Research Article

Minimizing the Makespan for a Two-Stage Three-Machine Assembly Flow Shop Problem with the Sum-of-Processing-Time Based Learning Effect

Win-Chin Lin

Department of Statistics, Feng Chia University, Taichung 40724, Taiwan

Correspondence should be addressed to Win-Chin Lin; linwc@fcu.edu.tw

Received 30 January 2018; Accepted 8 April 2018; Published 23 May 2018

Academic Editor: Manuel De la Sen

Copyright © 2018 Win-Chin Lin. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Two-stage production process and its applications appear in many production environments. Job processing times are usually assumed to be constant throughout the process. In fact, the learning effect accrued from repetitive work experiences, which leads to the reduction of actual job processing times, indeed exists in many production environments. However, the issue of learning effect is rarely addressed in solving a two-stage assembly scheduling problem. Motivated by this observation, the author studies a two-stage three-machine assembly flow shop problem with a learning effect based on sum of the processing times of already processed jobs to minimize the makespan criterion. Because this problem is proved to be NP-hard, a branch-and-bound method embedded with some developed dominance propositions and a lower bound is employed to search for optimal solutions. A cloud theory-based simulated annealing (CSA) algorithm and an iterated greedy (IG) algorithm with four different local search methods are used to find near-optimal solutions for small and large number of jobs. The performances of adopted algorithms are subsequently compared through computational experiments and nonparametric statistical analyses, including the Kruskal–Wallis test and a multiple comparison procedure.

1. Introduction

This study considers a two-stage assembly scheduling problem consisting of \( n \) jobs. Each of the \( n \) jobs has two parts (components) to be processed first, and the parts are then assembled on one assembly operation into an ordered product. At the first stage, two distinct parts can be fabricated at the same time in two dedicated machines, \( M_1 \) and \( M_2 \). Once the parts are finished, they are transmitted to the second stage, the assembly stage. There is a single assembly machine, \( M_3 \), in this stage to assemble parts into the ordered product. The assembly work can only start after the two parts are completed and delivered to the assembly line. The objective of the production plan is to minimize the maximum completion time among all the jobs, i.e., to minimize the makespan.

Wu et al. [1] studied the makespan minimization in the two-stage three-machine flow shop problem with position-based learning consideration based on Biskup [2]. Wu et al. [3] addressed a two-stage three-machine flow shop scheduling problem with a cumulated learning function to minimize the flowtime (or total completion time). They developed a branch-and-bound algorithm and proposed six versions of hybrid particle swarm optimization (PSO) to solve it. However, the sum-of-processing-times-based learning model (based on [4]) has yet to be considered in two-stage assembly flow shop environment. In light of the lack of studies on the sum-of-processing-times-based learning model, this research studies a two-stage three-machine assembly problem with a learning effect based on sum of the processing times of already processed jobs to minimize the makespan criterion.

The paper proceeds as follows. Section 2 presents some notations and problem definition. Section 3 provides some dominant properties and a lower bound to be used in a branch-and-bound method and then introduces a cloud theory-based simulated annealing (CSA) and four versions of the iterated greedy (IG) algorithm. Section 4 provides computational results. Section 5 is the conclusion.
1.1. Literature Survey. Invoking a real-world example of fire engine manufacturing and assembly procedure, Lee et al. [5] studied the issue of minimizing the makespan in the three-machine assembly-type flow shop scheduling problem. A fire engine consists of two parts, a body and a chassis, which are manufactured separately in two production lines in the plant; an engine, bought from outside, can be considered ready. Once the body and chassis are completed, they will be transmitted to the assembly line, where the end product, a fire engine, is built up. They proved the problem is strongly NP-complete, discussed some special polynomial-time solvable cases, and constructed exact and heuristics for them. Potts et al. [6] also demonstrated the complexity of the proposed problem with several machines in the first stage. They introduced several heuristic algorithms and presented the worst-case analysis of performances of these algorithms. Hariri and Potts [7] developed a branch-and-bound method, embedded with some dominance properties, for the considered problem with several machines in the first stage. Sun et al. [8] proposed 14 heuristics algorithms to solve the problem.

Allahverdi and Al-Anzi [9] proposed two evolutionary algorithms, a particle swarm optimization (PSO) and a tabu search, and a simple efficient algorithm to solve the considered problem. Koulamas and Kyparisis [10] studied a two-stage assembly flow shop scheduling problem with uniform parallel machines on one or both stages. They presented an asymptotically optimal heuristic to minimize the makespan, as the number of jobs increase. Their results improved earlier results in some situations. Fattahi et al. [11] considered the same problem but all machines in the first (possibly including two manufacturing stages) stage(s) can process all kinds of parts, named a hybrid flow shop in the first stage(s). They proposed several heuristics based on Johnson's rule and evaluated the performance of them. Fattahi et al. [12] also considered the hybrid flow shop problem but with setup time and presented a hierarchical branch-and-bound algorithm with several lower bounds and upper bounds. Hatami et al. [13] addressed a distributed assembly permutation scheduling problem in which the first stage comprised several identical flow shop factories. These factories, each with a permutation flow shop scheduling problem, produced parts which were later assembled by a single machine in the assembly stage. Komaki et al. [14] addressed a two-manufacturing-stage hybrid flow shop scheduling problem and one assembly stage to minimize the makespan, where the hybrid meant identical parallel machines or dedicated machines could present in the fabrication stages. They presented a lower bound, several heuristics, and two metaheuristic algorithms based on an artificial immune system. Their computational results showed the outperformance of the proposed lower bound and heuristic algorithms. Jung et al. [15] considered a slightly different two-stage assembly scheduling problem. A single manufacturing machine was used in the first stage to produce various types of parts which were assembled into various products (ordered by customers) in the second stage. The aforementioned studies addressed the problem with respect to the makespan criterion.

With regard to performance criteria other than the makespan for two stages or more stages assembly flow shop problems, Allahverdi and Aydilek [16] provided research works focusing on the following criteria: mean (or total) completion time, a weighted sum of mean completion time and makespan, maximum lateness, total tardiness, and bicriteria. Some of these studies were together with other machine settings, for example, setup times often treated as separate ones from processing times or due dates required by customers. For instance, Maleki-Darouknolaei et al. [17] minimized the weighted mean completion time and makespan for the proposed problem with sequence-dependent setup times and blocking times. For more studies on different criteria and different machine settings for two-stage or three-stage assembly flow shop problem, readers can refer to Al-Anzi and Allahverdi [18], Allahverdi and Al-Anzi [19], Al-Anzi and Allahverdi [20], Allahverdi and Al-Anzi [21], Fattahi et al. [11], Allahverdi and Aydilek [16], and Jung et al. [15], among others.

Several real-life examples or applications for two-stage and three-stage assembly problem can be found in Lee et al. [5], Potts et al. [6], Tozkapan et al. [22], Allahverdi and Al-Anzi [19], Al-Anzi and Allahverdi [20], Sung and Kim [23], Allahverdi and Al-Anzi [24], Cheng et al. [25], Fattahi et al. [11], Mozdgir et al. [26], Fattahi et al. [12], Hatami et al. [13], and Jung et al. [15], among others.

On the other hand, the learning concept has been widely addressed in different branches of industry, for example, product development, job rotation, and job scheduling problem [27]. The first application in scheduling literature appeared in airplane’s manufacturing, where the assembly costs reduced as the times of repetitions increased [28]. Yelle [29] gave a comprehensive survey on learning curves applied in many industries including manufacturing sector. In addition to the excellent review on scheduling with learning effect by Biskup [30], Anzanello and Fogliatto [27] also gave a review on the wide applicability of learning curve models in production systems. The work of Azzouz et al. [31] is a more recent survey paper in the scheduling literature providing applications of learning effects.

In scheduling techniques, Biskup [2] and Cheng and Wang [32] were some of the early researchers who proposed modeling a learning effect by a learning curve (function). Since then, many other learning curves (functions) in connection with a single machine, flow shop, or other scheduling problems, have been developed and modified.

Kuo and Yang work [4], one of the extensively discussed studies on the scheduling research community, proposed the sum-of-processing-times-based learning model, in which the real processing time of a job is relevant to the sum of the previous work times. Actually, the sum-of-processing-times based learning model is relevant to a long-winded and uninterrupted production process in which initial steps are set up. In manufacturing steel plates or bars by a foundry, the products are required to be inspected for surface defects. A single inspector is regarded as a processor or a system to work on those steel bars (jobs). After the processor finishes some work units (jobs), he will spend less time on processing follow-up work units as his experiences grow. Thus, Donderi and Mohanty [33] modeled the real processing time of a job as a function of the sum of the processing times of the already processed jobs.
Further modifications of the sum-of-processing-times based learning effect in scheduling problems were proposed by Cheng et al. [34], Yin et al. [35], Rudek [36], Wu et al. [37], Lu et al. [38], Wu and Wang [39], and Pei et al. [40], among others. The readers may refer to Azzouz et al. [31] for other commonly discussed learning models: position-based learning, truncated learning, and exponential learning.

2. Notation Definition and Problem Formulation

At the outset, we define some notations and then describe the proposal problem.

2.1. Notations

\( n \): total number of jobs

\( m = 3 \): total number of machines

\( J_i \): code for the \( i \)th job, \( i = 1, 2, \ldots, n \)

\( \omega \): set of \( n \) jobs

\( M_l \): machine code \( l, l = 1, 2, 3 \)

\( S \) and \( Sw \): complete job schedules (sequences)

\( S_1, S_2, S_A \), and \( S_J \): partial job schedules (sequences)

\( d \): number of jobs removed from a sequence \( S \)

2.2. Input Parameters

\( p_i, q_i, r_i \): component processing times of job \( J_i \) on machines \( M_1, M_2, \) and \( M_3 \), respectively, for \( i = 1, 2, \ldots, n \)

\( a \): learning index and \( a < 0 \)

\( T_0 \): initial temperature

\( T_f \): final temperature

\( \lambda \): annealing index

2.3. Decision Variables

\( T_{k_l} \): starting time of the \( k \)th job (in \( S \)) on machine \( M_l, l = 1, 2, 3 \)

\( C_{i_l}(S) \): completion time of job \( J_i \) on machine \( M_l, i = 1, 2, \ldots, n; l = 1, 2, 3 \)

\( C_{i_l}(S) \): completion time of the job in position \( i \) on machine \( M_l, l = 1, 2, 3 \)

\( C_{\text{max}}(S) \): completion time for the last job in \( S \) or makespan or \( C_{3_j[n]}(S) \)

2.4. Problem Formulation. We assume that a set of \( n \) jobs, \( \omega \), are available at time zero and will be processed on fixed three machines, \( M_1, M_2, \) and \( M_3 \). Each job \( J_i \) has three components to be processed on all three machines in three operations. Two components are first processed in parallel on machines \( M_1 \) and \( M_2 \) and then on machine \( M_3 \) for the assembly operation. Machines \( M_1 \) and \( M_2 \) have no idle times at the first stage of operation. A job to be processed on machine \( M_3 \) cannot start until it has been processed on machines \( M_1 \) and \( M_2 \). No machine can process more than one job at a time. Furthermore, preemption is not allowed; i.e., a job cannot be interrupted once it has begun processing. Sum of the component processing times on machines \( M_1, M_2, \) and \( M_3 \) are \( p_i(1 + \sum_{j=1}^{k-1} q_{ij})^a, q_i(1 + \sum_{j=1}^{k-1} q_{ij})^a, \) and \( r_i(1 + \sum_{j=1}^{k-1} l_{ij})^a, \) respectively, where \( a \) is a learning index and \( a < 0 \).

For the convenience of reading, we further let \( P_k = 1 + \sum_{j=1}^{k-1} P_{ij}, Q_k = 1 + \sum_{j=1}^{k-1} q_{ij}, R_k = 1 + \sum_{j=1}^{k-1} r_{ij}, F_k = (1 + \sum_{j=1}^{k-1} p_{ij})^a, Q_k^a = (1 + \sum_{j=1}^{k-1} q_{ij})^a, \) and \( R_k^a = (1 + \sum_{j=1}^{k-1} r_{ij})^a \).

Now, for a schedule \( S \), the completion time \( C_{3_j[k]}(S) \) of a job \( J_i \) to be scheduled on \( k \)th position on machine \( M_1 \) in \( S \) is calculated as \( C_{3_j[k]}(S) = \max\{T_{1_k} + p_i P_k, T_{2_k} + q_i Q_k, T_{3_k} + r_i R_k\} \), where \( T_{1_k}, T_{2_k}, \) and \( T_{3_k} \) denote the starting times of a job \( J_i \) (job here) to be assigned to the \( k \)th position on machines \( M_1, M_2, \) and \( M_3 \), respectively, in \( S \). The objective is to find a schedule \( S^* \) that minimizes the makespan; i.e., \( C_{\text{max}}(S^*) = \min_S \{C_{3_j[1]}(S), C_{3_j[2]}(S), \ldots, C_{3_j[n]}(S)\} \). For the Gantt chart of illustrating the considered two-stage three-machine environment, readers can refer to Lee et al. [5].

3. Solution Approach

A branch-and-bound (B&B) method is used to solve the proposed problem for small number of jobs with \( n = 8, 9, 10, 11 \). Several dominance properties and a lower bound (LB) are developed and incorporated into the branch-and-bound method to increase the search efficiency. Since the proposed problem is NP-hard, to solve the problem for large number of jobs, we utilize a cloud theory-based simulated annealing (CSA) algorithm and an iterated greedy (IG) algorithm with four different local search methods. It has been shown [7, 22, 23, 41] that the permutation schedules, in which the jobs are processed in the same order in all machines, are dominant with respect to the makespan criterion. Consequently, we limit our research to permutation schedules for the proposed problem. Using the three-field classified system in scheduling literature (see [42]), the proposed problem can be represented as AF2(2,1)/prmu, LE\( r / C_{\text{max}} \).

3.1. Dominance Properties and a Lower Bound. Dominance rules are commonly used to improve search efficiency of heuristics in the scheduling research, for example, Lee et al. [5], Potts et al. [6], and Tozkapan et al. [22]. In this section, we will develop some dominance (elimination) properties and a LB embedded in a B&B method for finding optimal solutions. Let \( S = \{S_1, i, j, S_2\} \) and \( S' = \{S_1, j, i, S_2\} \) denote two sequences in which \( S_1 \) and \( S_2 \) are partial job sequences (or empty) and there are at most \((k - 2)\) scheduled jobs in \( S_1, 1 \leq k \leq n \). To show \( S \) dominates \( S' \), it suffices to show that \( C_{3_j[S]}(S) < C_{3_j[S']} \).

In the following, we construct several dominance rules to assist in eliminating nodes in the B&B method. Two lemmas are provided to shorten the proof of the properties. The proofs of Lemma 1 and Properties 3 to 9 are given in the Appendix.

**Lemma 1.** If \( 0 < x \leq y, 0 < C, \) and \( a < 0, \) then \( x(C + y)^a \leq y(C + x)^a \).
Note that if $0 < x < y$, $0 < C$, and $a < 0$, then $x(C + y)^a > y(C + x)^a$.

**Lemma 2.** If $1 \leq \lambda, 0 \leq t$, and $a < 0$, then $0 \leq \lambda[1 - \left(1 + t\right)^y] - [1 - (1 + \lambda t)^y]$.

Note that Lemma 2 has already been proved by Kuo and Yang [4].

**Property 3.** If $p_i < p_j$, $q_i < q_j$, and $r_j < r_i$, $T_{3k} > \max[T_{1k} + p_i P_{i}^a, T_{2k} + q_i Q_{i}^a, T_{1k} + p_i P_{i}^a + p_j(R_i + p_j)^a > \max[T_{2k} + q_j Q_j^a + q_j(Q_k + q_j)^a, T_{3k} + r_{i} R_{i}^a]$, and $T_{3k} + p_i P_i^a + p_j(P_k + p_j)^a > \max[T_{2k} + q_j Q_j^a + q_j(Q_k + q_j)^a, T_{3k} + r_{i} R_{i}^a]$, then $S$ dominates $S'$.

**Property 4.** If $p_i < p_j$, $q_i < q_j$, and $r_j < r_i$, $T_{3k} > \max[T_{1k} + p_i P_i^a, T_{2k} + q_i Q_i^a, T_{2k} + q_i Q_i^a + q_j(Q_k + q_j)^a > \max[T_{1k} + p_i P_i^a + p_j(P_k + p_j)^a, T_{3k} + r_{i} R_{i}^a]$, and $T_{3k} + q_i Q_i^a + q_j(Q_k + q_j)^a > \max[T_{1k} + p_i P_i^a + p_j(P_k + p_j)^a, T_{3k} + r_{i} R_{i}^a]$, then $S$ dominates $S'$.

**Property 5.** If $p_i < p_j$, $q_i < q_j$, and $r_j < r_i$, $T_{3k} > \max[T_{1k} + p_i P_i^a + p_j(P_k + p_j)^a, T_{2k} + q_i Q_i^a + q_j(Q_k + q_j)^a]$, and $r_{i} R_{i}^a > \max[p_j(P_k + p_j)^a, q_j(Q_k + q_j)^a, r_{i} R_{i}^a]$, then $S$ dominates $S'$.

**Property 6.** If $p_i < p_j$, $r_j < r_i$, $T_{1k} + p_i P_i^a > \max[T_{2k} + q_j Q_j^a, T_{2k} + q_j Q_j^a + q_i Q_i^a + q_j Q_j^a]$, and $p_j(P_i + p_i)^a > \max[q_j(Q_k + q_j)^a, q_i(Q_k + q_j)^a, r_{i} R_{i}^a]$, then $S$ dominates $S'$.

**Property 7.** If $q_i < q_j$, $r_j < r_i$, $T_{3k} + q_j Q_j^a > \max[T_{1k} + p_i P_i^a, T_{1k} + p_i P_i^a + p_j(R_i + p_j)^a, T_{3k} + q_j Q_j^a + q_i Q_i^a]$, and $q_j(Q_k + q_j)^a > \max[p_j(P_k + p_j)^a, p_j(P_k + p_j)^a, r_{i} R_{i}^a]$, then $S$ dominates $S'$.

**Property 8.** If $p_i < p_j$, $q_i < q_j$, $r_j < r_i$, $T_{3k} + q_j Q_j^a > \max[T_{2k} + q_j Q_j^a + q_i Q_i^a, T_{3k} + q_j Q_j^a]$ and $r_{i} R_{i}^a > \max[p_j(P_k + p_j)^a, q_j(Q_k + q_j)^a, r_{i} R_{i}^a]$, then $S$ dominates $S'$.

The next property follows the same idea of Kuo and Yang [4], Theorem 1 thereof, but it is modified to accommodate the learning situation. Let $S = (S_A, S_t)$, where $S_A$ has $(k - 1)$ jobs, $1 \leq k \leq n$, and $S_t$ has $(n - k + 1)$ jobs. Let $r_{i(k)} = r_{i(k+1)} = \ldots = r_{i(n)}$ represent nondecreasing orders of processing times in jobs in $S_t$ for Machine 3.

**Property 10.** If $T_{3k} \geq \max[T_{1k} + \sum_{i=0}^{n} p_i(1 + \sum_{j=1}^{k-1} P_{i[j]}^a), T_{2k} + \sum_{i=0}^{n} q_i] + \sum_{i=1}^{k} q_i], (1 + \sum_{j=1}^{k} q_{i[j]}^a), T_{3k} + r_{i} R_{i}^a]$, then $(S_A, S_t)$ dominate $(S_A, S_t)$, where $S_{A(k)}$ is the partial schedule according to nondecreasing order of the processing times of $S_t$ in Machine 3. (Note that the complexity of Property 10 is $O(n \log(n))$.

As a job being placed in a schedule on the $k$th position, $1 \leq k \leq n$, a LB, based on similar idea of Lee et al. [5], can be acquired as follows.

\[
LB(S) = \max \left\{ T_{3k} + \sum_{i=0}^{n} p_i(1 + \sum_{j=0}^{i-1} r_{i[j]}^a) \right\} \\
+ \min_{i \in S_t} \left\{ \max \left\{ T_{3k} + T_{1k} + p_i \left( 1 + \sum_{j=0}^{k-1} P_{i[j]}^a \right), T_{2k} + q_j \left( 1 + \sum_{j=0}^{k-1} q_{i[j]}^a \right) \right\} - T_{3k} \right\} \\
\times \left( \sum_{i=1}^{n} p_i(1 + \sum_{j=0}^{i-1} p_{i[j]}^a), \sum_{i=1}^{n} q_i(1 + \sum_{j=0}^{i-1} q_{i[j]}^a) \right) + \min_{i \in S_t} \left\{ r_i \left( 1 + \sum_{j=1}^{n} r_j - r_i \right)^a \right\}.
\]

Note that $T_{11} = 0$, $T_{21} = 0$, $T_{31} = 0$, $p_{0(i)} = 0$, $q_{0(i)} = 0$, and $r_{0(i)} = 0$.  

For the branch-and-bound (B&B) method in this study, we employ the depth-first search [43] in the branching procedure: a branch is selected and explored systematically until it is pruned out or its end node is reached. The B&B method allocates jobs starting from the first position ($k = 1$) and in the forward to last position ($k = n$) allocation. To enhance the performance of the B&B method, the starting sequence (schedule) found by the algorithm IGLS3 (see more details in the next subsection) is considered as an incumbent sequence. Then, the depth-first search method is executed from the initial node and its descents from the tree, to renew the incumbent sequence. Apply Properties 3 to 9 to eliminate branches. If the LB of unsearched nodes is greater than or equal to the makespan of the incumbent sequence than eliminate the node and all nodes sprouting from it in the branch. For active nodes, Property 10 are applied to examine whether the order of unscheduled jobs can be allocated. Repeat the procedure until all nodes are explored.

The above B&B method is also used to evaluate the effectiveness of the adopted algorithms, a cloud-based annealing algorithm, and an iterated greed algorithm for small number of jobs.

For the study problem without learning consideration, AF2(2,1)//C_max has been shown to be NP-hard (see [5]), so does AF2(2,1)/pmumax,LF//C_max. In order to search for good quality solutions and shorten the computer CPU times,
we utilize a metaheuristic cloud theory-based simulated annealing (CSA) algorithm, Li et al. [44] and Li and Du [45], and four versions of the iterated greedy (IG) algorithm, Ruiz and Stützle [46], for (near-) optimal solutions. The details are introduced in the following subsections.

3.2. A Cloud Theory-Based Simulated Annealing Algorithm. The simulated annealing algorithm (SA), a random optimization method, was first proposed by Kirkpatrick [47]. SA has been frequently and successfully adopted in combinatorial optimization problems because of its capability of escaping from local minimum in the searching process. SA simulates the process of the physical annealing to the solution of an optimization problem. It starts by constructing an initial solution, \( S_0 \), from a high initial temperature, \( T_0 \), and selects a solution \( S_1 \) randomly in the neighborhood of \( S_0 \). Whether \( S_1 \) is accepted as an incumbent solution depends on the temperature and on \( C_{\text{max}}(S_0) \) and \( C_{\text{max}}(S_1) \), values of the objective function and the makespan. If \( C_{\text{max}}(S_1) \leq C_{\text{max}}(S_0) \), then \( S_1 \) is accepted and replaces \( S_0 \); otherwise, \( S_1 \) can also be accepted with a probability \( \exp[-(C_{\text{max}}(S_1) - C_{\text{max}}(S_0))/T] \). As the temperature \( T \) decreases gradually, SA repeats the above process and converges to a global solution (in theory) or finds a local solution at the end of the process. Allahverdi and Al-Anzi [21] successfully applied SA on an assembly flow shop problem. Although SA has been successfully applied to solve several optimization problems, it has also been found unable to solve some combinatorial optimization problems. Readers can refer to the reference listed in Boussaïd et al. [48].

The basic concept of the cloud theory has been proposed by Li et al. [44] and Li and Du [45]. Torabzadeh and Zandieh [49] proposed a cloud theory-based simulated annealing (CSA) algorithm for the 2-stage \( m \)-machine assembly flow shop scheduling problem with a bicriteria objective function. They showed that their CSA performed better than the best existing SA [21]. Studying the same scheduling problem but with learning consideration and a different criterion, we then adopt the CSA for the problem. For more details about CSA, readers may refer to Li and Du [45] and Lv et al. [50].

For the CSA to search better quality solutions, one local heuristic based on the idea of Johnson’s rule [51] was constructed. The resultant solution from the heuristic was then put in CSA as an initial solution. The Johnson-based local search heuristic, coded as \( J_{\text{mean}} \), is as follows.

The \( J_{\text{mean}} \) local search heuristic is shown as follows.

(1) For each \( J_{t,i} \), \( i = 1, 2, \ldots, n \), let \( A_{li} = (p_i + q_i)/2 \) and \( B_{li} = r_i \) be the processing times \( M_1 \) and \( M_2 \), \( N \), = \( \{J_{t,1}, J_{t,2}, \ldots, J_{t,n}\} \).

(2) Let \( s = 1, t = n \).

(3) (i) If \( \min\{A_{11}, A_{12}, \ldots, A_{16}, B_{21}, B_{22}, \ldots, B_{6n}\} = (\text{say}) A_{t,i} \), then assign \( J_{t,i} \) to the \( t \)th position and delete \( J_{t,i} \) from \( N \). Set \( s = s + 1 \), and go to (4).

(ii) If \( \min\{A_{11}, A_{12}, \ldots, A_{16}, B_{21}, B_{22}, \ldots, B_{6n}\} = (\text{say}) B_{t,i} \), then assign \( J_{t,i} \) to the \( t \)th position and delete \( J_{t,i} \) from \( N \). Set \( t = t - 1 \), and go to (4).

(4) Delete \( J_{t,i} \) from \( N \), if \( N \) is not empty, and then go to (3); otherwise, stop.

Note that the total computational complexity of \( J_{\text{mean}} \) algorithm, which is the same as that of Johnson’s rule, is \( O(n^2) \). The parameters aroused in CSA such as initial temperature \( T_0 \), final temperature \( T_f \), and annealing index \( \lambda \) need to be determined. We referred to the values of the corresponding parameters used in Allahverdi and Aydilek [16] and performed a pilot study similar to the study in Wu et al. (2017). Finally, we adopt the \( T_0 = 0.1 \), \( T_f = 10^{-5} \), and \( \lambda = 0.98 \). The procedure of CSA is as in Algorithm 1. In the CSA procedure (Algorithm 1), \( E_p \), the range of the cloud), \( H_p \) (the cloud drops’ dispersive degree), and a given certain value \( u_0 \) are parameters from a normal distribution in the cloud theory-based simulated annealing algorithm [50]. In addition, the number of iterations \( N_r \) can be raised to enhance the searching capability of CSA [16].

3.3. Iterated Greedy Algorithm. The iterated greedy (IG) algorithm was proposed by Ruiz and Stützle [46]. The great advantages of this method are as follows: it is simple to implement and has been confirmed to yield high-quality solutions for flow shop problems [52]. The IG has been extensively employed for the last ten years in the scheduling research, especially in the (permutation or hybrid) flow shop problem, for example, Pan and Ruiz [53], Msakni et al. [54], Dubois-Lacoste et al. [55], and Pan et al. [56]. In this study, we propose an IG algorithm with four different local search methods, i.e., four versions of IG, coded as IGLS1, IGLS2, IGLS3, and IGLS4.

When performing the IG, one randomly generates an initial solution and uses it repeatedly to execute destruction and construction cycles until a stopping criterion is reached [46]. In the destruction stage, \( d \) jobs are removed from the incumbent solution \( S \) to form a new partial sequence \( S_P \) with \( n - d \) jobs. Let \( S_R \) be the sequence with a size of \( d \) jobs created that contain the removed jobs according to the order of their appearances in \( S \). In the construction stage, we choose the jobs of \( S_R \) to be reinserted in the partial solution \( S_P \) using the Nawaz-Enscore-Ham (NEH) mechanism [57]. Each job in \( S_R \) has an assigned order, and the job with the first order is inserted in all possible positions of \( S_P \). The position with the best objective value is held. The same iteration is repeated for all remaining jobs in \( S_R \). Note that the complexity of NEH algorithm is \( O(n^3m) \).

Similar to Ruiz and Stützle [46], the IGLS algorithms utilize an acceptance criterion to determine whether a new solution should be kept or not. It adopts \( W = T' \cdot [\sum(p_i + q_i + r_i)] / (100 \cdot n \cdot m) \) as the temperature in the IG algorithm, where \( T' \) is a controllable variable with \( 0 < T' < 1 \). The procedure of IG algorithm is described as in Algorithm 2.

We used four different neighborhood structures as local search paths after the construction stage in the IG algorithm. The insertion neighborhood structure, LS3, is described in step (2) of the IG algorithm. The neighborhood structures for LS1, LS2, and LS4 are described as follows. Let \( S = (I_1, I_2, \ldots, I_n) \) be a solution (i.e., a sequence of \( n \) jobs) and \( C_{\text{max}}(S) \) be the objective value, i.e., the makespan, of the S.
Input initial and final temperature, \(T_0, T_f\), and annealing index \(\lambda\).
Randomly generate a solution, improved by (Johnson's rule based) \(J_{\text{mean}}\), then by a pairwise interchange, denote the solution \(S_0\). Calculate \(C_{\text{max}}(S_0)\).

\[
\text{Do while } \{T > T_f\} \quad \text{Output the final solution } S_{\text{best}}
\]

Set \(S_{\text{out}} = S_0\; r = 0\), and \(T = T_0\).

Let \(E_n = T, H_n = T\), and \(u_0 = 1 - T\).

Let \(E_n^* = \max[E_n + H_n - 1/3 \cdot \text{rand}(0, 1), \text{eps}]\) where \(\text{eps} < 10^{-8}\).

Let \(T^* = E_n^* \cdot \sqrt{-2 \ln(u_0)}\).

Set \(K = 0\).

\[
\text{Do while } \{K < N_\lambda\} \quad \text{Input initial and final temperature, } (T_0, T_f)\text{, a predetermined number of iterations is set at 20}
\]

Choose \(i\) and \(j\) randomly with \(i, j \in \{1, 2, \ldots, n\}\) and \(i \neq j\). Swap the \(i\)th job and \(j\)th job in \(S_0\) to obtain a new solution \(S_1\).

Calculate the makespans \(C_{\text{max}}(S_0)\) and \(C_{\text{max}}(S_1)\).

If \(C_{\text{max}}(S_1) < C_{\text{max}}(S_0)\), then replace \(S_0\) by \(S_1\); else, replace \(S_0\) by \(S_1\) when

\[
\exp[-(C_{\text{max}}(S_1) - C_{\text{max}}(S_0))/(C_{\text{max}}(S_0) \cdot T^*)] < U, \quad \text{where}
\]

\[
U = \text{rand}(0, 1).
\]

Let \(K = K + 1\).

\[
\text{End while}
\]

Let \(r = r + 1\) and \(T = T_0 \cdot \lambda^r\).

\[
\text{End while}
\]

Output a final solution \(S_{\text{out}}\).

**Algorithm 1: Procedure of the proposed CSA.**

**Initialization:**
Randomly generate an initial solution \(S\).

Improve \(S\) by a local search (e.g. LS3). The LS3 is as follows.

1. Randomly select one job among the \(n\) jobs.
2. Insert the selected job to \(n\) possible positions, keep the best (the lowest \(C_{\text{max}}\)) solution.
3. Repeat the steps until all \(n\) jobs have been selected and inserted back; update the best solution if better solution is found.
4. Output the final solution, \(S_0\).

Set \(S_0 = S_1\) and \(S_{\text{out}} = S_1\).

\[
\text{Do while } \{\text{iter} < M_{\text{max}}\} \quad \text{(a predetermined number of iterations } M_{\text{max}} \text{ is set equal to 30)}
\]

**Destruction stage.**
Divide \(S_1\) into \(S_D\) and \(S_R\), where \(S_D\) is a set of \(d\) removed jobs and \(S_R\) a set of the remaining \((n-d)\) jobs;

**Construction stage**
For the removed \(d\) jobs, execute the NEH mechanism to \(S_R\) one after another until it yields the best sequence, coded as \(S_2\).

Improve \(S_2\) by one of the four local search methods, LS1, LS2, LS3, LS4, coded as \(S_3\).

**Acceptance criterion**
If \(C_{\text{max}}(S_3) < C_{\text{max}}(S_1)\), then update \(S_1\) by \(S_3\);

If \(C_{\text{max}}(S_3) < C_{\text{max}}(S_{\text{best}})\), then Update \(S_{\text{best}}\) by \(S_3\);

Else if \(q \leq \exp(C_{\text{max}}(S_2) - C_{\text{max}}(S_1))/W\) then Update \(S_1\) by \(S_2\);

(Note that \(q = \text{rand}(0,1)\)).

**End while**

Output the final solution \(S_{\text{best}}\).

**Algorithm 2: An IG algorithm, IGLS3.**

**LS1:** swap the structures of adjacent positions.

1. Select \(k_1 = 1, k_2 = k_1 + 1\).

2. Swap the job numbers between the positions \(k_1\) and \(k_2\) in \(S\), and let \(S_0\) denote the new solutions.

3. If \(C_{\text{max}}(S_0)\) is less than \(C_{\text{max}}(S_1)\) then update \(S\) by \(S_0\).

4. Let \(k_1 = k_1 + 1\) and \(k_2 = k_2 + 1\). Repeat (2) and (3) until \(k_1 = n - 1\).

5. Repeat steps (1) to (4) until \(C_{\text{max}}(S_0)\) cannot be improved.
LS2: complete swap and best-first neighborhood structure.

1. Swaps each pair of positions (jobs) in $S$.
2. Calculate the resultant $C_{\text{max}}$.
3. Compare $C_{\text{max}}$'s for all possible $n(n+1)/2$ pairs.
4. Keep the best one, say $S_0$, with the lowest value of $C_{\text{max}}$.
5. If $C_{\text{max}}(S_0) < C_{\text{max}}(S_i)$, then interchange the job numbers between the pair of positions for the best; i.e., update $S$ by $S_i$.
6. Repeat steps (1) to (5) until $C_{\text{max}}(S_0)$ cannot be improved.
7. Return the best found solution, $S_0$.

LS4: first-improvement swap neighborhood structure is as follows.

1. Let $k_1 = 1$, $k_2 = k_1 + 1$.
2. Swap the job numbers between the positions $k_1$ and $k_2$ in $S$, and let $S_0$ denote the new solutions.
3. If $C_{\text{max}}(S_0)$ is less than $C_{\text{max}}(S_i)$ then update $S$ by $S_0$.
4. Let $k_2 = k_2 + 1$. Repeat (2) and (3) until $k_2 = n$.
5. Let $k_1 = k_1 + 1$. Repeat steps (2) to (4) until $k_1 = n-1$.
6. Repeat steps (1) to (5) until $C_{\text{max}}(S_0)$ cannot be improved.
7. Return the best found solution, denoted as $S_0$.

The ideas of the LS1, LS2, LS3, and LS4 are modified from three local searches proposed by Della Croce et al. [58]. The complexities of those three local searches are $O(\sqrt[4]{n})$. The proposed LS1, LS2, LS3, and LS4 are very fast; even their complexities are not easily found. As for other neighborhood structures and local search methods, one can refer to Li et al. [59] and Wang et al. [60, 61], among others.

### 4. Computational Experiments and Results

In this section, we present our experimental tests of examining the efficiency of all adopted algorithms. The branch-and-bound method was performed for small number of jobs; a CSA and an iterated greedy (IG) algorithm with 4 different local search methods (coded as IGLS1 IGLS2, IGLS3, and IGLS4) were performed for both small and large numbers of jobs. These algorithms were performed in Fortran (version 6.6, Compaq Visual Fortran) on a PC (Windows XP) with Intel(R) Core(TM) i7 Duo CPU 2.66-GHz and RAM size of 1.99 GB. The normal job processing times on machines $M_1$, $M_2$, and $M_3$, in line with the scheme of Lee et al. [5], were generated at random from a discrete uniform distribution $U(1, 100)$, respectively. All the parameters used in CSA adopted the settings of Wu et al. [1] while all the parameters in IG used the results of previous tests.

In the computational experiments for small number of jobs, $n$ was set at 8, 9, 10, and 11. Following the suggestion by Liu et al. [62], we set the learning index $\alpha$ at $\alpha = -0.1$, $-0.01$, and $-0.001$. The number of jobs removed from a sequence was set at $d = 2$, 3, and 4; no more than half of a number of jobs were removed. One hundred instances were tested for each combination of $n$, $\alpha$, and $d$. A total of 3600 instances were examined. Tables 1, 2, and 3 summarized the computational results. The optimal solutions were acquired by running the B&B method for the chosen small number of jobs. We report the average error percentage (AEP) here, where AEP $= 100\frac{[A_i - O_1^*]}{O_1^*}$, and $A_i$ was obtained by each metaheuristic and $O_1^*$ by the B&B method. Figure 1 shows the performance results of CSA and IGLS algorithms.

As for the capability of the B&B method, we report the maximum and average number of nodes and the maximum and average execution times (in seconds) in Table 1. As the numbers of job size ($n$) increased, the (mean and maximum) nodes increased; the CPU times increased dramatically as the $n$ increased (Table 1, columns 4 and 5). If the number of nodes goes beyond $10^6$, the B&B method would skip to the next instance, and instances solved out with less than $10^6$ nodes.
Table 2: The AEP of algorithms for small number of jobs.

<table>
<thead>
<tr>
<th>n</th>
<th>a</th>
<th>Algorithm</th>
<th>( d )</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>-0.01</td>
<td>IGLS1</td>
<td></td>
<td>0.000</td>
<td>0.013</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS2</td>
<td></td>
<td>0.000</td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS3</td>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS4</td>
<td></td>
<td>0.007</td>
<td>0.455</td>
<td>0.000</td>
<td>0.007</td>
<td>0.006</td>
<td>0.455</td>
<td>0.014</td>
<td>0.748</td>
<td>0.000</td>
<td>0.003</td>
<td>0.000</td>
<td>0.003</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>9</td>
<td>-0.01</td>
<td>IGLS1</td>
<td></td>
<td>0.002</td>
<td>0.105</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS2</td>
<td></td>
<td>0.001</td>
<td>0.047</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS3</td>
<td></td>
<td>0.043</td>
<td>2.277</td>
<td>0.000</td>
<td>0.028</td>
<td>0.004</td>
<td>0.175</td>
<td>0.031</td>
<td>0.634</td>
<td>0.000</td>
<td>0.003</td>
<td>0.000</td>
<td>0.003</td>
<td>0.000</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS4</td>
<td></td>
<td>0.007</td>
<td>0.455</td>
<td>0.000</td>
<td>0.007</td>
<td>0.006</td>
<td>0.455</td>
<td>0.014</td>
<td>0.748</td>
<td>0.000</td>
<td>0.003</td>
<td>0.000</td>
<td>0.003</td>
<td>0.000</td>
<td>0.003</td>
</tr>
<tr>
<td>10</td>
<td>-0.1</td>
<td>IGLS1</td>
<td></td>
<td>0.001</td>
<td>0.308</td>
<td>0.000</td>
<td>0.001</td>
<td>0.006</td>
<td>0.403</td>
<td>0.010</td>
<td>0.308</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS2</td>
<td></td>
<td>0.001</td>
<td>0.308</td>
<td>0.000</td>
<td>0.001</td>
<td>0.006</td>
<td>0.403</td>
<td>0.010</td>
<td>0.308</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS3</td>
<td></td>
<td>0.001</td>
<td>0.308</td>
<td>0.000</td>
<td>0.001</td>
<td>0.006</td>
<td>0.403</td>
<td>0.010</td>
<td>0.308</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGLS4</td>
<td></td>
<td>0.011</td>
<td>0.421</td>
<td>0.000</td>
<td>0.001</td>
<td>0.006</td>
<td>0.403</td>
<td>0.010</td>
<td>0.308</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.003</td>
</tr>
</tbody>
</table>

Table 3: The mean ranks of algorithms in Kruskal–Wallis test for small and large number of jobs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of obs.</th>
<th>Small n Mean Rank</th>
<th>Big n Mean Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGLS1</td>
<td>36</td>
<td>115.14</td>
<td>36</td>
</tr>
<tr>
<td>IGLS2</td>
<td>36</td>
<td>80.96</td>
<td>36</td>
</tr>
<tr>
<td>IGLS3</td>
<td>36</td>
<td>80.38</td>
<td>36</td>
</tr>
<tr>
<td>IGLS4</td>
<td>36</td>
<td>115.64</td>
<td>36</td>
</tr>
<tr>
<td>CSA</td>
<td>36</td>
<td>60.38</td>
<td>36</td>
</tr>
<tr>
<td>Total</td>
<td>180</td>
<td>90.50</td>
<td>180</td>
</tr>
</tbody>
</table>

*Note that average scores were used for ties.*
were recorded as feasible solution. The computational results are shown in the last column of Table 1. No matter what the number of jobs \( (n) \) or the learning index \( (a) \), all instances are solvable within \( 10^8 \) nodes. As for the impact of particular dominant properties on the efficiency of branch and bound (B&B), it can be easily done in the same way as that of Wu et al. [1].

For the proposed algorithms, CSA and IGLS, the AEPs are presented in Table 2. All AEPs of the CSA and IGLS algorithms increased slightly as the learning effect strengthened; i.e., the value of learning index \( (a) \) became larger, as shown in Table 2. As the amount of destruction \( (d) \) increased, the AEPs decreased as well; this is because the larger the value of \( d \) is, the wider the searching scope explores. Overall, the CSA (with mean AEP of 0.1%) performed the best; however, also note that the worst IGLS was with mean AEP of 1.5% (the last row of Table 2). The CPU times were all within 0.1 seconds; we thus omitted them here.

To compare whether the differences in the solution quality of the CSA and IGLS algorithms were statistically significant, the parametric or nonparametric statistical tests have commonly been employed [63]. There are three assumptions of the parametric analysis of variance (ANOVA), i.e., normality, homoscedasticity, and independence of the residuals. We carried out an ANOVA on AEPs and examined the assumption of normality for the residuals (using SAS 9.4). The value of the Kolmogorov–Smirnov D statistic was 0.18, and the asymptotic two-sided \( p \) value was less than 0.01; the level of significance was in general set at 0.05. Since the normality assumption was violated for the samples of AEP, we then used the nonparametric Kruskal–Wallis test on ranks of AEPs to test whether the populations of AEPs were the same. Table 3 (column 3) showed the mean ranks of the CSA and IGLS algorithms.

In Table 4 (column 2), with \( p \) value (Asymp. Sig) < 0.001, it is indicated that there were significantly statistical differences among the performances of the CSA and IGLS algorithms. Consequently, a multiple comparison method, which was based on pairwise two-sample rankings, was conducted to determine pairwise differences between algorithms. We employed the Dwass–Steel–Critchlow–Fligner (DSCF) procedure [64, 65]. As shown in Table 5, the mean ranks of AEPs could be separated significantly into two groups, at 0.05 level of significance. As illustrated in Table 5 (columns 2 and 3) of SDCF test, CSA (AEP = 0.001), IGLS2 (AEP = 0.004), and IGLS3 (AEP = 0.004) were in a better performance group and the IGLS1 (AEP = 0.015) and IGLS4 (AEP = 0.013) were in a worse performance group.

In the computational experiments for the large number of jobs, following Liu et al. [62], we also set the learning index \( a = -0.1, -0.01, \) and \(-0.001\); the number of jobs removed, \( d \), was set at 10, 15, and 20; the numbers of jobs was set at \( n = 40, 50, 60, \) and 70. One hundred instances were tested for each combination of \( n, d, \) and \( a \); a total of 3600 instances were examined. We calculated the relative percent deviation (RPD), \( \text{RPD} = 100[(A_i - A^*)/A^*]\% \), where \( A_i \) was obtained from each algorithm and \( A^* \) was the minimum value among the CSA and four IGLS algorithms.

As shown in Table 6, the mean RPDs of CSA and IGLS algorithms increased as the learning index increased from \(-0.001\) to \(-0.1\) regardless of the number of jobs. As for the impact of the number of removed jobs \( (d) \) on RPDs, the average RPDs were 0.0118, 0.0113, and 0.0113, for \( d = 10, 15, \) and 20, respectively. Figure 2 provided the boxplots of RPDs for each number of \( d \) with these five algorithms.

The boxplots of the RPDs displayed the performances of the CSA and IG algorithms for large numbers of jobs. The CSA algorithm performed slightly worse than those of the four IGLS types. We carried out another ANOVA on RPDs and examined the normality assumption for the residuals; the
p value was less than 0.01 from the Kolmogorov–Smirnov normality test. Therefore, the normality assumption was invalid for the RPD samples. A Kruskal–Wallis test on ranks of RPDs was subsequently used to check whether the RPD samples were from the same distribution. Table 3 (column 3) shows the mean ranks of the CSA and four IGLS algorithms for the large numbers of jobs. In Table 4 (column 3), the asymptotic p value is less than 0.0001. Consequently, the DSCF procedure was used to find pairwise differences between pairs of the CSA and IGLS algorithms. As displayed in Table 5 (columns 4 and 5), the IGLS algorithms were in one better performance group, whereas the CSA algorithm was the worst in large number of jobs case. Moreover, the boxplots of the RPDs (Figure 2) of the IGLS3 and IGLS2 showed less dispersion than the others. In other words, the IGLS2 and IGLS3 were both accurate and stable for solving the problem of the large numbers of jobs.

5. Conclusions and Future Studies

A makespan minimization problem for a two-stage threemachine assembly flow shop scheduling with sum-of-processed-times based processing times has been addressed. A branch-and-bound method embedded with a LB and several dominance properties was proposed for the small number of jobs \(n = 8, 9, 10, 11\). Since the problem is NP-hard, two algorithms, a cloud theory-based simulated annealing (CSA) algorithm and an iterated greedy (IG) algorithm with four local search methods, have been adopted to solve the problem for the large number of jobs \(n = 40, 50, 60, 70\). These algorithms seemed to perform well for both small number of jobs \(n = 8, 9, 10, 11\) and large number of jobs \(n = 40, 50, 60, 70\). The computational results validate the effectiveness, as claimed by researchers in the scheduling research community, of the proposed IG algorithm with four different local search methods. However, the CSA was still a competitive metaheuristic algorithm, especially for small-number-of-jobs problems in this study.

The occurrences of the study problem in many plant manufacturing environments have been exemplified by outstanding researchers such as these referred to in the introduction. Some potential research topics worthy of investigation are different learning curves including the position-based learning, an exponential learning, truncated learning, or general learning functions, especially different learning functions for different stages, i.e., at component manufacturing stage and at the assembly stage. This study considers two stages; however, there are several published papers that investigated three-stage flow shop assembly problem but all without learning effect. Therefore, extending to multistage assembly problem with learning consideration in some of these stages can be another future topic to pursue. One may generalize the properties for general nonincreasing learning curves, \(p_{jk} = p_f f(\sum_{i=1}^{k-1} p_{ij}),\) where \(f\) is a nonincreasing function describing the learning effect (similarly for \(q_i\) and \(r_j\)) or job based nonincreasing learning curve, \(p_{ijk} = p_{ij} - b_{ij} \min[k-1, q_{ij}].\) Readers may refer to Rudek [66–68] and Wang et al. [60, 61] for further details.

### Appendix

#### Proofs of Lemma 1 and Properties

**Proof of Lemma 1.** Because \(a < 0, \text{ then } 0 < (C+y)^a/(C+x)^a \leq 1, \text{ and } 1 \leq y/x, \text{ the result holds.}**

**Proof of Property 3.** The completion times of job \( Ji \) and job \( Jj \) in the positions of \( k \) and \( k + 1 \) on machines \( M_l, l = 1, 2, 3, \) in \( S \) are as follows.

\[
C_{i1}(S) = T_{1k} + p_i \left( 1 + \sum_{j=1}^{k-1} q_{ij} \right)^a = T_{1k} + p_i p_k^a.
\]

\[
C_{2i}(S) = T_{2k} + q_i \left( 1 + \sum_{j=1}^{k-1} q_{ij} \right)^a = T_{2k} + q_i q_k^a.
\]

\[
C_{i3}(S) = \max \left\{ C_{i1}(S), C_{2i}(S), T_{3k} \right\} + r_k R_k^a.
\]

\[
C_{ij}(S) = C_{i1}(S) + p_j \left( 1 + \sum_{j=1}^{k} p_{ij} \right)^a = T_{ik} + p_i p_k^a + p_j (p_k + p_j)^a.
\]
<table>
<thead>
<tr>
<th>$n$</th>
<th>$a$</th>
<th>$d$</th>
<th>IGLS1</th>
<th>IGLS2</th>
<th>IGLS3</th>
<th>IGLS4</th>
<th>CSA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>mean</td>
<td>max</td>
<td>mean</td>
<td>max</td>
<td>mean</td>
</tr>
<tr>
<td>40</td>
<td>-0.001</td>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>10</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>50</td>
<td>-0.001</td>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>10</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>60</td>
<td>-0.001</td>
<td>10</td>
<td>0.00</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>10</td>
<td>0.00</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>70</td>
<td>-0.001</td>
<td>10</td>
<td>0.00</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>10</td>
<td>0.00</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

\[
C_{2j}(S) = C_{2i}(S) + \left(1 + \sum_{j=1}^{k} q_{j|1}\right)^{a} + q_{j}(Q_{k} + q_{j})^{a} + r_{j}(R_{k} + r_{j})^{a} = \max \left[T_{1k} + p_{j}P_{k}, T_{2k} + q_{j}Q_{k}, T_{3k}\right]
\]

\[
C_{3j}(S) = \max \left[C_{1j}(S), C_{2j}(S), \max \left[C_{1i}(S), C_{2i}(S), T_{3k}\right]\right]
\]

\[
(C.1)
\]

\[
C_{2j}(S) = C_{2i}(S) + \left(1 + \sum_{j=1}^{k} q_{j|1}\right)^{a} = T_{2k} + q_{j}Q_{k}^{a} + r_{j}(R_{k} + r_{j})^{a} = \max \left[T_{1k} + p_{j}P_{k}, T_{2k} + q_{j}Q_{k}, T_{3k}\right]
\]

\[
C_{3j}(S) = \max \left[C_{1j}(S), C_{2j}(S), \max \left[C_{1i}(S), C_{2i}(S), T_{3k}\right]\right]
\]

\[
(C.1)
\]
The completion times of job \( fj \) and job \( fi \) in the positions of \( k \) and \( k + 1 \) on machines \( M_i, i = 1, 2, 3, \) in \( S^l \) are as follows.

\[
C_{ij} \left( S^l \right) = T_{ik} + p_j \left( 1 + \sum_{j=1}^{k-1} p_{ij} \right)^a = T_{ik} + p_j P_k^a.
\]

\[
C_{2j} \left( S^l \right) = T_{2k} + q_i \left( 1 + \sum_{j=1}^{k-1} q_{ij} \right)^a = T_{2k} + q_i Q_k^a.
\]

\[
C_{3j} \left( S^l \right) = \max \left\{ C_{1j} \left( S^l \right), C_{2j} \left( S^l \right), T_{3k} \right\} + r_j R_k^a.
\]

\[
C_{il} \left( S^l \right) = C_{ij} \left( S^l \right) + p_i \left( 1 + \sum_{j=1}^{k} p_{ij} \right)^a = T_{ik} + p_i \left( P_k + p_i \right)^a.
\]

\[
C_{2i} \left( S^l \right) = C_{2j} \left( S^l \right) + q_i \left( 1 + \sum_{j=1}^{k} q_{ij} \right)^a = T_{2k} + q_i Q_k^a.
\]

\[
C_{3i} \left( S^l \right) = \max \left\{ C_{1i} \left( S^l \right), C_{2i} \left( S^l \right), T_{3k} \right\} + r_i R_k^a.
\]

In the following, we claim that \( C_{3k} \left( S^l \right) - C_{3j} \left( S \right) > 0 \). The inequality \( T_{3k} > \max \left\{ T_{1k} + p_j P_k^a, T_{2k} + q_i Q_k^a \right\} \) implies that

\[
C_{3k} \left( S^l \right) = \max \left\{ T_{1k} + p_j P_k^a + p_i \left( P_k + p_i \right)^a, T_{2k} + q_i Q_k^a + q_i \left( Q_k + q_i \right)^a, T_{3k} + r_i R_k^a \right\}
\]

\[
= T_{1k} + p_j P_k^a + q_i Q_k^a + q_i \left( Q_k + q_i \right)^a, T_{3k} + r_i R_k^a
\]

\[
= T_{1k} + p_j P_k^a + q_i Q_k^a + q_i \left( Q_k + q_i \right)^a + r_i \left( R_k + r_i \right)^a.
\]

The last equality holds because \( T_{1k} + p_j P_k^a + p_i \left( P_k + p_i \right)^a \geq \max \left\{ T_{2k} + q_i Q_k^a + q_i \left( Q_k + q_i \right)^a, T_{3k} + r_i R_k^a \right\} \).

Since \( p_i < q_i, q_i < q_j, \) and \( T_{3k} > \max \left\{ T_{1k} + p_j P_k^a, T_{2k} + q_j Q_k^a \right\} \), then \( T_{3k} > \max \left\{ T_{1k} + p_j P_k^a, T_{2k} + q_j Q_k^a \right\} \).

Apply the last inequality to \( C_{3j} \left( S \right) \):

\[
C_{3j} \left( S \right) = \max \left\{ T_{1k} + p_j P_k^a + p_j \left( P_k + p_j \right)^a, T_{2k} + q_j Q_k^a + q_j \left( Q_k + q_j \right)^a, T_{3k} + r_j R_k^a \right\}
\]

\[
= T_{1k} + p_j P_k^a + q_j \left( Q_k + q_j \right)^a, T_{3k} + r_j R_k^a
\]

\[
= T_{1k} + p_j P_k^a + q_j \left( Q_k + q_j \right)^a + r_j \left( R_k + r_j \right)^a.
\]

The last equality comes from \( T_{1k} + p_j P_k^a + p_j \left( P_k + p_j \right)^a > \max \left( T_{2k} + q_j Q_k^a + q_j \left( Q_k + q_j \right)^a, T_{3k} + r_j R_k^a \right) \).

So,

\[
C_{3i} \left( S^l \right) - C_{3j} \left( S \right) = \left[ T_{1k} + p_j P_k^a + p_j \left( P_k + p_j \right)^a \right]
\]

\[
+ r_i \left( R_k + r_i \right)^a - \left[ T_{1k} + p_j P_k^a + p_j \left( P_k + p_j \right)^a \right]
\]

\[
+ r_j \left( R_k + r_j \right)^a - \left( p_i \left( P_k + p_i ^a \right) - p_j \left( P_k + p_j \right)^a \right)
\]

\[
= \left[ p_i \left( P_k + p_i \right)^a \right] \left[ 1 - \frac{p_j}{p_i} \right] + r_i \left( R_k + r_i \right)^a - r_j \left( R_k + r_j \right)^a
\]

\[
= \left( p_i \left( P_k + p_i \right)^a \right) \left[ \frac{p_j}{p_i} - 1 \right] + r_i \left( R_k + r_i \right)^a - r_j \left( R_k + r_j \right)^a.
\]

Letting \( \lambda = p_j/p_i > 1 \), and \( t = p_j/P_k, t > 0 \), then by applying Lemma 2, the first term of the last equation is greater than 0. Letting \( y = r_i, x = r_j \) (i.e., \( x < y \)), and \( C = R_k \), then Lemma 1 implies the second term of the last equation in \( C_{3k} \left( S^l \right) - C_{3j} \left( S \right) \) is greater than 0. Consequently, \( C_{3k} \left( S^l \right) - C_{3j} \left( S \right) > 0 \), as required.
\[ + \left[ r_i (R_k + r_j)^a - r_j (R_k + r_j)^a \right] \]
\[ = p_i p_k \left[ \frac{p_j}{p_i} - \frac{p_j}{p_k} \left( 1 + \frac{p_i}{p_k} \right)^a - \left( 1 + \left( 1 + \frac{p_j}{p_k} \right)^a \right) \right] \]
\[ + \left[ r_i (R_k + r_j)^a - r_j (R_k + r_j)^a \right] > 0. \quad (A.6) \]

Let \( \lambda = \frac{p_i}{p_j}, \lambda > 1 \) and \( t = \frac{p_i}{R_k}, t > 0; \) then by applying Lemma 2, the first term above is greater than 0. Directly applying Lemma 1, the second term above is greater than 0.

\[ \square \]

**Proof of Property 7.** The proof is similar to that of Property 6 with \( p_i \) and \( P_k \) now being replaced by \( q_i \) and \( Q_k \).

\[ \square \]

**Proof of Property 8.** The value of inside max in the \( C_{3j}(S) \) is equal to \( T_{ik} + p_i P_k^a \) because \( T_{ik} + p_i P_k^a > \max \{ T_{2k} + q_i Q_k^a, T_{3k} \} \) and \( a < q_j \). Applying the given inequality of \( r_i R_k^a \) to the first two terms of the outside max, we have \( C_{3j}(S) = \left[ T_{1k} + p_i P_k^a + r_i R_k^a \right] + r_j (R_k + r_j)^a \).

The value of max inside the \( C_{3j}(S') \) is equal to \( T_{ik} + p_j P_k^a \) because \( T_{ik} + p_i P_k^a > T_{1k} + p_j P_k^a > \max \{ T_{2k} + q_i Q_k^a, T_{3k} \} \) and \( q_i < q_j \).

Now,
\[ C_{3j}(S') = \max \left\{ T_{ik} + p_j P_k^a + p_i (P_k + p_i)^a, T_{2k} + q_i Q_k^a + q_l (Q_k + q_l)^a, T_{ik} + p_j P_k^a + r_i R_k^a \right\} \quad (A.7) \]
\[ + r_i (R_k + r_j)^a. \]

Note that \( r_i R_k^a > p_i (P_k + p_i)^a > p_i (P_k + p_j)^a, r_i R_k^a > q_i (Q_k + q_i)^a > q_i (Q_k + q_j)^a, \) and \( T_{ik} + p_j P_k^a > T_{1k} + p_j P_k^a > T_{2k} + q_i Q_k^a \). We have \( C_{3j}(S') = \left[ T_{1k} + p_j P_k^a + r_i R_k^a \right] + r_j (R_k + r_j)^a \).

Therefore,
\[ C_{3i}(S) - C_{3j}(S) = \left[ r_i R_k^a + r_i (R_k + r_j)^a \right] - \left[ r_i R_k^a + r_j (R_k + r_j)^a \right] \]
\[ = \left( 1 - \left( 1 + \frac{r_j}{r_i R_k^a} \right)^a \right). \quad (A.8) \]

Let \( \lambda = \frac{r_j}{r_i}, \lambda > 1, \) and \( t = \frac{r_j}{R_k}, t > 0. \) Applying Lemma 2, we obtain the desired \( C_{3i}(S') - C_{3j}(S) > 0. \)

\[ \square \]

**Proof of Property 9.** The proof is similar to that of Property 8 and thus omitted.

\[ \square \]

**Data Availability**

The data used to support the findings of this study are available from the corresponding author upon request.

**Conflicts of Interest**

The author declares that they have no conflicts of interest.

**Disclosure**

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

**Acknowledgments**

The author thanks Professor K. Lai for helping in editing the English language.

**References**


