Journal of Industrial and Management Optimization

https://www.aimspress.com/journal/jimo

JIMO, 22(4): 1609−1628.
DOI: 10.3934/jimo.2026059
Received: 03 January 2026
Revised: 19 February 2026
Accepted: 27 February 2026
Published: 09 March 2026

Research article

Algorithms for a 2-stage hybrid flow shop scheduling problem with two classes of jobs

Ju-Yong Lee[1] and BongJoo Jeong[2,*]

[1] Division of Business Administration & Accounting, Kangwon National University Kangwondaehak-gil, Chuncheon-si, Gangwon-do 24341, Republic of Korea
[2] Division of Business Administration & Accounting, Kongju National University 56, Gongjudaehak-ro, Gongju-si, Chungcheongnam-do 32588, Republic of Korea

* **Correspondence:** Email: jbj@kongju.ac.kr; Tel: +82-041-850-8432; Fax: +82-041-850-8429.

Abstract: This study addresses the complex 2-stage hybrid flow shop (HFS) scheduling problem, which involves urgent and normal jobs in a multi-agent context. The goal is to simultaneously minimize the makespan for normal jobs and the total tardiness for urgent jobs. Our proposed approaches include a Mixed-Integer Programming (MIP) model, list scheduling algorithms (MEDD, MEDD2), constructive algorithms (MNEH, MN-NEH, MFL, MN-MFL), and a simulated annealing (SA) metaheuristic to find practical solutions. Extensive experiments, using randomly generated instances which reflect real manufacturing environments, validate the algorithms' performance. The SA algorithm consistently exhibits a superior and robust performance across all scenarios, finding solutions remarkably close to the optimal. Additionally, MFL and MN-MFL show noteworthy efficiencies and stable performances, indicating their potential for practical applications in rapid decision-making environments.

Keywords: hybrid flow shop; multi-agent scheduling; urgent jobs; heuristic; simulated annealing
Mathematics Subject Classification: 90B35, 90C59, 90C29

1. Introduction

In today’s dynamic manufacturing environment, many manufacturing systems utilize hybrid flow shops (HFS) to produce various products through a series of processes, each of which is equipped with multiple machines. HFS represents a strategic approach to simultaneously achieve high productivity and flexibility in multi-variety, low-volume production settings. It combines the characteristics of both a flow shop and a parallel-machine system, given its multiple stages and multiple machines. Typical applications of HFS are found in industries such as textiles, chemicals, paper, and steel [1]. HFS presents unique challenges that necessitate innovative solutions to optimize the production efficiency. Optimizing such systems demands careful consideration of the job assignment, sequencing, and timing. Thus, efficient scheduling of production processes has become a critical factor to ensure operational competitiveness and customer satisfaction.

As the marketplace becomes more competitive, customer order management and rapid demand response capabilities are critical competitive factors. To address this environment, this study investigates HFS that involve two classes of jobs. Herein, jobs are divided into two distinct categories based on the urgency of customer orders: a normal class and an urgent class. Urgent jobs, by definition, require prompt processing more than normal jobs, thus reflecting the demand for rapid execution. Consequently, they generally have very tight due dates. Such urgent jobs are prevalent in many make-to-order manufacturing systems. A prime application of this study is found in semiconductor wafer fabrication, where jobs that require immediate processing are industrially referred to as 'hot lots' (e.g., high-priority prototypes or defect replacements). We adopt the generalized term 'urgent jobs' throughout this paper. These urgent jobs often account for 30–50% of the total production [2] and dynamically arrive at irregular intervals (non-zero release times), thus disrupting the pre-planned schedule of normal jobs. This environment forces shop floor managers into a critical trade-off: interrupting the continuous flow of normal jobs (thereby increasing makespan) versus delaying high-priority jobs. Consequently, this real-world scenario necessitates a multi-agent scheduling approach that explicitly models dynamic job arrivals ($r_i \geq 0$) and balances the conflicting goals of rapid delivery and throughput maximization via a weighted objective function. Multi-agent scheduling is a method where different job classes compete for limited resources to satisfy their own objectives. It focuses on balancing conflicting goals, such as meeting urgent deadlines while maintaining the overall efficiency of normal jobs. Therefore, effectively considering these urgent jobs is crucial to generate efficient schedules.

The consideration of urgent jobs further complicates the scheduling problems, thereby requiring the development of advanced algorithms that can effectively balance normal and urgent jobs. In general, normal jobs are scheduled to minimize the maximum completion time (makespan) to raise the throughput rate. However, urgent jobs are required to start processing as early as possible when they arrive at the workstations. To achieve this, each urgent job has a hypothetical due date, assuming that it is processed through all steps without immediately waiting upon arrival, that is, the due date of each urgent job is defined as the sum of its release time and total processing time. Hence, for urgent jobs, the objective of minimizing total tardiness can be considered as the scheduling measure. Due to these characteristics, balancing the needs of normal jobs and urgent jobs presents a complex trade-off between productivity and expediting urgent jobs. This is because if we only prioritize urgent jobs, then the overall productivity is likely to suffer. Therefore, this study considers the makespan for normal jobs and the total tardiness of urgent jobs as the scheduling measure. Since the general HFS scheduling problem is strictly nondeterministic polynomial-time hard (NP-hard) for all common scheduling measures [3], it is also NP-hard to minimize the makespan for normal jobs and the total tardiness of urgent jobs in HFS, which are considered in this study.

The HFS, which processes jobs of two classes depending on their urgency, can be classified as a multi-objective scheduling problem because it combines two types of measures (i.e., the makespan for normal jobs and the total tardiness for urgent jobs). Multi-objective scheduling problems have gained popularity as manufacturing environments become more complex, and metaheuristics are exclusively applied due to the NP-hard nature of the problems [1]. Liang et al. [4] investigated a flow shop with limited buffers between stages and introduced a self-adaptive differential evolution algorithm to minimize both the maximum tardiness and the makespan. Anjana et al. [5] addressed a flow shop with sequence-dependent setup times and presented four metaheuristics to simultaneously minimize the makespan and mean tardiness. Rifai et al. [6] studied a distributed reentrant permutation flow shop with sequence-dependent setup times to simultaneously minimize the makespan, tardiness, and production cost. Recently, multi-objective flow shop problems derived from energy consumption issues have been considered [7–9]. Furthermore, Yılmaz et al. [10] addressed a multi-objective HFS problem with a limited waiting time and machine capability constraints, threreby proposing Non-dominated Sorting Genetic Algorithm II (NSGA-II) based algorithms to minimize the makespan, average flow time, and workload imbalance. Most recently, Liu et al. [11] developed a hybrid NSGA-II algorithm for the multi-objective flexible flow shop scheduling problem to simultaneously optimize the makespan, customer satisfaction related to processing delays, and electricity costs. Expanding on these multifaceted goals, Jia et al. [12] addressed a 2-stage HFS with parallel batch machines to simultaneously optimize the productivity and energy efficiency, thus reflecting the growing importance of green manufacturing. Furthermore, Peng et al. [13] addressed the complexities of reentrant HFS by optimizing both the makespan and rescheduling stability within a multi-objective Pareto-based framework, utilizing a hybrid of meta-heuristics and Q-learning. A recent review on multi-objective problems in HFS was given in Neufeld et al [1].

However, the multi-objective considered in this study differs from conventional multi-objective problems. In typical multi-objective optimization, research often focuses on developing Multi-Criteria Decision-Making (MCDM) frameworks or smart algorithms to balance conflicting objectives for all tasks [14,15]. In contrast, this study considers a problem where each job class is governed by its own distinct objective function. This implies that instead of each job having multiple objectives to be simultaneously balanced, every job possesses a single, dedicated objective specifically determined by its assigned class. This is in contrast to typical multi-objective problems, where all jobs have multiple objectives. Due to this structural difference, conventional MCDM techniques for weighting cannot be directly applied to our model in the same manner. Consequently, the current problem is similar to a multi-agent problem, which is a specialized category within multi-objective problems [16]. In multi-agent problems, multiple agents are involved, and each agent manages jobs with a different objective function. Hence, assuming that jobs belonging to classes with different objectives are only handled by dedicated agents, this scenario is a multi-agent problem.

In the literature, most studies on the multi-agent problem have considered scheduling on a single machine. Baker and Cole Smith [17] examined the implications of minimizing an aggregate objective function of the makespan, total weighted completion time, and maximum lateness, and they demonstrated that it is NP-hard. Agnetis et al. [16] proposed algorithms to find nondominated schedules, such that a better schedule for one agent necessarily results in a worse schedule for the other agent. Ng et al. [18] studied a scheduling problem to minimize the total completion time of one agent with an upper limit on the number of tardy jobs of the other agent, and this problem was proven to be NP-hard even when not allowing jobs to get delayed [19]. Cheng et al. [20] focused on the feasibility model of minimizing the total weighted number of tardy jobs for each agent, and demonstrated that it is strongly NP-complete. However, they also showed that if the number of agents is fixed, then the problem can be solved in pseudo-polynomial time for integral weights and polynomial time for unit weights. Additionally, they (Cheng et al. [21]) studied a similar problem, where the objective functions of the agents were of the max-form. Liu and Tang [22] dealt with a two-agent problem where the processing time of each job was linearly deteriorated. In addition, Lee et al. [23] and Wan et al. [24] developed polynomial-time algorithms for single-machine problems with two agents.

In the field of flow shop studies with a multi-agent, Agnetis et al. [16] also studied minimizing the makespan of each agent for a flow shop. Lee et al. [25] developed a branch and bound algorithm and simulated annealing algorithms to minimize the total tardiness of the first agent, while the number of tardy jobs of the second agent was limited to zero. Lee et al. [26] used the same methodologies for a similar problem, with the difference that the first agent's objective was to minimize the total completion time. Mor and Mosheiov [27] proposed polynomial-time algorithms for three problems with different objective functions of the first agent: minimum maximum cost of all the jobs, minimum total completion time, and minimum number of tardy jobs, while the maximum allowable cost of the second agent was limited. Jeong and Shim [2] dealt with a re-entrant flow shop with two agents and proposed a lower bound scheme and several heuristic algorithms. Jeong et al. [28] introduced a two-machine flow shop problem with urgent jobs, and they proposed a branch and bound algorithm that included dominance properties, lower bound schemes, and heuristic algorithms. Jeong et al. [29] provided an iterated greedy algorithm and a simulated annealing for a flow shop with urgent jobs and limited waiting constraints. For hybrid flow shops, Lei and Guo [30] developed a novel shuffled frog-leaping algorithm to minimize the sum of the total tardiness of two agents. Additionally, Lei [31] proposed a two-phase neighborhood search algorithm to minimize the makespan of the first agent and the total tardiness of the second agent. Although Lei [31] hared the objectives of minimizing makespan and total tardiness in a multi-agent setting, it fundamentally differed in its optimization approach. Their method saught a set of Pareto-optimal solutions, which compelled shop floor managers to perform a secondary analysis to select one schedule among many alternatives. In contrast, our study adopts a weighted sum objective to facilitate rapid decision-making, which is critical to handle time-sensitive urgent jobs. By incorporating managerial priorities through pre-defined weights, our approach immediately yields a single, execution-ready schedule, thereby eliminating the decision latency associated with Pareto-based selection. Furthermore, their model assumed a static environment where all jobs were available at time zero (r_i=0). This assumption simplified the scheduling process by eliminating the need to manage machine idle times for incoming orders. In contrast, our study addresses a significantly more complex 'generalization' of the problem by incorporating arbitrary release times ($r_i \geq 0$) for urgent jobs. This dynamic characteristic forces the scheduler to make difficult trade-offs: deciding whether to keep a machine idle to accommodate an incoming urgent job (to minimize its tardiness) or to process a currently available normal job (to minimize makespan). This structural difference makes our problem far more challenging and closer to real-world manufacturing constraints than the static cases found in the literature. Therefore, Lei [31] cannot be viewed as a study that modeled urgent jobs, and it is difficult to directly apply their algorithm to problems that explicitly reflect urgent jobs. In this regard, our study has a clear academic contribution as the first attempt at scheduling a HFS that considers urgent jobs. The differences between this study and Lei's study are summarized in Table 1.

The primary objective of this study is to develop a novel scheduling algorithm that effectively addresses the HFS with jobs of two classes: normal jobs and urgent jobs. The algorithm is designed to achieve the following goals:

1) minimizing the makespan for normal jobs, thus ensuring efficient use of resources and a consistent production flow; and
2) minimizing the total tardiness of urgent jobs, thereby catering to time-sensitive demands, and enhancing customer satisfaction.

The results of this research will have a significant impact on manufacturing companies looking to improve their production scheduling processes. By providing effective and efficient scheduling algorithms that balance the optimization goals of normal and urgent jobs, this research will contribute to an improved production efficiency, responsiveness, and customer service.

Table 1. Comparison between this study and related multi-agent HFS studies.

Feature	Lei [31] (Existing)	This study (Proposed)
Job availability	All jobs available at t=0 (Static)	Dynamic release times ($r_i \geq 0$)
Objective function	Pareto Optimization (Multi-objective)	Weighted sum (Single-objective)
Output type	Set of non-dominated solutions	Single, execution-ready schedule
Focus	Exploring trade-off frontiers	Rapid decision-making for operations

2. Problem description

In this section, we describe the problem considered here with a mathematical model. The addressed problem involves n jobs (i=1, ..., n) in a 2-stage hybrid flow shop. Each job is categorized into one of two distinct classes: class-A or class-B. Here, class-A signifies the urgent jobs, while class-B refers to the normal (non-urgent) jobs. The classes of urgent and normal jobs are represented by J_A and J_B, respectively. In this research, that the following are assumed:

1) Once a job starts processing, it runs to completion without interruption;
2) Machines are reliable (no breakdowns);
3) All normal jobs are available at time 0;
4) Urgent jobs have known, non-negative release times; and
5) The due date for an urgent job i is established as the sum of its release time (r_i) and processing times on stage 1 (p_{i1}) and stage 2 (p_{i2}), denoting the earliest feasible completion time for the job i on the second stage as the aggregate of these times (r_i+p_{i1}+p_{i2}).

A schedule is feasible if it respects the assumptions above and satisfies the following:

1) Every job is assigned exactly once at Stage 1 and exactly once at Stage 2 to a specific machine and position. Processing on any machine must be non-preemptive, meaning that no time overlap occurs.
2) Jobs are ordered on each machine as a left-shift schedule (i.e., a contiguous (gap-free) sequence of position indices). *Note:* A left-shift schedule is a sequence where each job starts at its earliest possible time without altering the processing order. It eliminates avoidable gaps, though idle time may still occur due to the release times or precedence constraints.
3) The schedule must respect the release times (for urgent jobs) and stage precedence (a job's Stage 2 processing can only start after its Stage 1 completion).
4) Time consistency is maintained based on the machine order (earlier completion times govern subsequent job starts and completions).

To describe the problem more clearly, we use the following notations.

Parameters

α	weight for the total tardiness of urgent jobs (weight for the maximum completion time of normal jobs is $1-\alpha$)
M_s	number of identical parallel machines at stage s (s =1, 2)
p_{is}	processing time of job i in stage s (i = 1, …, n)

r_i	release time of job i ($i = 1, \dots, n$)
d_i	due date of job i ($d_i = r_i + p_{i1} + p_{i2}$ if job i is an urgent job, and d_i is a large number if job i is a normal job)
y_i	= 1 if job i is a normal job, otherwise(urgent job) 0
M	Big M

Decision Variables

Z_{ijks}	= 1 if job i is in the k-th position on j-th machine at s-th stage, otherwise 0.
S_{i2}	start time of the job i at stage 2
C_{jks}	completion time of k-th positioned job on a j-th machine in stage s (k=1,…n, j=1,…, M_s)
C_{max}	maximum completion time of normal jobs; i.e., makespan
τ_k	tardiness of job i (=0 if job i is a normal job, otherwise (urgent job) max{0, $C_{i2}-d_i$}

With the aforementioned notations, a mathematical formulation is given as follows.

Objective function:

$$\text{Min } \alpha \sum_{i=1}^{n} \tau_i + (1-\alpha) C_{\max} \tag{1}$$

Constraints:

$$\sum_{k=1}^{n} \sum_{j=1}^{M_s} Z_{ijks} = 1 \qquad \forall i = 1, \dots, n;\ \forall s = 1, 2 \tag{2}$$

$$\sum_{i=1}^{n} Z_{ijks} \le 1 \qquad \forall k = 1, \dots, n;\ \forall j = 1, \dots, M_s;\ \forall s = 1, 2 \tag{3}$$

$$\sum_{k=1}^{n} \sum_{i=1}^{n} Z_{ijks} \le n \qquad \forall j = 1, \dots, M_S;\ \forall s = 1, 2 \tag{4}$$

$$\sum_{i=1}^{n} Z_{ijks} \le \sum_{i=1}^{n} Z_{ij(k-1)s} \qquad \forall k = 2, \dots, n;\ \forall j = 1, \dots, M_s;\ \forall s = 1, 2 \tag{5}$$

$$C_{j(k-1)s} + \sum_{i=1}^{n} p_{is} Z_{ijks} \le C_{jks} \qquad \forall k = 2, \dots, n;\ \forall j = 1, \dots, M_s;\ \forall s = 1, 2 \tag{6}$$

$$\sum_{i=1}^{n} (r_i + p_{i1}) Z_{ijk1} \le C_{jk1} \qquad \forall k = 1, \dots, n;\ \forall j = 1, \dots, M_s \tag{7}$$

$$S_{i2} \ge C_{jk1} - \left(1 - Z_{ijk1}\right) M \qquad \forall i = 1, \dots, n, \forall j = 1, \dots, M_1;\ \forall k = 1, \dots, n \tag{8}$$

$$C_{jk2} \ge S_{i2} + p_{i2} - \left(1 - Z_{ijk2}\right) M \qquad \forall i = 1, \dots, n, \forall j = 1, \dots, M_2;\ \forall k = 1, \dots, n \tag{9}$$

$$\tau_i \ge C_{jk2} - d_i (1 - y_i) - (1 + y_i - Z_{ijk2}) M \qquad \forall i = 1, \dots, n, \forall j = 1, \dots, M_2;\ \forall k = 1, \dots, n \tag{10}$$

$$C_{\max} \ge C_{jk2} - \left(1 - \sum_{i=1}^{n} y_i Z_{ijk2}\right) M \qquad \forall j = 1, \dots, M_2;\ \forall k = 1, \dots, n \tag{11}$$

The objective function of this study, as presented in (1), is designed to minimize the weighted sum of the tardiness of all jobs and the maximum completion time of normal jobs. Note that normal jobs are definitely non-tardy because their due dates are set to a very large value. Equation (2) ensures that every job is scheduled exactly once at each stage, meaning that each job will be assigned to only one machine and position in the sequence at both stages of the process. Equation (3) ensures that each position on a machine, at any given time point within each stage, is occupied by at most one job. This implies that two jobs cannot be simultaneously scheduled at the same position on the same machine. Equation (4) ensures that the maximum number of jobs assigned to any machine at each stage does not exceed the total number of jobs n, thus ensuring no machine is allocated more than n jobs. Equation (5) represents that for a job to be assigned to the k-th position on any machine at any stage, there must be a job assigned to the (k−1)-th position on that machine at that stage. This ensures a sequential filling of positions on each machine. In our formulation, Equations (4) and (5) are not required for the mixed-integer programming (MIP) to be valid. We include them to strengthen the model and reduce the central processing unit (CPU) time, which leads to a faster solution performance. Although Equation (4) can be derived from Equation (3) and, given the objective, the model can be solved without Equation (5), including these constraints significantly reduces the CPU time. Equation (6) defines the completion time of a job at any position on any machine at any stage after the first. It ensures that the completion time of a job accounts for the sum of the completion time of the preceding job in the sequence and its own processing time. Equation (7) signifies that the completion time of a job in stage 1 must be greater than or equal to the sum of its release time and its processing time in stage 1. Equation (8) defines the earliest start time of a job at stage 2. It ensures that a job cannot start at stage 2 before it has completed stage 1. Equation (9) ensures that the completion time of any job at any position on any machine in stage 2 accounts for its start time and processing duration at that stage. This equation helps track when each job will finish in the last stage. Equation (10) defines the tardiness of a job in stage 2. If job i is an urgent job and is assigned to the k-th position on machine j in stage 2, then its tardiness is calculated as the difference between the completion time of that position on the machine and the due date of the job. For other cases (i.e., for normal jobs or if the job is not yet completed), the equation is invalidated by a large number, namely Big M, which ensures that tardiness is only considered for urgent jobs. Equation (11) defines the maximum completion time for normal jobs. For urgent jobs, this equation is effectively invalidated by a large number, thus ensuring that it only captures the latest finish time among all normal jobs. The parameter Big M is defined as the theoretical worst-case completion time: $M = max_i(r_i) + \sum_{i=1}^{n}\sum_{s=1}^{2} p_{is}$. This sufficiently large value guarantees that constraints remain valid by serving as a sufficient upper bound.

The proposed model involves $n^2(M_1 + M_2)$binary variables, indicating that the problem size quadratically $O(n^2)$ scales with the number of jobs. For instance, with n=20 and 2 machines per stage, the number of binary variables exceeds 1600. This complexity theoretically justifies the necessity of using heuristic algorithms, since exact solvers become computationally intractable for large instances.

3. Proposed algorithms

In this study, each stage consists of identical parallel machines. In such cases, determining how to assign jobs to machines based on a given scheduling sequence is a critical decision. One of the most commonly used rules for selecting machines is to assign jobs to the machine with the earliest possible time (EPT). However, selecting machines based on this rule may exclude an optimal solution from the solution space, thus preventing the discovery of the best possible outcome. These results are typically observed in scenarios where delaying specific jobs or balancing machine loads is necessary. Despite this limitation, the EPT-based selection rule is simple, efficient, and enables rapid decision-making, making it still the most preferred approach in real-world industrial settings. Accordingly, this study restricts the machine selection rule to EPT and proposes algorithms to determine the job sequence based on EPT.

3.1. List scheduling algorithms

In practice, list scheduling algorithms are frequently employed for multi-machine scheduling because they are intuitive, easy to implement, and easy for operators to control. These algorithms sequence jobs are based on their availability and specific priority criteria. Our research adopts two methods: Modified Earliest Due Date (MEDD) and its enhanced version, MEDD2. The MEDD is based on Bang and Jeong [32] and was further refined by Jeong et al. [28] to incorporate urgent job considerations; however, it has limitations in considering urgency weights, making it insufficient to accurately prioritize urgent jobs. To overcome this drawback and enhance the algorithm's efficiency and responsiveness to urgent jobs, we propose MEDD2, an advanced version of MEDD that better incorporates urgency weights and adjusts due dates of urgent and normal jobs based on their importance. This ensures that the scheduling process more accurately reflects the priorities and due dates of jobs. Let $C_2(\sigma j)$ denote the completion time at stage 2 when job j is appended to the end of a current partial sequence σ. In our study, the shortest processing time (SPT) rule used for tie-breaking refers to the sum of processing times across both stages.

Modified Earliest Due Date (MEDD) rule: d'_j,

where $d'_j=d_j$ for $j \in J_A$, and $d'_j=C_2(\sigma j)$ for $j \in J_B$

Modified Earliest Due Date2 (MEDD2) rule: d'_j,

where $d'_j=max(d_j, d_j(1-(\alpha-0.5))$ for $j \in J_A$, and $d'_j=C_2(\sigma j)(1+(\alpha-0.5))$ for $j \in J_B$

In MEDD, even when the priorities of urgent jobs increase, the due dates of normal jobs are fixed at their completion times at stage 2. This fixed due date does not adequately reflect the weights of urgent jobs. MEDD2 addresses this by proportionally delaying the temporary due dates of normal jobs to how much the urgent job weight (α) exceeds 0.5. For instance, in the current partial solution, if the completion time of normal job j at stage 2 is 10, then in MEDD, the due date for normal job j is fixed at 10 regardless of the weight of urgent jobs. In MEDD2, if the weight (α) of the urgent job is 0.7, then the due date for the normal job j becomes 12 (=10×(1+(0.7−0.5))). If the urgent job's weight (α) is 0.3, then the due date for the normal job j becomes 8 (=10×(1+(0.3−0.5))). This approach ensures that as the weight of urgent jobs increases, the due dates of normal jobs are pushed back, thus allowing urgent jobs to be processed earlier. Conversely, if the weight of urgent jobs is less than 0.5, then the due dates of urgent jobs are pushed back, thus allowing normal jobs to be scheduled ahead of urgent orders. The adaptation mechanism in MEDD2 is designed as a linear scaling function centered at the neutral weight of α=0.5. The term $(1 + (\alpha-0.5))$ acts as a relaxation factor. When $\alpha > 0.5$, this factor becomes greater than 1.0, effectively pushing back the hypothetical due dates of normal jobs. This mechanism linearly reduces the priority of normal jobs in proportion to the urgency weight α, thus allowing urgent jobs to be processed earlier.

3.2. Constructive heuristic algorithms

Typically, constructive heuristic algorithms generate solutions by incrementally placing jobs into the best positions within a partial sequence. The Nawaz-Enscore-Ham (NEH) algorithm (Nawaz et al. [33]) is a prominent algorithm for flow shop scheduling. It typically forms an initial seed sequence by sorting jobs by the total processing time, then iteratively inserts each job into its best position. However, the search space of NEH is limited due to its strong dependence on the initial seed and the fixed relative positions of the jobs once partial solutions have been formed. To overcome these limitations, the N-NEH and federated learning (FL) algorithms were developed. N-NEH (Puka et al. [34]) diversifies the search by employing multiple initial seed sequences via an *N*-list, thus broadening the construction order. In contrast, the FL algorithm (Framinan and Leisten [35]) extends NEH by performing pairwise interchanges after job insertion, dynamically altering relative positions within the partial sequence to explore more neighboring solutions. These traditional constructive algorithms were primarily designed for makespan minimization in flow shops. However, our study deals with HFS to minimize tardiness for urgent jobs as well as minimize the makespan for normal jobs. This necessitates modifying these algorithms, as the tardiness objective fundamentally differs from the makespan, which always increases upon job placement. This is because there is no increase in tardiness if the added job is completed within the due date. Since the placement of a job significantly affects subsequent unscheduled jobs, we modified the evaluation process for tardiness by temporarily completing the sequence with unscheduled jobs. Then, it enables a comprehensive assessment of each position where the unscheduled jobs will be placed. Furthermore, the initial seed sequence, which is typically based on the total processing times for the makespan, also needs to be adjusted due to the objectives for normal and urgent jobs. Thus, this study employs the better of the schedules obtained by the MEDD and MEDD2 rules as the initial seed. Based on these modifications to the evaluation and seed generation, we adapted NEH, N-NEH, and FL, thus resulting in MNEH, MN-NEH, and MFL, respectively. Additionally, we introduce MN-MFL, which combines N-NEH's *N*-list for initial sequence consideration with FL's pairwise interchanges after job insertion. Specifically, the pairwise interchange does not rely on complex termination conditions, but systematically evaluates all possible swaps for positions *i* and *j* satisfies $1 \leq i \leq j \leq |S|$. It enhances the algorithm efficiency by retaining a swap only if it improves the objective function, thereby exhaustively exploring the neighborhood to mitigate the local optima. This approach, while increasing complexity, significantly broadens the search space by simultaneously varying both the initial sequence and relative job positions. Detailed procedures for these algorithms are shown in Procedures 1−4.

Procedure 1. (*MNEH*)

Step 1. Obtain a seed sequence by selecting the better of the schedules obtained by MEDD and MEDD2. Let S^0 and S be the seed sequence and the current partial sequence, respectively, and set S=∅ and k=1.

Step 2. Select the k-th job in S^0.

Step 3. For r=1 to |S|+1, do the following:
Temporarily insert the selected job into the r-th position of S. Create a temporary complete sequence by appending the remaining unscheduled jobs from S^0 (in their original order) to this partial sequence. Obtain the objective function value for this temporary complete sequence to evaluate the performance of inserting the job at the r-th position.

Step 4. Select a partial schedule that results in the minimum objective function value in Step 3. Update S to be the selected (partial) schedule.

Step 5. Update k←k+1. If k<n, then go to Step 2; otherwise, stop (S is the final solution of this heuristic).

Procedure 2. (*MN-NEH*)

Step 1. Obtain a seed sequence by selecting the better of the schedules obtained by MEDD and MEDD2. Let S^0 and S be the seed sequence and the current partial sequence, respectively. Set S=∅, k=1, N=2, and initialize *N-list* with the first N jobs from S^0. (N-value was empirically set to 2 for efficient performance.)

Step 2. For each job j in the N-list, do the following:
Temporarily insert job j into every possible position (r=1 to |S|+1) of S. For each temporary insertion, create a temporary complete sequence by appending the remaining unscheduled jobs from S^0 (in their original order) to this partial sequence. Obtain the objective function value for this temporary complete sequence to evaluate the performance of inserting the job.

Step 3. Select a partial schedule that results in the minimum objective function value in Step 2. Update S to be the selected (partial) schedule and remove the inserted job from the *N-list.*

Step 4. If possible, add the next job from S^0 to the *N-list.*

Step 5. If *N-list*≠ ∅, then return to Step 2; otherwise, stop (S is the final solution of this heuristic).

Procedure 3. (*MFL*)

Steps 1-4. (the same as steps 1-4 of MNEH)

Step 5. If $|S|<3$, then go to Step 7. Otherwise, perform pairwise interchanges for all possible pairs of jobs (i and j, where $i=1$ to $|S|-1$ and $j=i+1$ to $|S|$) of S. For each generated sequence, create a temporary complete sequence by appending the remaining unscheduled jobs from S^0 (in their original order) to this partial sequence. Obtain the objective function value for this temporary complete sequence to evaluate the performance of the interchange on the partial sequence.

Step 6. Select a partial schedule that results in the minimum solution value in *Step* 5. Update S to be the selected (partial) schedule.

Step 7. Update $k \leftarrow k+1$. If $k<n$, then go to Step 2; otherwise, stop (S is the final solution of this heuristic).

Procedure 4. (*MN-MFL*)
Steps 1-4. (the same as steps 1-4 of MN-NEH)
Steps 5-6. (the same as steps 5-6 of MFL)
Steps 7. (the same as Step 5 of MN-NEH)

3.3. Meta-heuristic (Simulated Annealing)

Simulated Annealing (SA) is a widely recognized meta-heuristic algorithm to solve combinatorial problems. The following subsections detail the implementation of the proposed SA, including the representation of feasible schedules, the neighborhood generation mechanism, and the parameter configurations for reproducibility.

3.3.1. Solution representation

A feasible solution to this problem respects the assumptions provided in Section 2, that is, a sequence is left- shift (gap-free), and jobs are assigned by the earliest possible time. Thus, the feasible sequence can be represented as a permutation. For instance, for a problem with n jobs, the solution space is composed of $n!$ permutation schedules.

3.3.2. Initial solution

In this research, the initial solution for the SA algorithm is selected from the best solution provided by our developed list scheduling and constructive algorithms, that is, the initial solution S is set as the best solution derived from the MEDD, MEDD2, MNEH, MN-NEH, MFL, and MN-MFL.

3.3.3. Neighborhood solution and Acceptance criterion

SA operates by seeking enhancements to the current solutions, transitioning from a current solution (S) to a neighborhood solution (S') through local searches. Herein, we use pair-wise interchange to generate neighborhoods. In this mechanism, two jobs in the current sequence are selected, then their positions are swapped, thus generating a new neighborhood solution. This simple but effective local search allows the algorithm to explore neighborhood solutions and gradually improve the solution quality while maintaining the computational efficiency. Then, if the neighborhood solution S' shows an improvement over S, then the transition to S' is accepted (i.e., $S' \leftarrow S$). Conversely, if S' is inferior to S, then the transition is accepted based on the probability $\exp(-\Delta/T)$, where Δ is the difference between S' and S, and T is the current temperature parameter. This probabilistic transition enables the SA to escape the local optimum and enhances its ability to explore better solutions.

3.3.4. Temperature parameters and epoch length

SA begins with an initial temperature T_0. At a certain temperature T, SA attempts to find better neighborhood solutions. However, if no improvement is found after E iterations, where E denotes the epoch length, then the temperature is gradually decreased by a cooling function as $T \leftarrow uT$, where u is the cooling ratio. Typically, the cooling ratio is set to a high value ($0.9 \leq u < 1$), which implies a slow decrease in temperature. The process terminates either when the temperature hits a final value T_f or when the algorithm runs for 3600 seconds.

Through preliminary experiments to determine the best configuration of parameters, the configuration was set to $(T_0, u, T_f, E) = (100, 0.995, 0.01, 2n)$, thus achieving efficient solutions within a practical time. Given the vast number of potential parameter combinations, we adopted a focused calibration strategy by fixing the initial temperature (T_0) based on preliminary experiments and prioritizing the tuning of the cooling rate (u), which is the most critical factor that influences the algorithm performance. To determine the optimal value of u that balances the solution quality and the computational efficiency, we conducted a sensitivity analysis using instances with n=40 jobs, selected as representative medium-sized problems. Each scenario was executed 10 times for each candidate value ($u \in \{0.990, 0.995, 0.999\}$) to obtain the average performance metrics. The results demonstrated a clear trade-off: increasing u from 0.990 to 0.995 yielded a meaningful 2.56% improvement in the solution quality, while the average adjusted CPU time increased by 94.4%. However, further increasing u to 0.999 resulted in diminishing returns, thus only offering a marginal additional gain of 0.46% while the computation time surged by 408.1%. Consequently, we selected u=0.995 as the final parameter to ensure a robust performance within a practical timeframe. Note that, due to the nature of scheduling problems, the solution space expands as the size of the problem instances increases. Therefore, to cope with the size of the problem instance, we set the epoch length in a way that takes into account the size of the problem instance.

4. Computational results

To evaluate the performance of the proposed algorithms, a series of tests were conducted using randomly generated instances that reflect real manufacturing environments. The instance characteristics and data generation scheme were designed to be consistent with semiconductor manufacturing systems with urgent and normal jobs, following the problem setting adopted in Jeong and Shim [2]. The computational experiments were performed on a personal computer equipped with an i5 processor running at 3.5 GHz and 32 GB of memory, and all algorithms proposed in this study were implemented using the Python programming language. In the test instances, the processing times were randomly generated following a discrete uniform distribution within the range [1, 10]. The weight (α) for the total tardiness of urgent jobs was set at three levels: 0.3, 0.5, and 0.7, while the ratio (β) of urgent jobs among all jobs ($\beta = |J_A|/n$) was also tested at three levels: 0.3, 0.5, and 0.7. If the release times of urgent jobs are too large, then scheduling becomes simplified because urgent jobs can be immediately processed upon arrival without interfering with normal jobs. To prevent this simplification, the release times of urgent jobs were randomly generated within an upper limit, considering the number of machines and jobs, that is, the release times were randomly generated within the range [0, $(10 \times n)/(M_1+M_2)$], where n is the total number of jobs, and M_1 and M_2 represent the number of machines at each stage. This ensures that urgent jobs do not have excessively large release times while maintaining a realistic scheduling complexity.

Since the considered problem is NP-hard, it may fail to obtain optimal solutions for large-sized instances. Therefore, two categories of problem sizes were considered: small-sized and large-sized instances. In small-sized instances, the number of jobs was set to 6, 8, 10, and 12, with 2 or 3 machines per stage. In large-sized instances, the number of jobs was 40, 60, and 80, with 5 or 8 machines per stage. In all experimental instances, the number of machines per stage was the same across all stages. For small-sized instances, 10 instances were generated for each combination of n, M_1, M_2, α, and β values, which resultedt in a total of 720 instances. For large-sized instances, 540 instances were generated.

Table 2. Performance of algorithms in small-sized problems.

machines per stage	# of jobs		CPLEX	MEDD	MEDD2	MNEH	MN-NEH	MFL	MN-MFL	SA
2	6	AE[*]	0.000	0.353	0.305	0.144	0.086	0.048	0.045	0.023
		SD[**]	0.000	0.272	0.324	0.178	0.113	0.063	0.061	0.036
	8	AE	0.000	0.460	0.348	0.156	0.125	0.066	0.069	0.021
		SD	0.000	0.293	0.211	0.121	0.104	0.055	0.078	0.029
	10	AE	0.000	0.460	0.382	0.176	0.137	0.073	0.071	0.021
		SD	0.000	0.308	0.220	0.126	0.101	0.055	0.055	0.027
	12	AE	0.003	0.482	0.445	0.194	0.168	0.081	0.076	0.021
		SD	0.010	0.317	0.243	0.144	0.149	0.077	0.075	0.036
3	6	AE	0.000	0.219	0.212	0.122	0.086	0.036	0.034	0.015
		SD	0.000	0.212	0.226	0.170	0.137	0.070	0.070	0.033
	8	AE	0.000	0.411	0.332	0.177	0.129	0.062	0.065	0.024
		SD	0.000	0.252	0.223	0.150	0.109	0.062	0.063	0.031
	10	AE	0.000	0.537	0.423	0.211	0.177	0.099	0.091	0.029
		SD	0.002	0.319	0.214	0.115	0.098	0.075	0.066	0.033
	12	AE	0.016	0.560	0.471	0.230	0.200	0.117	0.106	0.021
		SD	0.036	0.382	0.263	0.164	0.153	0.118	0.113	0.034
Total		AE	0.002	0.435	0.365	0.176	0.139	0.073	0.069	0.022
		SD	0.014	0.309	0.251	0.148	0.125	0.077	0.076	0.032

*Note: * Average relative error ratio of algorithms to the optimal(or best) solution; ** Standard deviation of the relative error ratio of algorithms to the optimal(or best) solution.

The performances of the algorithms were first analyzed using small-sized instances. To assess the performance, the average error ratio from the optimal solution was calculated. The optimal solutions were obtained by solving the proposed mathematical model using CPLEX(v 22.1.1). If an optimal solution could not be found within 3600 seconds, then the best solution obtained within that time limit was used. The results are summarized in the following Table 2. The SA algorithm demonstrates the closest results to the optimal solution in most cases, with an average gap ranging from 1.47% to 2.92%, the smallest among all tested methods. In contrast, MEDD and MEDD2 exhibit significantly lower performances, with gaps of 21.9% to 56.0% and 21.2% to 47.1%, respectively, compared to SA. MNEH and MN-NEH show gaps of 12.2% to 23.0%, thus indicating lower performances than SA.

While MFL and MN-MFL achieve results comparable to SA in some instances, their average gaps range from 3.6% to 11.7% and 3.3% to 10.6%, respectively, both larger than that of SA. Additionally, even in cases where the optimal solution was not found within the time limit (i.e., when the optimal gap is nonzero), SA consistently maintains a superior performance compared to other algorithms. The stability of the proposed algorithms was further analyzed using the standard deviation (SD) of the relative error ratios. Although the EPT selection rule does not guarantee global optimality, the SA algorithm exhibited a remarkably low overall average SD of 0.032. Specifically, the SA algorithm consistently maintained stable SD values, ranging from 0.021 to 0.036 across different numbers of jobs. This suggests that SA maintains a highly consistent and robust performance regardless of the problem size, thereby effectively finding stable solutions near the optimal, even as the scheduling complexity increases. Additionally, the experimental results in Table 2 reveal that list scheduling algorithms (MEDD and MEDD2) exhibit significantly higher SD values compared to constructive algorithms, thus indicating a higher performance volatility. Among the constructive methods, MFL and MN-MFL provide the most consistent performances with lower and more stable SD across all tested instances. Note that the average error ratio can significantly underestimate the performance of algorithms in tardiness-based evaluations. For example, if a job has a due date of 100, then the optimal solution finishes at 101 (tardiness = 1), while a heuristic finishes at 102 (tardiness = 2). Despite the absolute difference being just 1, the error ratio calculates a 1.00 (100%), which is misleading.

The Shapiro-Wilk test results for all algorithms yielded *p*-values significantly lower than 0.001, thus rejecting the assumption of normality. This non-normality is primarily due to the floor effect, where the relative error ratios are lower-bounded by zero, causing the data to be heavily skewed toward the left and preventing a symmetric normal distribution. Consequently, the Kruskal-Wallis test, a non-parametric method, was employed to ensure a rigorous statistical comparison. The test yielded a *p*-value of 2.2e-16, thus indicating statistically significant differences. Subsequently, Dunn's post hoc test was conducted with the Bonferroni correction for multiple comparisons to ensure statistical rigor, and the results are summarized in Table 3. The test showed that there was no statistically significant difference between SA and the optimal solution ($p > 0.05$), thus suggesting that SA achieved a performance comparable to the optimal. Among the remaining algorithms, MFL and MN-MFL exhibited relatively better performances, with no significant difference observed between the two. Between MEDD and MEDD2, MEDD2 showed a slightly better average performance by more effectively reflecting the characteristics of the problem. However, the difference was not statistically significant based on the test results.

Table 3. The results of Dunn's post hoc test of algorithms.

	MEDD	MEDD2	MNEH	MN-NEH	MFL	MN-MFL	SA	Optimal
MEDD	—							
MEDD2	ns	—						
MNEH	*	*	—					
MN-NEH	*	*	ns	—				
MFL	*	*	*	*	—			
MN-MFL	*	*	*	*	ns	—		
SA	*	*	*	*	*	*	—	
Optimal	*	*	*	*	*	*	ns	—

*Note: ns: Not significant; *: Significant difference ($p < 0.05$).

Kruskal–Wallis tests were conducted to evaluate how the algorithm performance varied with α (weight of urgent jobs) and β (ratio of urgent jobs). Regarding the α-value dependency, most algorithms exhibited significant performance variations: MEDD (p = 1.23e−6), MEDD2 (p = 0.003), MNEH (p = 5.10e−6), MN-NEH (p = 0.002), and MN-MFL (p = 0.038). In contrast, SA (p = 0.275) and MFL (p=0.153) demonstrated robust and stable performances, thus indicating consistency across different α-values. For β-value dependency, most algorithms showed no significant performance variation: MEDD (p = 0.985), MEDD2 (p = 0.790), MNEH (p = 0.655), MN-NEH (p = 0.451), MFL (p = 0.552), and MN-MFL (p = 0.227). However, SA showed statistically significant differences (p = 0.0086). This variation is attributed to SA's notably superior performance when β=0.3, where it more frequently finds optimal solutions. At other values of β, SA's performance remained consistent, suggesting an enhanced ability in scenarios with fewer urgent jobs rather than degradation at other levels.

Overall, the parameter-based validation highlighted the remarkable stability of MFL and SA. MFL maintained a consistent performance across both α (p = 0.153) and β (p = 0.552) values. For SA, the performance remained stable across all α and β-values except for β=0.3. The observed variation at β = 0.3 was not a decline but rather a significant performance improvement, thus confirming the overall effectiveness of SA. The problem difficulty is primarily determined by resource competition among urgent jobs with overlapping release windows. In small-sized instances, β=0.3 results in minimal job conflicts, thus allowing the SA algorithm to frequently converge to the optimal solution. However, as β reaches 0.5 or higher, the increased density of urgent jobs sharply escalates the scheduling complexity due to an intense resource contention. This explains the significant performance advantage observed at lower β levels, which diminishes as the environment becomes more congested.

Table 4. CPU time of algorithms in small-sized problems.

machines per stage	# of jobs	CPLEX	MEDD	MEDD2	MNEH	MN-NEH	MFL	MN-MFL	SA
2	6	0.34*	0.00	0.00	0.01	0.02	0.02	0.03	6.19
	8	11.60	0.01	0.01	0.03	0.03	0.06	0.06	10.53
	10	1,133.41	0.01	0.01	0.05	0.07	0.12	0.14	16.33
	12	2,919.72	0.04	0.03	0.14	0.20	0.42	0.49	139.35
3	6	0.29	0.00	0.00	0.01	0.02	0.02	0.03	6.19
	8	37.10	0.01	0.01	0.02	0.03	0.06	0.06	10.70
	10	1,848.52	0.01	0.01	0.06	0.08	0.15	0.17	20.87
	12	3,600.00	0.03	0.03	0.13	0.19	0.40	0.46	133.58
Average		1,194.19	0.01	0.01	0.06	0.08	0.15	0.18	42.97

*Note: * seconds.

Table 4 presents the CPU time for each algorithm. All algorithms, with the exception of SA, completed their tasks within 1 second. While SA's processing time was comparatively longer, it never exceeded 150 seconds for any instance. The CPLEX began to encounter difficulties in finding solutions within 3600 seconds for instances with 10 or more jobs. Notably, for all instances that involve 3 machines per stage and 12 jobs, the optimal solution could not be determined within the time limit.

Next, the performances of the algorithms were analyzed for large-sized problems. As this problem is NP-hard, obtaining optimal solutions for large instances may not be feasible within a practical time. Therefore, we compared the heuristic algorithms by calculating the relative gap from the best solution found among all algorithms. The results are summarized in Table 5. As shown in the table, SA consistently yielded the best solutions for all problems. Excluding SA, MFL and MN-MFL demonstrated superior performances among the other algorithms. Additionally, MFL and MN-MFL demonstrated consistent stability, thereby maintaining low SD values between 0.072 and 0.081 regardless of the problem scale.

In large-sized problems, the CPU times of the algorithms are summarized in Table 6. SA took the longest time, but it was able to solve all problems within 1 hour. MFL and MN-MFL, which showed the best performances excluding SA, also solved all problems within 10 minutes.

Table 5. Performance of algorithms in large size problem.

machines per stage	# of jobs		MEDD	MEDD2	MNEH	MN-NEH	MFL	MN-MFL	SA
5	40	AE*	1.021	0.868	0.365	0.324	0.118	0.131	0.000
		SD**	0.548	0.283	0.144	0.131	0.070	0.078	0.000
	60	AE	1.043	0.941	0.402	0.361	0.112	0.120	0.000
		SD	0.576	0.299	0.194	0.177	0.069	0.077	0.000
	80	AE	1.009	0.973	0.342	0.298	0.094	0.094	0.000
		SD	0.556	0.341	0.148	0.138	0.053	0.048	0.000
8	40	AE	1.089	0.899	0.434	0.383	0.138	0.138	0.000
		SD	0.605	0.350	0.191	0.188	0.097	0.096	0.000
	60	AE	1.115	0.959	0.418	0.356	0.104	0.108	0.000
		SD	0.704	0.361	0.246	0.202	0.073	0.073	0.000
	80	AE	1.378	1.228	0.488	0.443	0.124	0.118	0.000
		SD	0.817	0.414	0.238	0.221	0.084	0.081	0.000
Total		AE	1.109	0.978	0.408	0.361	0.115	0.118	0.000
		SD	0.651	0.362	0.202	0.184	0.077	0.078	0.000

Note: Average relative gap of algorithms to the best solution; ** Standard deviation of the relative error ratio of algorithms to the optimal(or best) solution.

Table 6. CPU time of algorithms in large- sized problems.

machines per stage	# of jobs	MEDD	MEDD2	MNEH	MN-NEH	MFL	MN-MFL	SA
5	40	0.61*	0.62	2.72	4.18	23.17	24.65	334.76
	60	2.51	2.63	11.25	17.11	133.44	140.38	1051.14
	80	6.88	6.73	28.85	44.79	441.92	463.70	2414.95
8	40	1.04	1.06	5.14	7.99	45.08	47.87	649.55
	60	1.91	1.94	9.01	14.00	112.71	117.72	880.86
	80	6.13	6.42	28.96	44.19	466.11	476.73	2513.59
Average		3.18	3.23	14.32	22.05	203.74	211.84	1307.48

*Note: * seconds.

The experimental results indicate that, with respect to the number of machines per stage (M_s), the SA algorithm maintains a highly stable performance in small-sized problems regardless of variations in M_s. As observed in Tables 2 and 5, all algorithms show performance degradation as M_s increases and the search space expands. In particular, the list scheduling algorithms exhibit a relatively larger decline in performance compared to the constructive algorithms. In contrast, only MFL and MN-MFL show a marginal increase in error rates and, except for SA, demonstrate the most stable and reliable performance. Notably, these algorithms effectively manage the increased scheduling complexity associated with a larger number of machines.

In conclusion, the SA algorithm proposed in this study demonstrated an exceptionally strong performance within the given time constraint. While SA yielded the best results, MFL and MN-MFL also provided relatively good performances, suggesting they could be viable alternatives when the computational efficiency is a primary concern.

5. Conclusions

This study comprehensively addressed the complex scheduling problem of the HFS with urgent and normal jobs, which is a common scenario in modern manufacturing environments. This problem not only focuses on the production efficiency of the HFS but also possesses the characteristics of a multi-agent problem, requiring the simultaneous consideration of two conflicting objectives: minimizing the makespan for normal jobs and minimizing the total tardiness for urgent jobs. To effectively achieve these multi-faceted objectives, we proposed various approaches, including a MIP model, and evaluated their performance. Specifically, we developed list scheduling algorithms (such as MEDD and MEDD2) and constructive algorithms (including MNEH, MN-NEH, MFL, and MN-MFL), and further presented a SA metaheuristic based on these, thereby seeking practical solutions for the complex HFS.

To validate the performance of the proposed algorithms, we conducted extensive experiments using randomly generated problem instances of various sizes and characteristics that reflect real manufacturing environments. The experimental results demonstrated that the developed SA algorithm consistently exhibited highly superior and robust performances across all tested scenarios, thereby effectively finding solutions close to the optimal. Alongside SA, the MFL and MN-MFL algorithms also showed noteworthy efficiencies and stable performances, thus suggesting their potential for practical utilization even in environments that require rapid decision-making. While the average execution time of the SA algorithm for large-scale problems may appear unsuitable for real-time decision-making, it demonstrates highly acceptable performances in environments such as semiconductor manufacturing systems, where shift-based operations are standard and prioritizing the derivation of high-quality solutions takes precedence over instantaneous results.

Despite its academic contribution as the first study on a HFS that considers urgent jobs, several meaningful directions remain for future research. First, the present study adopted the EPT rule for machine assignment, which is widely used in practice but does not guarantee global optimality. Developing alternative machine selection strategies that overcome this limitation while maintaining computational efficiency constitutes an important extension. Second, this study focused on a static scheduling environment in which all jobs were known in advance. However, in real manufacturing systems, urgent jobs often dynamically arrive and may involve uncertainty in release times. In such dynamic environments, the proposed SA framework provides a robust foundation for rescheduling; the existing schedules serve as high-quality initial seeds, thus allowing for swift, event-driven updates via localized neighborhood exploration. Additionally, in this context, incorporating adaptive search algorithms (Ding et al. [36]) or learning-based mechanisms such as Neuroevolutionary approaches (Chen et al. [37]) to enhance the responsiveness under dynamic and uncertain conditions represents a promising direction for future research. Finally, while the SA algorithm's efficiency was verified in small-sized instances, ensuring absolute optimality in large-scale problems remains challenging due to the inherent complexity of the 2-stage HFS. Furthermore, as this study is the first to model the specific structure of urgent jobs with release times, no established metaheuristics such as a genetic algorithm (GA) or Tabu Search currently exist for direct comparison. Therefore, we expect that various metaheuristics will be developed based on this study in the future and that our work will serve as a foundational benchmark for such evaluations.

Author contributions

Ju-Yong Lee: conceptualization, methodology, software, validation, formal analysis, investigation, data curation, writing-original draft, writing-review & editing; BongJoo Jeong: conceptualization, methodology, resources, writing-original draft, writing-review & editing, supervision, project administration.

Use of Generative-AI tools declaration

AI tools used: Gemini (Google)

How were the AI tools used?

1. Coding Assistance: The AI tool was used to assist in generating and optimizing portions of the Python code used for data processing and numerical experiments. Specifically, it helped in debugging and refining the implementation of the proposed methodology. 2. Writing Assistance: The AI tool was employed to improve the grammatical accuracy and academic phrasing of the manuscript. It provided suggestions for more precise terminology and aided in the overall refinement of the English prose.

Where in the article is the information located?

The AI-assisted code contributes to the results presented in the 'Experimental Results' section. The writing assistance was applied throughout the manuscript to enhance linguistic quality.

Acknowledgments

This work was supported by the research grant of Kongju National University in 2022.

Conflict of interest

All authors declare no conflicts of interest in this paper.

Data Availability Statement

All experimental instances (processing times, release times, job classes) and detailed results are available at: https://bit.ly/4qVYbnr.

References

1. J. S. Neufeld, S. Schulz, U. Buscher, A systematic review of multi-objective hybrid flow shop scheduling, *Eur. J. Oper. Res.*, **309** (2023), 1–23. https://doi.org/10.1016/j.ejor.2022.08.009
2. B. J. Jeong, S. O. Shim, Heuristic algorithms for two-machine re-entrant flowshop scheduling problem with jobs of two classes, *J. Adv. Mech. Des. Syst. Manuf.*, **11** (2017), JAMDSM0062. https://doi.org/10.1299/jamdsm.2017jamdsm0062

3. T. Kis, E. Pesch, A review of exact solution methods for the non-preemptive multiprocessor flowshop problem, *Eur. J. Oper. Res.,* **164** (2005), 592–608. https://doi.org/10.1016/j.ejor.2003.12.026
4. J. Liang, P. Wang, L. Guo, B. Qu, C. Yue, K. Yu, et al., Multi-objective flow shop scheduling with limited buffers using hybrid self-adaptive differential evolution, *Memetic Comput.*, **11** (2019), 407–422. https://doi.org/10.1007/s12293-019-00290-5
5. V. Anjana, R. Sridharan, P. N. Ram Kumar, Metaheuristics for solving a multi-objective flow shop scheduling problem with sequence-dependent setup times, *J. Sched.,* **23** (2020), 49–69. https://doi.org/10.1007/s10951-019-00610-0
6. A. P. Rifai, S. T. W. Mara, A. Sudiarso, Multi-objective distributed reentrant permutation flow shop scheduling with sequence-dependent setup time, *Expert Syst. Appl.*, **183** (2021), 115339. https://doi.org/10.1016/j.eswa.2021.115339
7. Y. Han, J. Li, H. Sang, Y. Liu, K. Gao, Q. Pan, Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time, *Appl. Soft Comput.*, **93** (2020), 106343. https://doi.org/10.1016/j.asoc.2020.106343
8. C. Lu, L. Gao, J. Yi, X. Li, Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in China, *IEEE Trans. Ind. Inform.*, **17** (2021), 6687–6696. https://doi.org/10.1109/TII.2020.3043734
9. B. Zhang, Q. Pan, L. Gao, X. Li, L. Meng, K. Peng, A multiobjective evolutionary algorithm based on decomposition for hybrid flowshop green scheduling problem, *Comput. Ind. Eng.*, **136** (2019), 325–344. https://doi.org/10.1016/j.cie.2019.07.036
10. B. G. Yılmaz, Ö. F. Yılmaz, F. B. Yeni, Comparison of lot streaming division methodologies for multi-objective hybrid flowshop scheduling problem by considering limited waiting time, *J. Ind. Manag. Optim.,* **20** (2024), 3373–3414. https://doi.org/10.3934/jimo.2024058
11. X. Liu, X. Chen, Y. Zhao, X. Liu, C. C. Wu, W. C. Lin, A multi-objective flexible flow shop scheduling problem with an improved NSGA-II algorithm, *J. Ind. Manag. Optim.*, **21** (2025), 4041–4062. https://doi.org/10.3934/jimo.2025042
12. Z. H. Jia, T. Wu, H. Zhang, C. Liu, K. Li, An energy-efficient scheduling approach for a two-stage hybrid flow shop with parallel batch machines, *Eur. J. Oper. Res.,* **328** (2026), 762–784. https://doi.org/10.1016/j.ejor.2025.07.055
13. Q. Peng, K. Gao, Y. Fu, J. Li, H. F. Rahman, Integrating meta-heuristics and Q-learning for solving hybrid flow shop scheduling and rescheduling problems with reentrant, *J. Ind. Manag. Optim.,* **21** (2025), 6023–6043. https://doi.org/10.3934/jimo.2025121
14. M. Rauf, Z. Guan, S. Sarfraz, J. Mumtaz, S. Almaiman, E. Shehab, et al., Multi-criteria inventory classification based on multi-criteria decision-making (Mcdm) technique, In: *Advances in Manufacturing Technology*, **8** (2018), 343–348. https://doi.org/10.3233/978-1-61499-902-7-343
15. M. Rauf, Z. Guan, S. Sarfraz, J. Mumtaz, E. Shehab, M. Jahanzaib, et al., A smart algorithm for multi-criteria optimization of model sequencing problem in assembly lines, *Robot. Comput. Integr. Manuf.,* **61** (2020), 101844. https://doi.org/10.1016/j.rcim.2019.101844
16. A. Agnetis, P. B. Mirchandani, D. Pacciarelli, A. Pacifici, Scheduling problems with two competing agents, *Oper. Res.*, **52** (2004), 229–242. https://doi.org/10.1287/opre.1030.0092
17. K. R. Baker, J. Cole Smith, A multiple-criterion model for machine scheduling, *J. Sched.,* **6** (2003), 7–16. https://doi.org/10.1023/A:1022231419049
18. C. T. Ng, T. C. E. Cheng, J. J. Yuan, A note on the complexity of the problem of two-agent scheduling on a single machine, *J. Combin. Optim.*, **12** (2006), 387–394. https://doi.org/10.1007/s10878-006-9001-0

19. J. Y. T. Leung, M. Pinedo, G. Wan, Competitive two-agent scheduling and its applications, *Oper. Res.,* **58** (2010), 458–469. https://doi.org/10.1287/opre.1090.0744
20. T. C. E. Cheng, C. T. Ng, J. J. Yuan, Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs, *Theor. Comput. Sci.*, **362** (2006), 273–281. https://doi.org/10.1016/j.tcs.2006.07.011
21. T. C. E. Cheng, C. T. Ng, J. J. Yuan, Multi-agent scheduling on a single machine with max-form criteria, *Eur. J. Oper. Res.,* **188** (2008), 603–609. https://doi.org/10.1016/j.ejor.2007.04.040
22. P. Liu, L. Tang, Two-agent scheduling with linear deteriorating jobs on a single machine, In: *Lecture Notes in Computer Science,* Heidelberg: Springer Berlin Heidelberg, 2008. https://doi.org/10.1007/978-3-540-69733-6_63
23. W. C. Lee, Y. H. Chung, Z. R. Huang, A single-machine bi-criterion scheduling problem with two agents, *Appl. Math. Comput.,* **219** (2013), 10831–10841. https://doi.org/10.1016/j.amc.2013.05.025
24. L. Wan, J. Yuan, L. Wei, Pareto optimization scheduling with two competing agents to minimize the number of tardy jobs and the maximum cost, *Appl. Math. Comput.*, **273** (2016), 912–923. https://doi.org/10.1016/j.amc.2015.10.059
25. W.-C. Lee, S.-K. Chen, C.-C. Wu, Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem, *Expert Syst. Appl.,* **37** (2010), 6594–6601. https://doi.org/10.1016/j.eswa.2010.02.125
26. W. C. Lee, S. K. Chen, C. W. Chen, C. C. Wu, A two-machine flowshop problem with two agents, *Comput. Oper. Res.,* **38** (2011), 98–104. https://doi.org/10.1016/j.cor.2010.04.002
27. B. Mor, G. Mosheiov, Polynomial time solutions for scheduling problems on a proportionate flowshop with two competing agents, *J. Oper. Res. Soc.*, **65** (2014), 151–157. https://doi.org/10.1057/jors.2013.9
28. B. J. Jeong, Y. D. Kim, S. O. Shim, Algorithms for a two-machine flowshop problem with jobs of two classes, *Int. Trans. Oper. Res.,* **27** (2020), 3123–3143. https://doi.org/10.1111/itor.12530
29. B. J. Jeong, J. H. Han, J. Y. Lee, Metaheuristics for a flow shop scheduling problem with urgent jobs and limited waiting times, *Algorithms*, **14** (2021), 323. https://doi.org/10.3390/a14110323
30. D. Lei, X. Guo, A shuffled frog-leaping algorithm for hybrid flow shop scheduling with two agents, *Expert Syst. Appl.*, **42** (2015), 9333–9339. https://doi.org/10.1016/j.eswa.2015.08.025
31. D. Lei, Two-phase neighborhood search algorithm for two-agent hybrid flow shop scheduling problem, *Appl. Soft Comput.,* **34** (2015), 721–727. https://doi.org/10.1016/j.asoc.2015.05.027
32. J. Y. Bang, B. J. Jeong, A branch-and-bound algorithm to minimize total tardiness in a two-machine flowshop problem with different release time, *J. Adv. Mech. Des. Syst. Manuf.*, **13** (2019), JAMDSM0080. https://doi.org/10.1299/jamdsm.2019jamdsm0080
33. M. Nawaz, E. E. Enscore, I. Ham, A heuristic algorithm for the m-machine, n-job flow shop sequencing problem, *Omega*, **11** (1983), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9
34. R. Puka, J. Duda, A. Stawowy, I. Skalna, N-NEH algorithm for solving permutation flow shop problems, *Comput. Oper. Res.*, **132** (2021), 105296. https://doi.org/10.1016/j.cor.2021.105296
35. J. M. Framinan, R. Leisten, An efficient constructive heuristic for flowtime minimisation in permutation flow shops, *Omega*, **31** (2003), 311–317. https://doi.org/10.1016/S0305-0483(03)00047-1
36. L. Ding, Z. Guan, D. Luo, M. Rauf, W. Fang, An adaptive search algorithm for multiplicity dynamic flexible job shop scheduling with new order arrivals, *Symmetry*, **16** (2024), 641. https://doi.org/10.3390/sym16060641

37. Y. Chen, J. Zhang, M. Rauf, J. Mumtaz, S. Huang, Dynamic scheduling of hybrid flow shop problem with uncertain process time and flexible maintenance using NeuroEvolution of Augmenting Topologies, *IET Collab. Intell. Manuf.*, **6** (2024), e12119. https://doi.org/10.1049/cim2.12119