance. The rationale for this improvement lies in the generally weak local search capability of ACS; hence, incorporating a local search feature is beneficial within the algorithm's framework.

4.1 Initialization

In the case of our problem, we define $\tau_{u,j}$ as the pheromone trail, which represents the desirability of having job *u* and *j* in the same batch, which is fixed during the construction process. However, it could not be utilized directly since the number of jobs in a batch is uncertain until the batch is full or no space is available. Thus, we define a variable $\tau_{bm}^{u} = \sum_{j \in bm} \tau_{u,j} / |bm|$. It is the average value of pheromone trail $\tau_{u,j}$, where j denotes the one that has been assigned into the current batch b of machine m and job u is a candidate job. If bm is empty, the variable τ_{bm}^{u} which refers to the desirability of assigning job u into the current batch b on machine m is set to one.

As mentioned earlier, selecting a job that reduces the value of WIS the most during batch formation is optimal. Specifically, the next job chosen should preferably have a larger value of $\Lambda WIS(\pi_m^u)$. Accordingly, the heuristic information used in this process is calculated as follows,

$$\eta_{bm}^{u} = \begin{cases} s_u \cdot p_u - B \cdot (C_{m'} - C_m) \text{ if } \Lambda WIS(\pi_m^u) > 0 \\ \frac{1}{1+|s_u \cdot p_u - B \cdot (C_{m'} - C_m)|} \quad \text{otherwise} \end{cases} \tag{12}$$

which means the variable η_{bm}^{u} is directly proportional to $\Lambda WIS(\pi_m^u)$. We add one to the denominator to prevent division by zero for the case that $\Lambda WIS(\pi_m^u)$ is a non-positive value.

4.2 Candidate list strategy

The candidate list strategy is a key component of our ACS-PBPMs algorithm, designed to guide the decision-making process when forming batches. A candidate list is a set of jobs that are eligible to be added to the current batch based on two primary factors:

1) **Machine Capacity:** The total size of jobs in a batch cannot exceed the machine's capacity. Only jobs that fit within the remaining capacity of the batch are considered.
2) **Job Release Time:** Jobs must be available for processing at the time the batch starts. This ensures that no unnecessary delays are introduced while waiting for jobs to arrive.

The candidate list helps prioritize jobs that meet these two criteria, ensuring that the batch formation process is both efficient and compliant with the problem's constraints. For example, if a machine has limited capacity, the strategy will only consider smaller jobs that can fit within that limit. Similarly, if a job is not yet ready for processing, it will not be included in the candidate list.

For example, as illustrated in Figure 1, consider a machine with a capacity $B = 2$ and unit job sizes. After assigning job $J1$ to the first batch, a decision must be made on whether to wait for the incoming jobs $J2$, $J3$, and $J4$. If a non-delay candidate condition is applied, job $J1$ will be processed immediately, resulting in the best solution depicted in Solution 1. To enhance machine utilization and form a larger batch, alternative solutions explore different scenarios of waiting for $J2$, $J4$, and $J3$, respectively. As demonstrated, waiting for incoming jobs can lead to suboptimal outcomes (e.g., Solution 3), but it may also yield improved solutions (e.g., Solution 4). Therefore, it is essential to establish candidate conditions that help determine which jobs are worth waiting for.

Theorem: Let b be the last non-empty batch on machine m and u be an unassigned job whose size does not exceed the residual capacity of the batch. If the releasing time of job u satisfies the condition $r_u < S_{bm} + min\{p_u, P_{bm}\}$, then the objective value of assigning job u into batch b is equal to or greater than the value processing job *u* after batch b.

Proof: We will prove this by contradiction. We consider two scenarios: assigning job u into batch b versus processing job u after batch b.

Case 1: if job u is assigned to batch b, the resulting objective value is: $C_{max1} = r_u + \max\{p_u, P_{bm}\}$.

Case 2: if job u is processed after batch b, the resulting objective value is: $C_{max2} = S_{bm} + p_u + P_{bm}$.

Next, we analyze C_{max1} based on the relationship between p_u and P_{bm}. If $p_u \geq P_{bm}$, then $C_{max1} = r_u + p_u$. Suppose $C_{max1} \geq C_{max2}$ it requires $r_u - S_{bm} \geq P_{bm}$, which is contradictory to the given condition $r_u < S_{bm} + min\{p_u, P_{bm}\}$. Otherwise, $C_{max1} = r_u + P_{bm}$ if f $p_u < P_{bm}$. Suppose $C_{max1} \geq C_{max2}$, it needs $r_u - S_{bm} \geq P_{bm}$, which is contradictory either.

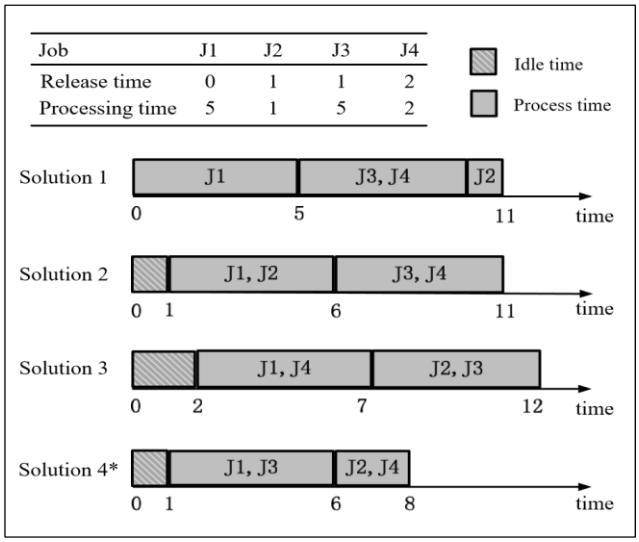


Figure 1. An example to illustrate the performance of four solutions with different candidate strategies

Solution 1 presents the result using the non-delay candidate strategy, and the other solutions denote the different cases assigning the incoming jobs $(J2, J4, J3)$ and $J1$ in one batch, respectively. Solution 4 achieves the best performance among all cases.

From the theorem, we can deduce that if the delay time of batch b is less than the minimum processing time between job u and batch b, it is advantageous to postpone processing batch b until job u arrives, allowing them to be combined into a single batch. For instance, as illustrated in Figure 1, job $J3$ is the only one that meets the candidate condition, and the objective value achieved in Solution 4, where $J3$ and $J1$ are assigned together in one batch, is the optimal among all solutions.

Based on the consideration, we propose the candidate list θ used in the ACS-PBPMs algorithm.

$$\theta = \left\{ u \middle| s_u \le \left(B - \sum\nolimits_{j \in bm} s_j \right) \text{and } r_u \le S_{bm} + min\{p_u, P_{bm}\} \right\} \tag{13}$$

This candidate list θ effectively balances reducing machine idle time with enhancing machine utilization. However, since the assignment process is continuous, it is not feasible to rely solely on current information to make optimal decisions with this candidate strategy. Therefore, we also implement a local search procedure to further enhance performance once a feasible solution has been established.

4.3 Solution construction and pheromone updating

The ACS-PBPMs employ a construction-based metaheuristic approach, similar to the solution construction policy utilized in the FMF-WIS heuristic. The specifics of this process are illustrated in Figure 2. The pseudo-random proportional rule for selecting the next job from the candidate list is defined as follows,

$$u = \begin{cases} \arg \underset{u \in \Theta}{max} [\tau_{bm}^u] \cdot [\eta_{bm}^u]^\beta & \text{if } q \le q_0 \\ \hat{u} & \text{otherwise} \end{cases} \tag{14}$$

where q_0 is a parameter in ACS-PBPMs that represents the probability of selecting job u with the largest combined pheromone trail and heuristic value. q is a random variable uniformly distributed in [0, 1]. The variable $\hat{u}$ can be calculated as follows.

$$\hat{u} = \begin{cases} \dfrac{[\tau_{bm}^{u}] \cdot [\eta_{bm}^{u}]^{\beta}}{\sum_{u \in \Theta}([\tau_{bm}^{u}] \cdot [\eta_{bm}^{u}]^{\beta})} & \text{if } u \in \theta \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

In ACS-PBPMs, pheromone trails are updated through two distinct processes, each occurring at different phases. The global updating rule applies only to the best-so-far solution at the conclusion of each iteration. Given the nature of batch processing machines (BPM), where multiple jobs may be grouped together in feasible solutions, the value of $\tau_{u,j}$ is reinforced when jobs u and j are assigned to the same batch, according to the following equation,

$$\tau_{u.i} = (1 - \rho) \cdot \tau_{u.i} + h_{u.i} \cdot \rho \cdot \Lambda\tau_{u.i} \tag{16}$$

where $\rho \in (0,1)$ is a parameter that adjusts the pheromone evaporation rate. $\Lambda\tau_{u,j}$ represents the amount of pheromone deposited based on the quality of the best-so-far solution.

$$\Lambda\tau_{u,j} = \begin{cases} \dfrac{1}{c_{max}} & \text{if jobs } i \text{ and } j \text{ are assigned in the same batch} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

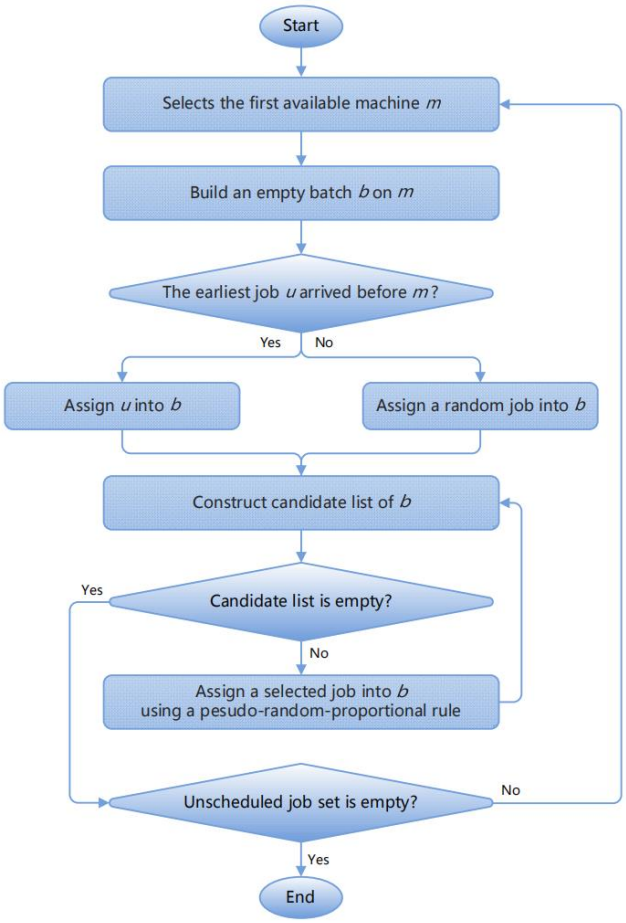


Figure 2. Flow chart of the solution construction process

The local updating rule is applied at each step of the construction process, following the formulation that incorporates the initial pheromone value τ_0.

$$\tau_{u.i} = (1-\rho)\cdot\tau_{u.i} + \rho\cdot\tau_0 \quad (18)$$

4.4 Local search procedure for batch processing

The objective C_{max} of the target problem depends heavily on the last batch b processed on machine m. This is because its completion time has a direct impact on the overall objective value. The completion time of batch b can be represented as $C_{bm} = S_{bm} + P_{bm}$.

To minimize C_{bm}, we can target two components: reducing the batch size S_{bm} or the batch processing time P_{bm}. The processing time P_{bm} is determined by the longest job within the batch, which can be decreased by reallocating the longest job to a different batch. The batch size S_{bm} is defined by the later of two values: the release time of the batch R_{bm} and the completion time of the previous batch $C_{(b-1)m}$. The analysis splits into two cases:

Case 1: If $S_{bm} = R_{bm}$, then S_{bm} is influenced by the latest job in the batch. In this case, moving the latest job to another batch can help reduce S_{bm}.

Case 2: If $S_{bm} = C_{(b-1)m}$, then it depends on the completion time of the previous batch. The completion time $C_{(b-1)m}$ can be expressed as $C_{(b-1)m} = S_{(b-1)m} + P_{(b-1)m}$.

To reduce S_{bm} we must address both $S_{(b-1)m}$ and $P_{(b-1)m}$. The longest job in the previous batch can be moved to another batch to decrease $P_{(b-1)m}$. For $S_{(b-1)m}$, the process will repeat recursively until we encounter a batch where the starting time is not equal to its previous batch's completion time.

Based on the above analysis, we propose a local search procedure that includes two job adjustment mechanisms: job insertion and job interchange.

- Job Insertion: This mechanism transfers a job from its current batch to an existing batch while respecting machine capacity constraints. Given its stricter limitations on capacity, this approach is prioritized to maximize machine utilization.
- Job Interchange: This mechanism involves swapping two jobs from different batches. While this offers more flexibility, it may not directly improve the objective as effectively as job insertion.

The proposed local search procedure is summarized in Algorithm 1. This algorithm systematically explores the potential job adjustments within the defined constraints, focusing primarily on the first class of jobs (the longest job in a batch starting at the previous batch's completion time) to effectively reduce C_{bm}.

Algorithm 1 Local Search procedure

Require:

π: Solution before local search;
Π: Solution after local search;
J: Current job with the longest processing time in B;
Φ_{ins}^{B}: Set of batches, each of which has enough capacity to accommodate J;
Ψ_{int}^{J}: Set of jobs, each of which can be interchanged with J without violate the machine capacity;

Ensure:

1: set $\Pi = \pi;\ \Omega = \emptyset$
2: **repeat**
3: $\Omega = \Omega \cup B_{bm}$
4: $b \leftarrow b - 1$
5: **until** $S_{bm} \neq C_{(b-1)m}$
6: **while** $\Omega \neq \emptyset$ **do**
7: $B = \Omega[0]$ and $\Omega = \Omega - B$
8: $\Phi_{ins}^{B} = InsertableBatch(J)$
9: **while** $\Phi_{ins}^{B} \neq \emptyset$ **do**
10: $B_{ins} = \Phi_{ins}^{B}[0]$ and $\Phi_{ins}^{B} = \Phi_{ins}^{B} - B_{ins}$
11: insert J into B_{ins} and recalculate C_{max} of Π
12: **if** $C_{max}(\Pi) < C_{max}(\pi)$ **then**
13: $\pi = \Pi$ and return to step 1
14: **else**
15: $\Pi = \pi$
16: **end if**
17: **end while**
18: $\Psi_{int}^{J} = InterchangeableJob(J)$

19: **while** $\Psi_{int}^{J} \neq \emptyset$ **do**
20: $J_{int} = \Psi_{int}^{J}[0]$ and $\Psi_{int}^{J} = \Psi_{int}^{J} - J_{int}$
21: interchange J_{int} with J and recalculate C_{max} of Π
22: **if** $C_{max}(\Pi) < C_{max}(\pi)$ **then**
23: $\pi = \Pi$ and return to step 1
24: **else**
25: $\Pi = \pi$
26: **end if**
27: **end while**
28: **end while**
29: **return** $\boldsymbol{\Pi}$

5. COMPUTATIONAL RESULTS

5.1 Experimental instances

To assess the performance of the proposed algorithm, we generated a series of random problem instances. For each job j, the integer processing time p_j was drawn from a uniform distribution in the range [1,10]. Given that the constraints of job size s_j and release time r_j have a significant impact on algorithm performance, we utilized different levels for these constraints to evaluate their effects, as detailed in Table 6.

It is important to note that r_{max} represents the maximum release time, which is determined using the following equation,

$$r_{max} = R * \frac{E(p) * E(s) * n}{B * k} \tag{19}$$

where $E(p)$ and $E(s)$ denote the expected value for processing time and job size, respectively, while the factor R indicates the relative frequency of job arrivals. When $R = 0$, all jobs become available simultaneously at time zero; as R increases, job arrivals are spread over a longer interval. For each combination of problem parameters, five instances were generated for job counts $n = 10,20,50,100,200$. In total, this resulted in 150 problem instances used in our experiments.

Each category of problems is assigned a unique run code. The details regarding the number of machines and processing time of jobs are excluded, as these parameters remain constant. For example, a problem with 20 jobs, where job release times are generated with $R = 0.5$, and job sizes generated from a uniform distribution [4,8], is designated as J20S3R1.

Table 6. Summary of parameters used to generate instances

Factors	Levels
n	$10, 20, 50, 100, 200$
p_j	$U[1,10]$
r_j	$U[0, r_{max}]$
s_j	$U[1,10], U[2,4], U[4,8], R = 0.5, 1.0$
B	10
k	2

5.2 Sensitivity analysis

To evaluate the robustness of the proposed ACS-PBPMs algorithm, we conducted a sensitivity analysis on key parameters, including the pheromone evaporation rate (ρ), heuristic information parameter (β), and ant population size (λ).

For the pheromone evaporation rate ρ, it was tested across values ranging from 0 to 1 in increments of 0.1, and the ultimate value was set to 0.1. This choice effectively balanced the retention of useful information from previous iterations with the exploration of new solutions, thereby stabilizing the search process and preventing premature convergence. The heuristic information parameter (β), which modulates the influence of pheromone trails versus heuristic information, was tested across values from 1 to 6. The experimental results indicated that the ACS-PBPMs performed best for most instances

when $\beta = 4$. For the number of ants λ, we tested several options $(5,15,30, n/2, n, 2n)$ and found $\lambda = 15$ achieved a favorable balance between solution quality and computation time.

Additionally, the initial pheromone τ_0 is another critical factor influencing the exploration and exploitation balance in the search process. A low τ_0 can cause the search to quickly converge on the initial solutions generated by the ants, while a high τ_0 may lead to many iterations being wasted while waiting for pheromone evaporation to sufficiently reduce the values. To balance the searchability of ACS-PBPMs, we set the initial pheromone value to $\tau_0 = 1/LB$, where LB is derived from the employed lower bound in the subsequent subsection.

Regarding the stopping criterion, we observed that solutions showed minimal improvement after 100 iterations, prompting us to set $l_{max} = 100$ as one termination rule for ACS-PBPMs. Additionally, we implemented another termination criterion based on whether the best solution found matched the lower bound, aimed at reducing computational costs.

5.3 Numerical results and comparison analysis

A computational study was conducted to evaluate the efficiency and effectiveness of the proposed algorithms, which were implemented in C++. The main goal of this study was to compare the performance of the proposed algorithms against optimal solutions, with the latter derived from the MILP model discussed in Section 3.

To assess the effectiveness of the proposed algorithms, it is essential to compare the solutions they generate with the optimal values obtained via CPLEX. However, due to the complexity of the problem, CPLEX faced challenges in finding optimal solutions within a reasonable time frame, particularly for larger problem sizes. In pilot experiments, CPLEX was unable to achieve optimal solutions for most of these instances, often terminating after a runtime of 3600 seconds. Given these constraints, a tighter lower bound (LB) revised by Chen *et al.* (2010) was employed as an alternative benchmark to validate the proposed algorithms' effectiveness. Figure 3 shows the comparison of this LB against CPLEX across a series of instances.

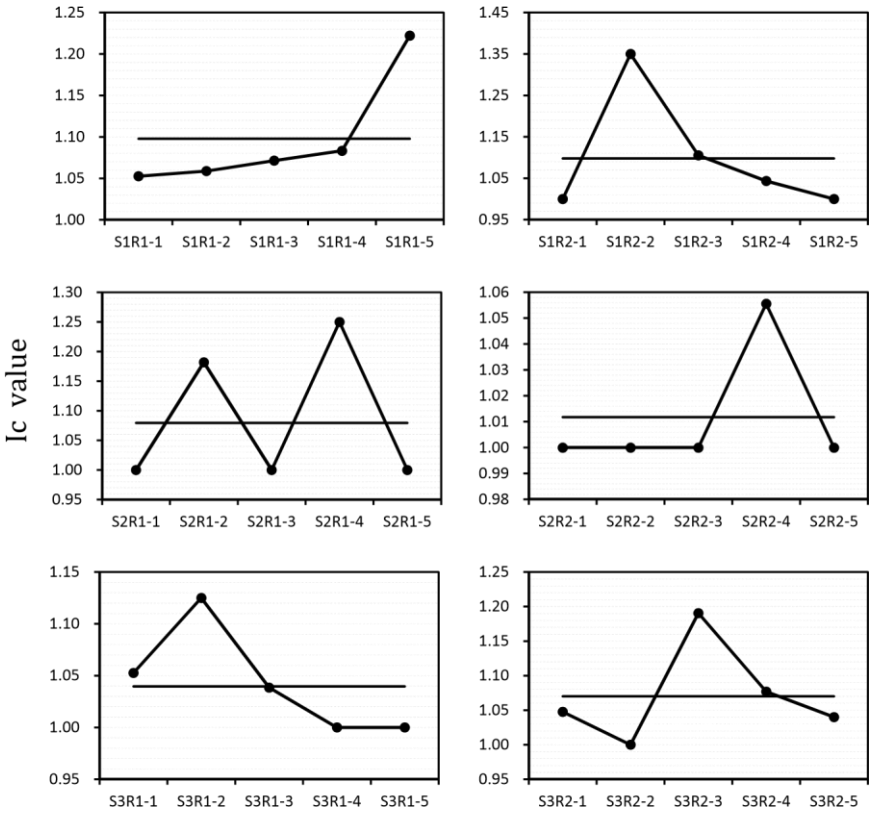


Figure 3. Comparison of CPLEX and LB for small problems with 10 jobs

In Figure 3, each sub-figure illustrates the performance of the lower bound (LB) on five instances for a problem configuration with ten jobs. The symbol l_c represents the ratio calculated as $l_c = CPLEX/LB$, meaning the reported optimal value from CPLEX divided by the LB value. A smaller l_c indicates better quality for the LB. The straight line in each sub-figure denotes the average values of l_c across the five instances with the corresponding problem configuration.

As observed, the LB performs best in the S2R2 problem configuration, even matching the optimal values (where $l_c = 1.0$ in some instances. From the perspective of average values, all average l_c values for each configuration fall within the interval [1.01, 1.10], suggesting that the LB is highly tight for the target problem. Consequently, we employed it to evaluate the effectiveness of our proposed algorithms for other larger problems.

The proposed algorithms were compared against two competitive meta-heuristic algorithms from related research (Beldar *et al.*, 2024; Arroyo *et al.*, 2019). The algorithm that performs best is designated as the benchmark algorithm, referred to as BEST-META in this paper. All algorithms (BE, FMF-WIS, BEST-META, and ACS-PBPMs) were evaluated by measuring the relative gap between the solutions found and the lower bound (LB). The gap percentage Gap^{LB} for algorithm A is defined as,

$$Gap^{LB}(A) = \frac{C^A_{max} - LB}{LB} * 100\% \tag{20}$$

where C^A_{max} denotes the makespan value obtained from algorithm A. A smaller value of $Gap^{LB}(A)$ indicates better performance of algorithm A.

Table 7 presents the average results for LB, BE, FMF-WIS, BEST-META, and ACS-PBPMs across different categories. For each category, five instances were randomly generated, and the average values over these instances are reported. Columns 1 and 2 display the run code and the value of LB for each category. Columns 3 and 4 report the performance of the heuristics BE and FMF-WIS using the relative difference calculated by Equation (20). Columns 5 and 6 show the mean performance and mean runtime of BEST-META, with each instance tested ten times. Similarly, Columns 7 and 8 report the performance and runtime of ACS-PBPMs, also tested ten times.

Since the two heuristics, BE and FMF-WIS, are not iterative or parallel, their computation time is significantly less than that of the intelligence algorithms, even for larger job instances. Therefore, the runtimes for these heuristics are not included in the result tables. Column 9 provides the difference between the algorithms BE and FMF-WIS, calculated as $\Lambda_1 = Gap^{LB}(BE) - Gap^{LB}(FMF - WIS)$, while Column 10 reports the difference between BEST-META and ACS-PBPMs as $\Lambda_2 = Gap^{LB}(BEST - META) - Gap^{LB}(ACS - PBPMs)$. Note that larger values of Λ_1 and Λ_2 indicate better performance for FMF-WIS and ACS-PBPMs, respectively. Figures 4-6 illustrate the average performance of the four algorithms in relation to the number of jobs, job sizes, and release times, with the bars representing the average performance of different algorithms.

As shown in Figure 4 and Table 7, the proposed ACS-PBPMs algorithm outperforms the other algorithms. In contrast, the heuristic BE consistently exhibits the worst performance across all categories. For smaller problems with 20 and 50 jobs, the performance ranking is ACS-PBPMs, BEST-META, FMF-WIS, and BE. Both heuristics demonstrate inferior performance compared to the metaheuristic algorithms. However, for larger problems with 100 and 200 jobs, the ranking shifts to ACS-PBPMs, FMF-WIS, BEST-META, and BE, indicating that FMF-WIS outperforms BEST-META in this context.

Furthermore, as the problem size increases, the average value of Λ_1 rises from 2.01 to 4.56, while Λ_2 increases from 2.95 to 8.55, suggesting that the proposed algorithms deliver superior performance compared to BE and BEST-META. Notably, this improvement is more pronounced for the ACS-PBPMs algorithm. For smaller problems, the performances of ACS-PBPMs and BEST-META are similar due to the limited search spaces.

As problem size increases, the solution spaces expand significantly, often exhibiting exponential growth. This expansion necessitates the exploration of a greater number of potential solutions. In this context, ACS-PBPMs demonstrates superior performance compared to BEST-META, effectively navigating larger search spaces to uncover potentially better solutions. This advantage is largely due to the unique features incorporated into ACS-PBPMs, which enhance its capability to find near-optimal solutions amidst expansive search areas.

Table 7. Comparison of BE, FMF-WIS, BEST-META, and ACS-PBPMS in different categories

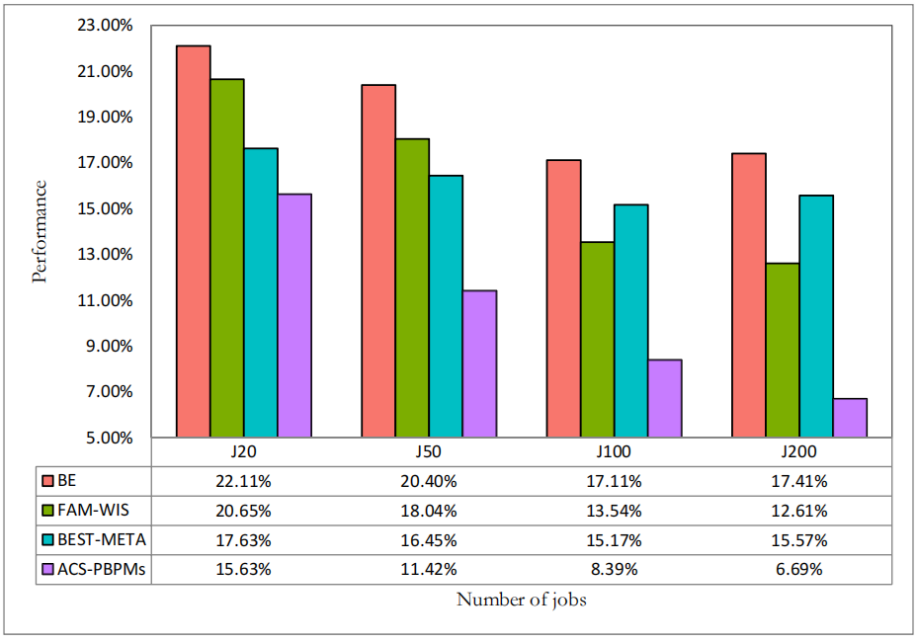
Run Code	LB	BE (%)	FMF-WIS(%)	BEST-META(%)	Time	ACS-PBPMs(%)	Time	Λ_1(%)	Λ_2(%)
J20S1R1	34.20	20.47	21.05	18.13	4.64	18.13	3.51	-0.58	0.00
J20S1R2	37.20	25.81	26.88	18.28	3.93	16.67	3.59	-1.08	1.61
J20S2R1	20.60	31.07	32.04	28.16	2.67	23.30	3.45	-0.97	4.85
J20S2R2	25.80	33.33	16.28	24.03	2.81	13.18	3.47	17.05	10.85
J20S3R1	41.00	9.27	10.73	8.29	4.03	8.78	3.58	-1.46	-0.49
J20S3R2	44.60	12.11	13.00	10.31	3.76	9.42	3.46	-0.90	0.90
Average	33.90	22.01	20.00	17.87	3.64	14.91	3.51	2.01	2.95
J50S1R1	73.20	13.11	15.57	12.02	9.10	9.84	23.07	-2.46	2.19
J50S1R2	84.20	29.22	28.98	25.42	9.38	20.67	21.80	0.24	4.75
J50S2R1	43.20	25.00	23.15	22.22	6.00	9.72	22.41	1.85	12.50
J50S2R2	50.20	32.67	14.74	21.91	6.12	7.97	20.22	17.93	13.94
J50S3R1	90.80	9.25	9.69	7.93	11.13	6.83	25.32	-0.44	1.10
J50S3R2	107.60	11.90	13.01	9.85	10.13	8.74	26.53	-1.12	1.12
Average	74.87	20.19	17.52	16.56	8.64	10.63	23.22	2.67	5.93
J100S1R1	157.00	9.43	11.46	8.28	30.48	8.03	106.11	-2.04	0.25
J100S1R2	163.00	24.05	18.16	24.23	32.76	15.46	108.83	5.89	8.77
J100S2R1	87.60	22.83	19.86	19.41	16.03	6.85	107.51	2.97	12.56
J100S2R2	97.40	29.16	13.76	25.26	15.38	5.34	95.18	15.40	19.92
J100S3R1	189.00	5.40	6.56	5.08	35.66	4.13	129.77	-1.16	0.95
J100S3R2	196.40	11.20	10.79	10.29	34.22	7.94	131.72	0.41	2.34
Average	148.40	17.01	13.43	15.42	27.42	7.96	113.19	3.58	7.47
J200S1R1	302.00	8.74	10.33	8.21	84.67	7.28	648.69	-1.59	0.93
J200S1R2	322.75	23.63	15.96	23.01	72.80	13.59	674.76	7.67	9.42
J200S2R1	168.20	23.31	19.86	20.69	57.85	6.06	632.39	3.45	14.63
J200S2R2	190.00	34.47	15.00	27.89	53.15	4.53	580.11	19.47	23.37
J200S3R1	394.20	3.20	4.16	3.20	91.48	3.15	825.40	-0.96	0.05
J200S3R2	395.50	8.22	8.91	7.46	94.96	4.58	807.22	-0.70	2.88
Average	295.44	16.93	12.37	15.08	75.82	6.53	694.76	4.56	8.55

Figure 5 and Table 7 illustrate that for problems with large job sizes (S3), there is minimal performance difference among the four algorithms. In the S3 category, 40% of jobs are required to fit into exactly one batch (those with sizes 7 or 8), while the remaining 60% must be efficiently assigned to batches. Since a majority of individual jobs need to be allocated to a single batch, the feasible search space is considerably smaller than that of the S1 and S2 problems. Consequently, this diminishes the performance disparities among the four algorithms. However, although the number of jobs per batch is lower, the mechanism of the proposed candidate list strategy must evaluate more constrained options, increasing the effort to find optimal solutions. Additionally, the pheromone update mechanism iteratively accumulates sufficient pheromone data to guide the search, further increasing computational effort. These algorithmic components, combined with the inherent complexity of the problem, result in longer computational times.

In contrast, for problems with small job sizes (S2), the performance differences among the four algorithms are more pronounced. Specifically, FMF-WIS and ACS-PBPMs deliver superior results compared to BE and BEST-META. Similar to the outcomes in larger problems, FMF-WIS outperforms BEST-META in this category as well. In S2, since each batch can accommodate at least two jobs, the feasible search space is significantly larger than in the other cases. The strong performance of ACS-PBPMs in this context further underscores its effectiveness in finding solutions within extensive solution spaces, leveraging the problem-specific knowledge integrated into its design.

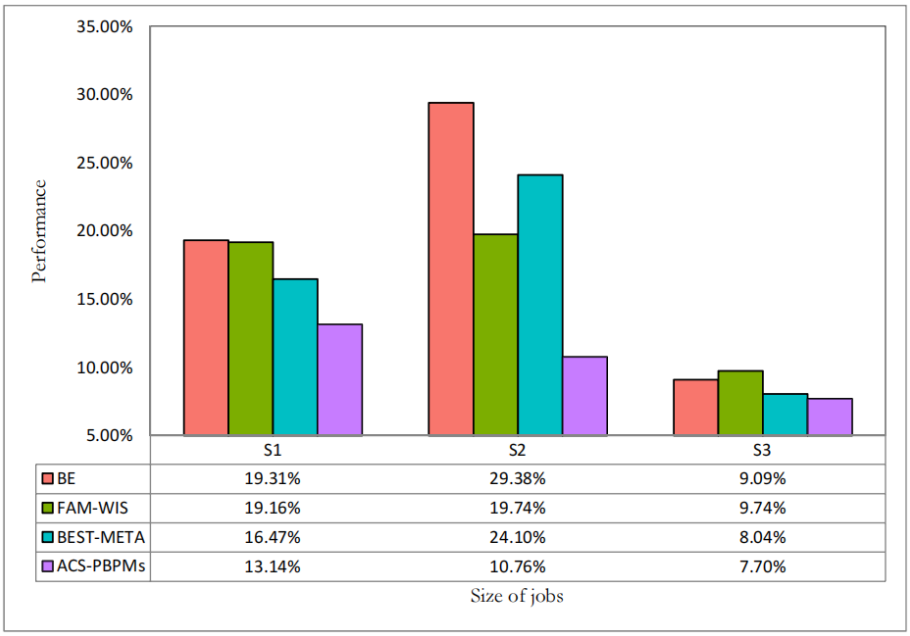
As illustrated in Figure 6 and Table 7, the performance difference among the four algorithms is more pronounced in the R2 category compared to R1. In the R2 category, job release times are generated over a longer interval, meaning jobs arrive less frequently than in R1. Notably, the performance of BE and BEST-META is relatively unsatisfactory in this category compared to R1. This phenomenon can be attributed to the fact that both BE and BEST-META utilize a best-fit rule for batch

formation, which does not take release times into account. When jobs arrive frequently, the impact of release time is minimal; however, with longer intervals, this oversight can lead to significantly poorer performance in certain instances.



	J20	J50	J100	J200
BE	22.11%	20.40%	17.11%	17.41%
FAM-WIS	20.65%	18.04%	13.54%	12.61%
BEST-META	17.63%	16.45%	15.17%	15.57%
ACS-PBPMs	15.63%	11.42%	8.39%	6.69%

Figure 4. Performance of four algorithms for different numbers of jobs



	S1	S2	S3
BE	19.31%	29.38%	9.09%
FAM-WIS	19.16%	19.74%	9.74%
BEST-META	16.47%	24.10%	8.04%
ACS-PBPMs	13.14%	10.76%	7.70%

Figure 5. Performance of four algorithms for different sizes of jobs

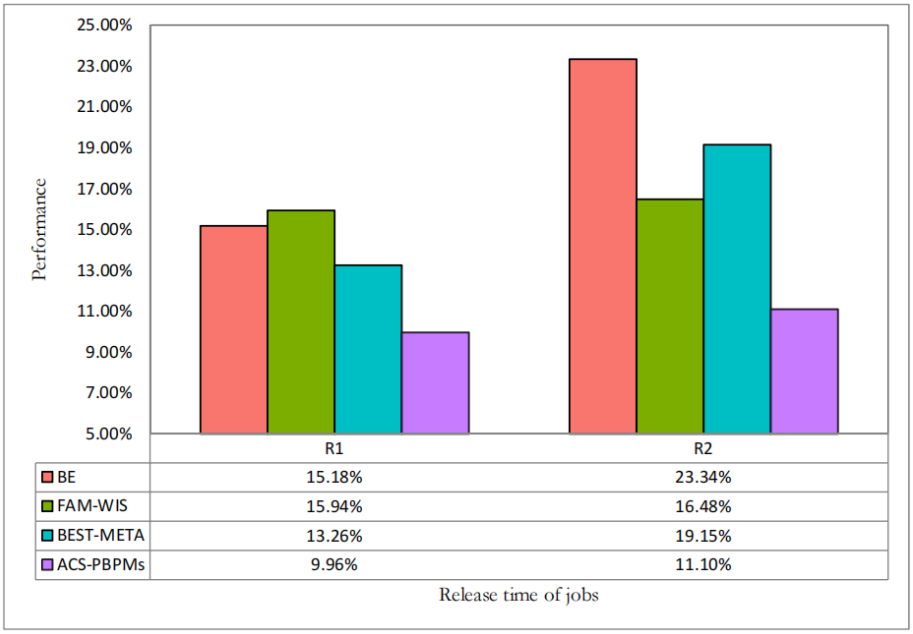


Figure 6. Performance of four algorithms for different release times of jobs

In contrast, FMF-WIS and ACS-PBPMs exhibit superior performance regarding release times. This is largely due to their use of the Weighted Insertion Strategy (WIS) for batch formation, which considers both release time and job size constraints. Additionally, the candidate selection strategy in ACS-PBPMs is specifically designed to accommodate release time constraints, enhancing its effectiveness in addressing R2 problems.

Regarding computational cost, ACS-PBPMs generally require less time than BEST-META for most instances within the J20 problem category. However, as the number of jobs increases, the computational cost of ACS-PBPMs can surpass that of BEST-META. This increase is primarily due to the differing encoding mechanisms employed by the two algorithms. Despite the higher computational cost, the unique encoding strategies and search techniques of ACS-PBPMs enable it to maintain reasonable computational times while delivering high-quality solutions.

The superior performance of the ACS-PBPMs algorithm can be attributed to three key factors. First, it integrates batch formation and sequencing decisions within a unified framework, addressing the interdependencies often neglected by traditional methods. Second, the candidate list strategy dynamically balances machine utilization and job waiting times, enabling efficient decision-making even under complex constraints. Third, the algorithm incorporates a tailored local search procedure that refines solutions by reducing machine idle times and optimizing batch configurations. These enhancements significantly improve both solution quality and computational efficiency. While ACS-PBPMs perform well, it has potential limitations. Its computational cost increases with problem size, making scalability for very large-scale problems challenging. Moreover, its performance relies on problem-specific knowledge, which may reduce flexibility for other applications.

6. CONCLUSIONS

In this paper, we address critical scheduling challenges in semiconductor manufacturing, focusing on optimizing batch processing to enhance efficiency and sustainability. Our work has the potential to impact broader semiconductor operations, enabling scalable, cost-effective, and environmentally sustainable practices to meet growing global demand. Specifically, we tackle the scheduling problem at burn-in workstations, often bottlenecks in final testing facilities, by proposing an integrated approach that improves machine utilization and reduces makespan. We model this challenge as a scheduling problem on multiple PBPMs with non-identical job sizes and dynamic arrivals. We develop a constructive heuristic and an ACS-PBPMs algorithm that utilizes problem-specific knowledge. To enhance the effectiveness of the ACS-PBPMs algorithm, we introduce a candidate list mechanism that accounts for two key problem constraints, thereby reducing the search space. Furthermore, we implement a local search procedure tailored to the unique characteristics of the solution, further improving the performance of ACS-PBPMs. Our proposed approaches are benchmarked against CPLEX and two competitive algorithms to

evaluate their effectiveness. Experimental results demonstrate that the ACS-PBPMs algorithm outperforms the others, particularly for problems with extensive search spaces.

While this study provides a robust framework for semiconductor scheduling, future research could explore several specific extensions. First, the model could be adapted to accommodate unrelated parallel machines with varying capacities and speeds, capturing the complexity of diverse manufacturing environments. Second, incorporating sequence-dependent setup times and incompatible job families would address practical challenges in semiconductor production. Third, developing real-time batch processing algorithms to handle unpredictable job arrivals and machine breakdowns could significantly enhance responsiveness. These extensions would strengthen the framework's relevance to complex, real-world scenarios, bridging the gap between theoretical optimization and practical implementation in semiconductor manufacturing.

ACKNOWLEDGEMENT

This research was supported by the Fundamental Research Funds for the Central Universities (Grant No. B240207057,423180), Guangdong Provincial Key Laboratory (Grant No.2020B121201001), the National Natural Science Foundation of China (Grant No.62106098/62272210).

REFERENCES

Arroyo, J. E. C., Leung, J. Y.-T., & Tavares, R. G. (2019). An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times. *Engineering Applications of Artificial Intelligence*, *77*, 239–254.

Bektur, G. (2022). Distributed flow shop scheduling problem with learning effect, setups, non-identical factories, and eligibility constraints. *International Journal of Industrial Engineering: Theory, Applications and Practice*, *29*(1).

Beldar, P., Battarra, M., & Laporte, G. (2024). Non-identical parallel machines batch processing problem to minimize the makespan: Models and algorithms. *Computers & Operations Research*, 106708.

Blum, C. (2024). Ant colony optimization: A bibliometric review. *Physics of Life Reviews*, *51*, 87–95.

Chang, P. Y., Damodaran, P., & Melouk, S. (2004). Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. *International Journal of Production Research*, *42*(3), 11–20.

Chen, H., Du, B., & Huang, G. Q. (2010). Metaheuristics to minimise makespan on parallel batch processing machines with dynamic job arrivals. *International Journal of Computer Integrated Manufacturing*, *23*(10), 942–956.

Durasevic, M., & Jakobovic, D. (2023). Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: A survey. *Artificial Intelligence Review*, *56*(4), 3181–3289.

Fowler, J. W., & Mönch, L. (2022). A survey of scheduling with parallel batch (p-batch) processing. *European Journal of Operational Research*, *298*(1), 1–24.

Gahm, C., Wahl, S., & Tuma, A. (2022). Scheduling Parallel Serial-batch Processing Machines with Incompatible Job Families, Sequence-dependent Setup Times and Arbitrary Sizes. *International Journal of Production Research*, *60*(17), 5131–5154.

Gonçalves Rodrigues, C., Elmo de Menezes, J., & Lajovic Carneiro, M. (2022). Global market overview of semiconductor industry: Visão geral do mercado mundial da indústria de semicondutores. *RGC*, *16*, 490–497.

Jia, Z. H., Yan, J. H., Leung, J. Y. T., Li, K., & Chen, H. P. (2019). Ant Colony Optimization Algorithm for Scheduling Jobs with Fuzzy Processing Time on Parallel Batch Machines with Different Capacities. *Applied Soft Computing*, *75*, 548–561.

Lee, C.-Y., Uzsoy, R., & Martin-Vega, L. A. (1992). Efficient Algorithms for Scheduling Semiconductor Burn-In Operations. *Operations Research*, *40*(4), 764–775.

Liu, J., Anavatti, S., Garratt, M., & Abbass, H. A. (2022). Modified continuous ant colony optimisation for multiple unmanned ground vehicle path planning. *Expert Systems with Applications*, *196*, 116605.

Muter, İ. (2020). Exact algorithms to minimize makespan on single and parallel batch processing machines. *European Journal of Operational Research*, *285*(2), 470–483.

Nguyen, A. H., & Sheen, G. J. (2023). A decomposition-based heuristic algorithm for parallel batch processing problem with time window constraint. *International Journal of Industrial Engineering: Theory, Applications and Practice*, *30*(2).

Park, H., Shin, Y., & Moon, I. (2024). Multiple-objective scheduling for batch process systems using stochastic utility evaluation. *International Journal of Industrial Engineering: Theory, Applications and Practice*, *31*(2), 413–428.

Saber, M., Abdelhamid, A., & Ibrahim, A. (2023). Metaheuristic Optimization Review: Algorithms and Applications. *Journal of Artificial Intelligence and Metaheuristics*, *3*(1), 21-30.

Saemi, S., Komijan, A. R., Tavakkoli-Moghaddam, R., & Fallah, M. (2022). Solving an integrated mathematical model for crew pairing and rostering problems by an ant colony optimization algorithm. *European Journal of Industrial Engineering*, *16*(2), 215–240.

Uzsoy, R. (1994). Scheduling a single batch processing machine with nonidentical job sizes. *International Journal of Production Research*, *32*(7), 1615–1635.

Uzsoy, R., & Yang, Y. (1997). Minimizing Total Weighted Completion Time on a Single Batch Processing Machine. *Production and Operations Management*, *6*(1), 57–73.

Wang, J., Tang, H., & Lei, D. (2024). A feedback-based artificial bee colony algorithm for energy-efficient flexible flow shop scheduling problem with batch processing machines. *Applied Soft Computing*, *153*, 111254.

Wang, Q., Huang, N., Chen, Z., Chen, X., Cai, H., & Wu, Y. (2023). Environmental data and facts in the semiconductor manufacturing industry: An unexpected high water and energy consumption situation. *Water Cycle*, *4*, 47–54.

Wu, W., Hayashi, T., Haruyasu, K., & Tang, L. (2023). Exact algorithms based on a constrained shortest path model for robust serial-batch and parallel-batch scheduling problems. *European Journal of Operational Research*, *307*(1), 82-102.

Xiao, X., Ji, B., Yu, S. S., & Wu, G. (2024). A tabu-based adaptive large neighborhood search for scheduling unrelated parallel batch processing machines with non-identical job sizes and dynamic job arrivals. *Flexible Services and Manufacturing Journal*, *36*(2), 409–452.

Zhang, H., Yang, Y., & Wu, F. (2022b). Just-in-time single-batch-processing machine scheduling. *Computers & Operations Research*, *140*, 105675.

Zhang, X., Shan, M., & Zeng, J. (2022). Parallel Batch Processing Machine Scheduling Under Two-Dimensional Bin-Packing Constraints. *IEEE Transactions on Reliability*, *72*(3), 1265-1275.

Zhou, S., Jia, Z., Jin, M., & Du, N. (2022). Minimizing makespan on parallel batch processing machines with two-dimensional rectangular jobs. *Computers & Industrial Engineering*, *167*, 108167.

Zhou, S., Jin, M., & Du, N. (2020). Energy-efficient scheduling of a single batch processing machine with dynamic job arrival times. *Energy*, *209*, 118420.

Zhou, S., Xing, L., Zheng, X., Du, N., Wang, L., & Zhang, Q. (2021). A Self-Adaptive Differential Evolution Algorithm for Scheduling a Single Batch-Processing Machine with Arbitrary Job Sizes and Release Times. *IEEE Transactions on Cybernetics*, *51*(3), 1430–1442.