Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 8520503, 13 pages
https://doi.org/10.1155/2019/8520503

Research Article

An Improved Discrete Artificial Bee Colony Algorithm for Flexible Flowshop Scheduling with Step Deteriorating Jobs and Sequence-Dependent Setup Times

Hua Xuan, Huixian Zhang, and Bing Li

School of Management Engineering, Zhengzhou University, Zhengzhou 450001, Henan, China

Correspondence should be addressed to Hua Xuan; hxuan@zzu.edu.cn

Received 16 July 2019; Revised 4 November 2019; Accepted 13 November 2019; Published 12 December 2019

Academic Editor: Francisco Chicano

This paper studies a flexible flowshop scheduling problem with step-deteriorating jobs and sequence-dependent setup times (FFSP-SDJ&SDST) where there are multiple unrelated parallel machines at each stage. The actual processing time of each job is modeled as a step function of its starting time. An integer programming model is first formulated with the objective of minimizing the total weighted completion time. Since this problem is NP-complete, it becomes an interesting and challenging topic to develop effective approximation algorithms for solving it. The artificial bee colony (ABC) algorithm has been successfully applied to solve both continuous and combinatorial optimization problems with the advantages of fewer control parameters and ease of implementation. So, an improved discrete artificial bee colony algorithm is proposed. In this algorithm, a dynamic generation mechanism of initial solutions is designed based on job permutation encoding. A genetic algorithm and a modified variable neighborhood search are introduced, respectively, to obtain new solutions for the employed and onlooker bees. A greedy heuristic is proposed to generate the solutions of the scout bees. Finally, to verify the performance of the proposed algorithm, an orthogonal test is performed to optimize the parameter settings. Simulation results on different scale problems demonstrate that the proposed algorithm is more effective compared against several presented algorithms from the existing literatures.

1. Introduction

Scheduling problem is one of key issues in many manufacturing systems. Flexible flowshop scheduling problem (FFSP) widely arises in discrete industry and process industry such as semiconductors, electronics manufacturing, cosmetics, and pharmaceuticals [1]. In the classical FFSP, the processing times of jobs are always assumed to be known and fixed. However, it might not be true in realistic industrial processes such as steel production and equipment maintenance [2]. The actual processing time may become longer due to equipment wear, physical properties of jobs, fatigue of the workers, and so on. This phenomenon is known as scheduling with deteriorating jobs. In addition, it often happens that some tasks such as equipment cleaning and tool replacement are performed when a machine handles different jobs. That is, a setup time occurs between adjacent jobs on the same machine and it always depends on the job sequence [3]. Therefore, this paper considers FFSP with step-deteriorating jobs and sequence-dependent setup times where step deterioration means that if the starting time of a job is more than a prespecified deteriorating date, it will require extra penalty time to complete its tasks successfully. The scheduling goal is to determine the assignment of jobs to machines and the sequence of jobs on each machine.

Scheduling with deterioration has been presented and defined first by Gupta and Gupta [4] and Browne and Yechiali [5]. The current research studies on deteriorating job scheduling problems mostly assume that the job processing times are linear functions of their starting times. For single-machine scheduling, Wang and Liu [6] considered a group scheduling problem with setup times. A modified Lawler's algorithm was proposed to minimize the total weighted completion time and the maximum cost simultaneously. In terms of unrelated parallel machine scheduling, Wang and Li [7] considered modifying maintenance activities with the objective of

minimizing a cost function of total completion (waiting) time, total absolute differences in completion (waiting) time, and total machine load. They proved that the problem can be solved in polynomial time. For flowshop scheduling, Wang et al. [8] presented a multiverse optimizer algorithm so that the total tardy time is minimized. There is a relatively little research on step deterioration. To solve identical parallel machine scheduling, Guo et al. [2] considered sequence-dependent setup times and applied a hybrid discrete cuckoo algorithm to minimize total tardiness; Cheng et al. [9] presented a modified weight-combination search algorithm and a variable neighborhood search to minimize total completion time. For two-stage FFSP with identical parallel machines at stage 1 and a single batching machine at stage 2, Gong and Tang [10] proposed a heuristic algorithm to minimize makespan plus the total penalty cost of batching-machine utilization ratio where the job processing time at stage 2 is a step function of its waiting time between the two stages.

No setup is commonly considered in many shop scheduling problems [11, 12]. However, separated setup times have to be included in scheduling decision so as to describe realistic production process more accurately. Recently, the sequence-dependent setup times (SDST) have attracted increasing attention among researchers. For the flowshop with SDST, Wang et al. [13] presented a hybrid local search algorithm to minimize makespan; Wang and Li [14] applied a hybrid chaotic biogeography-based optimization to minimize the total weighted tardiness; Li and Li [15] developed multiobjective local search based decomposition to solve permutation flowshop with the objective of minimizing makespan and total flowtime. Regarding the multistage FFSP with SDST, Pedro et al. [16] presented an improved genetic algorithm to minimize makespan. Marichelvam et al. [17] used a discrete firefly algorithm to minimize total tardiness. Pan et al. [18] developed nine metaheuristics based on trajectory and population to minimize makespan. Santosa and Riyanto [19] applied hybrid differential evolution-variable neighborhood search to minimize makespan and maximum lateness simultaneously.

In past decades, many efforts have been made on heuristic and metaheuristic algorithms to obtain near-optimal solutions with high quality in a reasonable computational time [20, 21]. The artificial bee colony (ABC) algorithm is one of the population-based evolutionary metaheuristics, proposed by Karaboga [22], which simulates the intelligent foraging behavior of honey bees in nature. It has been applied to solve some shop scheduling problems. Chow et al. [23] used this algorithm to solve the flowshop makespan problem. Li and Ma [24] presented a novel discrete ABC (DABC) algorithm for the permutation flowshop scheduling with sequence-dependent setup times in order to minimize makespan and total flowtime. Peng et al. [25] improved the ABC algorithm to solve FFSP with serial-batching machines on the last stage in which the objective is minimizing sojourn time, earliness of casting starting penalty, and tardiness of cast starting penalty. Cui and Gu [26] proposed an improved DABC algorithm for FFSP to minimize makespan. Li et al. [27] used an improved DABC algorithm for hybrid FFSP with dynamic operation skipping so that the average sojourn time, the penalties of tardiness and earliness, and the penalty of the skipping rate are minimized.

To the best of our knowledge, most research studies on scheduling with deterioration concentrate on linear deteriorating jobs and consider the machine environments such as single machine, parallel machine, flowshop, and two-stage FFSP. Little work has been carried out on a multistage FFSP with step-deteriorating jobs. This problem is NP-complete even if there is only one single machine [28]. In this paper, the more complicated multistage FFSP with step-deteriorating jobs and sequence-dependent setup times (FFSP-SDJ&SDST) is studied to minimize the sum of weighted completion times. A new improved DABC (IDABC) algorithm is then developed to obtain near-optimal solutions.

The rest of this paper is organized as follows. Section 2 describes the problem and formulates a mathematical model. Section 3 develops an IDABC algorithm. In Section 4, computational experiments are performed and the algorithm comparisons are provided. Finally, the conclusions are summarized in Section 5.

2. Problem Formulation

2.1. Problem Description. In the FFSP-SDJ&SDST, a set of n jobs $\{J_1, J_2, \ldots, J_n\}$ has to be processed on S stages, and each stage k has m_k unrelated machines ($m_k \geq 2$ for at least one stage). These unrelated machines are continuously available from time zero. Each job should be started at its release time r_i ($r_i > 0$). Job j can be handled on any one of the m_k machines at stage k, and the basic processing time is different on each machine. A setup time δ_{fijk} occurs when job f is handled immediately before job i on machine j at stage k. $\delta_{0ijk} = 0$ means that no setup time is required if job i is the first job processed on machine j at stage k. At stage k, job i has a basic constant processing time P_{ijk} on machine j if its starting time St_{ijk} is no more than its deteriorating date h_{ijk}; otherwise, it requires an extra penalty time b_{ijk}. Transfer times between two successive stages have to be considered because job delivery is often carried out by using conveyors, carts, and so on. The goal of the problem is to minimize the total weighted completion time.

2.2. Problem Definition and Notation. To describe the above problem, the following notations are introduced:

(1) Parameters

n: number of jobs
S: number of stages
m_k: number of machines on stage k
$T_{k,k+1}$: delivery time of jobs from stage k to stage $k+1$
$r_{i:}$ release time of job i
w_i: weight of job i
δ_{fijk}: setup time when job f is processed immediately before job i on machine j at stage k
h_{ijk}: deteriorating date of job i on machine j at stage k

b_{ijk}: deteriorating penalty time of job i on machine j at stage k
P_{ijk}: basic processing time of job i on machine j at stage k

(2) Decision variables

C_{ik}: completion time of job i at stage k
St_{ijk}: starting time of job i on machine j at stage k
P'_{ijk}: actual processing time of job i on machine j at stage k
X_{ijk}: binary, set to 1 if job i is assigned to machine j at stage k; 0 otherwise
Y_{fijk}: binary, set to 1 if job f is handled immediately before job i on machine j at stage k; 0 otherwise

2.3. Model Formulation. Applying the above notations, the studied problem can be formulated as an integer programming model, as shown below:

$$\text{Min } O = \sum_{i=1}^{n} w_i C_{iS}, \tag{1}$$

subject to

$$\sum_{j=1}^{m_k} X_{ijk} = 1, \quad i = 1, \ldots, n,\ k = 1, \ldots, S, \tag{2}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m_k} Y_{0ijk} = m_k, \quad k = 1, \ldots, S, \tag{3}$$

$$P'_{ijk} = \begin{cases} P_{ijk}, & St_{ijk} \le h_{ijk}, \\ P_{ijk} + b_{ijk}, & St_{ijk} > h_{ijk}, \end{cases} \tag{4}$$

$$C_{ik} \ge St_{ijk} + P'_{ijk}, \quad i = 1, \ldots, n,\ j = 1, \ldots, m_k,\ k = 1, \ldots, S, \tag{5}$$

$$C_{i1} - \sum_{j=1}^{m_1} X_{ij1} P'_{ij1} \ge r_i, \quad i = 1, \ldots, n, \tag{6}$$

$$\sum_{f=1}^{n} \sum_{i=1}^{n} Y_{fijk} St_{ijk} \ge \sum_{f=1}^{n} \sum_{i=1}^{n} Y_{fijk} \left(C_{fk} + \delta_{fijk}\right), \quad f \ne i,\ j = 1, \ldots, m_k,\ k = 1, \ldots, S, \tag{7}$$

$$C_{i,k+1} - C_{i,k} \ge T_{k,k+1} + \sum_{j=1}^{m_{k+1}} X_{ij,k+1} P'_{ij,k+1}, \quad i = 1, \ldots, n,\ k = 1, \ldots, S, \tag{8}$$

$$X_{ijk}, Y_{fijk} \in \{0, 1\}, \quad f = 0, \ldots, n-1, i = 1, \ldots, n, j = 1, \ldots, m_k, k = 1, \ldots, S. \tag{9}$$

The objective function (1) indicates the minimum total weighted completion time. Constraints (2) ensure that each job has to be processed in all stages and at each stage, a job can select only one available machine. Constraints (3) guarantee that all m_k machines are scheduled at each stage when $n \ge \max\{m_1, \ldots, m_S\}$. Constraints (4) define the actual processing time of a job in consideration of step-deteriorating effect. Constraints (5) indicate the relation of the completion time and beginning time of a job. Constraints (6) make sure that the beginning time of a job at stage 1 must be larger than or equal to its release time. Constraints (7) ensure that, for two consecutive jobs on the same machine, the next one can be started only after the completion of the preceding one plus the setup time between the two jobs. Constraints (8) indicate that, for two adjacent operations of a job, the latter one can be processed only after the former one has been finished and transferred to the corresponding machine. Constraints (9) represent the value ranges of decision variables.

3. Improved Discrete Artificial Bee Colony Algorithm

The ABC algorithm is originally used to solve continuous function optimization. However, as mentioned above, the considered FFSP-SDJ&SDST is a discrete combination problem. Therefore, it is necessary to make discretization improvements so as to solve FFSP-SDJ&SDST. So, an improved discrete artificial bee colony (IDABC) algorithm is presented to solve the NP-complete problem.

3.1. The Framework of the Proposed IDABC Algorithm. The ABC algorithm starts with the generation of initial solutions and then performs an iteration of employed bee phase, onlooker bee phase, and scout bee phase. Since the considered problem belongs to discrete function optimization, this paper utilizes job sequence as solution representation and presents an IDABC algorithm. To make the algorithm more effective, a genetic algorithm (GA) is adopted including single-point crossover and multiple insertion mutation for employed bees; a modified variable neighborhood search (MVNS) using perturbation and search procedures is designed for onlooker bees; a greedy heuristic (GH) with destruction and reconstruction procedures is proposed for scout bees. The above modification or improvement is performed in the novel IDABC algorithm to obtain better food sources. The process of the IDABC algorithm is shown in Figure 1. The detailed steps are provided as follows:

Step 1: parameter setting phase. Set the following system parameters: maximum iteration number $I_{\max}$, population size *Nump*, iteration number *limit* related to food sources to be abandoned, crossover probability *CP*, and mutation probability *MP*

Step 2: initialization of population. Randomly generate *Nump* job sequences (food sources) as initial solutions

Step 3: solution evaluation. Compute fitness values of food sources and memorize the best solution

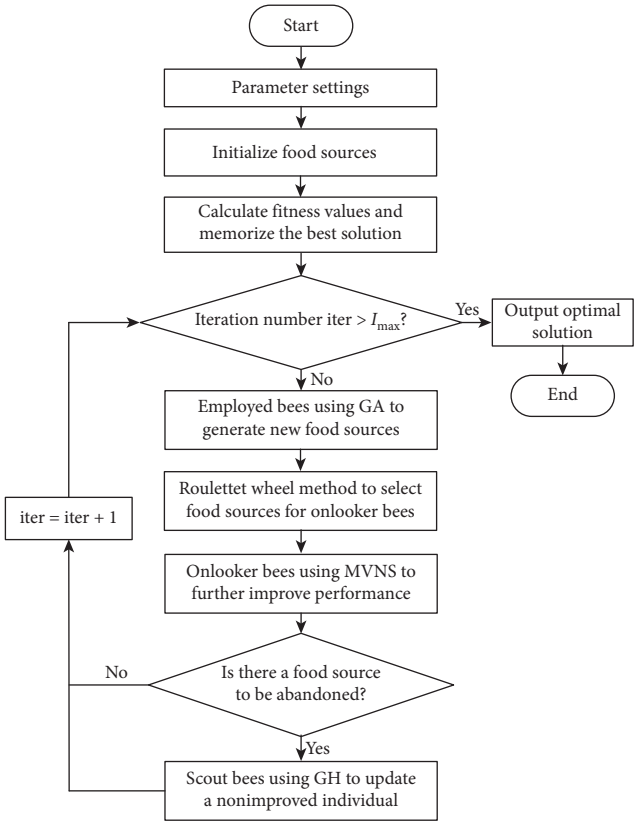

Figure 1: Framework of the proposed IDABC algorithm.

Step 4: if the termination condition is met, output the optimal solution; otherwise, perform Step 5~Step 9

Step 5: employed bee phase. Utilize GA to generate the new food sources

Step 6: calculate the fitness value of each newly generated food sources and use the roulette wheel method to select food sources for onlooker bees

Step 7: onlooker bee phase. Use MVNS to search round the selected food sources. Calculate the fitness values and find the better solutions

Step 8: if there is a food source to be abandoned, perform Step 9; otherwise, go to Step 4

Step 9: scout bee phase. For the food sources without improvement during the *limit* iterations, a new solution is generated by using GH

3.2. Encoding and Decoding Procedures. The FFSP-SDJ&SDST considers the sequence-dependent setup times and unrelated parallel machines. The scheduling goal consists of assigning the jobs to the appropriate machines at each stage and ordering the jobs assigned to the same machine. Thus, a dynamic generation mechanism of initial solutions is proposed where job sequence-based representation for stage 1 is applied to encode the solutions and the decoding scheme containing two steps is applied. The initial solutions of the population are then obtained.

4	5	1	3	2

Figure 2: An instance with five jobs.

3.2.1. Encoding Solutions. To obtain *Nump* food sources, each job sequence at stage 1 is yielded from discrete domain. So, a random perturbation procedure based on job numbers is developed to construct the initial job sequence $\omega^1 = \{\omega^1(1), \omega^1(2), \ldots, \omega^1(i), \ldots, \omega^1(n)\}$, where $\omega^k(i)$ denotes the job on the ith position at stage k. Figure 2 illustrates an encoding scheme for an instance with five jobs where $\omega^1(1) = 4$.

3.2.2. Decoding Food Sources. Because the goal is to minimize the total weighted completion time, the job with the earlier completion time at the immediately preceding stage is allowed to have a higher priority in order to improve on-time delivery between adjacent stages. So, the job sequence at stage $k(k > 1)$ is determined according to their completion times at stage $k - 1$.

At stage 1, according to the above generated job sequence, the job $\omega^1(1)$ is firstly assigned to the machine with the shortest actual processing time. To balance the utility of all the machines, the job $\omega^1(2)$ chooses one machine with the shortest actual processing time from the unscheduled machines until the job $\omega^1(m_1)$ is arranged. For the unscheduled n-m_1 jobs, compute the possible completion time of job $\omega^1(i)$ $(i > m_1)$ equal to the sum of the completion time of the immediately preceding job on the same machine, the setup time and its actual processing time, and assign it to the machine with the shortest completion time.

The job sequence at stage $k(k > 1)$ is obtained in the increasing order of the completion time of all jobs at stage $k - 1$. If some jobs have the same completion time, order these jobs randomly. Similar with stage 1, the first m_k job is, respectively, arranged on the m_k machines with the shortest actual processing time based on the sequence so that all machines are used. When it turns to $\omega^k(i)$ $(i > m_k)$, the starting time of job $\omega^k(i)$ is set to the maximum between its completion time at stage $k - 1$ plus the delivery time and the completion time of the immediately preceding job on the same machine plus the setup time. So, its completion time can be computed by its starting time plus the actual processing time. The assigned machine is decided with the shortest completion time. The process continues until all jobs are arranged in all stages, as shown in Figure 3.

3.2.3. Population Initialization. As described above, an individual is denoted as a permutation of all jobs at stage 1. To generate a population, a simple random procedure is adopted where *Nump* individuals (food sources) are consistent with the size of employed bee or onlooker bee.

To clearly illustrate the FFSP-SDJ&SDST and the decoding procedure, a simple example is introduced with 5 jobs and 3 stages where each stage has three machines. The release time and weight of jobs are given by $r_i = [3, 1, 4, 2, 3]$ and $w_i = [5, 6, 9, 7, 3]$; delivery time $T_{k,k+1}$ is, respectively, set to 3 and 2. For simplicity, it is assumed that the setup time

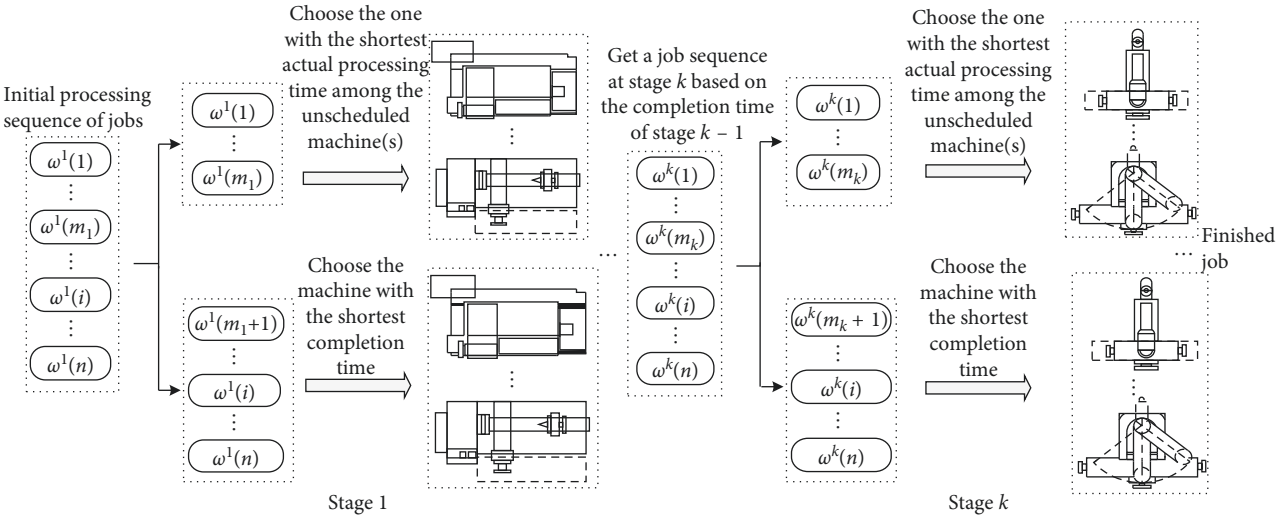

Figure 3: Decoding scheme using the rule of minimum processing/completion time of the free machine.

δ_{fijk} is dependent on jobs and irrelevant to machines and denoted as δ'_{fi}. The basic processing time P_{ijk}, deteriorating date h_{ijk}, deteriorating penalty time b_{ijk}, and setup time δ'_{fi} are, respectively, given below:

Stage1 Stage2 Stage3

M11 M21 M31 M12 M22 M32 M13 M23 M33

$$P_{ijk} = \begin{bmatrix} 17 & 7 & 8 & 16 & 12 & 13 & 20 & 13 & 10 \\ 6 & 4 & 17 & 16 & 10 & 20 & 16 & 11 & 9 \\ 17 & 6 & 12 & 16 & 11 & 10 & 7 & 14 & 14 \\ 5 & 13 & 11 & 14 & 16 & 12 & 11 & 14 & 19 \\ 19 & 10 & 19 & 12 & 15 & 7 & 12 & 15 & 16 \end{bmatrix}$$

$$h_{ijk} = \begin{bmatrix} 10 & 20 & 36 & 26 & 22 & 25 & 22 & 33 & 31 \\ 59 & 40 & 53 & 98 & 45 & 56 & 25 & 44 & 21 \\ 37 & 23 & 44 & 10 & 20 & 20 & 28 & 12 & 30 \\ 61 & 87 & 3 & 33 & 29 & 41 & 66 & 23 & 80 \\ 15 & 9 & 7 & 68 & 104 & 13 & 68 & 25 & 53 \end{bmatrix} \quad (10)$$

$$b_{ijk} = \begin{bmatrix} 9 & 2 & 9 & 5 & 2 & 9 & 1 & 5 & 5 \\ 6 & 10 & 1 & 10 & 9 & 7 & 3 & 1 & 4 \\ 10 & 1 & 4 & 2 & 6 & 4 & 2 & 10 & 10 \\ 1 & 8 & 3 & 3 & 6 & 6 & 2 & 10 & 4 \\ 5 & 9 & 9 & 2 & 2 & 5 & 3 & 5 & 2 \end{bmatrix}$$

1 2 3 4 5 job

$$\delta'_{fi} = \begin{bmatrix} 0 & 3 & 4 & 2 & 5 & 1 \\ 2 & 0 & 2 & 3 & 2 & 2 \\ 4 & 2 & 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 0 & 3 & 4 \\ 5 & 1 & 1 & 4 & 0 & 5 \end{bmatrix}$$

The initial job sequence [4, 5, 1, 3, 2] at stage 1 is given and consistent with the instance illustrated in Figure 2. The machine allocation and completion time of all jobs are computed as follows. The Gantt chart of a solution is drawn in Figure 4, where each operation is denoted by a rectangle labeled with a job number.

Stage 1:

$\omega^1 = \{4, 5, 1, 3, 2\}$
$C_{41} = r_4 + \min\{P_{411}, P_{421}, P_{431}\} = 2 + \min\{5, 13, 11\} = 7$
$X_{411} = 1, X_{421} = X_{431} = 0$
$C_{51} = r_5 + \min\{P_{421}, P_{431}\} = 3 + \min\{10, 19\} = 13$
$X_{521} = 1, X_{511} = X_{531} = 0$
$C_{11} = r_1 + P_{131} = 3 + 8 = 11$
$X_{131} = 1, X_{111} = X_{121} = 0$
$C_{31} = \min\{C_{41} + \delta'_{43} + P_{311}, C_{51} + \delta'_{53} + P_{321}, C_{11} + \delta'_{13} + P_{331}\} = \min\{7 + 5 + 17, 13 + 1 + 6, 11 + 4 + 12\} = 20$ $(C_{41} + \delta'_{43} < h_{311}, C_{51} + \delta'_{43} < h_{321}, C_{11} + \delta'_{13} < h_{331})$
$X_{321} = 1, X_{311} = X_{331} = 0$
$C_{21} = \min\{C_{41} + \delta'_{42} + P_{211}, C_{31} + \delta'_{32} + P_{221}, C_{11} + \delta'_{12} + P_{231}\} = \{7 + 4 + 6, 20 + 2 + 4, 11 + 3 + 17\} = 17$ $(C_{41} + \delta'_{42} < h_{211}, C_{31} + \delta'_{32} < h_{221}, C_{11} + \delta'_{12} < h_{231})$
$X_{211} = 1, X_{221} = X_{231} = 0$

Stage 2:

$\omega^2 = \{4, 1, 5, 2, 3\}$
$C_{42} = C_{41} + T_{12} + \min\{P_{412}, P_{422}, P_{432}\} = 7 + 3 + \min\{14, 16, 12\} = 22$ $(C_{41} + T_{12} < h_{412}, C_{41} + T_{12} < h_{422}, C_{41} + T_{12} < h_{432})$
$X_{432} = 1, X_{412} = X_{422} = 0$
$C_{12} = C_{11} + T_{12} + \min\{P_{112}, P_{122}\} = 11 + 3 + \min\{16, 12\} = 26$ $(C_{11} + T_{12} < h_{112}, C_{11} + T_{12} < h_{122})$
$X_{122} = 1, X_{112} = X_{132} = 0$
$C_{52} = C_{51} + T_{12} + P_{512} = 13 + 3 + 12 = 28$
$(C_{51} + T_{12} < h_{512})$
$X_{512} = 1, X_{522} = X_{532} = 0$
$C_{22} = \min\{\max\{C_{21} + T_{12}, C_{52} + \delta'_{52}\} + P_{212}, \max\{C_{21} + T_{12}, C_{12} + \delta'_{12}\} + P_{222}, \max\{C_{21} + T_{12}, C_{42} + \delta'_{42}\} + P_{232}\} = \min\{\max\{17 + 3, 28 + 1\} + 16, \max\{17 + 3, 26 + 3\} + 10, \max\{17 + 3, 22 + 4\} + 20 = 39$

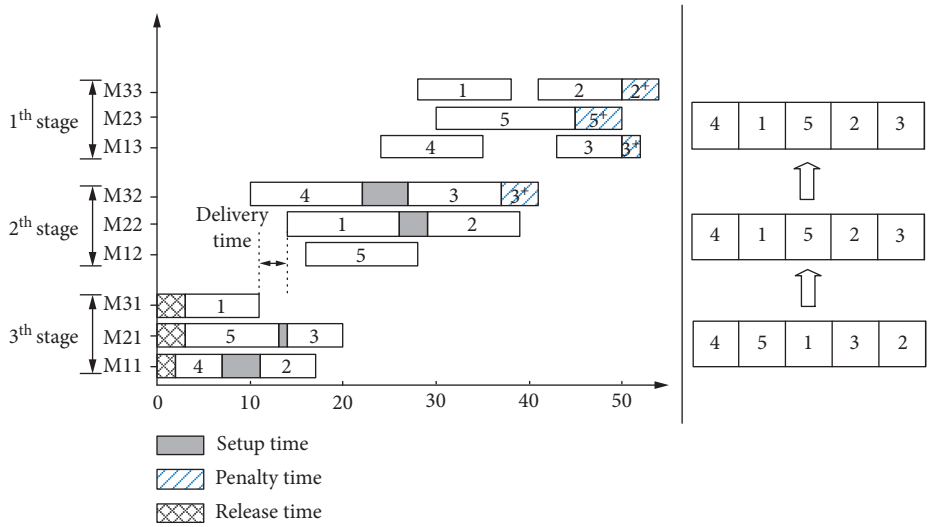

Figure 4: Example Gantt chart.

$(\max\{C_{21} + T_{12},\ C_{52} + \delta'_{52}\} < h_{212},\ \max\{C_{21} + T_{12},\ C_{12} + \delta'_{12}\} < h_{222},\ \max\{C_{21} + T_{12},\ C_{42} + \delta'_{42}\} < h_{232})$
$X_{222} = 1,\ X_{212} = X_{232} = 1$
$C_{32} = \min\{\max\{C_{31} + T_{12},\ C_{52} + \delta'_{53}\} + P_{312} + b_{312},\ \max\{C_{31} + T_{12},\ C_{22} + \delta'_{23}\} + P_{322} + b_{322},\ \max\{C_{31} + T_{12},\ C_{42} + \delta'_{43}\} + P_{332} + b_{332}\} = \min\{\max\{20 + 3, 28 + 1\} + 16 + 2,\ \max\{20 + 3,\ 39 + 2\} + 11 + 6,\ \max\{20 + 3,\ 22 + 5\} + 10 + 4\} = 41$ $(\max\{C_{31} + T_{12}, C_{52} + \delta'_{53}\} > h_{312}, \max\{C_{31} + T_{12}, C_{22} + \delta'_{23}\} > h_{322}, \max\{C_{31} + T_{12},\ C_{42} + \delta'_{43}\} > h_{332})$
$X_{332} = 1,\ X_{312} = X_{322} = 0$

Stage 3:

$\omega^3 = \{4,\ 1,\ 5,\ 2,\ 3\}$
$C_{43} = C_{42} + T_{23} + \min\{P_{413},\ P_{423},\ P_{433}\} = 22 + 2 + \min\{11,\ 14,\ 19\} = 35$ $(C_{42} + T_{23} < h_{413},\ C_{42} + T_{23} < h_{423},\ C_{42} + T_{23} < h_{433})$
$X_{413} = 1,\ X_{423} = X_{433} = 0$
$C_{13} = C_{12} + T_{23} + \min\{P_{123},\ P_{133}\} = 26 + 2 + \min\{13,\ 10\} = 38$
$(C_{12} + T_{23} < h_{123},\ C_{12} + T_{23} < h_{133})$
$X_{133} = 1,\ X_{113} = X_{123} = 0$
$C_{53} = C_{52} + T_{23} + P_{523} + b_{523} = 28 + 2 + 15 + 5 = 50$
$(C_{52} + T_{23} > h_{523})$
$X_{523} = 1,\ X_{513} = X_{533} = 0$
$C_{23} = \min\{\max\{C_{22} + T_{23},\ C_{43} + \delta'_{42}\} + P_{213} + b_{213},\ \max\{C_{22} + T_{23},\ C_{53} + \delta'_{52}\} + P_{223} + b_{223},\ \max\{C_{22} + T_{23},\ C_{13} + \delta'_{12}\} + P_{233} + b_{233}\} = \min\{\max\{39 + 2,\ 35 + 4\} + 16 + 3,\ \max\{39 + 2,\ 50 + 1\} + 11 + 1,\ \max\{39 + 2,\ 38 + 3\} + 9 + 4\} = 54$
$(\max\{C_{22} + T_{23},\ C_{43} + \delta'_{42}\} > h_{213},\ \max\{C_{22} + T_{23},\ C_{53} + \delta'_{52}\} > h_{223},\ \max\{C_{22} + T_{23},\ C_{13} + \delta'_{12}\} > h_{233})$
$X_{233} = 1,\ X_{213} = X_{223} = 0$
$C_{33} = \min\{\max\{C_{32} + T_{23},\ C_{43} + \delta'_{43}\} + P_{313} + b_{313},\ \max\{C_{32} + T_{23},\ C_{53} + \delta'_{53}\} + P_{323} + b_{323},\ \max\{C_{32} + T_{23},\ C_{23} + \delta'_{23}\} + P_{333} + b_{333}\} = \min\{\max\{41 + 2,\ 35 + 5\} + 7 + 2,\ \max\{41 + 2,\ 50 + 1\} + 14 + 10,\ \max\{41 + 2,\ 54 + 2\} + 14 + 10\} = 52$ $(\max\{C_{32} + T_{23},\ C_{43} + \delta'_{43}\} > h_{313},\ \max\{C_{32} + T_{23},\ C_{53} + \delta'_{53}\} > h_{323},\ \max\{C_{32} + T_{23},\ C_{23} + \delta'_{23}\} > h_{333})$
$X_{313} = 1,\ X_{323} = X_{333} = 0$

Therefore, the total weighted completion time is:

$$\sum_i w_i C_{iS} = 5 \times 38 + 6 \times 54 + 9 \times 52 + 7 \times 35 + 3 \times 50 = 1377.$$

3.3. Employed Bee Phase. In the employed bee phase, the new solutions are yielded by performing a genetic algorithm in order to find a neighboring food source from the current food source. The detailed GA is described below.

3.3.1. Fitness Evaluation and Selection Operation. Because the objective of this model is to minimize total weighted completion time, the reciprocal of the objective function is used as the fitness function. Therefore, the fitness function of the yth food source is estimated by $f(y) = 1/\sum_i w_i C_{iS}$. The roulette wheel selection is applied to determine the offspring where the individual with the higher fitness value has the higher probability to be selected.

3.3.2. Single-Point Crossover Operator

Step 1: randomly generate a real number Rc from the interval [0, 1] and compare it with the crossover probability CP. If $Rc \le CP$, two food sources $A = \{\omega_A(1), \omega_A(2), \ldots, \omega_A(i), \ldots, \omega_A(n)\}$ and $B = \{\omega_B(1), \omega_B(2), \ldots, \omega_B(i), \ldots, \omega_B(n)\}$ are picked out randomly from the population after the selection operation.

Step 2: a crossover position θ $(0 < \theta < n)$ is generated randomly, and the single-point crossover is performed on A and B. As shown in Figure 5, the genes after position θ in A (B) are copied to A' (B') by keeping their position unchanged. The lacking genes in A' (B') are filled with the first θ genes in B (A) by maintaining their relative orders.

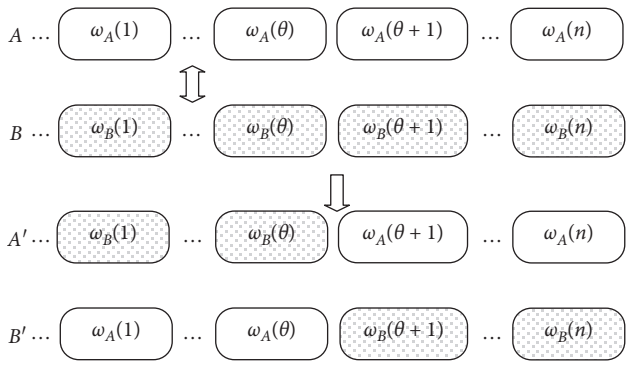

Figure 5: Crossover operator.

Step 3: find the job number (denoted as $\mu(\theta_1)$) in $\{\omega_B(1), \ldots, \omega_B(\theta)\}$ which emerges repeatedly in $\{\omega_A(\theta+1), \ldots, \omega_A(n)\}$ for A'. Similarly, find the job number $\nu(\theta_2)$ from $\{\omega_A(1), \ldots, \omega_A(\theta)\}$ with the same as $\{\omega_B(\theta+1), \ldots, \omega_B(n)$ for B'.

Step 4: replace $\mu(\theta_1)$ in the θ_1th position with $\nu(\theta_2)$ in A' and $\nu(\theta_2)$ in the θ_2th position with $\mu(\theta_1)$ in B'. Therefore, two new food sources are formed.

3.3.3. Multi-Insertion Mutation Operator. A real number Rm is randomly produced from the interval [0, 1]. If Rm is not larger than mutation probability MP, two job positions (θ_1 and θ_2) are randomly chosen in a permutation $Z = \{\omega(1), \omega(2), \ldots, \omega(n)\}$. Suppose that IT is an insertion number. Insert job $\omega(\theta_1)$ into $(\theta_2 + q - 1)$th ($q \in \{1, \ldots, IT\}$) position, as shown in Figure 6. If $\theta_2 + q - 1 > n$, let $\theta_2 + q - 1$ equal to 1. Thus, IT new food sources are obtained to increase the diversity of population.

3.4. Onlooker Bee Phase. This phase uses the same roulette wheel selection applied in the employed bee phase for the food source returned by the employed bee. An onlooker bee chooses a food source ω depending on its probability value $\mathrm{FP}_\omega = f(\omega)/\sum_y f(y)$.

The modified variable neighborhood search (MVNS) is then applied for the above food sources. Firstly, the following neighborhood structures NS_φ ($\varphi = 1, 2, 3$) are designed to help the onlooker bee explore possible promising neighbors.

3.4.1. Exchange Mutation. Two randomly generated positions (θ_1 and θ_2) are chosen and the corresponding jobs are swapped, as shown in Figure 7.

3.4.2. 2-Opt Invert Mutation. Two randomly selected jobs ($\omega(\theta_1)$ and $\omega(\theta_2)$) are picked out and the subsequence between positions θ_1 and θ_2 are inverted, as illustrated in Figure 8.

3.4.3. Fragment Insert Mutation. Randomly select two different jobs $\omega(\theta_1)$ and $\omega(\theta_2)$ and remove the subsequence between positions θ_1 and θ_2. A randomly selected position τ from the remaining positions is chosen, and the subsequence between θ_1 and θ_2 is inserted into position τ, as shown in Figure 9.

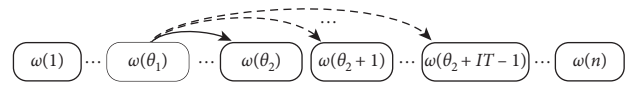

Figure 6: Mutation operator.

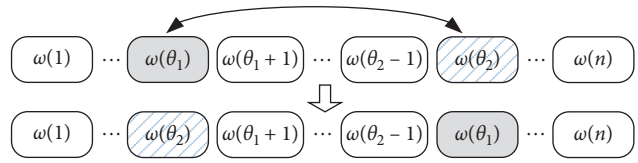

Figure 7: Exchange mutation.

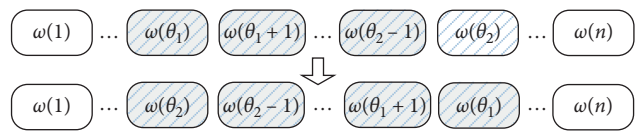

Figure 8: 2-opt invert mutation.

Next, the new neighboring solution obtained by applying these search rules is evaluated and compared to the primary food source. If the new solution obtained by MVNS is better than the primary one, the primary food source is replaced and the new food source becomes a new member in the population.

The procedure of MVNS is listed in Algorithm 1.

3.5. Scout Bee Phase. If a food source has no improvement during *limit* iterations, this employed bee will become a scout bee. A greedy heuristic is developed to produce a new food source including two steps.

3.5.1. Destruction Procedure. Job $\omega(\theta)$ is randomly chosen and removed from the food source $Z = \{\omega(1), \omega(2), \ldots, \omega(\theta-1), \omega(\theta), \omega(\theta+1), \ldots, \omega(n)\}$ so that a remaining food source $Z - \theta = \{\omega(1), \omega(2), \ldots, \omega(\theta-1), \omega(\theta+1) \ldots, \omega(n)\}$ is generated.

3.5.2. Reconstruction Procedure. The selected job $\omega(\theta)$ is sequentially inserted into another randomly selected position of Z-θ, and the corresponding fitness value is calculated. Compare these new generated neighboring solutions and choose the best one with the largest fitness value as the current solution.

4. Experimental Testing and Analyses

In order to verify the performance of the presented algorithm, an extensive experimental comparison with GA, VNS, and GH are provided. All algorithms are coded in

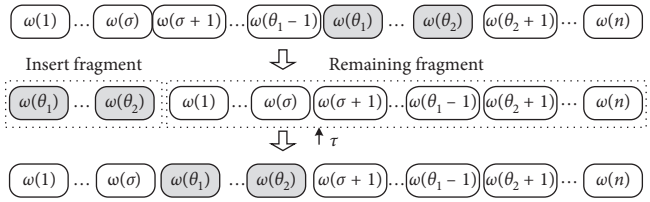

Figure 9: Fragment insert mutation.

Initialization: define NS_φ ($\varphi = 1, 2, 3$)
for every food source ω **do**
Perturbation:
  $x = 1$;
  while $x < 3$;
    execute NS_1 to ω
    $x = x + 1$;
  end while
Search:
  $\varphi = 1$
  while the stopping criterion is not met
    $\sigma =$ execute NS_φ to ω
    If $f(\sigma) > f(\omega)$
      $\omega = \sigma$;
      $\varphi = \varphi$;
    else
      $\varphi = \varphi + 1$;
    end if
  end while
end for

Algorithm 1: Procedure of MVNS.

MATLAB R2014a and run on a Micro Core i5-4210U, 1.7 GHz PC with 4.00 GB. The orthogonal test method is used to optimize the parameters in the IDABC algorithm. The maximum iteration number is set to 100, and the longest running time is 800 s. Test instances are randomly generated as follows:

(1) Job number $n \in \{10, 40, 70, 100, 120\}$ and stage number $S \in \{2, 3, 5\}$. The setting of m_k at stage k is divided into two cases: X and Y. In case X, $m_k = 3$ for $k = 1, \ldots, S$ and in case Y, $m_k \in U[1, 5]$.

(2) The basic processing time, job release time, delivery time between stages, setup time, and job weight are generated from the following discrete uniform distributions: $P_{ijk} \in U[1, 20]$, $r_i \in U[1, 5]$, $T_{k,k+1} \in U[1, 6]$, $\delta_{fijk} \in U[1, 6]$, and $w_i \in U[1, 10]$.

(3) The penalty time b_{ijk} is produced according to the uniform distribution $U[1, 20\psi]$ where $\psi = 0.5$.

(4) Parameter Δ is determined by equation (11), where $\overline{P}_{ik}$ represents the average processing time of job i at stage k defined as equation (12). The deterioration dates are randomly generated from a uniform distribution on the intervals $H_1 \in U[1, 0.7\Delta]$ and $H_2 \in U[0.3\Delta, \Delta]$.

$$\Delta = \sum_{i=1}^{n} \sum_{k=1}^{S} \overline{P_{ik}}, \tag{11}$$

$$\overline{P_{ik}} = \frac{\sum_{j=1}^{m_k} P_{ijk}}{m_k}. \tag{12}$$

(5) Based on the simulation experiments of different parameter values, the insertion number IT applied for employed bees is set to $IT = n \times 20\%$. The number of cycles is used as the stopping criterion of MVNS for onlooker bees and it is fixed to 5.

4.1. Parameter Settings. In the presented IDABC algorithm, four main parameters are included, i.e., *Nump, limit, CP, and MP*. They are optimized through orthogonal experiments where each parameter is viewed as a factor changing at three different levels (see Table 1), resulting in $3^2 = 9$ experiments.

An instance $30 \times 3 \times 3$ ($n = 30$, $S = 3$, and $m_k = 3$) is taken as an example, and every parameter combination is tested ten times. Table 2 gives the orthogonal parameter table $L_9(3^4)$ with nine groups of the parameters, and the mean objective values where Q_y ($y = 1, 2, 3$) in column z ($z = 1, 2, 3, 4$) denotes the average of the results for three sets of factor z at level y. According to the experimental results, the parameters are set as follows: $Nump = 80$, $limit = 8$, $CP = 0.8$, and $MP = 0.5$.

4.2. Simulation Results and Analyses. To make a fair comparison, all the algorithms utilize the same CPU time limit as a termination criterion. All of the cited parameters $\{n, S, m_k, h_{ijk}\}$ mentioned above result in a total of $5 \times 3 \times 2 \times 2 = 60$ different combinations. For each combination, ten random instances are generated. Thus, there are 360 small- and medium-sized instances and 240 large-sized instances to test these algorithms. The relative percentage increase (RPI) is defined as follows:

$$\mathrm{RPI} = \frac{O_\zeta - O_{\mathrm{best}}}{O_{\mathrm{best}}} \times 100, \tag{13}$$

where O_ζ is the objective value obtained by different metaheuristic algorithm and O_{best} is the best objective value found by all algorithms.

Therefore, average RPI (ARPI) can be computed and used as a performance measure to compare the performance of the algorithms. Tables 3 and 4 list the testing results for different sized problems where columns Avg and Min represent the average and minimum of 10 runs of each algorithm, respectively.

It can be concluded from Tables 3 and 4 that the proposed IDABC algorithm performs better than GA, GH, and VNS consistently. Precisely, the IDABC algorithm has the least ARPI 2.00% and 2.46%, respectively, for small-medium-sized problems and large-sized problems, which are smaller than the ARPI of the GA, GH, and VNS, 4.75% and 7.77%, 6.17% and 9.01%, and 5.18% and 8.16%. This indicates that the results obtained by the presented algorithm are

Table 1: Factor settings of orthogonal experiments.

Factor level	Factors			
	Nump	*limit*	*CP*	*MP*
1	60	3	0.2	0.2
2	80	5	0.5	0.5
3	90	8	0.8	0.8

Table 2: Orthogonal array and average objective value.

	Factor				Average objective value
	Nump	*limit*	*CP*	*MP*	
1	60	3	0.2	0.2	7328.7
2	60	5	0.5	0.5	7319.5
3	60	8	0.8	0.8	7325.3
4	80	3	0.5	0.8	7330.6
5	80	5	0.8	0.2	7312.3
6	80	8	0.2	0.5	7302.7
7	90	3	0.8	0.5	7317.0
8	90	5	0.2	0.8	7335.4
9	90	8	0.5	0.2	7308.5
Q_1	7324.5	7325.4	7322.3	7316.5	
Q_2	7315.2	7322.4	7319.5	7313.1	
Q_3	7320.3	7312.2	7318.3	7330.4	

Table 3: Comparison of the four algorithms for small and medium instances.

$n \times S \times m_k \times h$	GA			GH			VNS			IDABC			CPU time
	Avg	Min	ARPI	Avg	Min	ARPI	Avg	Min	ARPI	Avg	Min	ARPI	
$n \times S \times m_k \times H_1$													
$10 \times 2 \times X \times H_1$	1754.4	1710	3.32	1735.8	1704	2.23	1725.6	1704	1.63	1709.4	**1698**	0.07	25.52
$10 \times 2 \times Y \times H_1$	778.2	**765**	2.74	787.8	**765**	2.98	776.7	**765**	1.53	765.3	**765**	0.04	27.19
$10 \times 3 \times X \times H_1$	1081.8	1065	1.73	1081.2	**1053**	2.67	1130.4	1071	7.35	1056.9	**1053**	0.37	49.66
$10 \times 3 \times Y \times H_1$	1592.8	**1552**	3.32	1594.4	1572	2.73	1577.2	**1552**	1.62	1564.8	**1552**	0.82	42.11
$10 \times 5 \times X \times H_1$	4236.1	4135	2.62	4225.1	4119	3.05	4225.9	4101	3.07	4216.0	**4100**	2.83	78.03
$10 \times 5 \times Y \times H_1$	4430.3	**4354**	1.75	4435.9	4368	1.88	4363.8	**4354**	0.22	4358.2	**4354**	0.10	81.33
$40 \times 2 \times X \times H_1$	17268.0	16626	3.35	17857.8	17622	11.01	17208.6	16836	6.98	16736.4	**16808**	4.04	146.63
$40 \times 2 \times Y \times H_1$	16813.6	16192	4.14	17879.2	17480	10.75	17276.0	16872	7.01	16409.6	**16144**	1.65	139.72
$40 \times 3 \times X \times H_1$	39717.0	38140	4.55	40121.3	39170	5.61	40267.2	38860	5.99	38991.2	**37990**	2.63	262.19
$40 \times 3 \times Y \times H_1$	16295.5	15995	3.39	16700.5	16345	5.97	16326.0	15875	3.59	15999.5	**15760**	1.52	189.74
$40 \times 5 \times X \times H_1$	17268.0	**16060**	7.52	17357.8	16626	8.08	17208.6	16836	7.15	16736.4	16086	4.21	346.06
$40 \times 5 \times Y \times H_1$	45633.6	44064	6.29	45929.7	44784	6.99	44965.8	44379	4.74	44474.4	**42930**	3.60	482.23
$70 \times 2 \times X \times H_1$	7312.4	7115	8.70	7445.0	7204	10.67	7397.3	7247	9.96	7163.6	**6727**	6.49	294.42
$70 \times 2 \times Y \times H_1$	42046.9	40474	8.64	43504.3	41874	12.41	43201.9	42287	11.62	40496.4	**38703**	4.63	281.67
$70 \times 3 \times X \times H_1$	39585.9	38406	8.68	39309.6	38298	7.93	39376.8	37869	8.11	37216.2	**36423**	2.18	496.63
$70 \times 3 \times Y \times H_1$	67027.4	64744	6.36	67655.0	65840	7.35	66655.0	64952	5.76	63835.7	**63033**	1.29	429.50
$70 \times 5 \times X \times H_1$	51786.4	50873	7.44	51987.9	51003	7.84	51972.9	51030	7.81	48987.1	**48201**	1.61	734.33
$70 \times 5 \times Y \times H_1$	81461.2	80544	7.00	81714.2	80664	7.33	81300.2	80210	6.79	77476.8	**76134**	1.76	789.32
Average	25338.3	24601	5.09	25629.0	25027	6.53	25386.4	24822	5.61	24344.1	**23803**	2.21	272.02
$n \times S \times m_k \times H_2$													
$10 \times 2 \times X \times H_2$	1126.0	**1108**	1.62	1120.0	**1108**	1.08	1122.4	**1108**	1.30	1108.4	**1108**	0.04	25.52
$10 \times 2 \times Y \times H_2$	1020.6	999	2.47	1029.6	1011	3.37	1020.9	1014	2.50	1001.7	**996**	0.58	27.19
$10 \times 3 \times X \times H_2$	469.9	461	2.37	470.5	461	2.51	468.0	**459**	1.96	460.4	**459**	0.31	49.66
$10 \times 3 \times Y \times H_2$	3003.0	**2926**	2.63	2974.0	**2926**	0.71	2946.3	**2926**	0.69	**2926.4**	2926	0.01	42.11
$10 \times 5 \times X \times H_2$	4823.6	4802	2.02	4831.9	4801	2.20	4856.8	4802	2.72	4804.1	**4728**	1.61	78.03
$10 \times 5 \times Y \times H_2$	4261.6	4200	2.05	4274.4	4200	2.36	4215.2	**4176**	0.94	4176.8	**4176**	0.02	81.33
$40 \times 2 \times X \times H_2$	8677.2	8304	4.76	8999.1	8607	8.65	8837.4	8676	6.69	8530.8	**8232**	2.99	146.63
$40 \times 2 \times Y \times H_2$	6795.0	6507	5.30	7229.7	7050	12.04	6890.4	6711	6.78	6614.4	**6453**	2.50	139.72
$40 \times 3 \times X \times H_2$	22859.4	22218	7.41	25345.8	22872	10.64	22761.0	22110	6.95	21937.8	**21282**	3.08	262.19
$40 \times 3 \times Y \times H_2$	3179.7	3090	3.30	3248.5	3195	5.54	3196.8	3127	3.86	3122.6	**3078**	1.45	189.74

Table 3: Continued.

$n \times S \times m_k \times h$	GA			GH			VNS			IDABC			CPU time
	Avg	Min	ARPI	Avg	Min	ARPI	Avg	Min	ARPI	Avg	Min	ARPI	
$40 \times 5 \times X \times H_2$	39642.2	39099	4.21	39505.9	38739	3.89	39667.5	39494	4.32	38536.0	**38025**	1.34	346.06
$40 \times 5 \times Y \times H_2$	34203.4	33327	4.14	34980.4	34174	6.50	34365.8	33782	4.63	33717.6	**32884**	2.66	482.23
$70 \times 2 \times X \times H_2$	52081.9	50935	5.76	53207.0	52353	8.05	52888.6	52086	7.40	50476.3	**49245**	2.50	294.42
$70 \times 2 \times Y \times H_2$	68669.4	65230	7.97	68177.7	66990	7.20	68780.6	66240	8.14	66003.8	**63600**	3.78	281.67
$70 \times 3 \times X \times H_2$	50135.6	48936	4.57	51162.1	49422	6.71	50552.6	49806	5.44	48794.7	**47946**	1.77	496.63
$70 \times 3 \times Y \times H_2$	24507.3	24162	5.15	24948.9	24729	7.04	24859.2	24399	6.66	24277.6	**23308**	4.15	429.50
$70 \times 5 \times X \times H_2$	61215.0	60342	7.73	61804.5	61083	8.77	61518.9	60497	8.26	57294.3	**56823**	0.83	734.33
$70 \times 5 \times Y \times H_2$	54368.4	53136	5.93	55100.4	53332	7.35	54555.6	53724	6.29	52626.0	**51328**	2.53	789.32
Average	24502.2	23877	4.32	24911.7	24281	5.81	24639.1	24174	4.75	23689.4	**23144**	1.79	272.02

Table 4: Comparison of the four algorithms for large instances.

$n \times S \times m_k \times h$	GA			GH			VNS			IDABC			CPU time
	Avg	Min	ARPI	Avg	Min	ARPI	Avg	Min	ARPI	Avg	Min	ARPI	
$n \times S \times m_k \times H_1$													
$100 \times 2 \times X \times H_1$	85002.6	80394	8.89	86004.0	83364	10.18	86182.8	85224	10.41	81648.6	**78060**	4.60	563.30
$100 \times 2 \times Y \times H_1$	164413.6	162340	8.69	165181.4	164110	9.20	164004.7	160820	8.42	154190.4	**151270**	1.93	570.34
$100 \times 3 \times X \times H_1$	46886.7	45298	6.64	47590.7	45573	8.24	47375.1	45339	7.75	44838.0	**43968**	1.98	674.16
$100 \times 3 \times Y \times H_1$	162497.4	162030	7.42	163215.7	162620	7.90	163404.8	162140	8.02	154372.7	**151270**	2.05	691.13
$100 \times 5 \times X \times H_1$	111831.0	103973	9.28	113080.3	109763	10.50	111534.2	108734	8.99	105269.5	**102336**	2.87	800.00
$100 \times 5 \times Y \times H_1$	27299.4	26998	8.45	27476.4	27194	9.15	27325.2	26392	8.55	25724.9	**25173**	2.19	800.00
$120 \times 2 \times X \times H_1$	67987.9	64257	8.94	69349.3	68724	11.13	68148.8	65922	9.20	64419.3	**62406**	3.23	755.67
$120 \times 2 \times Y \times H_1$	181648.0	174368	8.44	182875.2	175072	9.18	182338.4	176600	8.86	172412.8	**167504**	2.93	759.46
$120 \times 3 \times X \times H_1$	250104.5	244510	11.00	254437.3	251670	12.93	249121.0	242280	10.57	232410.3	**225310**	3.15	800.00
$120 \times 3 \times Y \times H_1$	156395.6	155288	8.31	158088.0	157040	9.48	157724.8	157128	9.23	147907.2	**144400**	2.43	800.00
$120 \times 5 \times X \times H_1$	353503.5	349220	7.68	355892.7	351950	8.40	534840.2	351250	8.08	331862	**328300**	1.08	800.00
$120 \times 5 \times Y \times H_1$	317292.8	303048	6.99	318821.6	312832	7.50	314614.4	311240	6.08	302820.8	**296576**	2.11	800.00
Average	160405.3	155977	8.39	161834.4	159159	9.48	175551.2	157756	8.68	151489.7	**148048**	2.55	734.51
$n \times S \times m_k \times H_2$													
$100 \times 2 \times X \times H_2$	30774.4	30210	4.67	31132.6	30408	5.89	30855.8	30238	4.94	30072.0	**29402**	2.28	563.30
$100 \times 2 \times Y \times H_2$	36300.3	**34101**	6.45	36957.0	36483	8.38	36704.4	36333	7.63	35929.5	34662	5.36	570.34
$100 \times 3 \times X \times H_2$	46820.3	44035	9.09	48176.5	46020	12.25	47780.5	45895	11.32	42967.4	**42920**	0.11	674.16
$100 \times 3 \times Y \times H_2$	96520.6	93430	5.07	97543.1	93552	6.18	97208.4	93372	5.82	92908.5	**91886**	1.13	691.13
$100 \times 5 \times X \times H_2$	233298.4	229788	9.26	235390.2	222711	10.24	233203.8	227548	9.22	215706.6	**213516**	1.03	800.00
$100 \times 5 \times Y \times H_2$	26102.1	25189	9.10	26158.0	25455	9.33	25978.5	25208	8.58	24693.1	**23925**	3.21	800.00
$120 \times 2 \times X \times H_2$	146292.3	141300	3.81	150106.5	148113	6.52	148338.0	144729	5.26	143712.9	**140992**	1.98	755.67
$120 \times 2 \times Y \times H_2$	110558.5	107555	9.29	109579.0	108495	8.32	108672.5	107310	7.42	105177.0	**101165**	3.97	759.46
$120 \times 3 \times X \times H_2$	244468.6	238810	6.30	248267.6	237611	7.96	246137.8	240340	7.03	235697.3	**229970**	2.49	800.00
$120 \times 3 \times Y \times H_2$	73899.5	70101	7.02	74268.4	72523	7.55	74104.5	70653	7.31	70228.2	**69054**	1.70	800.00
$120 \times 5 \times X \times H_2$	282084.6	271104	8.07	283192.5	271120	8.50	279734.2	270824	7.18	266991.9	**261000**	2.30	800.00
$120 \times 5 \times Y \times H_2$	258686.4	245560	7.55	267808.8	264159	11.34	264243.0	257390	9.86	247565.3	**240536**	2.92	800.00
Average	132150.5	127599	7.14	134048.4	129721	8.54	132746.8	129153	7.63	125970.8	**123252**	2.37	734.51

closer to the optimal or near-optimal solutions. For all instances, the average objective values of IDABC outperform consistently the other algorithms, and the best optimal or near-optimal solutions are almost found by IDABC except two combinations ($40 \times 5 \times X$ and $100 \times 2 \times Y$) where the IDABC is slightly worse than the GA.

When the scale of the problem increases, the gap of ARPI values become bigger compared with the other algorithms. For the two tested deterioration dates of H_1 and H_2, it can be seen that the algorithms under H_2 perform better.

To explain the difference statistically among the algorithms, the analysis of variance (ANOVA) is carried out to check the results. The method of residual diagnosis is applied, and the results support the main hypotheses of the ANOVA. The means plot of ARPI and least significant deviation (LSD) intervals for all the algorithms are shown in Figure 10 with LSD intervals at a 95% confidence level. It can be seen from the figure that the intervals obtained by the IDABC algorithm differ significantly from the other algorithms and it provides better stability.

4.3. IDABC Effect Experiments. This section will further study the relationship between the algorithm performance and iteration number. Take $n = \{10, 70, 120\}$, $S = 3$ and $m_k = 3$, and the job deterioration date is set to H_1. Figure 11

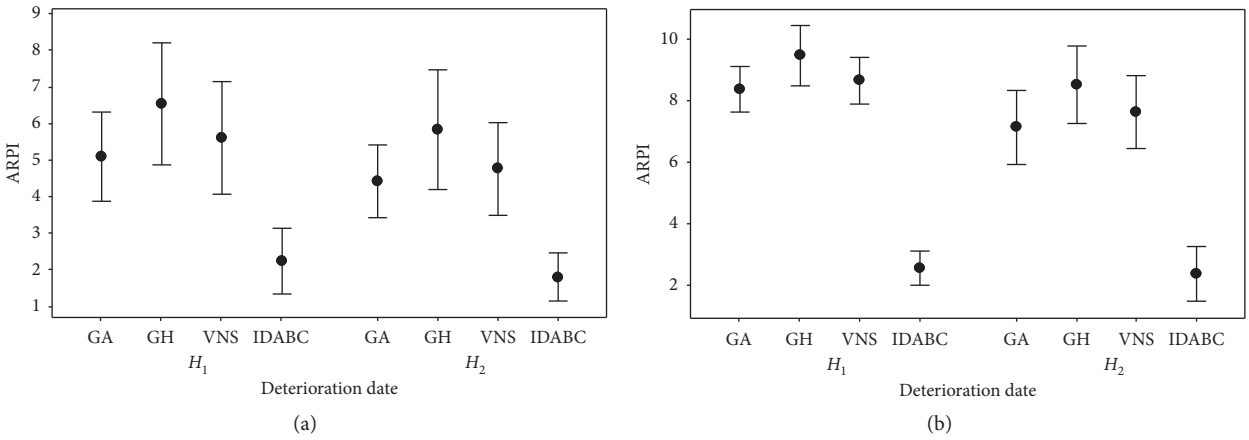

Figure 10: Means plot and LSD intervals with a 95% confidence level for different algorithms. (a) Small- and medium-scale problems. (b) Large-scale problems.

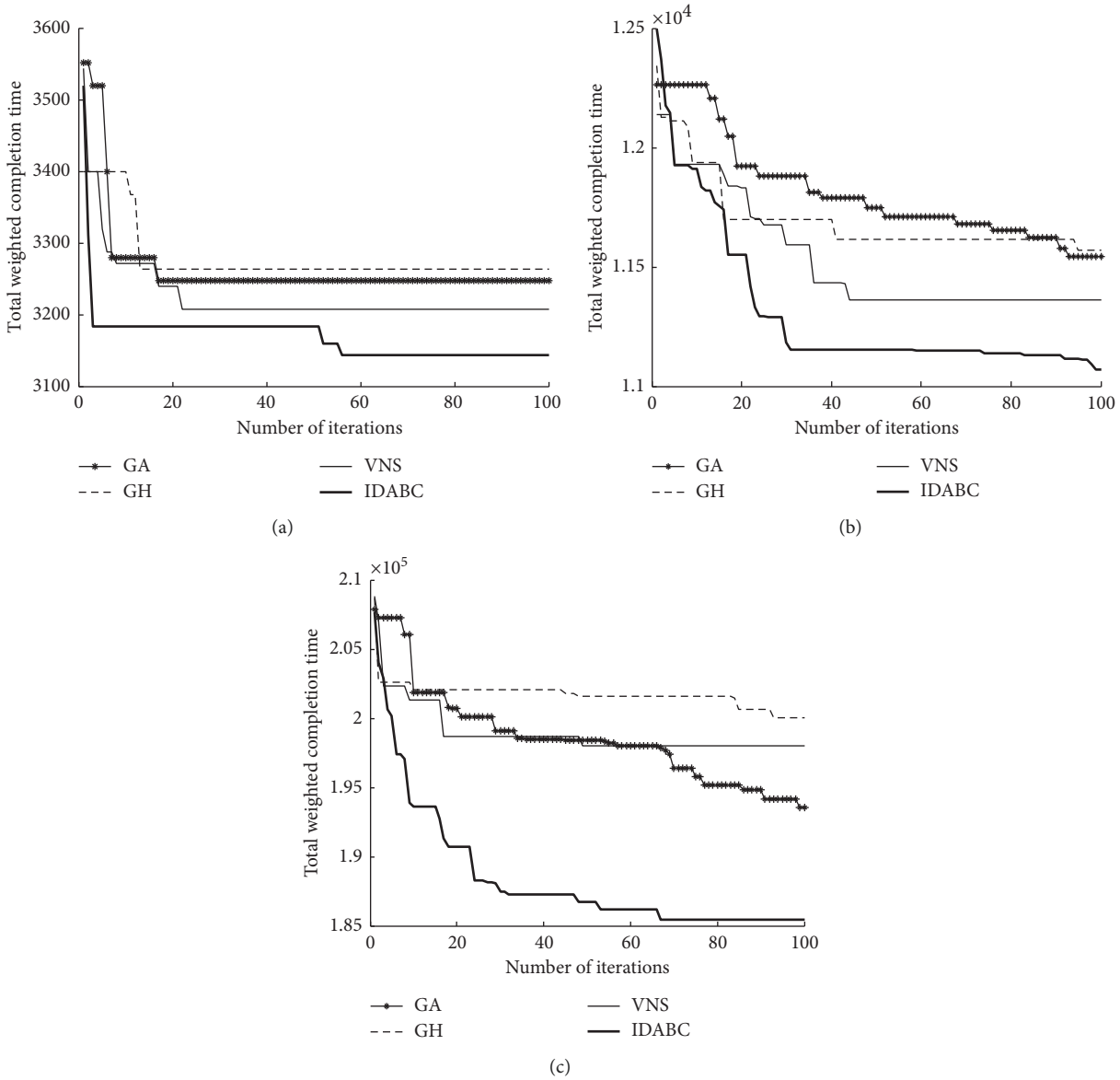

Figure 11: The convergence of different problem sizes. (a) 10 jobs. (b) 70 jobs. (c) 120 jobs.

shows the convergence trend of the three instances for the four algorithms. As seen in the figure, as the job size increases, the IDABC can find better objective values within a less iteration number and have a faster convergence.

5. Conclusions

In this paper, the FFSP with step-deteriorating jobs and sequence-dependent setup times is studied, and the integer programming model is established. An improved discrete artificial bee colony (IDABC) algorithm is then presented for minimum total weighted completion time. In the IDABC algorithm, a permutation-based encoding scheme is applied, decoding with the dynamic generation mechanism to obtain the initial solutions. A genetic algorithm is designed to new offspring solutions in the employed bee phase; a modified variable neighborhood search with perturbation and searching operators is then developed in onlooker bee phase to generate new neighborhood food sources; scouting bees adopt a greedy heuristic to further find better solutions for the nonimproved individual. The comparison with GA, GH, and MVNS verifies the effectivity and efficiency of the IDABC algorithm, especially for large-scale problems.

Some future work may be done from the following aspects: (1) our study involves the single-objective problem of total weighted completion time; the application of the developed algorithm may be extended to solve other single or multiobjective scheduling problems; (2) it is interesting to present some heuristic algorithms to generate initial population instead of the random procedure used in the paper so as to further improve the algorithm performance.

Data Availability

All data generated or analyzed during this study are included in this article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was partly supported by the National Natural Science Foundation of China (grant nos. U1804151 and U1604150).

References

[1] S.-W. Lin, K.-C. Ying, and C.-Y. Huang, "Multiprocessor task scheduling in multistage hybrid flowshops: a hybrid artificial bee colony algorithm with bi-directional planning," *Computers & Operations Research*, vol. 40, no. 5, pp. 1186–1195, 2013.

[2] P. Guo, W. Cheng, and Y. Wang, "Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm," *Engineering Optimization*, vol. 47, no. 11, pp. 1564–1585, 2015.

[3] J. Y. Xu, N. Q. Dong, and S. S. Gu, "Multi-objective permutation flowshop with sequence-dependent setup time," *Computer Integrated Manufacturing Systems*, vol. 19, no. 12, pp. 3170–3176, 2013.

[4] J. N. D. Gupta and S. K. Gupta, "Single facility scheduling with nonlinear processing times," *Computers & Industrial Engineering*, vol. 14, no. 4, pp. 387–393, 1988.

[5] S. Browne and U. Yechiali, "Scheduling deteriorating jobs on a single processor," *Operations Research*, vol. 38, no. 3, pp. 495–498, 1990.

[6] J.-J. Wang and Y.-J. Liu, "Single-machine bicriterion group scheduling with deteriorating setup times and job processing times," *Applied Mathematics and Computation*, vol. 242, pp. 309–314, 2014.

[7] J.-B. Wang and L. Li, "Machine scheduling with deteriorating jobs and modifying maintenance activities," *The Computer Journal*, vol. 61, no. 1, pp. 47–53, 2018.

[8] H. Wang, M. Huang, and J. Wang, "An effective metaheuristic algorithm for flowshop scheduling with deteriorating jobs," *Journal of Intelligent Manufacturing*, vol. 30, no. 7, pp. 2733–2742, 2019.

[9] W. Cheng, P. Guo, Z. Zhang, M. Zeng, and J. Liang, "Variable neighborhood search for parallel machines scheduling problem with step deteriorating jobs," *Mathematical Problems in Engineering*, vol. 2012, Article ID 928312, 20 pages, 2012.

[10] H. Gong and L. X. Tang, "A two-stage flexible flowshop problem with deterioration," *Computing and Combinatorics, Proceedings*, vol. 5092, pp. 651–660, 2008.

[11] X. Li and M. Yin, "A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem," *International Journal of Production Research*, vol. 51, no. 16, pp. 4732–4754, 2013.

[12] X. Wang and L. Tang, "A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem," *Computers & Operations Research*, vol. 79, pp. 60–77, 2017.

[13] Y. Wang, X. Li, and Z. Ma, "A hybrid local search algorithm for the sequence dependent setup times flowshop scheduling problem with makespan criterion," *Sustainability*, vol. 9, no. 12, pp. 1–35, 2017.

[14] Y. Wang and X. Li, "A hybrid chaotic biogeography based optimization for the sequence dependent setup times flowshop scheduling problem with weighted tardiness objective," *IEEE Access*, vol. 5, pp. 26046–26062, 2017.

[15] X. Li and M. Li, "Multiobjective local search algorithm-based decomposition for multiobjective permutation flow shop scheduling problem," *IEEE Transactions on Engineering Management*, vol. 62, no. 4, pp. 544–557, 2015.

[16] G. G. Pedro, C. Andrés, and F.-C. Lario, "An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan," *Expert Systems with Applications*, vol. 39, no. 9, pp. 8095–8107, 2012.

[17] M. K. Marichelvam, A. Azhagurajan, and M. Geetha, "Minimisation of total tardiness in hybrid flowshop scheduling problems with sequence dependent setup times using a discrete firefly algorithm," *International Journal of Operational Research*, vol. 32, no. 1, pp. 114–126, 2018.

[18] Q.-K. Pan, L. Gao, X.-Y. Li, and K.-Z. Gao, "Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times," *Applied Mathematics and Computation*, vol. 303, pp. 89–112, 2017.

[19] B. Santosa and O. A. W. Riyanto, "Hybrid differential evolution-variable neighborhood search to solve multiobjective hybrid flowshop scheduling with job-sequence dependent setup time," *Lecture Notes in Computer Science*, vol. 9712, pp. 587–598, 2016.

[20] X. P. Wang, Z. M. Dong, and L. X. Tang, "Multiobjective differential evolution with personal archive and biased self-adaptive mutation selection," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018, In press.
[21] L. X. Tang, X. P. Wang, and Z. M. Dong, "Adaptive multi-objective differential evolution with reference axis vicinity mechanism," *IEEE Transactions on Cybernetics*, vol. 49, no. 9, pp. 3571–3585, 2019.
[22] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Technical Report TR06, Computer Engineering Department, Erciyes University, Kayseri, Turkey, 2005.
[23] H. Y. Chow, S. Hasan, and S. A. Bareduan, "Basic concept of implementing artificial bee colony (ABC) system in flow shop scheduling," *Applied Mechanics and Materials*, vol. 315, pp. 385–388, 2013.
[24] X. T. Li and S. Ma, "Multiobjective discrete artificial bee colony algorithm for multiobjective permutation flow shop scheduling problem with sequence dependent setup times," *IEEE Transactions on Engineering Management*, vol. 64, no. 2, pp. 149–165, 2017.
[25] K. Peng, Q. Pan, and B. Zhang, "An improved artificial bee colony algorithm for steelmaking–refining–continuous casting scheduling problem," *Chinese Journal of Chemical Engineering*, vol. 26, no. 8, pp. 1727–1735, 2018.
[26] Z. Cui and X. Gu, "An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems," *Neurocomputing*, vol. 148, pp. 248–259, 2015.
[27] J.-q. Li, Q.-k. Pan, and P.-y. Duan, "An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping," *IEEE Transaction on Cybernetics*, vol. 46, no. 6, pp. 1311–1324, 2016.
[28] T. C. E. Cheng and Q. Ding, "Single machine scheduling with step-deteriorating processing times," *European Journal of Operational Research*, vol. 134, no. 3, pp. 623–630, 2001.