Journal of Industrial and Management Optimization

JIMO, 22(12): 6475–6502.
DOI: 10.3934/jimo.2026217
Received: 29 September 2026
Revised: 02 November 2026
Accepted: 05 November 2026
Published: 12 November 2026

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

Research article

Two-stage three-machine assembly flowshop scheduling with step learning: exact and metaheuristic approaches

Chin-Chia Wu[1], Juin-Han Chen[2,*], Xin-Gong Zhang[3], Chao-Chung Kang[4], Ching-Yuan Huang[1], Jian-You Xu[5] and Jatinder N.D. Gupta[6]

[1] Department of Statistics, Feng Chia University, Taichung, 40724, Taiwan
[2] Department of Industrial Engineering and Management, National Quemoy University, Kinmen County 892, Taiwan
[3] College of Mathematics Science, Chongqing Normal University, Chongqing, China
[4] Department of Business Administration, Providence University, Taichung, 433303, Taiwan
[5] College of Information Science and Engineering, Northeastern University, Shenyang 110819, China
[6] College of Business, The University of Alabama in Huntsville, Huntsville, AL 35899, USA

* **Correspondence:** Email: jhchen@nqu.edu.tw.

Abstract: In this paper, we addressed a scheduling challenge in a two-stage, three-machine flow-shop assembly problem with step learning. In this setting, each job had a machine-specific learning date, after which its processing time was reduced if processing began on or after that date. Our goal was to devise a schedule that minimized the makespan for the given jobs. This problem was categorized as NP-hard, as it could be reduced to a similar problem without learning dates. To address this issue, we formulated a mixed-integer programming model and developed two lemmas along with a lower bound, which were incorporated into a branch-and-bound algorithm to achieve an optimal solution. Additionally, we proposed two heuristics derived from the Johnson rule, including enhanced versions utilizing a swapping scheme. We also applied these Johnson-based heuristics in an iterated greedy algorithm, a population-based genetic algorithm, and simulated annealing algorithm to effectively approximate the solutions. Finally, we performed numerical experiments to assess the efficacy of all the proposed methods.

Keywords: scheduling; iterated greedy algorithm; population-based genetic algorithm; simulated annealing algorithm; learning dates
Mathematics Subject Classification: 90B35

1. Introduction

The existence of learning phenomena has been widely acknowledged in many manufacturing systems. Learning effects play an important role in production scheduling because processing times may decrease as workers gain experience, complete training programs, or become familiar with manufacturing operations. Two commonly studied learning effects are position-based learning and time-dependent learning. Position-based learning describes improvements associated with the sequence or position of jobs (Biskup [1]), whereas time-dependent learning reflects improvements associated with elapsed processing time (Koulamas and Kyparisis [2]; Kuo and Yang [3]).

Two-stage assembly scheduling problems (TSASPs) arise in manufacturing industries, including automobile assembly, textile manufacturing, and electronic component production. In these environments, individual components are processed on separate machines during the first stage and subsequently combined or assembled in the second stage. Efficient coordination between the processing and assembly stages is essential for reducing production delays, improving machine utilization, and minimizing the makespan. However, scheduling becomes increasingly challenging when processing times are affected by learning effects.

In practical manufacturing environments, workers may achieve improvements in processing efficiency after completing specific training programs or gaining sufficient operational experience. Such improvements can be represented by step-learning effects, where processing times decrease after predetermined learning dates. Unlike continuous learning models, step-learning models capture discrete productivity improvements associated with specific training milestones or operational changes. Ignoring these effects may lead to inaccurate processing-time estimates and inefficient production schedules. Therefore, incorporating step-learning effects into two-stage assembly scheduling is important for developing more realistic and efficient scheduling decisions.

Despite the extensive literature on scheduling with learning effects, relatively limited attention has been given to two-stage, three-machine assembly flowshop scheduling with job- and machine-specific step-learning dates. The main challenge arises from the interaction between assembly precedence constraints, job sequencing decisions, and changes in processing times after learning dates. Furthermore, the combinatorial complexity of the problem makes it difficult to obtain optimal solutions within reasonable computational time, particularly for large-scale instances.

To address these challenges, we investigate a two-stage, three-machine assembly flowshop scheduling problem with step-learning effects, aiming to minimize the makespan. The main contributions are summarized as follows:

1. We formulate a mathematical model that incorporates job- and machine-specific step-learning dates into a two-stage assembly scheduling environment.

2. We develop two dominance properties and a lower bound to improve the computational efficiency of an exact branch-and-bound (B&B) algorithm.
3. We propose two Johnson-rule-based heuristics with swapping improvements and develop three metaheuristic approaches, including iterated greedy (IG), genetic algorithm (GA), and simulated annealing (SA), to obtain high-quality solutions.
4. We conduct extensive computational experiments to evaluate the effectiveness of the proposed solution methods for small- and large-sized instances.

The computational results provide quantitative evidence of the effectiveness of the proposed algorithms. For example, the J3V_IG algorithm achieves an average error percentage (AEP) of only 0.030% for small-sized instances, as reported in the computational results. This finding demonstrates the ability of the proposed approach to obtain near-optimal scheduling solutions while considering step-learning effects.

The remainder of this paper is organized as follows: In Section 2, we review the related literature and identify the research gap. In Section 3, we describe the proposed scheduling problem and its mathematical formulation. In Section 4, we present the dominance properties and lower bound incorporated into the B&B algorithm. In Section 5, we introduce the proposed heuristics and metaheuristic algorithms. In Section 6, we discuss the parameter settings of the algorithms. In Section 7, we present the computational results and performance comparisons. Finally, in Section 8, we conclude the paper and suggest directions for future research.

2. Literature review

2.1. Two-stage assembly scheduling with learning effects

The integration of learning concepts into production scheduling has attracted considerable attention over the past few decades. Biskup [1] and Cheng and Wang [4] were among the pioneering contributors to scheduling research incorporating learning effects. Subsequent studies investigated various forms of learning effects in different scheduling environments (Koulamas and Kyparisis [2]; Kuo and Yang [3]; Wang [5]; Yin et al. [6]; Wang and Wang [7]; Niu et al. [8]). Comprehensive reviews of scheduling problems involving learning effects can be found in Azzouz et al. [9], Biskup [10], and Janiak et al. [11].

Most early studies focused primarily on single-machine or conventional flowshop scheduling environments. Relatively limited research has addressed two-stage assembly scheduling problems (TSASPs) with learning effects. Wu et al. [12] investigated a TSASP with position-based learning and developed a branch-and-bound (B&B) algorithm, simulated annealing (SA), cloud theory-based simulated annealing (CSA), and hybrid NEH heuristics. Lin [13] further investigated related scheduling problems with learning effects, employing exact and metaheuristic approaches to optimize scheduling performance.

Zou et al. [14] studied a TSASP with sum-of-processing-times-based learning effects to minimize the makespan. They developed a B&B algorithm, modified Johnson-rule-based heuristics, a genetic algorithm (GA), and CSA. Azzouz et al. [15] investigated a TSASP with truncated position-based learning, proposing a B&B algorithm and six iterated greedy (IG) algorithms to minimize the makespan. Similarly, Wu et al. [16] considered a TSASP with truncated cumulative processing-time-

based learning effects and developed a B&B algorithm, dynamic differential evolution (DDE), a hybrid IG algorithm, and a GA.

Other researchers have extended two-stage assembly scheduling to more complex production environments. Wang et al. [17] investigated distributed two-stage assembly scheduling with maintenance considerations, while Rostami and Yousefzadeh [18] examined decentralized two-stage assembly scheduling. Lin and Chen [19] considered dynamic scheduling in a TSASP to minimize total tardiness using a Markov decision process. Moreover, Ying et al. [20] developed a mixed-integer linear programming model and an iterated epsilon-greedy reinforcement learning algorithm for distributed assembly hybrid flowshop scheduling problems.

These studies demonstrate the importance of developing effective solution methods for two-stage assembly scheduling. However, most learning-based TSASP models consider position-based, cumulative processing-time-based, or truncated learning functions rather than job- and machine-specific step-learning dates.

2.2. Step-learning scheduling problems

Step-learning effects represent discrete improvements in processing efficiency after specific learning dates or milestones. Unlike position-based or continuously time-dependent learning effects, step-learning models assume that processing times decrease when predefined learning conditions are satisfied. Such models are particularly relevant to manufacturing environments in which productivity improvements occur after worker training, operational adjustments, or technological upgrades.

Kim and Oran [21] investigated a single-machine scheduling problem with step-learning effects to minimize total completion time. They established the NP-hardness of the problem and developed an LP-based heuristic and a mixed-integer programming approach. Xu et al. [22] examined a two-machine flowshop scheduling problem with step-learning effects to minimize the makespan. They proposed a B&B algorithm, two Johnson-rule-based heuristics, and a population-based genetic algorithm.

Atsmony et al. [23] considered single-machine scheduling with job-dependent step-learning dates to minimize the makespan. They established NP-hardness and developed pseudo-polynomial dynamic programming algorithms under different learning-date and machine-idleness conditions. Song et al. [24] investigated scheduling with step learning and job rejection, developing a pseudo-polynomial-time algorithm.

Researchers have further explored different step-learning scheduling environments. Phosavanh and Oron [25] investigated single-machine scheduling with step-learning effects to minimize total weighted late work. They established strong NP-hardness for the general problem and proposed pseudo-polynomial algorithms for special cases involving common learning dates or due dates. Jiang et al. [26] examined single-machine scheduling with distinct learning dates and a common learning ratio. They developed a polynomial-time algorithm for the preemptive case and an approximation algorithm with a theoretical performance guarantee for the non-preemptive case.

Although these studies have advanced the understanding of step-learning scheduling, they have mainly considered single-machine or conventional two-machine environments. The incorporation of

job- and machine-specific step-learning dates into two-stage, three-machine assembly flowshop scheduling remains relatively unexplored.

2.3. Developments and research gap

Researchers have addressed increasingly complex production scheduling problems involving maintenance, setup times, environmental considerations, and learning effects.

Kong et al. [27] studied green scheduling for two-stage photo-etching and acid-etching processes in semiconductor manufacturing. They developed a hybrid metaheuristic to improve production efficiency while considering environmental impacts. Kazemi et al. [28] investigated integrated two-stage multi-factory assembly scheduling with preventive and corrective maintenance, proposing mathematical models and a decomposition algorithm to minimize the makespan. Lu et al. [29] examined two-stage hybrid flowshop scheduling with sequence-dependent setup times in semiconductor manufacturing and developed a customized variable neighborhood search algorithm.

Other researchers have incorporated learning effects into different scheduling environments. Lv and Wang [30] considered two-machine flowshop scheduling with release dates and truncated learning effects, developing a B&B algorithm, heuristics, simulated annealing, and tabu search to minimize total completion time. Zhang et al. [31] investigated single-machine scheduling with DeJong's time-dependent learning effect and linear resource consumption, proposing optimality properties, a B&B algorithm, and heuristic methods for minimizing a weighted combination of makespan, total completion time, and resource costs.

Gu and Wang [32] studied a two-machine no-wait flowshop scheduling problem with learning effects and group technology. They established NP-hardness, identified polynomially solvable special cases, and developed exact and approximation-based solution methods. Gu et al. [33] further examined a two-machine no-wait flowshop scheduling problem with learning effects to minimize the makespan, developing polynomial algorithms for special cases, a B&B algorithm, and several heuristics.

Lu and Li [34] investigated single-machine scheduling with resource allocation, learning, and deterioration effects, establishing optimality properties and polynomial-time solution procedures. Lv and Wu [35] considered just-in-time single-machine scheduling with learning and deterioration effects and past-sequence-dependent delivery times, deriving polynomial-time solutions for different due-window assignments. Wang [36] investigated due-window assignment scheduling with truncated learning effects, position-dependent weights, and past-sequence-dependent setup times. Zhang et al. [37] studied group parallel-batching scheduling with position-dependent learning and time-dependent deterioration, proposing exact and heuristic methods for complex sequencing decisions.

These contributions demonstrate the growing importance of incorporating realistic manufacturing characteristics into production scheduling models. Nevertheless, most researchers focus on single-machine, conventional flowshop, or distributed assembly environments. Comparatively little attention has been given to two-stage, three-machine assembly flowshop scheduling with job- and machine-specific step-learning dates.

In contrast to other researchers, we simultaneously consider assembly precedence constraints and discrete processing-time reductions associated with individual learning dates. To address the resulting scheduling complexity, we develop dominance properties, a lower bound, an exact B&B algorithm,

Johnson-rule-based heuristics, and three metaheuristic approaches. These methods provide exact and approximate solutions for minimizing the makespan in the proposed scheduling environment.

A comparison of the scheduling environments, learning characteristics, optimization objectives, and solution methods considered in the related literature is summarized in Table 1.

Table 1. Comparison of related scheduling studies.

Study	Scheduling environment	Learning effect	Objective	Solution methods
Wu et al. [12]	Two-stage assembly	Position-based	Makespan	B&B, heuristics
Zou et al. [14]	Two-stage assembly	Sum-of-processing-times-based	Makespan	B&B, GA, CSA
Xu et al. [22]	Two-machine flowshop	Step-learning	Makespan	B&B, heuristics, GA
Atsmony et al. [23]	Single-machine	Step-learning	Makespan	Dynamic programming
Present study	Two-stage, three-machine assembly	Job- and machine-specific step-learning	Makespan	B&B, IG, GA, SA, heuristics

As summarized in Table 1, researchers have investigated various two-stage assembly scheduling problems with different learning effects and optimization methods. However, relatively limited attention has been given to two-stage, three-machine assembly flowshop scheduling with job- and machine-specific step-learning dates. To address this research gap, we develop dominance properties, a lower bound, an exact branch-and-bound algorithm, and several heuristic and metaheuristic approaches to minimize the makespan.

3. Definitions and problem description

First, we introduce some symbol definitions that are used in this study, as outlined below.

n = Number of jobs.

J_j = Job code j, j=1, 2, …, n.

N=$\{J_1, J_2, \ldots, J_n\}$ = A set of n jobs.

M_a, M_b, and M_c = three machine codes.

σ = Job schedule.

π_1, π_2 = Partial job schedules

[ ] = Indicates the position of a job in a certain sequence

u_{aj}, u_{bj}, u_{cj} = Original processing times of job J_j on machines M_a, M_b, and M_c, j=1, 2, …, n.

v_{aj}, v_{bj}, v_{cj} = reduced processing times of job $J_j$due to the learning dates on machines M_a, M_b, and M_c, j=1, 2, …, n.

$LD_{aj}, LD_{bj}, LD_{cj}$ = Learning dates for job J_j on machines M_a, M_b, and M_c, j=1, 2, …, n.

$C_{aj}(\sigma), C_{bj}(\sigma), C_{cj}(\sigma)$ = Completion times of job J_j in schedule σ on machines M_a, M_b, and M_c, j=1, 2, …, n.

s_a, s_b, s_c = Start times of processing job J_j on machines M_a, M_b, and M_c.

$C_{max}(\sigma)$= Maximum completion time (makespan). That is,

$$C_{max}(\sigma) = \max\{C_{c[1]}(\sigma), C_{c[2]}(\sigma), \ldots, C_{c[n]}(\sigma)\}$$

In this section, we describe the following problem model: We have a set of n jobs, each of which needs to be processed on three distinct machines. At time zero, all n jobs are available for processing. Each job J_j has three components, each with a normal processing time: u_{aj} for machine M_a, u_{bj} for machine M_b, and u_{cj} for machine M_c.

The processing rules are as follows:

i. Each job must be processed sequentially on all three machines.
ii. A job can only start processing on machine M_c after it has been completed on machines M_a and M_b.
iii. Machines M_a and M_b cannot have idle time and cannot process multiple jobs simultaneously.
iv. Once a job starts on any machine, it cannot be interrupted.

In this study, the processing times are subject to a learning effect. Let v_{kj} (where $v_{kj} \leq u_{kj}$) represent the reduced processing time for job J_j on machine M_k due to the learning effect, which occurs at learning dates LD_{kj}. Specifically, $LD_{aj}, LD_{bj},$ and LD_{cj} denote the learning dates for job J_j on machines M_a, M_b, and M_c, respectively. According to the three-field notation proposed by Graham et al. [38], the problem can be simply recorded as *F2// ass.*, $v_{aj} \leq u_{aj}, v_{bj} \leq u_{bj}, v_{cj} \leq u_{cj}$ $LD_{aj}; LD_{bj};$ $LD_{cj}//C_{max}$.

The described scheduling problem can be formulated as a mixed-integer programming model as follows:

Decision Variables:

C_{aj}: Completion time of J_j on M_a.
C_{bj}: Completion time of J_j on M_b.
C_{cj}: Completion time of J_j on M_c.
s_{aj}, s_{bj}, s_{cj}: Start times of job J_j on machines M_a, M_b, and M_c, respectively.
x_{aj}, x_{bj}, x_{cj}: Binary variables where $x_{aj} = 1$ if job J_j starts after the learning date LD_{aj} on machine M_a, and similarly for machines M_b and M_c

Objective Function:

Minimize the maximum completion time (makespan):

$$\min C_{max} = max\{C_{cj}\}, \forall j = 1,2, \ldots, n$$

Constraints:

i. Completion time for each job on each machine, considering the learning date:

$$C_{aj} = s_{aj} + x_{aj} \cdot v_{aj} + (1 - x_{aj}) \cdot u_{aj}$$

$$C_{bj} = s_{bj} + x_{bj} \cdot v_{bj} + (1 - x_{bj}) \cdot u_{bj}$$

$$C_{cj} = max\{C_{aj}, C_{bj}, s_{cj}\} + x_{cj} \cdot v_{cj} + (1 - x_{cj}) \cdot u_{cj}$$

ii. Start times must be greater than or equal to the completion times of the previous jobs on the same machine (ensuring no overlap):

$$s_{a[j]} \geq C_{a[j-1]} \text{ or } s_{a[k]} \geq C_{a[k-1]};$$

$$s_{b[j]} \geq C_{b[j-1]} \text{ or } s_{b[k]} \geq C_{b[k-1]};$$

$$s_{c[j]} \geq C_{c[j-1]} \text{ or } s_{c[k]} \geq C_{c[k-1]},$$

$$\forall j \neq k.$$

iii. The job sequence must follow the order $\{M_a, M_b\} \rightarrow M_c$, meaning that a job can start only on M_c after it is completed on $\{M_a, M_b\}$:

$$s_{cj} \geq \max\{C_{aj}, C_{bj}\}, \forall j$$

iv. The learning dates constraint:

$$s_{aj} \geq LD_{aj}, s_{bj} \geq LD_{bj}, s_{cj} \geq LD_{cj}, \forall j$$

if the corresponding $x_{aj}, x_{bj}, x_{cj} = 1$

4. Properties and lower bounds

Given schedules $\sigma = (\pi_1, J_i, J_j, \pi_2)$ and $\sigma' = (\pi_1, J_j, J_i, \pi_2)$, and assuming s_a, s_b, s_c represent the completion times of the last jobs in the partial subsequence π_1, the completion times of jobs J_i *and* J_j in schedules $\sigma = (\pi_1, J_i, J_j, \pi_2)$ and $\sigma' = (\pi_1, J_j, J_i, \pi_2)$ can be obtained as follows:

Property 1: If two adjacent jobs *i* and *j* in schedules σ and σ' with the following conditions: $s_a > \max\{LD_{ai}, LD_{aj}\}$, $s_b > \max\{LD_{bi}, LD_{bj}\}$, $s_c > \max\{s_a + v_{aj}, s_b + v_{bj}\}$, $\min\{s_a + v_{aj}, s_b + v_{bj}\} > \max\{LD_{ci}, LD_{cj}\}$, $\min\{s_a + v_{aj} + v_{ai}, s_b + v_{bj} + v_{bi}\} > s_c + v_{ci}$, $v_{ai} < v_{aj}$, $v_{bi} < v_{bj}$, and $v_{cj} < v_{ci}$, then schedule σ dominates schedule σ'.

Proof: This notes that $C_{ai}(\sigma) = \begin{cases} s_a + u_{ai}, & \text{if } s_a < LD_{ai} \\ s_a + v_{ai}, & \text{if } s_a \geq LD_{ai} \end{cases}$, $C_{bi}(\sigma) = \begin{cases} s_b + u_{bi}, & \text{if } s_b < LD_{bi} \\ s_b + v_{bi}, & \text{if } s_b \geq LD_{bi} \end{cases}$, and $C_{ci}(\sigma) = max\{C_{ai}(\sigma), C_{bi}(\sigma), s_c\} + \begin{cases} u_{ci}, & \text{if } max\{C_{ai}(\sigma), C_{bi}(\sigma), s_c\} < LD_{ci} \\ v_{ci}, & \text{if } max\{C_{ai}(\sigma), C_{bi}(\sigma), s_c\} \geq LD_{ci} \end{cases}$.

$C_{aj}(\sigma) = C_{ai}(\sigma) + \begin{cases} u_{aj}, & \text{if } C_{ai}(\sigma) < LD_{aj} \\ v_{aj}, & \text{if } C_{ai}(\sigma) \geq LD_{aj} \end{cases}$, $C_{bj}(\sigma) = C_{bi}(\sigma) + \begin{cases} u_{bj}, & \text{if } C_{bi}(\sigma) < LD_{bj} \\ v_{bj}, & \text{if } C_{bi}(\sigma) \geq LD_{bj} \end{cases}$, and $C_{cj}(\sigma) = \max\{C_{aj}(\sigma), C_{bj}(\sigma), C_{ci}(\sigma)\} + \begin{cases} u_{cj}, & \text{if } \max\{C_{aj}(\sigma), C_{bj}(\sigma), C_{ci}(\sigma)\} < LD_{cj} \\ v_{cj}, & \text{if } \max\{C_{aj}(\sigma), C_{bj}(\sigma), C_{ci}(\sigma)\} \geq LD_{cj} \end{cases}$.

This also notes that $C_{aj}(\sigma') = \begin{cases} s_a + u_{aj}, & \text{if } s_a < LD_{aj} \\ s_a + v_{aj}, & \text{if } s_a \geq LD_{aj} \end{cases}$, $C_{bj}(\sigma') = \begin{cases} s_b + u_{bj}, & \text{if } s_b < LD_{bj} \\ s_b + v_{bj}, & \text{if } s_b \geq LD_{bj} \end{cases}$, and $C_{cj}(\sigma') = \max\{C_{aj}(\sigma'), C_{bj}(\sigma'), s_c\} + \begin{cases} u_{cj}, & \text{if } \max\{C_{aj}(\sigma'), C_{bj}(\sigma'), s_c\} < LD_{cj} \\ v_{cj}, & \text{if } \max\{C_{aj}(\sigma'), C_{bj}(\sigma'), s_c\} \geq LD_{cj} \end{cases}$.

$C_{ai}(\sigma') = C_{aj}(\sigma') + \begin{cases} u_{ai}, & \text{if } C_{aj}(\sigma') < LD_{ai} \\ v_{ai}, & \text{if } C_{aj}(\sigma') \geq LD_{ai} \end{cases}$, $C_{bi}(\sigma') = C_{bj}(\sigma') + \begin{cases} u_{bi}, & \text{if } C_{bj}(\sigma') < LD_{bi} \\ v_{bi}, & \text{if } C_{bj}(\sigma') \geq LD_{bi} \end{cases}$, and $C_{ci}(\sigma') = \max\{C_{ai}(\sigma'), C_{bi}(\sigma'), C_{cj}(\sigma')\} + \begin{cases} u_{ci}, & \text{if } \max\{C_{ai}(\sigma'), C_{bi}(\sigma'), C_{cj}(\sigma')\} < LD_{ci} \\ v_{ci}, & \text{if } \max\{C_{ai}(\sigma'), C_{bi}(\sigma'), C_{cj}(\sigma')\} \geq LD_{ci} \end{cases}$.

To show that σ can dominates schedule σ', one needs to show that $C_{cj}(\sigma) < C_{ci}(\sigma')$.

The given conditions $v_{ai} < v_{aj}$and $v_{bi} < v_{bj}$, imply that $s_c > \max\{s_a + v_{aj}, s_b + v_{bj}\} > \max\{s_a + v_{ai}, s_b + v_{bi}\}$, thus apply the $s_a > \max\{LD_{ai}, LD_{aj}\}$, $s_b > \max\{LD_{bi}, LD_{bj}\}$, and $s_c > \max\{s_a + v_{ai}, s_b + v_{bi}\}$, one has that $C_{aj}(\sigma) = s_a + v_{ai} + v_{aj}$, $C_{bj}(\sigma) = s_b + v_{bi} + v_{bj}$, and $C_{ci}(\sigma) = s_c + \begin{cases} u_{ci}, & \text{if } s_c < LD_{ci} \\ v_{ci}, & \text{if } s_c \geq LD_{ci} \end{cases} = s_c + v_{ci}$. The last equation is from the condition $s_c > \max\{s_a + v_{aj}, s_b + v_{bj}\} \geq \min\{s_a + v_{aj}, s_b + v_{bj}\} > \max\{LD_{ci}, LD_{cj}\}$. Applying $\min\{s_a + v_{aj} + v_{ai}, s_b + v_{bj} + v_{bi}\} > s_c + v_{ci}$ and $s_c > \max\{LD_{ci}, LD_{cj}\}$, one has

$$C_{cj}(\sigma) = \max\{s_a + v_{ai} + v_{aj}, s_b + v_{bi} + v_{bj}\} + v_{cj} \quad (1)$$

Now apply the inequalities $s_a > \max\{LD_{ai}, LD_{aj}\}$, $s_b > \max\{LD_{bi}, LD_{bj}\}$, and $s_c > \max\{s_a + v_{aj}, s_b + v_{bj}\} \geq \min\{s_a + v_{aj}, s_b + v_{bj}\} > \max\{LD_{ci}, LD_{cj}\}$ on the schedule σ', one has $C_{cj}(\sigma') = s_c + v_{cj}$, $C_{ai}(\sigma') = s_a + v_{aj} + v_{ai}$, and $C_{bi}(\sigma') = s_b + v_{bj} + v_{bi}$. In addition, from $\max\{s_a + v_{aj} + v_{ai}, s_b + v_{bj} + v_{bi}\} > s_c + v_{ci} > s_c + v_{cj}$, one obtains that

$$C_{ci}(\sigma') = \max\{s_a + v_{aj} + v_{ai}, s_b + v_{bj} + v_{bi}\} + v_{ci} \quad (2)$$

Therefore, comparing Equation (1) with Equation (2), one has $C_{cj}(\sigma) = \max\{s_a + v_{ai} + v_{aj}, s_b + v_{bi} + v_{bj}\} + v_{cj} < \max\{s_a + v_{aj} + v_{ai}, s_b + v_{bj} + v_{bi}\} + v_{ci} = C_{ci}(\sigma')$. Thus, one has the required result, $C_{cj}(\sigma) < C_{ci}(\sigma')$.

Property 2: If two adjacent jobs i and j in schedules σ and σ' with the following conditions:$s_a + u_{aj} < \min\{LD_{ai}, LD_{aj}\}$,$s_b + u_{bj} < \min\{LD_{bi}, LD_{bj}\}$,$s_c > \max\{s_a + u_{aj}, s_b + u_{bj}\}$,$\max\{s_a + u_{aj} + u_{ai}, s_b + u_{bj} + u_{bi}\} < \min\{LD_{ci}, LD_{cj}\}$, $min\{s_a + u_{aj} + u_{ai}, s_b + u_{bj} + u_{bi}\} > s_c + u_{ci}$, $u_{ai} < u_{aj}$, $u_{bi} < u_{bj}$, and $u_{cj} < u_{ci}$, then schedule σ dominates schedule σ'.

Proof: Apply the conditions $s_a + u_{aj} < \min\{LD_{ai}, LD_{aj}\}$ and $u_{ai} < u_{aj}$, $s_b + u_{bj} < \min\{LD_{bi}, LD_{bj}\}$ and $u_{bi} < u_{bj}$, and $s_c > \max\{s_a + u_{aj}, s_b + u_{bj}\}$on the schedule σ, respectively, one has $C_{ai}(\sigma) = s_a + u_{aj}$, $C_{bj}(\sigma) = s_b + u_{bj}$, and $C_{ci}(\sigma) = s_c + \begin{cases} u_{ci}, & \text{if } s_c < LD_{ci} \\ v_{ci}, & \text{if } s_c \geq LD_{ci} \end{cases} \leq s_c + u_{ci}$ (because the $v_{ci} \leq u_{ci}$). Furthermore, one has $C_{aj}(\sigma) = s_a + u_{ai} + u_{aj}$, $C_{bj}(\sigma) = s_b + u_{bi} + u_{bj}$. Apply $min\{s_a + u_{aj} + u_{ai}, s_b + u_{bj} + u_{bi}\} > s_c + u_{ci}$, and $\max\{s_a + u_{aj} + u_{ai}, s_b + u_{bj} + u_{bi}\} < LD_{cj}$, one has that

$$C_{cj}(\sigma) = max\{s_a + u_{ai} + u_{aj}, s_b + u_{bi} + u_{bj}\} + u_{cj} \quad (3)$$

Now apply the inequalities $s_a + u_{aj} < \min\{LD_{ai}, LD_{aj}\}$, $s_b + u_{bj} < \min\{LD_{bi}, LD_{bj}\}$, and $s_c > \max\{s_a + u_{aj}, s_b + u_{bj}\}$ on the schedule σ', one has that $C_{aj}(\sigma') = s_a + u_{aj}$, $C_{bj}(\sigma') = s_b + u_{bj}$, and $C_{cj}(\sigma') = s_c + \begin{cases} u_{cj}, & \text{if } s_c < LD_{cj} \\ v_{cj}, & \text{if } s_c \geq LD_{cj} \end{cases} \leq s_c + u_{cj}$. Furthermore, $C_{ai}(\sigma') = s_a + u_{aj} + u_{ai}$, $C_{bi}(\sigma') = s_b + u_{bj} + u_{bi}$. In addition, apply the conditions $min\{s_a + u_{aj} + u_{ai}, s_b + u_{bj} + u_{bi}\} > s_c + u_{ci} (> s_c + u_{cj})$ and$\max\{s_a + u_{aj} + u_{ai}, s_b + u_{bj} + u_{bi}\} < LD_{ci}$, one has that

$$C_{ci}(\sigma') = max\{s_a + u_{aj} + u_{ai}, s_b + u_{bj} + u_{bi}\} + u_{ci} \quad (4)$$

Therefore, comparing Equation (3) with Equation (4), and the given $u_{cj} < u_{ci}$, one obtains that $C_{cj}(\sigma) < C_{ci}(\sigma')$.

Next, as an extension to the concept of constructing a lower bound proposed by Lee et al. [39], we propose one lower bound as follows:

$LB(\sigma) = \max\left\{s_c + \sum_{j \in US} v_{cj} + \min_{j \in US}\{\max\{s_c, s_a + v_{aj}, s_b + v_{bj}\} - s_c\}, \max\{\sum_{j=1}^{n} v_{aj}, \sum_{j=1}^{n} v_{bj}\} + \min_{j \in US}\{v_{cj}\}\right\}$, where $\sigma = (PS, US)$, and where s_a, s_b, s_c represent the completion times of the last jobs in the partial subsequence's *PS*.

The two dominance properties and the proposed lower bound are incorporated into the B&B algorithm to improve its computational efficiency. Specifically, the dominance properties eliminate partial schedules that cannot lead to better solutions, while the lower bound enables the pruning of branches that cannot improve the current best solution. Together, these mechanisms reduce the number of nodes explored and improve the efficiency of the B&B search.

5. Heuristics and metaheuristics

Heuristic methods are problem-solving techniques that leverage practical experience and intuitive approaches to find solutions rather than adhering to a rigid algorithmic process. Compared with more formal algorithms, these methods can often yield quicker results, which is especially advantageous when dealing with complex issues and constrained resources. Complex problems frequently lack simple solutions, making heuristic methods an efficient way to address them. Consequently, there are numerous valid reasons for employing heuristic techniques in addressing challenging problems.

A permutation-based representation is adopted for the proposed metaheuristic algorithms. Each solution is represented by a sequence of job indices, where each job appears exactly once. The same job order is used for the first-stage machines, while the second-stage assembly machine follows the corresponding job sequence, subject to the completion of both first-stage operations. For example, a chromosome (2, 4, 1, 3) represents the processing order of four jobs. The makespan is calculated by considering machine availability, assembly precedence constraints, and job- and machine-specific step-learning dates.

The Johnson rule optimally minimizes the makespan in a two-machine flow-shop scheduling problem. However, it cannot provide an optimal solution for the proposed TSASP with learning dates. Notably, the method has still been widely applied to solve many variants of the related flowshop scheduling models. Inspired by these observations, we consider a two-stage heuristic phase incorporating Johnson's rule. In phase one, we adopt to set $P_{1i} = (u_{ai} + u_{bi})/2$ and $P_{2i} = u_{ci}$, i$\in \Omega$ or $P_{1i} = (v_{ai} + v_{bi})/2$ and $P_{2i} = v_{ci}$, i$\in \Omega$ to produce an initial solution for each case; in phase two, we further apply a swapping scheme to improve the initial solution obtained from phase one. The details are summarized as follows:

Approximate Method 1:

Step 1: Set an initial seed including $\{u_{ai}, u_{bi}, u_{ci}$ i $\in \Omega = \{1, 2, \dots, \text{n}\}\}$.

Step 2: Input $P_{1i} = (u_{ai} + u_{bi})/2$ and $P_{2i} = u_{ci}$, where i$\in \Omega$.

Step 3: Perform the Johnson rule on the set of $\{u_{ai}, u_{bi}, u_{ci}$ i $\in \Omega = \{1, 2, \dots, \text{n}\}\}$ to obtain the final complete schedule σ

Step 4: reset σ as an initial seed σ_1 and the best schedule σ^*

Step 5: Do $i_1 = 1$ to n

Do $i_2 = 1$ to n

If $i_1 \neq i_2$ then

Swap the i_1 position job and the i_2 position job in σ_1 to generate a new job sequence σ_2.

Determine if the new sequence σ_2 is better than σ^*, then replace σ^* with σ_2.

Endif

End do

End do

Step 6: Output the final best solution σ^*.

Approximate Method 2:

Step 1: Set an initial seed including $\{v_{ai}, v_{bi}, v_{ci}$ i $\in \Omega = \{1, 2, \dots, \text{n}\}\}$.

Step 2: Input $P_{1i} = (v_{ai} + v_{bi})/2$ and $P_{2i} = v_{ci}$, where i$\in \Omega$.

The remaining steps are the same as those in Steps 3~6 of *Approximate Method 1*.

Additionally, three metaheuristic algorithms are employed to find high-quality approximate solutions for the proposed problem. Heuristic methods are generally noted for their simplicity and ease of construction. In contrast, metaheuristic algorithms are more complex and require the intelligent implementation of stochastic search strategies. The literature indicates that metaheuristic algorithms are widely utilized by domestic and international researchers to address combinatorial optimization problems. Notably, the simulated annealing (SA) algorithm, genetic algorithms (GAs), iterated greedy (IG) algorithms, and bat algorithm (BA) have been successfully applied to obtain high-quality approximate solutions for combinatorial problems (Iyer and Saxena [40]; Ribas et al. [41]; Ruiz and Stützle [42]; Shehab et al. [43]; Xiong and Xing [44]; Li et al. [45]; Li et al. [46]; Li et al. [47]; Shao et al. [48]). The details of the SA, GA, and IG algorithms are separately discussed below.

To solve the proposed problem, we adapt the SA algorithm. The implementation details are described below, and the corresponding flowchart is presented in Figure 1.

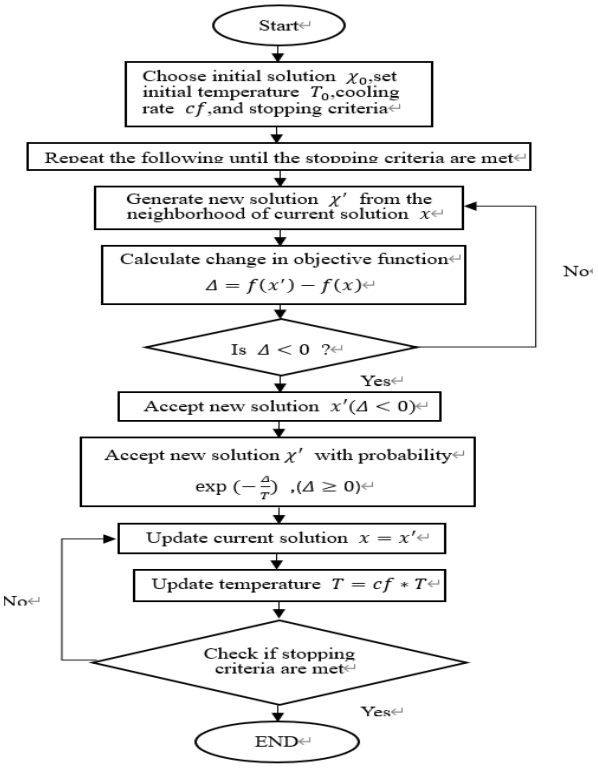


Figure 1. Flowchart of the simulated annealing algorithm.

Steps of the simulated annealing algorithm

Step 1. Initialization: Select an initial solution χ_0, set the initial temperature T_0, and determine the cooling rate α and stopping criteria.

Step 2. The following steps are repeated until the stopping criteria are met:

- **Generate a new solution**: Randomly generate a new solution x' from the neighbourhood of the current solution x.
- **Calculation of the objective function change**: Compute the change in the objective function value$\Delta E = f(x') - f(x)$

- **Accept a new solution**: Decide whether to accept the new solution on the basis of the following rules:
  - If $\Delta E < 0$, the new solution x' is better than the current solution; thus, accept the new solution.
  - If $\Delta E \geq 0$, accept the new solution x' with a probability of $e^{-\frac{\Delta E}{T}}$ (where T is the current temperature).
- **Update the temperature**: Update the current temperature according to the predetermined cooling rate $T = \alpha T$ (typically $0 < \alpha < 1$).
- **Check the stopping criterion**: If the stopping criterion is met (e.g., the temperature decreases to a minimum value or the maximum number of iterations is reached), stop the algorithm and return the current solution; otherwise, continue.

Step 3. Return optimal solution: At the end of the algorithm, return the final best solution.

The genetic algorithm adopts a permutation-based chromosome representation, where each gene represents a job and appears exactly once. For example, the chromosome (2, 4, 1, 3, 5, 6) specifies the job processing sequence. The second-stage assembly follows the same sequence, subject to the completion of both first-stage operations. The makespan is evaluated by considering machine availability, precedence constraints, and step-learning effects. The proposed GA utilizes a population-based search strategy. The corresponding flowchart is presented in Figure 2, and the key components are defined as follows.

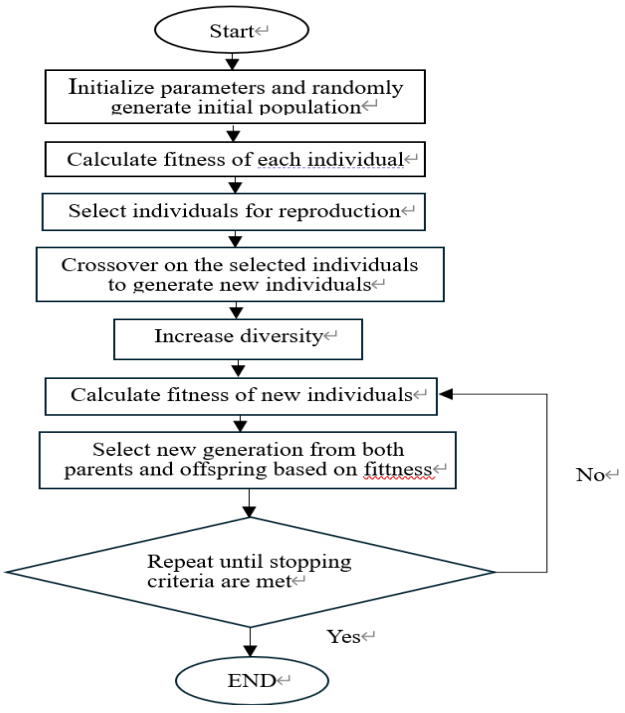


Figure 2. Flowchart of the heuristic-based genetic algorithm.

Steps of the heuristic-based genetic algorithm

Step 1: Initialization:

- Set initial parameters (such as the population size, crossover rate, and mutation rate).

- Input the best population obtained from the heuristics as the initial population.
- Perform a pairwise interchange to create an initial population.

Step 2: Evaluation: Calculate the fitness of each individual (evaluate the quality of the individuals according to the objective function).

Step 3: Selection: Select individuals for reproduction on the basis of their fitness via a roulette wheel selection.

Step 4: Crossover: Perform crossover operations on the selected individuals to generate new individuals (offspring), usually via single-point crossover, multipoint crossover, or uniform crossover methods.

Step 5: Mutation: Apply shift mutation operations to the crossover individuals to increase population diversity and prevent premature convergence.

Step 6: Evaluate the new population: Calculate the fitness of the newly generated individuals.

Step 7: Select a new generation: Select individuals with higher fitness values from the parent and offspring populations to form the new generation.

Repeat: Repeat steps 3–7 until the maximum number of generations is met.

Return results: When the algorithm ends, the best individual in the current population is returned as the final best solution.

The IG algorithm iterates through destruction and construction phases to improve the solution quality. The main steps and flowchart involved in the iterative greedy algorithm are as follows, and the corresponding flowchart is presented in Figure 3.

The details of the iterated greedy algorithm

Step 1: Initialization

- **Generate the initial solution**: Start by generating an initial solution. This can be done via a simple greedy approach or any other heuristic that provides a feasible starting point.
- **Evaluate the initial solution**: Evaluate the initial solution on the basis of a predefined objective function to determine its quality.

Step 2: Iteration

- **Destruction phase**: Remove a subset of elements from the current solution. The selection of elements to remove can be random, based on some criteria, or a combination of both.
- **Construction phase**: Reconstruct the solution by reinserting the removed elements or by adding new elements, aiming to improve the solution. This phase typically involves a greedy heuristic to ensure that the newly constructed solution is at least as good as the previous one.

Step 3: Acceptance criterion

- **Evaluate new solution**: Assess the quality of the newly constructed solution.
- **Update the best solution**: If the new solution is better than the current best solution, update the best solution.
- **Acceptance rule**: Decide whether to accept the new solution on the basis of predefined criteria. This could be based on improvement, probabilistic acceptance, or other heuristic rules.

Step 4: Termination

- **Check stopping condition**: Determine if the algorithm should terminate on the basis of stopping criteria such as a maximum number of iterations, a time limit, or convergence criteria (e.g., no improvement over a certain number of iterations).
- **Return the best solution**: Once the stopping condition is met, return the best solution found during the iterations.

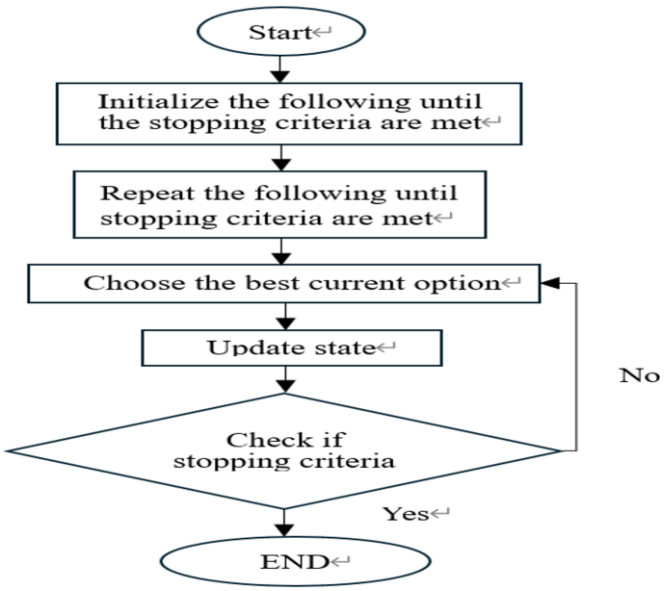


Figure 3. Flowchart of the iterated greedy algorithm.

6. Exploring the values of the parameters used in the SA, GA, and IG algorithm

6.1. Small-job case

To explore parameter optimization via the three algorithms, SA, the GA, and IG algorithm, we conduct experiments in a scenario with a small sample size (n = 12). We generate 100 randomly tested instances for this purpose. The experimental setup involves three machines, each with a fixed weight of $w_1 = 0.5, w_2 = 0.5,$ and $w_3 = 0.5$.

For each machine k and job j, the original processing times u_{kj} are generated from a uniform distribution U(51, 100). The processing times v_{kj} under the learning schedule are generated from a uniform distribution U(1, 50). Additionally, the learning dates LD_{kj} for job j are generated from a uniform distribution $U\left(1, w_k \sum_{j=1}^{n} u_{kj}\right)$, where k=1, 2, 3, and j=1,2..., n.

For the values of the parameters used in the SA algorithm, in the parameter exploration of the simulated annealing method, three variables are considered: initial temperature (T_i), cooling factor (cf.), and improvement repetitions (N_r).

First, T_i is varied, starting from 10^5 with an interval of 10, down to 10^{-5}, while keeping cf. fixed at 0.99 and Nr fixed at 20. The results in Figure 4 (a) show that the trend of AEP reaches its minimum value when T_i is 10^4. Therefore, T_i is set to 10^4, and the next step is to vary cf.

With T_i fixed at 10^4 and N_r fixed at 20, cf. is varied from 0.1 with an interval of 0.1 up to 0.9, and then from 0.91 with an interval of 0.01 up to 0.99. The results in Figure 4 (b) indicate that AEP quickly converges as cf. increases, reaching its minimum value at a cf. of 0.99. Therefore, cf. is set to 0.99.

Finally, N_r is varied while keeping T_i at 10^4 and cf. at 0.99. Nr is varied from 5 with an interval of 5 to 110. The results in Figure 4 (c) show that as N_r increases, AEP converges, reaching its minimum value at N_r = 60. Therefore, the final parameters are set to $T_i = 10^4$, cf. = 0.99, and N_r = 60.

For the values of the parameters used in the GA, in the parameter exploration of the genetic algorithm, three variables are considered: population size (nsize), mating pool size (gsize), and mutation rate (p).

First, nsize is varied, starting from 8 with an interval of 2 to 28, while gsize is fixed at 500 and p is fixed at 0.2. The results in Figure 4 (d) indicate that the AEP reaches its minimum value when nsize is 28. Therefore, nsize is set to 28, and the next step is to vary gsize.

With nsize fixed at 28 and p fixed at 0.2, gsize is varied from 400 with an interval of 50 to 1600. The results in Figure 4 (e) show that AEP reaches its minimum value when gsize is 1400. However, to keep the CPU time for the three algorithms at approximately 0.05, a relatively lower value of gsize = 700 is selected for further parameter exploration.

Finally, p is varied while keeping nsize at 28 and gsize at 700. p is varied from 0.01 with an interval of 0.01 to 0.40. The results in Figure 4 (f) indicate that as p increases, AEP converges, reaching its minimum value at p = 0.38. Therefore, this parameter value is selected.

In summary, the final parameters for the genetic algorithm are set to nsize = 28, gsize = 700, and p = 0.38.

For the values of the parameters used in the IG algorithm, in the parameter exploration of the greedy algorithm, three variables are considered: the temperature parameter (T), the number of iterations (No_repeat), and the number of recombinations (No_d).

First, T is varied, starting from 0.05 with an interval of 0.05 to 1, while No_repeat is fixed at 1000 and No_d is fixed at 4. The results in Figure 4 (g) show that AEP reaches its minimum value when T is 0.6. Therefore, T is set to 0.6 for the next step of varying No_repeat.

With T fixed at 0.6 and No_d fixed at 4, No_repeat is varied from 50 with an interval of 50 to 1000. The results in Figure 4 (h) indicate that AEP converges as the number of iterations increases, reaching its minimum value at a No_repeat of 950. Therefore, No_repeat is set to 950.

Finally, No_d is varied while keeping T at 0.6 and No_repeat at 950. No_d is varied from 1 with an interval of 1 to 8. The results in Figure 4 (i) show that AEP reaches its minimum value when No_d is 2. Therefore, No_d is set to 2.

In summary, the final parameters for the greedy algorithm are set to T = 0.6, No_repeat = 950, and No_d = 2.

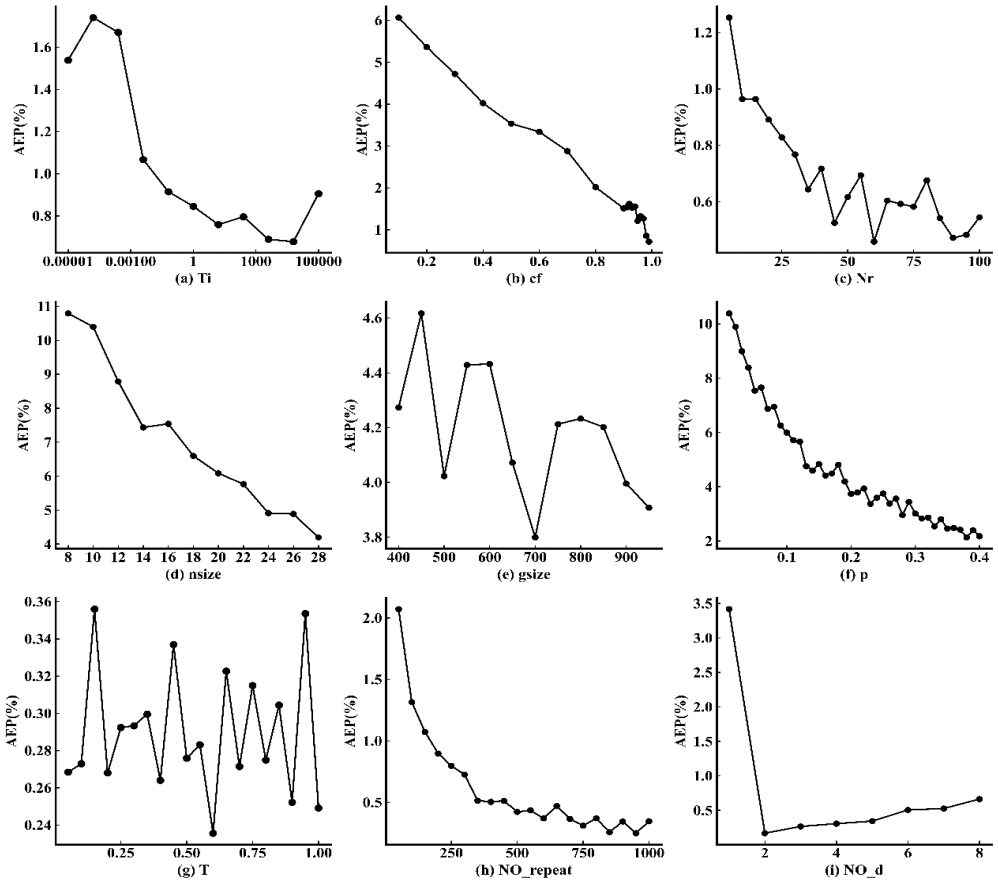


Figure 4. Exploring the values of parameters used in SA, GA, and IG (small job size).

6.2. Large number of jobs

To explore parameter optimization via the three algorithms, SA, GA, and IG, we conduct experiments in a scenario with a large number of jobs (n = 120). We also generate 100 randomly tested instances for this purpose. The experimental setup involves three machines, each with a fixed weight of $w_1 = 0.5, w_2 = 0.5,$ and $w_3 = 0.5$. The designs of each machine k and job j, along with the original processing times u_{kj}, processing times v_{kj}, and learning dates LD_{kj} for job j, are similar to the parameters used in the case of exploring small number of jobs.

In the parameter exploration for SA algorithm with a large n, only the improvement factor (Nr) is explored, whereas other variables (Ti and cf.) remain unchanged on the basis of our previous experimental tests. Therefore, the values of Ti and cf. obtained from the small n parameter exploration are subsequently used for the large number of jobs case. By varying Nr to explore the performance GAP of the large n, the values are sampled from 100 to 1200 with an interval of 50. The results in Figure 5(a) show that the value is lowest when Nr = 1200.

In the parameter exploration for a GA with a large n, the result for the parameter mutation rate p obtained from the small n parameter exploration is used. On the other hand, to match the CPU time of the other algorithms, the population size (gsize) is increased from 700 in the small number of jobs case to 1400 in the large number of jobs case. We vary the population size (nsize) to explore the performance of GAPs with a large n; nsize is sampled from 100 to 1200 with an interval of 50. The results in Figure

5 (b) show that the value is lowest when nsize = 100. Therefore, nsize = 100 is selected as the parameter exploration result.

In the parameter exploration for the IG algorithm with a large n, our focus is on exploring the performance of GAPs with a large n by varying the number of iterations (No_repeat). Parameter No_d is set to 20 by default, and further explanation of No_d will be provided later. By varying No_repeat, the values are sampled from 1000 to 2000 with an interval of 100. The results in Figure 5 (c) show that the value is lowest when No_repeat = 1800. Therefore, No_repeat = 1800 is chosen as the parameter exploration result.

In the case of a large n, we initially set No_d to 20. However, the performance of GAP not ideal. Therefore, as a second step, we explore the No_d values, starting from 2 and sampling every 2 units up to 26. Figure 5 (d) shows that the value is lowest when No_d = 2. This result is consistent with the findings from the small n parameter exploration. In the subsequent data simulation section, it is also demonstrated that using No_d = 2 yields a better relative percent deviation (RPD than No_d = 20 in the case of a large n, especially in scenarios with fewer resampling instances.

Finally, the effect of temperature parameter T is explored by varying T from 0.4 with an interval of 0.05 to 0.95. The results in Figure 5 (e) show that the lowest value is achieved when T is 0.80.

Therefore, the final parameters for the SA algorithm are set to $T_i = 10^4$, cf. = 0.99, and N_r =1200; the final parameters for the GA are set to nsize = 100, gsize = 700, and p = 0.38; and the final parameters for the IG algorithm are set to T = 0.8, No_repeat =1800, and No_d = 2 for the large job case.

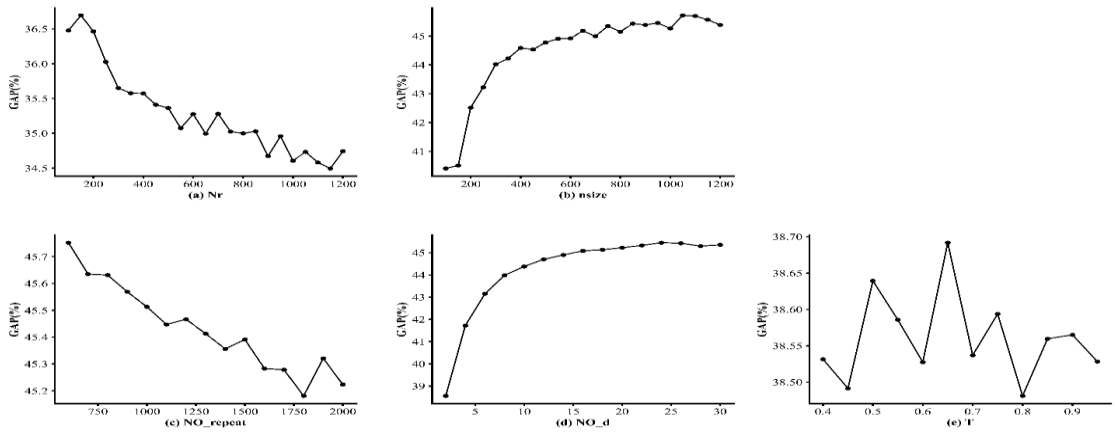


Figure 5. Exploring the values of parameters used in SA, GA, and IG (large job size).

7. Computational simulation results of all the proposed algorithms

In this section, we evaluate the efficacy of eight proposed algorithms, J3U_pi, J3V_pi, J3U_IG, J3V_IG, J3U_GA, J3V_GA, J3U_SA, J3V_SA, and the B&B algorithm, using two sets of test instances. These algorithms are implemented in Fortran and executed on a PC with a 3.00-GHz Intel Core i7-9700 CPU and 32.0 GB of RAM running Windows 10. The test instances comprise two sets: one with 8, 10, and 12 jobs (small cases) and another with 80, 100, and 120 jobs (large cases). Adopting

the similar designs of Xu et al. [22], the experimental setup includes three parameters, w_1, w_2, and w_3, for controllable parameters on three machines, with weights w_k set at 0.25, 0.5, and 0.75 for k = 1, 2, 3. The processing times u_{kj} are drawn from a uniform distribution U(51, 100), whereas the learning schedule times v_{kj} are drawn from U(1, 50). Learning dates LD_{kj} are generated from $U\left(1, w_k \sum_{j=1}^{n} u_{kj}\right)$. Each case includes 100 randomly generated instances, resulting in a total of 81 problem combinations and 8100 instances for small and large numbers of jobs cases. To ensure reproducibility, fixed random seeds are used to generate the test instances and control the stochastic components of the algorithms. All competing algorithms are evaluated using the same sets of problem instances to ensure fair and consistent comparisons.

The AEP and RPD are separately used to evaluate the performances of eight proposed algorithms for small and large numbers of jobs, where the AEP and RPD are defined as $100 * \sum[\frac{Hsol_i - BBsol_i}{BBsol_i} / 100]\%$ and $100 * \sum[\frac{Hsol_i - msol_i}{msol_i} / 100]\%$, respectively, the value $Hsol_i$ is obtained from the heuristics, $BBsol_i$ is produced from the B&B algorithm, and $msol_i$ is the smallest value among eight heuristics or metaheuristics. Additionally, the B&B algorithm is halted if the number of nodes searched exceeds 10^9, proceeding to the next instance in such cases.

Table 2. The performance of B&B at small size jobs.

			node	CPU-Time		
n	W_1	mean	max	mean	max	Total FS
8	0.25	909.3	7528.6	0.0053	0.0486	900
	0.5	791.0	6212.4	0.0040	0.0451	900
	0.75	1807.7	11453.0	0.0090	0.0608	900
10	0.25	37169.4	425439.4	0.1400	1.1758	900
	0.5	24090.6	349492.4	0.1015	1.2542	900
	0.75	54320.0	1405839.0	0.2083	2.1371	900
12	0.25	2058207.7	27236409.3	9.1206	113.2535	900
	0.5	1045904.7	14746970.8	4.9034	66.2257	900
	0.75	3230357.0	45222969.7	14.1293	186.1007	900
	W_2					900
8	0.25	1094.2	8017.7	0.0058	0.0521	900
	0.5	810.3	6533.7	0.0044	0.0399	900
	0.75	1603.5	10642.7	0.0081	0.0625	900
10	0.25	33615.3	360119.8	0.1289	1.1875	900
	0.5	23074.6	1046629.6	0.0959	0.9930	900
	0.75	58890.0	774021.6	0.2250	2.2559	900
12	0.25	1985991.0	24420781.9	8.3003	98.8143	900
	0.5	1469342.5	21664038.6	6.3408	87.9722	900
	0.75	2879135.8	41121529.3	13.5122	178.7934	900
	W_3					900
8	0.25	1393.8	9273.6	0.0068	0.0538	900
	0.5	889.8	7809.2	0.0046	0.0521	900
	0.75	1224.4	8111.2	0.0070	0.0486	900
10	0.25	43397.2	1355244.0	0.1672	1.9497	900
	0.5	33510.6	397804.6	0.1281	1.3333	900
	0.75	38672.1	427722.3	0.1545	1.4937	900
12	0.25	2544483.7	34445773.1	11.3553	146.5365	900
	0.5	1773529.3	27518744.9	7.8862	112.7309	900
	0.75	2016456.4	25241831.8	8.9117	106.3125	900
Total mean		717061.9		3.1802		

In this section, we first present the performance results of the branch-and-bound (B&B) algorithm, as shown in Table 2, alongside the eight heuristics or metaheuristics detailed in Table 2. The data in Table 1, specifically Columns 3 and 5, reveal that the B&B algorithm generally requires fewer mean nodes or less CPU time when w_1=0.5 (or $w_2 = 0.5$ or $w_3 = 0.5$) than the other values (0.25 and 0.75), regardless of the number of jobs n. Additionally, the mean number of nodes increases with n, with a significant increase observed as n approaches 12.

Column 7 in Table 2 shows that the number of feasible solutions (denoted as *FS*) remains constant at 900. This consistency indicates that the B&B algorithm can effectively solve the problem with a reasonable mean node count or CPU time for *n* less than 12. However, as *n* approaches 12, the mean number of nodes and the number of CPU seconds increase sharply, reflecting the typical behavior of NP-hard problems.

A comparison of the performance of the proposed heuristics (J3U_pi and J3V_pi) with that of the six metaheuristics (J3U_IG, J3V_IG, J3U_GA, J3V_GA, J3U_SA, and J3V_SA) for a small number of jobs (n=8, 10, and 12), as detailed in Table 3 and illustrated in Figure 6, reveals the following insights: On average, the AEP values for the J3U_pi and J3V_pi heuristics are 6.891 and 5.917, respectively. In comparison, the mean AEP values for the metaheuristics are 0.035 (J3U_IG), 0.030 (J3V_IG), 1.060 (J3U_GA), 1.045 (J3V_GA), 0.280 (J3U_SA), and 0.277 (J3V_SA).

Table 3. The performances of AEPs for two heuristics and six metaheuristics at a small job size.

		J3U_pi		J3V_pi		J3U_IG		J3V_IG		J3U_GA		J3V_GA		J3U_SA		J3V_SA	
n	W_1	mean	mean	mean	max	mean	max	mean	max	mean	max	mean	max	mean	max	mean	max
8	0.25	6.69	22.65	4.92	18.62	0.00	0.34	0.00	0.44	0.52	7.17	0.44	6.41	0.16	2.92	0.15	2.87
	0.5	6.54	22.50	5.48	19.03	0.00	0.30	0.00	0.48	0.51	6.46	0.58	7.35	0.20	3.85	0.23	3.80
	0.75	5.65	21.76	4.88	17.66	0.00	0.16	0.00	0.10	0.50	6.56	0.44	5.96	0.21	3.59	0.18	3.26
10	0.25	6.96	24.50	5.77	19.86	0.02	1.25	0.01	0.51	0.97	8.88	1.07	8.67	0.25	3.95	0.27	4.55
	0.5	7.34	23.46	6.24	19.16	0.04	1.93	0.02	1.53	1.23	7.65	1.22	7.61	0.41	4.70	0.34	4.03
	0.75	6.26	21.23	5.71	21.11	0.02	1.18	0.01	0.88	0.97	6.86	0.88	6.34	0.23	3.69	0.25	3.72
12	0.25	7.48	24.34	6.94	22.61	0.06	1.65	0.05	1.81	1.42	8.91	1.58	10.10	0.29	3.96	0.27	3.39
	0.5	8.46	21.51	7.17	19.97	0.12	2.29	0.12	2.34	2.03	10.27	1.95	9.65	0.46	4.41	0.47	4.16
	0.75	6.92	21.88	6.42	18.40	0.06	1.72	0.06	1.59	1.60	8.67	1.45	7.95	0.35	3.78	0.32	3.58
	W_2																
8	0.25	6.46	23.01	5.04	19.40	0.00	0.00	0.01	0.75	0.48	7.35	0.40	6.78	0.17	3.93	0.17	3.34
	0.5	6.46	21.31	5.23	16.82	0.00	0.29	0.00	0.18	0.57	6.23	0.60	6.96	0.22	3.35	0.22	3.51
	0.75	5.97	22.59	5.06	19.08	0.01	0.51	0.00	0.09	0.47	6.60	0.47	5.99	0.18	3.07	0.17	3.07
10	0.25	7.11	25.40	5.76	21.61	0.01	0.67	0.01	0.79	0.90	7.17	0.88	6.93	0.24	3.90	0.21	3.39
	0.5	7.20	23.67	6.38	18.90	0.03	1.86	0.02	1.16	1.25	9.07	1.30	8.11	0.35	4.32	0.32	4.18
	0.75	6.25	20.12	5.59	19.62	0.03	1.83	0.02	0.98	1.01	7.15	0.99	7.59	0.30	4.10	0.32	4.72
12	0.25	7.72	24.85	6.82	20.96	0.07	1.86	0.06	1.75	1.60	9.40	1.48	9.36	0.29	3.83	0.27	3.74
	0.5	8.03	22.11	7.26	20.32	0.12	2.18	0.11	2.49	1.88	9.04	1.92	9.74	0.43	4.04	0.48	4.09
	0.75	6.25	20.12	5.59	19.62	0.03	1.83	0.02	0.98	1.01	7.15	0.99	7.59	0.30	4.10	0.32	4.72
	W_3																
8	0.25	5.56	20.36	4.98	18.87	0.00	0.26	0.00	0.47	0.43	6.22	0.49	6.93	0.17	3.28	0.19	3.26
	0.5	6.75	22.90	5.32	18.90	0.00	0.43	0.00	0.48	0.63	7.32	0.57	6.80	0.20	3.76	0.20	3.61
	0.75	6.57	23.65	4.93	17.52	0.00	0.11	0.00	0.07	0.46	6.65	0.40	6.00	0.19	3.33	0.18	3.07
10	0.25	6.33	23.03	5.83	21.40	0.02	1.52	0.01	1.13	1.03	7.81	1.13	8.25	0.29	4.51	0.30	4.74
	0.5	7.31	25.95	6.07	18.92	0.04	1.75	0.02	1.13	1.15	7.97	1.20	7.55	0.34	4.36	0.32	4.40
	0.75	6.93	20.21	5.82	19.81	0.01	1.09	0.01	0.67	0.98	7.61	0.84	6.83	0.26	3.45	0.24	3.15
12	0.25	7.12	21.41	6.87	20.01	0.08	1.90	0.08	2.40	1.64	8.85	1.79	10.70	0.36	4.27	0.36	4.19
	0.5	8.20	23.28	7.19	20.17	0.10	2.01	0.09	1.81	1.87	10.02	1.85	9.34	0.40	4.19	0.40	3.79
	0.75	7.54	23.04	6.47	20.81	0.07	1.74	0.06	1.52	1.54	8.98	1.34	7.67	0.34	3.69	0.31	3.16
Total Mean		6.89		5.92		0.04		0.03		1.06		1.05		0.28		0.28	

Among these, the J3U_IG and J3V_IG algorithms demonstrate a slightly better performance than the J3U_SA and J3V_SA algorithms, and they also outperform the J3U_GA and J3V_GA algorithms. Overall, the J3U_IG, J3V_IG, J3U_GA, J3V_GA, J3U_SA, J3V_SA heuristics outperform the J3U_pi and J3V_pi heuristics.

Figure 6 presents a box plot illustrating the distributions of the AEP across the eight algorithms. The data indicate that the J3U_pi and J3V_pi heuristics present the widest range of values, as measured by the interquartile range (the difference between the third and first quartiles). The J3U_GA and J3V_GA heuristics, as well as the J3U_SA and J3V_SA heuristics, have intermediate ranges. In contrast, the J3U_IG and J3V_IG heuristics display the smallest range, indicating that they perform the most consistently and with the least variability.

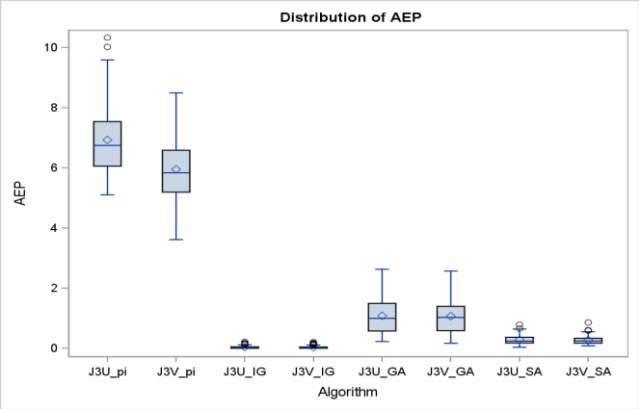


Figure 6. Box of distribution of AEP for eight proposed algorithms.

Table 4. The performances of RPDs for two heuristics and six metaheuristics at large job size.

		J3U_pi		J3U_pi		J3U_IG		J3V_IG		J3U_GA		J3V_GA		J3U_SA		J3V_SA	
n	W_1	mean	max	mean	max	mean	max	mean	max	mean	max	mean	max	mean	max	mean	max
80	0.25	11.71	22.22	11.67	22.01	1.62	5.51	1.60	5.18	2.89	6.83	2.97	6.93	0.21	3.01	0.19	2.86
	0.5	11.47	22.51	12.24	21.75	1.96	5.55	1.94	5.70	3.07	6.95	3.08	7.96	0.25	3.04	0.26	3.34
	0.75	10.38	20.35	11.47	23.25	1.44	5.14	1.42	5.15	2.87	6.56	2.90	6.59	0.19	3.51	0.21	3.43
100	0.25	11.98	22.01	12.14	22.55	1.74	5.51	1.69	5.27	3.40	7.65	3.44	7.40	0.23	3.74	0.21	3.03
	0.5	11.64	20.55	12.28	21.31	1.95	5.67	1.95	5.92	3.44	7.58	3.47	7.57	0.29	3.52	0.26	3.44
	0.75	10.59	20.21	12.01	23.27	1.57	4.76	1.58	5.04	3.38	6.80	3.34	7.23	0.24	3.27	0.26	3.91
120	0.25	12.37	21.93	12.45	21.00	1.78	5.12	1.76	5.42	3.86	7.59	4.00	7.98	0.24	3.45	0.25	3.25
	0.5	11.94	21.70	12.47	20.16	1.96	5.38	2.00	5.46	3.90	7.51	4.00	8.39	0.28	3.86	0.31	3.72
	0.75	10.85	22.19	12.61	24.28	1.65	4.73	1.64	5.19	3.74	7.07	3.76	7.67	0.26	3.28	0.28	2.99
	W_2																
80	0.25	11.69	22.39	11.89	22.80	1.53	5.15	1.52	5.18	2.84	7.04	2.97	7.38	0.18	2.44	0.19	2.98
	0.5	11.60	22.46	11.84	21.24	2.00	6.06	1.97	6.04	3.06	7.09	3.09	7.39	0.26	3.88	0.25	3.57
	0.75	10.27	20.23	11.65	22.96	1.50	4.99	1.46	4.80	2.93	6.20	2.90	6.73	0.21	3.24	0.21	3.09
100	0.25	11.91	21.38	12.07	20.79	1.74	5.36	1.75	5.49	3.45	7.48	3.48	7.88	0.25	3.76	0.25	3.07
	0.5	11.75	21.49	12.26	22.73	2.01	5.81	2.00	5.70	3.50	7.35	3.52	7.33	0.27	3.68	0.26	3.76
	0.75	10.54	19.90	12.11	23.61	1.50	4.77	1.48	5.03	3.27	7.20	3.25	6.99	0.23	3.09	0.23	3.56
120	0.25	12.27	21.92	12.63	21.00	1.76	4.83	1.75	5.39	3.89	7.66	3.98	8.16	0.23	3.28	0.25	3.18
	0.5	12.01	23.37	12.59	21.39	2.04	5.81	2.06	5.82	3.92	7.55	3.99	8.39	0.31	4.08	0.30	3.37
	0.75	10.87	20.53	12.31	23.06	1.60	4.59	1.60	4.85	3.69	6.96	3.78	7.49	0.24	3.24	0.29	3.40
	W_3																
80	0.25	11.11	20.17	12.67	23.36	1.63	5.13	1.64	4.99	3.07	6.69	3.23	7.15	0.21	3.07	0.23	3.31
	0.5	11.31	20.78	11.98	22.52	2.00	5.74	1.97	5.77	3.09	7.24	3.12	7.31	0.23	3.52	0.23	3.44
	0.75	11.14	24.12	10.73	21.12	1.39	5.33	1.34	5.26	2.66	6.40	2.60	7.03	0.21	2.97	0.20	2.88
100	0.25	11.64	20.71	13.16	23.18	1.76	5.19	1.77	5.40	3.71	7.49	3.71	7.85	0.28	3.71	0.25	3.81
	0.5	11.35	20.22	12.34	23.40	2.00	5.59	1.99	5.98	3.46	7.68	3.46	7.38	0.25	3.39	0.27	3.28
	0.75	11.22	21.85	10.93	20.54	1.50	5.16	1.46	4.84	3.04	6.86	3.08	6.97	0.23	3.42	0.21	3.30
120	0.25	11.80	21.84	13.52	22.00	1.78	4.96	1.76	4.91	4.19	7.80	4.30	8.51	0.28	3.51	0.27	2.97
	0.5	11.53	21.34	12.70	23.12	2.07	5.42	2.12	5.71	3.84	7.22	3.90	7.98	0.25	3.25	0.31	3.79
	0.75	11.82	22.63	11.32	20.33	1.54	4.85	1.52	5.44	3.47	7.16	3.55	7.55	0.25	3.83	0.26	3.20
Total mean		11.44		12.15		1.74		1.73		3.39		3.44		0.24		0.25	

To evaluate the performance of the RPDs for the two heuristics and six metaheuristics across a large number of jobs sets (n = 80, 100, and 120), the results are summarized in Table 4 and Figure 7. Table 4 shows that the J3U_SA and J3V_SA heuristics consistently achieve the lowest average RPDs, regardless of the value of n. Specifically, the average RPDs are as follows: for the J3U_pi and J3V_pi heuristics, the averages are 11.43 and 12.15; for the J3U_IG and J3V_IG heuristics, the averages are 1.742 and 1.731; for the J3U_GA and J3V_GA heuristics, the averages are 3.394 and 3.440; and for the J3U_SA and J3V_SA heuristics, the averages are 0.243 and 0.248.

Additionally, Figure 7 displays a box of the average RPDs for the two heuristics and six metaheuristics. The box plot reveals that the distributions of average RPDs for the J3U_SA and J3V_SA algorithms are more tightly clustered, with the smallest range and interquartile spread compared with those of the other methods.

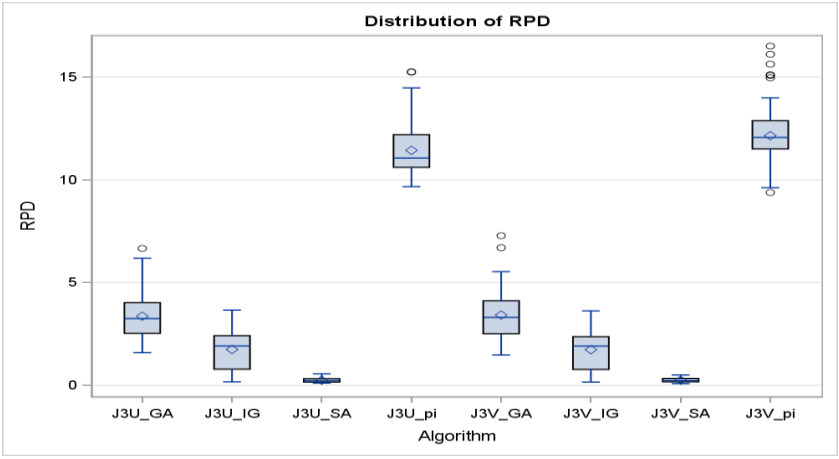


Figure 7. Box plot of distribution of RPDs for eight proposed algorithms.

A linear model is developed to analyze the differences between the two heuristics and six metaheuristics on the basis of the average AEP and RPD observations. To determine if these differences are statistically significant, an analysis of variance (ANOVA) is conducted via SAS 9.4. However, the assumption of normality for the linear model is not met. The Shapiro–Wilk test for normality yields a p value less than 0.0001, with test statistics of 0.9346 for the AEP and 0.9714 for RPD. These results indicate that the AEP and RPD samples do not follow a normal distribution at the significance level of $\alpha = 0.05$.

To further examine the pairwise differences among the two heuristics (J3U_pi and J3V_pi) and the six metaheuristics (J3U_IG, J3V_IG, J3U_GA, J3V_GA, J3U_SA, J3V_SA), we apply the WNMT (Wilcoxon, Nemenyi, McDonald, Thompson), procedure (Holland et al. 2014). Table 5, specifically Columns 3 and 5, displays the sums of the ranks for the AEP and RPD across 81 instance problems for each of these methods.

For the small number of jobs case, Table 5 shows that there is no statistically significant difference between the rank sums of J3U_pi (638.00) and J3V_pi (577.00) at the 0.05 significance level. Similarly, no significant differences are found between the rank sums of the three pairs J3U_IG (130.50) and J3V_IG (113.50), J3U_GA (446.00) and J3V_GA (440.00), and J3U_SA (283.00) and J3V_SA

(248.00). However, significant differences are observed among the pairs (J3U_pi, J3V_pi), (J3U_IG, J3V_IG), (J3U_GA, J3V_GA), and (J3U_SA, J3V_SA). Notably, the pair (J3U_IG, J3V_IG) has the smallest AEP and rank sum for the smaller number of jobs.

For the large number of jobs case, Table 5 indicates that at the 0.05 significance level, there is no statistically significant difference between the rank sums of J3U_pi (585.00) and J3V_pi (630.00). Similarly, no significant differences are observed among the pairs J3U_IG (289.00) and J3V_IG (278.00), J3U_GA (436.00) and J3V_GA (455.00), and J3U_SA (121.00) and J3V_SA (122.00). However, significant differences are noted among the pairs (J3U_pi, J3V_pi), (J3U_IG, J3V_IG), (J3U_GA, J3V_GA), and (J3U_SA, J3V_SA). Notably, the pair (J3U_SA, J3V_SA) has the smallest RPD and rank sum for large number of jobs.

Table 5. Sum of ranks for two heuristics and six metaheuristics.

Algorithm	No. of obs./ Small n	Rank-sum R_i, $(i = 1,2,\dots,8)$	No. of obs./ Large n	Rank-sum R_i, $(i = 1,2,\dots,8)$
$J3U_pi$	81	638.00	81	585.00
$J3V_pi$	81	577.00	81	630.00
$J3U_IG$	81	130.50	81	289.00
$J3V_IG$	81	113.50	81	278.00
$J3U_GA$	81	446.00	81	436.00
$J3V_GA$	81	444.00	81	455.00
$J3U_SA$	81	283.00	81	121.00
$J3V_SA$	81	248.35	81	122.00

*Note that any two algorithms i and j are significant different if $|R_i - R_j| \geq 95$ for small n, and if $|R_i - R_j| \geq 95$ (for large n) base on the WNMT test

Finally, with respect to reporting the computational times of the eight methods, Figure 8 presents box plots illustrating the CPU times (in seconds) for the two heuristics and six metaheuristics with a large number of jobs. Compared with the other methods, the J3U_IG and J3V_IG algorithms result in significantly longer execution times. For small number of jobs instances, the CPU time measurements are not provided, as each algorithm completes its execution in less than one second.

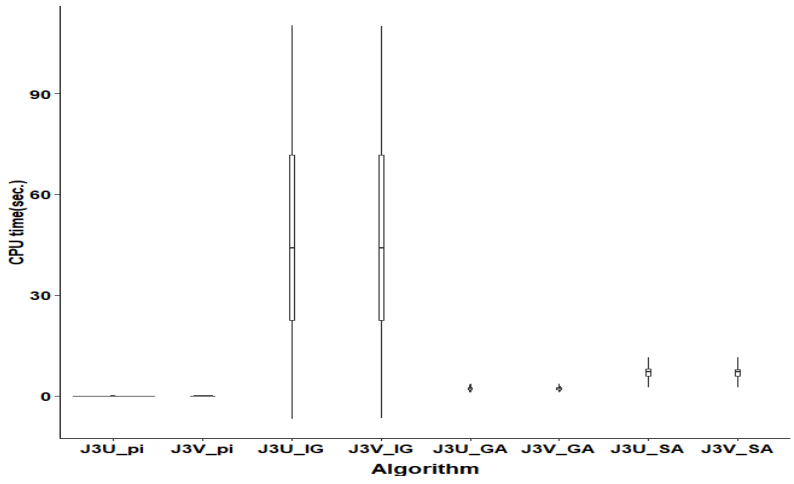


Figure 8. Box plot of CPU times (in sec.) for eight algorithms at a large job case.

8. Conclusions and suggestions

In a two-stage flow shop assembly scheduling problem, jobs typically progress sequentially through two stages or sets of operations. The main challenge is to determine the optimal sequence for processing these jobs at both stages to achieve the desired performance outcomes. A key objective is often to minimize the maximum completion time for all jobs, which is commonly known as the makespan.

However, the impact of the step learning effect in a two-stage flow shop assembly setting has not been extensively explored. The step-learning effect refers to improvements that occur at specific intervals or steps in a process rather than continuously throughout the process. Motivated by this gap, we investigate the two-stage flow shop assembly scheduling problem with the goal of minimizing the makespan.

To address this problem, we propose several approaches: a branch-and-bound (B&B) method with two properties and a lower bound, along with two heuristics (J3U_pi and J3V_pi) and six metaheuristics (J3U_IG, J3V_IG, J3U_GA, J3V_GA, J3U_SA, J3V_SA). These methods are evaluated for small and large numbers of job cases.

Our results indicate that the B&B method was effective in finding optimal solutions for up to 12 jobs. For the small number of jobs cases, the J3U_IG and J3V_IG heuristics provided high-quality near-optimal solutions with the smallest average error percentage (AEP) among the eight methods. In contrast, the J3U_SA and J3V_SA metaheuristics achieved the lowest relative deviation percentage (RDP) for the large job case. Furthermore, our analysis revealed that the differences in performance between the J3U_IG and J3V_IG, J3U_GA heuristics, and J3V_GA, and J3U_SA and J3V_SA heuristics were not statistically significant at the 0.05 level for the small and large numbers of job cases.

In future studies, researchers could extend the proposed model to consider multiple scheduling objectives, such as minimizing a weighted combination of makespan and mean completion time. Another promising direction is to incorporate learning dates into two-agent or distributed two-stage assembly flowshop scheduling environments (Xiong and Xing [44]). Furthermore, systematic comparisons with MILP solvers and lower bounds could be conducted to provide a more comprehensive analysis of the optimality gaps of the proposed algorithms. Finally, sensitivity analyses of key model parameters, including learning dates, processing-time reductions, and job sizes, could be performed to evaluate the robustness and practical applicability of the proposed methods.

Author contributions

Chin-Chia Wu and Juin-Han Chen conceived the research problem and developed the scheduling model. Xin-Gong Zhang and Chao-Chung Kang contributed to the theoretical analysis and algorithm development. Ching-Yuan Huang and Jian-You Xu assisted with algorithm implementation, numerical experiments, and data analysis. Jatinder N.D. Gupta contributed to the research methodology. Chin-Chia Wu and Juin-Han Chen prepared the original manuscript. All authors reviewed, edited, and approved the final version of the manuscript.

Acknowledgments

The authors sincerely thank the editor and three anonymous referees for their valuable comments and constructive suggestions. This work was supported in part by National Science and Technology Council of Taiwan (NSTC) under Grant No. NSTC 115-2410-H-035 -026 and NSTC 112-2221-E-035-060-MY2, in part by the Natural Science Foundation of Chongqing under Grant No. CSTB2023NSCQ-LZX0005, and in part by the National Natural Science Foundation of China, grant number 72271048.

Statements and Declarations

- ✓ Data Availability: The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.
- ✓ Code Availability: The Fortran programming codes used in this study are available from the corresponding author on reasonable request.

Use of Generative-AI tools declaration

The authors declare that generative artificial intelligence (AI) tools were used in the preparation of this manuscript.

AI tools used: ChatGPT (OpenAI).

How were the AI tools used? The AI tool was used solely for English language polishing, grammatical corrections, and improving the clarity and readability of the authors' original text. It was not used to generate research data, develop mathematical models, conduct numerical experiments, or draw scientific conclusions. All AI-assisted modifications were carefully reviewed and finalized by the authors.

Where in the article is the information located? Throughout the manuscript, primarily for linguistic polishing and improving the clarity and consistency of the presentation.

Conflict of interest

The authors have no relevant financial or non-financial interests to disclose.

References

1. D. Biskup, Single-machine scheduling with learning considerations, *Eur. J. Oper. Res.*, **115** (1999), 173–178. https://doi.org/10.1016/S0377-2217(98)00246-X
2. C. Koulamas, G. J. Kyparisis, Single-machine and two-machine flowshop scheduling with general learning functions, *Eur. J. Oper. Res.*, **178** (2007), 402–407. https://doi.org/10.1016/j.ejor.2006.01.030

3. W. H. Kuo, D. L. Yang, Minimizing the total completion time in a single-machine scheduling problem with a time-dependent learning effect, *Eur. J. Oper. Res.*, **174** (2006), 1184–1190. https://doi.org/10.1016/j.ejor.2005.03.020
4. T. C. E. Cheng, G. Wang, Single machine scheduling with learning effect considerations, *Ann. Oper. Res.*, **98** (2000), 273–290. https://doi.org/10.1023/A:1019216726076
5. J. B. Wang, Single-machine scheduling problems with the effects of learning and deterioration, *Omega*, **35** (2007), 397–402. https://doi.org/10.1016/j.omega.2005.07.008
6. Y. Yin, D. Xu, K. Sun, H. Li, Some scheduling problems with general position-dependent and time-dependent learning effects, *Inf. Sci.*, **179** (2009), 2416–2425. https://doi.org/10.1016/j.ins.2009.02.015
7. J. B. Wang, J. J. Wang, Single machine scheduling with sum-of-logarithm-processing-times based and position-based learning effects, *Optim. Lett.*, **8** (2014), 971–982. https://doi.org/10.1007/s11590-012-0494-4
8. Y. P. Niu, L. Wan, J. B. Wang, A note on scheduling jobs with extended sum-of-processing-times-based and position-based learning effect, *Asia-Pac. J. Oper. Res.*, **32** (2015), 1550001. https://doi.org/10.1142/S0217595915500013
9. A. Azzouz, M. Ennigrou, L. Ben Said, Scheduling problems under learning effects: classification and cartography, *Int. J. Prod. Res.*, **56** (2018), 1642–1661. https://doi.org/10.1080/00207543.2017.1355576
10. D. Biskup, A state-of-the-art review on scheduling with learning effect, *Eur. J. Oper. Res.*, **188** (2008), 315–329. https://doi.org/10.1016/j.ejor.2007.05.040
11. A. Janiak, T. Krysiak, R. Trela, Scheduling problems with learning and ageing effects: a survey, *Decis. Mak. Manuf. Serv.*, **5** (2011), 19–36. https://doi.org/10.7494/dmms.2011.5.1.19
12. C. C. Wu, J. Y. Chen, W. C. Lin, K. Lai, S. C. Liu, P. W. Yu, A two-stage three-machine assembly flow shop scheduling with learning consideration to minimize the flowtime by six hybrids of particle swarm optimization, *Swarm Evol. Comput.*, **41** (2018), 97–110. https://doi.org/10.1016/j.swevo.2018.01.012
13. W. C. Lin, Minimizing the makespan for two-stage three-machine assembly flow shop problem with the sum-of-processing-time based learning effect, *Discrete Dyn. Nat. Soc.*, **2018** (2018), 8170294. https://doi.org/10.1155/2018/8170294
14. Y. Zou, D. J. Wang, W. C. Lin, J. Y. Chen, P. W. Yu, W. H. Wu, et al., Two-stage three-machine assembly scheduling problem with sum-of-processing-times based learning effect, *Soft Comput.*, **24** (2020), 5445–5462. https://doi.org/10.1007/s00500-019-04301-y
15. A. Azzouz, P. A. Pan, P. H. Hsu, W. C. Lin, S. C. Liu, L. Ben Said, et al., A two-stage three-machine assembly scheduling problem with a truncation position-based learning effect, *Soft Comput.*, **24** (2020), 10515–10531. https://doi.org/10.1007/s00500-019-04561-8
16. C. C. Wu, X. Zhang, A. Ameni, W. L. Shen, S. R. Cheng, P. H. Hsu, et al., Metaheuristics for two-stage flow-shop assembly problem with a truncation learning function, *Eng. Optim.*, **53** (2021), 843–866. https://doi.org/10.1080/0305215X.2020.1757089
17. J. Wang, D. Lei, J. Cai, An adaptive artificial bee colony with reinforcement learning for distributed three-stage assembly scheduling with maintenance, *Appl. Soft Comput.*, **117** (2022), 108371. https://doi.org/10.1016/j.asoc.2021.108371

18. M. Rostami, A. Yousefzadeh, A gamified teaching-learning based optimization algorithm for a three-echelon supply chain scheduling problem in a two-stage assembly flow shop environment, *Appl. Soft Comput.*, **146** (2023), 110598. https://doi.org/10.1016/j.asoc.2023.110598
19. W. Lin, J. Chen, Deep reinforcement learning for dynamic scheduling of two-stage assembly flowshop, in: *Advances in Swarm Intelligence*, Springer, (2021), 248–258. https://doi.org/10.1007/978-3-030-78811-7_25
20. K. C. Ying, S. W. Lin, K. F. Chen, Minimising makespan in distributed assembly hybrid flowshop scheduling problems, *Int. J. Prod. Res.*, **63** (2025), 1674–1691. https://doi.org/10.1080/00207543.2024.2383781
21. E. S. Kim, D. Oron, Minimizing total completion time on a single machine with step improving jobs, *J. Oper. Res. Soc.*, **66** (2015), 1481–1490. https://doi.org/10.1057/jors.2014.91
22. J. Y. Xu, W. C. Lin, Y. W. Chang, Y. H. Chung, J. H. Chen, C. C. Wu, A two-machine learning-date flowshop scheduling problem with heuristics and population-based GA to minimize the makespan, *Mathematics*, **11** (2023), 4060. https://doi.org/10.3390/math11194060
23. M. Atsmony, B. Mor, G. Mosheiov, Single machine scheduling with step-learning, *J. Sched.*, **27** (2024), 227–237. https://doi.org/10.1007/s10951-022-00763-5
24. J. Song, C. Miao, F. Kong, Scheduling with step learning and job rejection, *Oper. Res. Int. J.*, **25** (2025), 6. https://doi.org/10.1007/s12351-024-00887-w
25. J. Phosavanh, D. Oron, Minimizing total weighted late work with step-learning on a single machine, *Discrete Appl. Math.*, **391** (2026), 277–293. https://doi.org/10.1016/j.dam.2026.04.045
26. Y. Jiang, H. Lin, J. Dong, T. C. E. Cheng, M. Ji, Single-machine scheduling with learning dates, *Eur. J. Oper. Res.*, (2026). https://doi.org/10.1016/j.ejor.2026.08.033
27. M. Kong, Y. Zhang, J. Xu, W. Wang, S. Lu, A. M. Fathollahi-Fard, A green scheduling model for two-stage photo-etching and acid-etching collaboration in semiconductor manufacturing, *J. Ind. Inf. Integr.*, **41** (2024), 100655. https://doi.org/10.1016/j.jii.2024.100655
28. H. Kazemi, M. Nourelfath, A. M. Fathollahi-Fard, M. Gendreau, Integrated two-stage multi-factory assembly scheduling with maintenance considerations, *Int. J. Prod. Res.*, **63** (2025), 2133–2149. https://doi.org/10.1080/00207543.2024.2396512
29. S. Lu, X. Zhang, M. Kong, A. M. Fathollahi-Fard, Two-stage hybrid flow shop scheduling with sequence-dependent setup times in semiconductor manufacturing: a customized variable neighborhood search, *Ann. Oper. Res.*, (2025). https://doi.org/10.1007/s10479-025-06541-8
30. D. Y. Lv, J. B. Wang, Research on two-machine flow shop scheduling problem with release dates and truncated learning effects, *Eng. Optim.*, **57** (2025), 1828–1848. https://doi.org/10.1080/0305215X.2024.2372633
31. K. F. Zhang, J. F. Zhang, J. B. Wang, Single-machine linear resource-dependent consumption scheduling with DeJong's time-dependent learning effect, *J. Ind. Manag. Optim.*, **22** (2026), 4422–4448. https://doi.org/10.3934/jimo.2026154
32. J. M. Gu, J. B. Wang, Minimising total completion time in no-wait flow shop scheduling with learning effects and group technology, *J. Oper. Res. Soc.*, (2026). https://doi.org/10.1080/01605682.2026.2721567

33. J. M. Gu, L. Lin, J. B. Wang, The no-wait flow shop problem with makespan objective and learning effects, *J. Ind. Manag. Optim.*, **22** (2026), 5275–5307. https://doi.org/10.3934/jimo.2026182
34. Y. Y. Lu, M. H. Li, A unified analysis for single-machine scheduling with resource allocations, learning and deteriorating effects simultaneously, *Comput. Appl. Math.*, **45** (2026), 388. https://doi.org/10.1007/s40314-026-03784-4
35. D. Y. Lv, Y. B. Wu, Single-machine just-in-time scheduling problem with learning/deterioration effects and past-sequence-dependent delivery times, *J. Oper. Res. Soc.*, (2026). https://doi.org/10.1080/01605682.2026.2731156
36. L. Y. Wang, Due-window assignment scheduling problems with position-dependent weights, truncated learning effects and past-sequence-dependent setup-times, *Symmetry*, **18** (2026), 396. https://doi.org/10.3390/sym18030396
37. L. H. Zhang, J. B. Wang, P. Ji, Group parallel-batching scheduling with position-dependent learning and time-dependent deterioration effects, *Comput. Appl. Math.*, **46** (2027), 45. https://doi.org/10.1007/s40314-026-03899-8
38. R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Discrete Math.*, **5** (1979), 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X
39. C. Y. Lee, T. C. E. Cheng, B. M. T. Lin, Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem, *Manage. Sci.*, **39** (1993), 616–625. https://doi.org/10.1287/mnsc.39.5.616
40. S. K. Iyer, B. S. Saxena, Improved memetic genetic algorithm for the permutation flowshop scheduling problem, *Comput. Oper. Res.*, **31** (2004), 593–606. https://doi.org/10.1016/S0305-0548(03)00016-9
41. I. Ribas, R. Companys, X. Tort-Martorell, An iterated greedy algorithm for the flowshop scheduling problem with blocking, *Omega*, **39** (2011), 293–301. https://doi.org/10.1016/j.omega.2010.07.007
42. R. Ruiz, T. Stützle, An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, *Eur. J. Oper. Res.*, **187** (2008), 1143–1159. https://doi.org/10.1016/j.ejor.2006.07.029
43. M. Shehab, M. A. Abu-Hashem, M. K. Y. Shambour, A. I. Alsalibi, O. A. Alomari, J. N. D. Gupta, et al., A comprehensive review of bat inspired algorithm: variants, applications, and hybridization, *Arch. Comput. Methods Eng.*, **30** (2023), 765–797. https://doi.org/10.1007/s11831-022-09817-5
44. F. Xiong, K. Xing, Meta-heuristics for the distributed two-stage assembly scheduling problem with bi-criteria of makespan and mean completion time, *Int. J. Prod. Res.*, **52** (2014), 2743–2766. https://doi.org/10.1080/00207543.2014.884290
45. Y. Z. Li, Q. K. Pan, J. Q. Li, L. Gao, M. F. Tasgetiren, An adaptive iterated greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems, *Swarm Evol. Comput.*, **63** (2021), 100874. https://doi.org/10.1016/j.swevo.2021.100874
46. H. Li, X. Li, L. Gao, An iterated greedy algorithm with acceleration of job allocation probability for distributed heterogeneous permutation flowshop scheduling problem, *Swarm Evol. Comput.*, **88** (2024), 101580. https://doi.org/10.1016/j.swevo.2024.101580

47. Q. Y. Li, Q. K. Pan, L. Gao, H. Y. Sang, X. X. Zhang, W. M. Li, The constrained permutation flowshop problem: an effective two-stage iterated greedy algorithm to minimize weighted tardiness, *Swarm Evol. Comput.*, **91** (2024), 101696. https://doi.org/10.1016/j.swevo.2024.101696
48. Z. Shao, W. Shao, D. Pi, Effective heuristics and metaheuristics for the distributed fuzzy blocking flow-shop scheduling problem, *Swarm Evol. Comput.*, **59** (2020), 100747. https://doi.org/10.1016/j.swevo.2020.100747



AIMS Press

© 2026 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0)