Hybrid Artificial Bee Colony Algorithm for a Parallel Batching Distributed Flow-Shop Problem With Deteriorating Jobs

Jun-Qing Li, *Member, IEEE*, Mei-Xian Song, Ling Wang, Pei-Yong Duan, Yu-Yan Han, Hong-Yan Sang, and Quan-Ke Pan

***Abstract*—In this article, we propose a hybrid artificial bee colony (ABC) algorithm to solve a parallel batching distributed flow-shop problem (DFSP) with deteriorating jobs. In the considered problem, there are two stages as follows: 1) in the first stage, a DFSP is studied and 2) after the first stage has been completed, each job is transferred and assembled in the second stage, where the parallel batching constraint is investigated. In the two stages, the deteriorating job constraint is considered. In the proposed algorithm, first, two types of problem-specific heuristics are proposed, namely, the batch assignment and the right-shifting heuristics, which can substantially improve the makespan. Next, the encoding and decoding approaches are developed according to the problem constraints and objectives. Five types of local search operators are designed for the distributed flow shop and parallel batching stages. In addition, a novel scout bee heuristic that considers the useful information that is collected by the global and local best solutions is investigated, which can enhance searching performance. Finally, based on several well-known benchmarks and realistic industrial instances and via comprehensive computational comparison and statistical analysis, the highly effective performance of the proposed algorithm is favorably compared against several algorithms in terms of both solution quality and population diversity.**

***Index Terms*—Artificial bee colony (ABC), deteriorating job, distributed flow shop (DFS), parallel batching.**

Manuscript received March 26, 2019; revised July 28, 2019; accepted September 21, 2019. This work was supported in part by the National Science Foundation of China under Grant 61773192, Grant 61873328, and Grant 61803192, in part by the National Natural Science Fund for Distinguished Young Scholars of China under Grant 61525304, in part by the Shandong Province Higher Educational Science and Technology Program under Grant J17KZ005, in part by the Dongguan Innovative Research Team Program under Grant 2018607202007, in part by the Open Project of Henan Key Laboratory of Intelligent Manufacturing of Mechanical Equipment, Zhengzhou University of Light Industry under Grant IM201906, and in part by the major basic research projects in Shandong under Grant ZR2018ZB0419. This article was recommended by Associate Editor L. Tang. *(Corresponding authors: Ling Wang; Pei-Yong Duan.)*

J.-Q. Li is with the School of Information and Engineering, Shandong Normal University, Jinan 250014, China, and also with the School of Computer Science, Liaocheng University, Liaocheng 252059, China (e-mail: lijunqing@lcu-cs.com).

M.-X. Song, Y.-Y. Han, and H.-Y. Sang are with the School of Computer Science, Liaocheng University, Liaocheng 252059, China.

L. Wang is with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: wangling@tsinghua.edu.cn).

P.-Y. Duan is with the School of Information and Engineering, Shandong Normal University, Jinan 250014, China (e-mail: duanpeiyong@sdnu.edu.cn).

Q.-K. Pan is with the College of Mechanical and Electrical Engineering and Automation, Shanghai University, Shanghai 200444, China.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2019.2943606

I. Introduction

FLOW-SHOP problems (FSPs) have been investigated in recent years and have been utilized in many realistic industrial applications, such as steelmaking systems [1]–[3] and cloud systems [4]. With the development of distributed manufacturing and parallel machines, distributed FSPs (DFSPs) have also been investigated [5]. A DFSP is a distributed generalization of the classical FSP and can be considered an NP-hard problem, where a set N of n jobs is to be processed on a set F of f factories, each of which contains the same set M of m machines [5]. Each job can be assigned to any factory to be processed on the set of M machines of the assigned factory via m operations. In each factory, the scheduling problem can be considered the classical FSP.

In addition, after the distributed manufacturing process, all jobs should be assembled in a processing center [6]. In the realistic production horizon, there are typically parallel machines in the assembly center. Parallel batching and deteriorating jobs should also be considered to align the problem with the actual production scenario [7]. However, no literature is available in which parallel batching and deteriorating constraints are considered.

The artificial bee colony (ABC) has been widely used to solve many types of scheduling problems [8], such as flow shop, hybrid flow shop, and flexible job-shop scheduling problems [2]. The typical advantages of the ABC algorithm are as follows.

1) The ABC consists of three main parts: a) the employed bee; b) the onlooker bee; and c) the scout bee. The employed bee and the onlooker bee conduct the exploitation search, while the scout bee completes the exploration task. Therefore, the ABC algorithm is simple and easy to implement for many continuous and discrete optimization problems.
2) There are fewer parameters in the ABC algorithm.
3) Many applications of the ABC algorithm have demonstrated that the ABC algorithm is efficient for many

types of scheduling problems. Therefore, in this article, we use the ABC algorithm to solve a DFSP with several special and realistic constraints.

The main contributions are as follows: 1) two types of problem-specific structures are presented; 2) two heuristics that consider the problem-specific structures are proposed, namely, batch assignment and right-shifting rules; 3) encoding and decoding approaches are developed according to the problem constraints and objectives; 4) five types of local search operators are designed for the distributed flow shop (DFS) and the parallel batching stages; and 5) a novel scout bee heuristic that considers the useful information that is collected by the global and local best solutions is investigated, which can enhance the searching ability.

The remainder of this article is organized as follows. Section II presents a brief literature review. Section IV describes the problem-specific heuristics. Section V presents the framework of the proposed algorithm. Section VI presents the experimental results and compares them with those of algorithms from the literature to evaluate the performance of the proposed algorithm. Finally, Section VII presents the concluding remarks and discusses future research directions.

II. Literature Review

A. Distributed Flow Shop

Naderi and Ruiz [5] first studied the DFS scheduling problems in 2010, in which the makespan was to be minimized and developed six types of the mixed-integer linear programming (MILP) models. Hatami *et al.* [6] investigated the distributed assembly flow-shop scheduling problems in supply-chain systems and solved them using a variable neighborhood descent (VND) algorithm. In recent years, many metaheuristics have been used to solve DFSPs, which can be classified into several categories according to the problem features.

In DFSPs of the first type, the makespan or the maximum completion time is minimized. To solve problems of this type, researchers have utilized many types of metaheuristics, such as the iterated greedy (IG) algorithm [9], [10]; the scatter search (SS) method by Naderi and Ruiz [11]; the hybrid immune algorithm (HIA) by Xu *et al.* [12]; the chemical reaction optimization (CRO) algorithm by Bargaoui *et al.* [13]; the tabu search algorithm (TS) by Gao *et al.* [14]; and the estimation of the distribution algorithm (EDA) by Wang *et al.* [15].

In DFSPs of the second type, additional constraints are imposed. Wang *et al.* [16] considered a DFSP with machine breakdown constraints. Rifai *et al.* [17] solved distributed re-entrant permutation flow-shop scheduling. Lin and Ying [18] considered a distributed no-wait flow-shop scheduling problem. Ying *et al.* [19] studied a no-idle DFSP. Deng and Wang [20] applied a competitive memetic algorithm to a multiobjective DFSP. To solve assembly DFSPs, many metaheuristics have been proposed, such as the VND algorithm [6]; the backtracking search [21]; the heuristics and metaheuristics [22], [23]; and the biased-randomized metaheuristic [24].

According to the above literature, researchers have studied DFSPs with many types of realistic constraints and the assembly process has also been considered in many studies. However, parallel batching in the assembly stage for DFSPs and deteriorating jobs should also be considered for a realistic production system.

B. Deteriorating Constraints

Deteriorating constraints have been investigated in recent years. The current literature mainly focuses on FSPs with deteriorating jobs. Lu [25] developed an exact solution algorithm for a no-idle flow-shop problem with deteriorating constraints. Bajestani and Beck [26] designed a two-stage coupled algorithm to minimize the flow-shop makespan. Pei *et al.* [7] studied the use of serial batching scheduling of deteriorating jobs in a two-stage supply chain to minimize the makespan. Cheng *et al.* [27] investigated a bicriteria hierarchical optimization of two-machine flow-shop scheduling. Lee *et al.* [28] considered minimizing the total tardiness via a branch-and-bound algorithm. Wang *et al.* [29] developed a multiverse optimizer (MVO) algorithm for the problem. Fu *et al.* [30] investigated the fireworks algorithm for a two-objective stochastic flow-shop problem. Pei *et al.* [31] utilized a hybrid BA-VNS algorithm for coordinated serial-batching scheduling. Lu *et al.* [32] investigated a hybrid ABC-TS algorithm for the unrelated parallel-batching machine scheduling problem.

According to the above literature on deteriorating the constrained scheduling problems, fewer studies have been conducted in which deteriorating constraints in DFSPs are considered.

C. Parallel Batch Scheduling

The parallel batch scheduling problems have been investigated for practical applications, such as the steelmaking casting process. In recent years, many types of metaheuristics have been proposed for and applied to problems of this type. Ham *et al.* [33] developed a constraint programming approach to consider parallel batching machines. Gao *et al.* [34] investigated a two-agent parallel batch scheduling problem. Arroyo *et al.* [35] studied unrelated parallel batch machines with unequal job release times. Zhou *et al.* [36] solved the problem while considering the electricity consumption cost. Tang *et al.* [37] considered a two-agent scheduling problem with deteriorating jobs. Li [38] solved this problem with inclusive processing set restrictions and nonidentical capacities. Zhang *et al.* [39] considered a realistic parallel batch-processing problem in fabric dyeing processes.

According to the above literature on the parallel batch scheduling problems, less work has been conducted on DFSPs in which parallel batch processing is considered.

D. Metaheuristics

During recent years, many types of metaheuristics have been proposed for and applied to many scheduling problems. The ABC algorithm is one of the classic metaheuristics that have been applied to many continuous optimization and scheduling problems. Karaboga and Ozturk [8] utilized the ABC algorithm for a data clustering problem. Gao *et al.* [40]

improved the classical ABC via a modified search equation and orthogonal learning. To solve the discrete scheduling problems, Li *et al.* [41] solved a DFSP with a distance coefficient in a prefabricated system. Pan *et al.* [42] applied a discrete ABC for the lot-streaming FSP. Li *et al.* [43] solved the reverse logistics network system by using the ABC algorithm. In addition, Liu *et al.* [44] studied the crowd evacuation problem in buildings by using the ABC algorithm. Gong *et al.* [45] solved the blocking lot-streaming FSP. Zhou and Yao [46] applied the ABC algorithm to a QoS-based cloud manufacturing system. Li *et al.* [47] investigated the flexible job-shop scheduling problem (FJSSP) with maintenance activities.

According to our analysis of the published literature on metaheuristics, compared with other recently published metaheuristics, such as the teaching-learning-based optimization (TLBO) algorithm [48], the artificial fish swarm (AFS) algorithm [49], the invasive weed optimization (IWO) algorithm [50], and the Jaya algorithm [51], the ABC algorithm has been widely applied to many types of optimization problems.

E. Motivations

From the analysis of DFSPs and various realistic constraints, including deteriorating jobs and parallel batching, it is concluded that the DFSPs have been recently investigated in many studies in the literature, and various studies also focused on realistic constraints. However, there is less literature in which realistic constraints are considered in typical DFSPs. In practice, in the production of a steelmaking casting system, there are typically two stages: 1) the first stage can be considered as a DFSP and 2) the second stage can be considered as a parallel batching problem. The deteriorating job constraint should be considered in realistic systems of this type. Therefore, in this article, to solve the DFSP, we combine the ABC algorithm and the features of other metaheuristics with the two realistic constraints.

III. Problem Description

In this article, we consider a typical process that is carried out in many industrial processes, such as steelmaking and textile processes. In the considered problem, there are two main stages.

1) In the first stage, there are several factories that vary in terms of their processing capabilities. Each job should select one factory to complete its work in the first stage. In each factory, the canonical permutation flow-shop scheduling problem is embedded, namely, there are several jobs to be processed in each factory with the same fixed processing sequence from the first to the last machine.
2) All of the jobs that are processed in the first stage should be processed in the second stage, where they should select exactly one machine from several parallel machines. Several jobs should be grouped into a batch to undergo the same process, and the deteriorating processing time of all jobs was considered.

A. Problem Formulation

1) Assumptions:

1) All jobs are released at time zero and must be processed from the first to the second stage.
2) All machines are available at time zero and remain continuously available over the entire production horizon.
3) The normal processing times for each job are predefined.
4) In the first stage, each job should select exactly one factory and be processed from the first to the last machine.
5) In the second stage, there are several parallel machines, where all jobs must be assigned to a specified batch and each batch must be assigned to a specified machine.
6) Each batch in the last stage can handle at most c jobs in parallel mode simultaneously.
7) The longest normal processing time for all jobs in the same batch is set as the normal processing time of the batch.
8) The job-processing deterioration effect is considered, where the actual processing time is calculated as follows: $p_{ij}^A = p_{ij} + \beta t$, where t is the starting time of job J_j on M_i, and β is a common deterioration rate for all jobs.
9) There are sufficiently many buffers between the two continuous stages such that after the completion of the first stage, any job can be passed to the next stage immediately.

2) Notation:

Indices:

i	Index of the machines.
j	Index of the jobs.
e	Index of the batches.
f	Index of the factories.
k	Index of the stages.

Parameters:

n	Total number of jobs.
m	Total number of machines.
m_1	Total number of machines in the first stage.
m_2	Total number of machines in the second stage.
s	Total number of factories.
c	Batch size.
J_j	Job j, for $j = 1, 2, \ldots, n$.
M_i	Machine i, for $i = 1, 2, \ldots, m$.
O_{ij}	ith operation of job j.
n_i	Number of jobs that are processed on M_i, $i = 1, 2, \ldots, m$.
J_{ir}	rth job that is processed on M_i. $i = 1, 2, \ldots, m$, $r = 1, 2, \ldots, m$.
B_i	Number of batches on M_i. $i = 1, 2, \ldots, m$; $\lceil (n_i/c) \rceil \le B_i \le n_i$.
b_{ei}	ith batch that is processed on M_i. $i = 1, 2, \ldots, m$, $e = 1, 2, \ldots, B_i$.
$\lvert b_{ei} \rvert$	Number of jobs in b_{ei}.
p_{ij}	Normal processing time of J_j on M_i. $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$.
p_{ij}^A	Actual processing time of J_j on M_i. $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$.

p^A_{ei} Actual processing time of b_{ei}. $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$.
β Common deterioration rate for all jobs.
L Very large number.

Decision Variables:

x_{jf} Binary value that is set to 1 when job J_j is assigned to factory f and otherwise set to 0.
z_{ij} Binary value that is set to 1 when job J_j is processed on M_i and otherwise set to 0.
y_{ije} Binary value that is set to 1 when job J_j is assigned to the batch e on machine M_i and otherwise set to 0.
SJ_{jk} Processing starting time of J_j in stage k, $k = 1, 2$, $j = 1, 2, \ldots, n$.
$S(O_{ij})$ Processing starting time of O_{ij}. $i = 1, 2, \ldots, m, j = 1, 2, \ldots, n$.
$S(b_{ei})$ Processing starting time of b_{ei}. $i = 1, 2, \ldots, m$.
CJ_{jk} Processing completion time of J_j in stage k, $k = 1, 2, j = 1, 2, \ldots, n$.
$C(O_{ij})$ Processing completion time of O_{ij}. $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$.
$C(b_{ei})$ Processing completion time of b_{ei}. $i = 1, 2, \ldots, m$.
$C^i_{\max}$ Completion time of M_i. $i = 1, 2, \ldots, m$.
$C_{\max}$ Makespan.

For a specified schedule, let $C^i_{\max}$ and $C_{\max}$ denote the completion times of the machine and the makespan, respectively. Then, $C_{\max} = \max_{1 \le i \le m} C^i_{\max}$.

The main tasks of this problem are as follows:

1) assign jobs to special factories;
2) schedule jobs in the assigned factories;
3) assign jobs to special batches;
4) assign batches to special machines;
5) schedule batches on the assigned machines.

According to the three-field notation of Graham *et al.*, formally, the considered problem is denoted as DF, R_m, $VM|p-\text{batch}, t_0$, and $p^A_{ij} = p_{ij} + \beta t | C_{\max}$. In the first field, DF represents the DFS, and R_m represents the unrelated parallel machines. In the second field, p-batch represents the parallel batching in the second stage, t_0 indicates that the parallel machines in the second stage are available at time t_0, and $p^A_{ij} = p_{ij} + \beta t$ is the expression for the normal processing time.

Based on the notation, the considered problem can be represented as a mixed-integer programming model as follows:

$$\text{Minimize} \quad C_{\max} \tag{1}$$

$$\text{s.t.} \quad \sum_{f=1}^{s} x_{jf} = 1, j = 1, 2, \ldots, n \tag{2}$$

$$\sum_{i=m_1}^{m_2} z_{ij} = 1, j = 1, 2, \ldots, n \tag{3}$$

$$\sum_{e=1}^{B_i} y_{ije} = 1, i = m_1 + 1, m_1 + 2, \ldots, m \quad j = 1, 2, \ldots, n \tag{4}$$

$$\sum_{j=1}^{n} y_{ije} \le c, i = m_1 + 1, m_1 + 2, \ldots, m; \forall e \tag{5}$$

$$C(b_{ei}) = S(b_{ei}) + p^A_{ei}, i = m_1 + 1, m_1 + 2, \ldots, m \quad \forall e \tag{6}$$

$$C^i_{\max} \ge C(b_{ei}), i = m_1 + 1, m_1 + 2, \ldots, m; e = m \tag{7}$$

$$C_{\max} \ge C^i_{\max}, i = m_1 + 1, m_1 + 2, \ldots, m \tag{8}$$

$$S(b_{ei}) \le S(O_{ij}) \times y_{ije} + L \times (1 - y_{ije}) \quad i = m_1 + 1, m_1 + 2, \ldots, m; \tag{9}$$

$$C(b_{ei}) \ge C(O_{ij}) \times y_{ije}, i = m_1 + 1, m_1 + 2, \ldots, m \quad j = 1, 2, \ldots, n; e = 1, 2, \ldots, B_i. \tag{10}$$

The objective is to minimize the makespan. Constraint (2) ensures that each job can be assigned to exactly one factory in the first stage. Constraint (3) ensures that each job can be assigned to exactly one machine in the second stage. Constraint (4) ensures that each job can be assigned to exactly one batch in the second stage. Constraint (5) ensures that each batch can contain at most c jobs simultaneously. Constraint (6) calculates the completion time of each batch. Constraints (7) and (8) ensure that the makespan is larger than the completion time of each machine at the second stage and the completion time of the last batch on each machine. Constraints (9) and (10) ensure that the starting time of each batch is earlier than the starting time of all jobs that are assigned to the batch and that the completion time of each batch is larger than the completion time of all jobs that are assigned to the batch.

B. Realistic Problem Example

Fig. 1 presents a detailed illustration of the considered realistic DFSP in a steelmaking casting system, where both the deteriorating job and parallel batching constraints are imposed. There are four blast furnaces, each of which can provide a specified quantity of molten iron. Those pouring molten iron are transported by many torpedo cars (TPCs). Each TPC or charge or job should be transported to one of the available factories, and the factories differ in terms of their processing abilities. In each factory, the molten iron in the TPC should be processed through at least three stages: 1) slag skimming; 2) desulfurization; and 3) reladling stages. All of the factories have the same processing operations for each TPC. After being processed at the assigned factory, the TPC will be transferred to the second stage, where one machine should be selected from the parallel batch machines for a continuous casting procedure. On the assigned batch machine, a specified number of charges should be processed in a batch. It is concluded from Fig. 1 that the complete processes in the specified steelmaking casting system can be considered as a typical DFSP with parallel batching in the last stage. In addition, the deteriorating job constraint should be considered in the realistic production systems of this type.

Fig. 2 presents a Gantt chart for an example solution for a realistic problem of this type, where there are two factories and each has three processes for each charge. In the last stage, there are two parallel batch machines. The objective of the considered problem is to compute the maximum completion time or makespan for the parallel machines in the last stage.

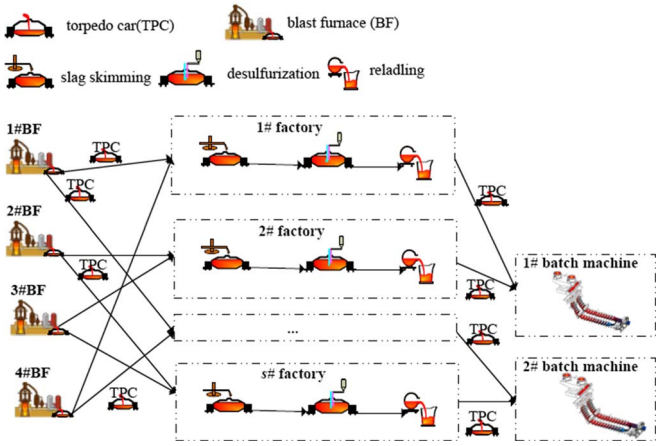


Fig. 1. Realistic DFSP problem in a steelmaking system.

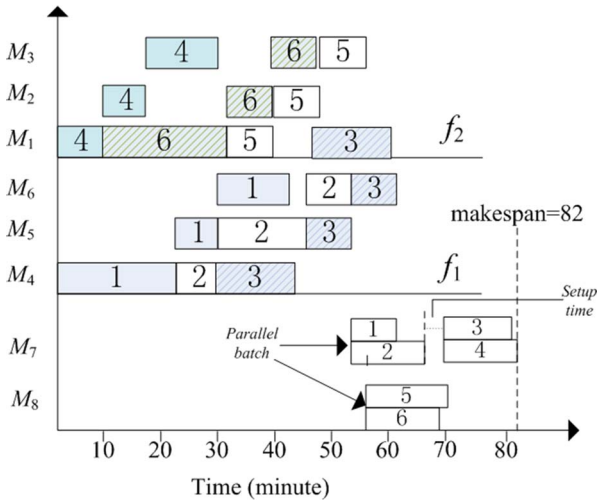


Fig. 2. Gantt example for the problem.

In this example, the fitness value, namely, the makespan, is 82 min.

IV. Problem-Specific Heuristics

In this section, two problem-specific lemmas that are related to the batch assignment and the right shifting are presented and proven. Then, two heuristics that utilize these two lemmas are designed, which can improve the fitness value of a specified solution and, therefore, improve the performance of the algorithm.

A. Batch-Related Lemmas

Considering the problem structure that is related to the parallel batching in the last stage, we propose three lemmas: Lemmas 1 and 2 are applied to minimize the processing time of a specified batch via a right-shifting procedure and Lemma 3 is used to group the jobs into batches, namely, to complete the batch assignment process.

Lemma 1: Consider a batch b_{ei} with several jobs $\{J_{b_{ei_1}}, J_{b_{ei_2}}, \ldots, J_{b_{ei_{|b_{ei}|}}}\}$ that are sorted in ascending order according to their starting times: $S(J_{b_{ei_1}}) \le S(J_{b_{ei_2}}) \le \cdots \le S(J_{b_{ei_{|b_{ei}|}}})$. Without considering other batches on the same machine, simply right shifting the jobs to postpone the starting time and minimizing the processing time of the entire batch does not minimize the completion time of the batch.

Proof: Suppose batch b_{ei} contains the set of jobs $\{J_{b_{ei_1}}, J_{b_{ei_2}}, \ldots, J_{b_{eir}}\}$ that have completion times that are earlier than $C(b_{ei})$, namely, $C(J_{b_{ei_1}}) \le C(J_{b_{ei_2}}) \le \cdots < C(b_{ei})$. Given $C(b_{eir}) = \max_{k=1\to r} C(J_{b_{ei_k}})$ and $\chi_{ei} = C(b_{ei}) - C(b_{eir})$, we can right shift all of the earlier rth jobs by χ_{ei} time units. Then, the starting time of b_{ei} can be calculated as $S'(b_{ei}) = S(b_{ei}) + \chi_{ei}$, and the batch completion time can be calculated as follows:

$$\begin{aligned} C'(b_{ei}) &= S'(b_{ei}) + p^A_{ei} = p'_{ei} + (1+\beta)S'(b_{ei}) \\ &= p'_{ei} + (1+\beta)(S(b_{ei}) + \chi_{ei}) \\ &= p_{ei} - \chi_{ei} + (1+\beta)S(b_{ei}) + (1+\beta)\chi_{ei} \\ &= C(b_{ei}) + \beta\chi_{ei} > C(b_{ei}). \end{aligned}$$

Thus, the lemma is proven. ■

Lemma 2 (Batch Right-Shifting Lemma): Consider a schedule $\pi = (b_{ei}, b_{e+1,i})$, where $b_{ei} = \{J_{b_{ei_1}}, J_{b_{ei_2}}, \ldots, J_{b_{eir}}\}$ and $b_{e+1,i} = \{J_{b_{e+1,i_1}}, J_{b_{e+1,i_2}}, \ldots, J_{b_{e+1,ir}}\}$. Suppose all jobs are sorted in ascending order according to their starting times. The two batches are scheduled consecutively, namely, $S(b_{e+1,i}) = C(b_{e,i})$. Suppose that prior to scheduling, the two batches overlap with each other, namely, $S^-(b_{e+1,i}) < C(b_{e,i})$, where $S^-(b_{e+1,i})$ is the earliest starting time among all jobs whose completion times are in $b_{e+1,i}$ in the first stage. Then, the parts of jobs in $b_{e+1,i}$ are right shifted by $\gamma_{e+1,i} = \min(\chi_{e+1,i}, C(b_{e,i}) - S^-(b_{e+1,i}))$ time units and, subsequently, all of batch $b_{e+1,i}$ is left shifted by $\gamma_{e+1,i}$ time units, thereby decreasing the makespans of the two batches.

Proof: Suppose batch $b_{e+1,i}$ contains the set $\{J_{b_{e+1,i_1}}, J_{b_{e+1,i_2}}, \ldots, J_{b_{e+1,ir}}\}$ of jobs that have completion times that are earlier than $C(b_{e+1,i})$, namely, $C(J_{b_{e+1,i_1}}) \le C(J_{b_{e+1,i_2}}) \le \cdots < C(b_{e+1,i})$. Given $C(b_{e+1,ir}) = \max_{k=1\to r} C(J_{b_{e+1,i_k}})$ and $\chi_{e+1,i} = C(b_{e+1,i}) - C(b_{e+1,ir})$, we can right shift the earlier r jobs by $\gamma_{e+1,i} = \min(\chi_{e+1,i}, C(b_{e,i}) - S^-(b_{e+1,i}))$ time units and, subsequently, left shift all of batch $b_{e+1,i}$ by $\gamma_{e+1,i}$ time units. ■

Then, the schedule completion time is calculated as follows:

$$\begin{aligned} C'(b_{e+1,i}) &= C(b_{ei}) + p^{A'}_{e+1,i} = C(b_{ei}) + p^A_{e+1,i} - \gamma_{e+1,\mathrm{i}} \\ &= C(b_{e+1,i}) - \gamma_{e+1,\mathrm{i}} < C(b_{e+1,i}). \end{aligned}$$

Thus, the lemma is proven.

Lemma 3 (Batch Assignment Lemma): Let $\theta_{j2} = CJ_{j2} + \sum_{i=m_1+1}^{m} p_{ij} \cdot z_{ij}$ be the possible starting time of J_j in the second stage if only the completion time in the first stage is considered. In the last stage, on a specified machine, the jobs in the batches should be performed consecutively in ascending order of their φ_{j2} values.

Proof: See Fig. 3. Suppose that there exists an optimal scheduling $\pi = \{S_1, b_{ei}, \ldots, b_{hi}, S_2\}$, with $JS_{\alpha i}, JS_{\gamma i} \in b_{ei}$, $JS_{\varphi i} \in b_{hi}$, and $\theta_{\alpha 2} < \theta_{\varphi 2} < \theta_{\gamma 2}$. Consider a schedule $\pi' = \{S_1, b_{ei} \backslash JS_{\gamma i} \cup JS_{\varphi i}, \ldots, b_{hi} \backslash JS_{\varphi i} \cup JS_{\gamma i}, S_2\}$, which we obtain by swapping the two jobs in the two batches. From $C(b_{ei}) = \max_{j \in b_{ei}} \theta_{j2}$, it follows that $C'(b_{ei}) < C(b_{ei})$ and the

Algorithm 1: Batch Assignment Heuristic

Step 1. Sort all jobs in ascending order of their θ_{j2} values, such that $\theta_{12} \le \theta_{22} \le \cdots \le \theta_{n2}$, and store them in a job list.
Step 2. Group the first c jobs in batch b_{ei} and remove them from the job list.
Step 3. (1) Set the starting time of b_{ei} as the earliest starting time of all jobs in b_{ei}, namely, $S(b_{ei}) = \min_{j \in b_{ei}} S(O_{ij})$. (2) Set the completion time of b_{ei} as the maximum completion time of all jobs, namely, $C(b_{ei}) = \max_{j \in b_{ei}} C(O_{ij})$. (3) Set the normal processing time of b_{ei} as the difference between the completion and the starting time of the batch, namely, set $p_{ei} = C(b_{ei}) - S(b_{ei})$.
Step 4. Repeat steps 2 and 3 until only $n - (n/c - 1)c$ jobs remain in the job list.
Step 5. Put the remaining $n - (n/c - 1)c$ jobs in a batch.

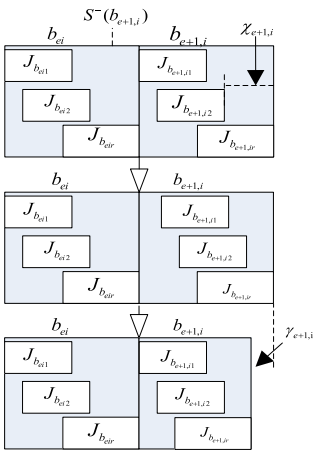


Fig. 3. Example of the right-shift heuristic.

starting time of b_{hi} in π' can be left shifted by $\delta = \theta_{\gamma 2} - \theta_{\varphi 2}$ time units. Therefore

$$\begin{aligned} C'(b_{hi}) &= p_{hi}^{A'} + S(b'_{hi}) = p_{hi} + \delta + (1+\beta)(S(b_{hi}) - \delta) \\ &= p_{hi} + (1+\beta)(S(b_{hi})) - \beta\delta \\ &= C(b_{hi}) - \beta\delta < C(b_{hi}). \end{aligned}$$

$C^i_{\max}$ will not increase after the exchange, namely, $C^i_{\max}(\pi) \ge C^i_{\max}(\pi')$. Thus, the lemma is proven. ■

B. Batch Assignment Heuristic

All jobs will differ in terms of their completion times in the first stage. Therefore, after transporting them to the second parallel-batching stage, the jobs that have similar completion times should be grouped into the same batch, which is the main strategy of Lemma 3. Based on Lemma 3, we proposed a batch assignment heuristic, which is presented as Algorithm 1. The first step of this heuristic is to sort the jobs in ascending order of their θ_{j2} values. Then, the first c jobs are selected for a batch and the process is repeated until all jobs have been assigned.

The time complexity of the heuristic is $O(n \log n)$.

C. Right-Shifting Heuristic

After assigning each job to a batch, the next task is to schedule each batch on the assigned machine. Based on Lemmas 1 and 2, the processing time of a batch will be decreased by right shifting or postponing several jobs in the batch. Therefore, we propose a right-shifting heuristic that is based on Lemmas 1 and 2, which is presented as Algorithm 2. The time complexity of the heuristic is $O(n \log n)$.

Algorithm 2: Right-Shifting Heuristic

Step 1. Sort all batches in ascending order of their p_{ei} values, such that $p_{1i} \le p_{2i} \le \cdots \le p_{Bi}$, and store them in a batch list.
Step 2. Fetch the first batch b_{ei} in the batch list and schedule it on the assigned machine via the following steps:
(1) Obtain the following three values: $S^-(b_{ei})$, which is the starting time of b_{ei}; $\xi(b_{ei})$, which is the maximum completion time of all jobs in b_{ei} in the first stage; and $I(M_i)$, which is the earliest idle time of machine M_i. Let $\tau(b_{ei}) = \min(\xi(b_{ei}), I(M_i))$ and let $\chi_{ei} = \min_{j \in b_{ei} \wedge S(O_{ij}) < S^-(b_{ei})} C(O_{ij}) - C(b_{ei})$ be the minimum completion difference value over all jobs for which the starting time is earlier than that of the batch.
(2) Based on Lemma 2, if $S^-(b_{ei}) < \tau(b_{ei})$, let $\gamma_{ei} = \min(\chi_{ei}, \tau(b_{ei}) - S^-(b_{ei}))$ and right-shift the jobs that satisfy the condition $j \in b_{ei} \wedge S(O_{ij}) < S^-(b_{ei})$ by γ_{ei} time units; otherwise, let $S(b_{ei}) = S^-(b_{ei}) + \gamma_{ei}$, $C(b_{ei}) = C(b_{ei})$, and $I(M_i) = C(b_{ei})$.
(3) If $S^-(b_{ei}) \ge \tau(b_{ei})$, let $S(b_{ei}) = \tau(b_{ei})$, $C(b_{ei}) = S(b_{ei}) + p^A_{ei}$, and $I(M_i) = C(b_{ei})$.
(4) Remove batch b_{ei}.
Step 3. Repeat step 2 until there no batches remain in the list.

V. Framework of the Proposed Algorithm

A. Encoding

To encode a solution for the considered problem, the following tasks should be performed: 1) in the first stage, assign suitable factories for each job; 2) in the first stage, schedule each job on the assigned factory; and 3) in the second stage, assign suitable machines for each batch. Considering these three tasks, we proposed a well-designed encoding approach.

The encoding approach consists of three parts: 1) the first part assigns the jobs to each factory in the first stage; 2) the second part specifies the scheduling sequence for all jobs assigned to each factory; and 3) the third part assigns the machine for each batch.

As shown in Fig. 4(a), the number of first dimensions for a solution is equal to the number of factories; each position corresponds to a factory and has a dimension of values. The second dimension contains a series of numerical values, where each value in the position denotes the scheduled job in the corresponding factory. According to Fig. 4(a), the first three jobs, namely, J_1, J_2, and J_3, are processed in the first factory, whereas J_4, J_6, and J_5 are assigned to the second factory. The scheduling sequence of each job in each factory is also specified in the encoding representation in the first stage.

According to Fig. 4(b), in the second stage, the solution representation assigns the batch to each machine and the

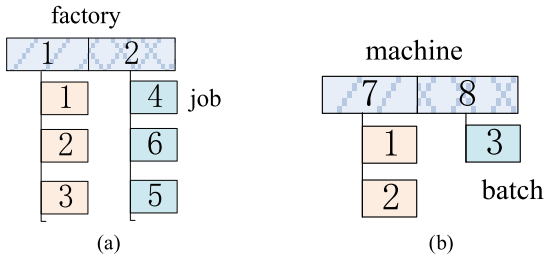


Fig. 4. Encoding. Coding for the (a) first stage and (b) second stage.

scheduling sequence of each batch on each machine is specified. For example, for machine M_7, two batches are assigned and the scheduling sequence is the first batch followed by the second batch.

B. Decoding Method

We decode a specified solution representation to complete the following tasks.

1) In the first stage, we assign each job to suitable factories and schedule each job according to the properties that are specified in Section III and the constraints that are presented in (2)–(10).
2) In the second stage, we assign each job to suitable batches using the batch assignment heuristic (see Section IV-B), we right shift each job in each batch by using the right-shifting heuristic (see Section IV-C), we assign each batch to suitable machines, and we schedule each batch on the assigned machines. The detailed decoding process is presented in Algorithm 3.

C. Local Search Operators

Based on the proposed coding scheme, two local search operators are presented in this section.

For the first part of the solution representation, two search operators are embedded: 1) the swap and 2) insert operators.

For the swap operator, one of the following two types is selected randomly: 1) LS-I, which randomly selects two positions x and y from the job list in a randomly selected factory and swaps the two elements in the two positions and 2) LS-II, which randomly selects two positions, namely, x and y, from the job lists in two factories that are also randomly selected and swaps the two elements in the two positions.

For the insert operator, the following two search operators are embedded: 1) LS-III, which randomly selects two positions, namely, x and y, from the job list in a randomly selected factory and inserts the job in position y to position x and 2) LS-IV, which randomly selects a job and inserts it into a different factory.

For the second part of the solution representation, the swap operator, namely, LS-V, is utilized as follows: first, two batches are randomly selected from two machines that are also selected randomly and, subsequently, the two batches are swapped.

D. Employed Bee Phase

The employed bee is used to perform a local search for each assigned food source or solution. Improving and enhancing the local search performance is the main task of the employed bee. The proposed algorithm will enhance the exploitation performance by combining the proposed heuristics, namely, the batch assignment heuristic, the right-shifting heuristic, and the local search heuristics. The employed bee phase is implemented in Algorithm 4.

Algorithm 3: Decoding Method

Step 1. In the first stage, for each factory, schedule each assigned job in the canonical permutation flow shop scheduling problem as follows: (1) for the first machine in the factory, schedule the jobs one by one according to their occurrence sequence in the first part of the solution representation, as illustrated in Fig. 4 (a), and (2) for the other machines in the factory, schedule each job immediately after its completion on the previous machine.
Step 2. In the second stage, perform the following steps:
Step 2.1. Assign each job to a suitable batch via the batch assignment rule, which was discussed in Section IV-B.
Step 2.2. For each machine in the last stage, schedule each assigned batch in the second part of the solution representation according to the right-shifting rule, which was discussed in Section IV-C.
Step 3. Calculate the last completion time for each machine in the second stage and obtain the maximum completion time as the makespan value.

Algorithm 4: Employed Bee Heuristic

Step 1. For each solution, perform the following steps:
Step 2. For the first stage, apply the four local search operators, i.e., LS-I, LS-II, LS-III, and LS-IV, in a random way.
Step 3. In the second stage, perform the following steps:
Step 3.1. Assign each job to a suitable batch via the batch assignment rule, which is discussed in Section IV-B.
Step 3.2. For each machine in the last stage, schedule each assigned batch in the second part of the solution representation according to the right-shifting rule, which is discussed in Section IV-C.
Step 3.3 Perform the local search by applying operator LS-V, which is discussed in Section V-C.
Step 4. Calculate the last completion time for each machine in the second stage and obtain the maximum completion time as the makespan value.
Step 5. Replace the best solution that has been found so far and the current solution with the newly generated solution if the newly generated solution is better than the best and current solutions.

E. Onlooker Bee Phase

The onlooker bee is used to perform a local search around the better solution that was found by the employed bee. The canonical selection approach of the onlooker bee is the roulette wheel approach. However, ordering the solutions will require additional computations. Therefore, in this article, we use a simple onlooker bee selection method to select the solution for each onlooker bee that is based on comparison with two randomly selected solutions. The steps are presented in detail in Algorithm 5.

F. Scout Bee Phase

The scout bee is used to replace a solution that has not improved after a specified number of iterations. The main objective of the scout is to improve the global search

Algorithm 5: Onlooker Bee Heuristic

Step 1. Randomly select two solutions in the current population, namely, p_1 and p_2.
Step 2. Select the better solution as the onlooker bee and apply similar operators to those of the employed bee.
Step 3. Perform similar elitist and update procedures to those in Step 5 in Algorithm 4.

Algorithm 6: Onlooker bee heuristic

Input: the current population
Output: fa scout bee solution

In the current population, select the following $k+1$ solutions: (1) the global best solution that has been found so far and (2) the local best solutions that have been found by each solution.
Inspired by the EDA algorithm, construct a probability model based on the $k+1$ solutions as follows:
  for each factory **do**
    Calculate the minimum number of jobs that are assigned to it by the specified $k+1$ solutions and store these numbers in a set: $nf = \{f_1, f_2, \ldots, f_s\}$.
  end
  Form a partial matrix, namely, *ss*, where the number of columns for the first factory is f_1, and that for the second factory is f_2, and so on.
  Calculate the matrix transpose, namely, ss^{T}, of the partial matrix *ss*.
  Calculate the probability matrix, namely, *pm*, to record the occurrence number for each job in each row.
  Calculate the matrix *pa* to record the selection probability for each job in each row.
  While unprocessed rows exist **do**
    Select the candidate job that has the maximum selection probability in each row
    Set the selection probability for the selected job to zero or unavailable in the following rows.
  end
  for the remaining unselected jobs **do**
    Find the optimal position in each factory
    Schedule it to construct the final neighboring solution
    Set the constructed solution as the scout bee solution
  end
Generate a solution, namely, x, by the probability model.
Set solution x as the current scout bee and replace the solution that has the longest iterations without any improvement with x.

performance of the canonical ABC algorithm and to avoid early convergence. In the canonical ABC, a randomly generated solution is used as the scout bee. However, a random solution always contains less useful information, and the solutions that are discarded from the generations always contain additional valuable information. Inspired by the EDA algorithm, in which several best solutions are selected and a probability model is constructed and used to generate a new solution, in this article, we proposed a novel scout bee heuristic that considers the useful information that is collected by the global and local best solutions. The main steps of the proposed scout bee are described in Algorithm 6. The main component of the scout bee is to generate the probability model, with the computational time complexity as follows: 1) in the first step, the time complexity is $O(k \cdot s)$, where s is the total number of factories; 2) the time complexities for the following five steps are all the same, namely, $O(n^2)$, where n is the total number of jobs; and 3) the time complexity for processing the remaining unscheduled jobs is $O(smn^3)$, where m is the total number of machines. If all solutions in the current population are assigned the same number of jobs for each factory, then there will not be any unscheduled jobs. Therefore, in the worst case, the time complexity of the scout bee is $O(smn^3)$.

Algorithm 7: General Framework of I_{ABC}.

Input: system parameters
Output: the best solution

for $i = 1$ *to* P_{s} **do**
  Initialize the i^{th} solution randomly.
  Evaluate it and insert it into the initial population.
end
Record the best solution that has been found so far.
While the stopping criterion is not satisfied **do**
  Employed bee phase
  for $i = 1$ to P_{size} **do**
    Perform the exploitation task (see Section V-D).
    Update the local and global best solutions.
  end
  Onlooker bee phase
  for $i = 1$ to P_{size} **do**
    Randomly select two solutions and select the best as the onlooker.
    Perform the exploitation task (see Section V-E)
    Update the local and global best solutions.
  end
  Scout bee phase
  If a solution in the population has not been improved during the limit trials, abandon it and assign a scout bee (see Section V-F) around it.
end
Output the best solution

G. Framework of the Proposed Algorithm

The framework of I_{ABC} is presented in Algorithm 7.

VI. Experimental Results

This section discusses the computational experiments that are conducted to evaluate the performance of the proposed algorithm. Our algorithm was implemented in C++ on an Intel Core i7 3.4-GHz PC with 16 GB of memory. To evaluate the effectiveness and efficiency of the proposed algorithm, after 30 independent runs, the resulting best solutions were collected for performance comparisons.

The performance measure is the relative percentage increase (RPI), which is calculated as follows:

$$\mathrm{RPI}(C) = \frac{C_c - C_b}{C_b} \times 100 \tag{11}$$

where C_b is the best solution that has been found by all of the compared algorithms and C_c is the best solution that has been generated by a specified algorithm.

A. Experimental Instances

To evaluate the performance of the proposed algorithm, we select 360 relatively large-scale instances from a website (http://soa.iti.es/rruiz), where the number of jobs is $n = \{20, 50\}$, the number of machines is $m = \{5, 10, 20\}$, and the number of factories is $f = \{2, 3, 4, 5, 6, 7\}$. To render the instances suitable for realistic systems, we extend these 360 instances by considering the distance index, which is a real number that is generated from the uniform distribution [0.5, 1.5]. After considering the distance index, the final normal processing time of each job is computed as follows:

$$p'_{i,j,f} = p_{i,j,f} \times DI_{i,f} \tag{12}$$

where $p'_{i,j,f}$ is the final processing time that is required for considering the distance cost index ($DI_{i,f}$) and $p_{i,j,f}$ is the processing time that is provided on the website (http://soa.iti.es/rruiz). The notation $\lfloor . \rfloor$ denotes the transformation of a real number into an integer by maintaining the integral part. For example, $\lfloor 4.5 \rfloor$ corresponds to a processing time of 4. Then, in the last stage, the parallel machine number is randomly generated in [2, 3], and the maximum number of jobs in each batch is also randomly generated in [2, 3]. The normal processing time for each job in the last stage is randomly generated in [50, 100]. The deterioration rate for the processing time of each job in each stage (β) is set to 0.1.

In addition, we generate ten types of instances that are based on the realistic production process of a typical steel-making casting horizon in China. The ten types of generated instances differ in terms of scale, with 20, 30, 50, 80, 100, 120, 150, 200, 300, and 500 charges or jobs. There are two factories, which differ in terms of their processing capabilities, in the first stage. In each factory, the assigned charges undergo three phases: 1) lag skimming; 2) desulfurization; and 3) reladling process (see Fig. 1). The processing time of each charge differs according to the assigned factory; therefore, the instances are more similar to the real production scenario. The normal processing times for each charge in the first stage are randomly generated in [30, 50] min according to the realistic data. In the second stage, there are two parallel continuous casting machines, which can process charges in a batch. The batch has a maximum number of jobs, which is randomly generated in $[n/10, n/10+5]$. The normal processing time for each job in the last stage is randomly generated in [50, 100], according to the realistic data in the steel-making system. The deterioration rate for the processing time of each job in each stage is set to 0.1.

B. Experimental Parameters

The stopping criterion for each instance depends on the instance scale, which is 50 *nms* ms, where n is the total number of jobs, m is the total number of machines, and s is the total number of factories. The two key parameters are the population size (P_s) and the limited iterations without any improvement for the scout bee (L_s). According to our preliminary experiments, the values of the two parameters are set as follows: $P_s \in \{20, 100, 150, 200\}$ and $L_s \in \{10, 20, 30, 50, 100, 200\}$.

TABLE I
ANOVA TABLE ON RPI FOR THE PARAMETERS

Source	ss	DF	MS	F	Prob>F
L_S	0.03565	5	0.00713	3.72	0.0036
Ps	0.18679	3	0.06226	32.52	0
Ps*Ls	0.0193	15	0.00129	0.67	0.8074
Error	0.22975	120	0.00191		
total	0.47149	143			

Similar to [52]–[59], we have utilized the design-of-experiments (DOEs) method for the parameter tuning of the proposed algorithm. However, rather than using the DOE Taguchi method, we adopted a full factorial design in which the two parameters were used as factors. The full factorial design yields a total of 24 distinct combinations of the two parameters. Due to the randomness of the improvement procedure, we utilized the parameter calibration instances that are provided on the website (http://soa.iti.es/rruiz) to test the factor combinations and we performed 5 runs per instance. The experiments were carried out on a cluster of computers with Intel Core *i*7 3.4-GHz PCs and 16 GB of memory. The PRI values were computed and used as the response variable of the experiments.

The results were analyzed via a multiway analysis of variance (ANOVA), where the two parameters are used as the controllable factors. To evaluate the ANOVA model hypothesis, namely, normality, homoscedasticity, and independence, the standardized residuals were checked and can be accepted. A major advantage of the ANOVA technique is that it calculates the magnitude of the F-ratio, where a large F-ratio indicates that the analyzed factor has a substantial effect on the response variable.

Table I lists the results of the analysis. The results demonstrate that the two parameters all have significant factors (p-value<0.05). The parameter P_s, with an F-ratio value 32.52, has a significant effect on the performance of the proposed algorithm. The parameter L_s, with an F-ratio value 3.72, has a minor effect compared with P_s. Meanwhile, the factor interaction between the two parameters is nonsignificant, with a p-value of greater than 0.05.

Plots of the 95% confidence intervals for the PRI value under selections of parameter P_s and L_s are provided in Fig. 5. According to Fig. 5(a), $P_s = 20$ yields a much better RPI value than other values. According to Fig. 5(b), $L_s = 0$ yields a much worse RPI value compared to nonzero values. This result demonstrates the importance of incorporating the scout bee heuristic into the proposed algorithm. Fig. 5(b) also shows that $L_s = 10$ yields higher performance. Based on the results of this test, parameters P_s and L_s were set to 20 and 10, respectively.

C. Efficiency of the Scout Bee Heuristic

To investigate the performance of the scout bee heuristic, which is discussed in Section V-F, we realize three types of scout bee heuristics: 1) the proposed scout method (S-I); 2) the canonical scout bee heuristic, which replaces a solution that is without improvement after a specified number of iterations with a random solution (S-II); and 3) the scout bee approach, which replaces the solution with several local search operators around the best solution that has been found so far (S-III).

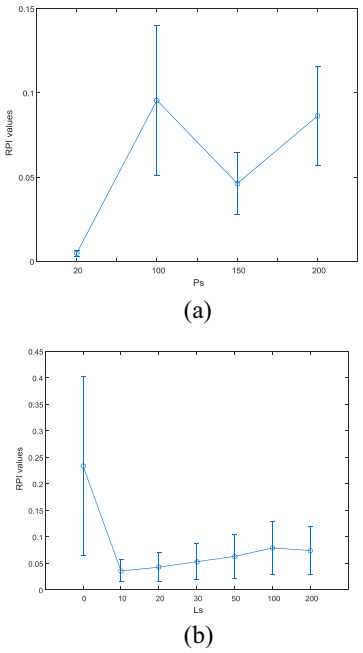


Fig. 5. The 95% confidence interval for different selections of (a) P_S and (b) L_S.

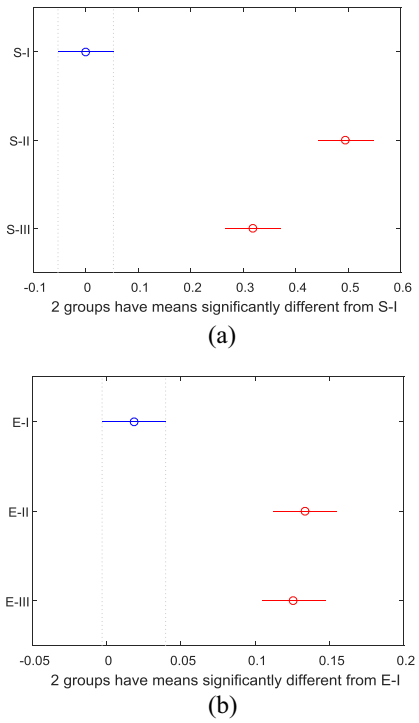


Fig. 6. Multicomparison results for the (a) scout and (b) employed bee heuristics.

For more than 30 independent runs, the average fitness value for each instance is computed. The fitness values of the best solutions that are found by the compared three types of scout bee heuristics are set as C_b in (11) to compute the RPI values. To determine whether the results that were obtained by the three compared heuristics differ significantly, we apply the Friedman test [52] and the Holm multiple comparison test [53] as *post hoc* procedures for pairwise comparison [56]. Fig. 6(a) presents the pairwise comparison results that were obtained by applying the Holm multiple comparison tests. It is concluded from Fig. 6(a) that the proposed scout bee outperforms the compared heuristics significantly.

The main reason why the proposed heuristic outperforms the other heuristics is that learning from the global and local best solutions enhances the searching performance of the scout bee and, thus, improves both the local and global search performances.

D. Efficiency of the Local Search Heuristic

To investigate the effectiveness of the local search heuristic that was discussed in Section V-C, we perform the following detailed comparisons.

We implement three types of local search heuristics: 1) the proposed heuristic with the five types of operators that are discussed in Section V-C, namely, E-I; 2) the local search with swap operators, namely, E-II; and 3) the local search with insertion operations, namely, E-III. The other components of the three compared algorithms are the same. Fig. 6(b) presents the pairwise comparison results that were obtained by applying the Holm multiple comparison tests. It is concluded that the proposed four types of local search operators outperform the others in solving the considered problem.

The main advantages of the proposed local search operators are as follows: 1) for the DFSP in the first stage, we utilize five types of local search operators that consider both the local and global search abilities and 2) for the parallel batching problem in the second stage, we utilize the swap operator to perturb the machine assignment for each batch to improve the local search performance of the proposed algorithm.

E. Efficiency of the Right-Shifting Heuristic

To investigate the performance of the right-shifting heuristic, which was discussed in Section IV-C, we realize two types of heuristics: 1) the proposed right-shifting method with consideration of the right-shift operator (R-I) and 2) the method with consideration only of the batch starting time (R-II). Fig. 7 presents the mean and 95% Fisher's least-significant difference (LSD) interval for the right-shifting heuristics. It is concluded that the proposed right-shifting heuristic significantly outperforms the heuristic that does not consider the right-shift operator.

The main advantage of the proposed right-shifting heuristic is that by considering the completion time of each job in the first stage, the proposed right-shifting heuristic makes each job in the same batch as compact as possible; therefore, the deterioration time of each batch and, thus, the completion time can be decreased.

F. Comparisons With Various Versions of ABC

This section investigates various versions of the ABC algorithm from the literature, which include the canonical ABC algorithm [8] (hereafter, called C_{ABC}), which was developed

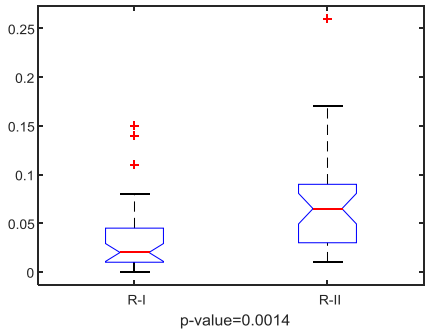


Fig. 7. Mean and 95% LSD interval for the right-shifting heuristic.

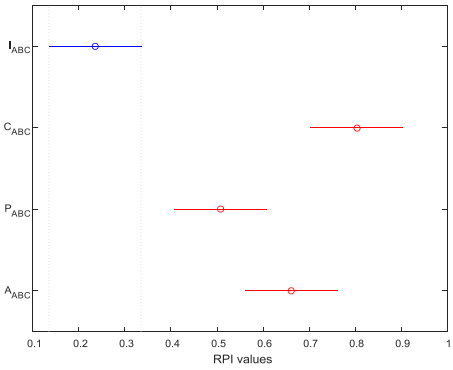


Fig. 8. Multicompare results for I_{ABC}, C_{ABC}, P_{ABC}, and A_{ABC}.

to solve the problem that is considered in this article; the discrete ABC algorithm proposed by Pan *et al.* [42] (hereafter, called P_{ABC}), which has been demonstrated to be an efficient version of ABC; and the ABC+TS algorithm, which was proposed in [32] (hereafter, called A_{ABC}) and is an efficient algorithm for solving parallel batching with deteriorating jobs.

These compared algorithms have not been applied to solve the problem that is considered in this article; therefore, we implement these algorithms according to their references, except that the following components are embedded: the encoding approaches in Section V-A, the batch assignment heuristic that was proposed in Section IV, and the four types of local search operators in Section V-C. For fair comparison of the algorithms, we record the best value for each instance after running for 200 *nms* ms.

Table II reports the detailed comparison results of four types of ABC algorithms. According to Table II: 1) in comparison with the other three ABC algorithms, the proposed I_{ABC} algorithm obtained all optimal RPI values for the 36 instances; 2) on average, the proposed I_{ABC} algorithm obtained an RPI value of 0.13, which is substantially better than the value that was obtained by the second-best performer, namely, P_{ABC}, of 0.51; and 3) the comparison results demonstrate the efficiency and robustness of the proposed I_{ABC} algorithm. Fig. 8 shows that the proposed algorithm performs significantly better than the other three ABC algorithms.

To evaluate the convergence performance of the proposed algorithm, we plot the convergence curves for the compared algorithms. Fig. 9 reports the convergence curves for solving one of the instances of case 12, in which there are 3 factories, 50 jobs, and 20 machines. According to Fig. 9, the proposed algorithm converges faster than the compared algorithms, namely, C_{ABC}, P_{ABC}, and A_{ABC}.

TABLE II
COMPARISONS WITH THE OTHER THREE TYPES OF ABC ALGORITHMS

Inst	makespan				Time(s)			
	I_{ABC}	C_{ABC}	P_{ABC}	A_{ABC}	I_{ABC}	C_{ABC}	P_{ABC}	A_{ABC}
Case1	**0.13**	0.42	0.29	0.33	1.41	4.12	4.01	5.58
Case2	**0.11**	0.37	0.27	0.30	2.10	6.25	8.03	10.36
Case3	**0.12**	0.31	0.26	0.27	3.33	14.32	10.90	22.09
Case4	**0.09**	0.37	0.17	0.29	8.80	8.68	15.54	15.33
Case5	**0.07**	0.28	0.09	0.17	12.10	15.59	30.57	31.08
Case6	0.44	0.23	**0.07**	0.14	26.75	34.22	59.14	58.42
Case7	**0.08**	0.78	0.58	0.70	4.37	5.66	5.39	7.60
Case8	**0.37**	0.87	0.62	0.70	4.07	11.20	9.21	17.32
Case9	**0.38**	0.90	0.69	0.74	4.58	19.86	9.33	34.07
Case10	**0.17**	0.56	0.27	0.46	15.28	11.19	22.36	22.44
Case11	**0.15**	0.48	0.18	0.36	19.35	15.92	41.91	47.40
Case12	0.20	0.45	**0.15**	0.30	40.51	47.88	78.53	98.48
Case13	**0.01**	0.62	0.46	0.57	4.03	6.94	9.34	11.96
Case14	**0.17**	1.18	0.87	1.00	4.63	19.62	15.79	22.11
Case15	**0.74**	1.10	0.84	0.91	8.78	27.55	20.27	40.97
Case16	**0.13**	0.61	0.36	0.53	22.21	20.38	27.79	29.52
Case17	**0.29**	0.79	0.41	0.64	34.51	29.56	50.87	62.72
Case18	**0.24**	0.72	0.31	0.56	46.83	77.05	104.24	128.37
Case19	**0.00**	0.56	0.42	0.51	4.51	7.30	7.45	12.77
Case20	**0.14**	1.17	0.80	1.01	5.79	10.23	17.28	28.87
Case21	1.04	1.01	**0.64**	0.76	6.70	32.52	12.66	60.50
Case22	**0.07**	0.60	0.37	0.50	32.64	18.71	37.10	34.85
Case23	**0.23**	0.91	0.52	0.74	55.16	40.79	66.73	79.78
Case24	**0.35**	0.94	0.41	0.70	69.36	61.07	128.82	155.63
Case25	**0.00**	0.53	0.40	0.49	6.35	10.90	11.70	14.63
Case26	**0.07**	1.12	0.81	0.95	5.27	20.30	21.66	32.80
Case27	**0.70**	1.96	1.48	1.60	4.68	47.48	16.36	66.80
Case28	**0.02**	0.60	0.41	0.52	35.22	28.63	34.22	41.90
Case29	0.12	0.94	**0.54**	0.79	54.08	48.17	70.15	88.07
Case30	**0.36**	1.10	0.50	0.87	89.39	124.91	135.70	186.93
Case31	**0.00**	0.51	0.40	0.47	6.23	8.86	15.23	12.26
Case32	**0.13**	1.17	0.82	1.00	4.60	20.50	22.17	38.22
Case33	**0.76**	2.02	1.38	1.62	5.35	36.00	31.34	83.19
Case34	**0.02**	0.59	0.37	0.51	31.95	30.99	41.20	49.65
Case35	**0.24**	1.09	0.64	0.92	64.61	39.64	82.97	104.07
Case36	**0.32**	1.07	0.47	0.85	94.84	103.68	159.27	217.55
mean	**0.24**	0.80	0.51	0.66	23.34	29.63	39.87	54.84

* values in bold mean the best ones

G. Comparisons With the Presented Efficient Algorithms

For fair comparison of the proposed algorithm and the other efficient algorithms, we select two sets: 1) the first set includes two algorithms: a) the hybrid simulated annealing (SA) algorithm (Lee *et al.*, 2014 [28]) and the SS algorithm (Naderi and Ruiz, 2014 [11]) and 2) two efficient nonpopulation-based methods that have been proposed

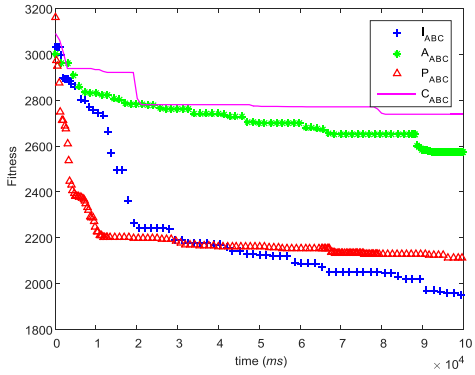


Fig. 9. Convergence curve for case 12.

in recent years, namely, the IG algorithm (Ruiz *et al.*, 2019 [59]) and the variable neighborhood search (VNS) algorithm (Menéndez *et al.* 2017 [60]). The main reason for selecting the two comparison algorithms is that the SA is an efficient algorithm to solve the FSPs with deteriorating jobs and the SS is an efficient algorithm for the DFSP. The other two compared algorithms, namely, IG and VNS, are nonpopulation-based algorithms, which have also been applied to solve the flow-shop scheduling problems. Because there is no efficient algorithm to solve the DFSP with parallel batching and deteriorating jobs, we select the four efficient algorithms for deteriorating jobs or for DFSPs. Detailed comparisons of the RPI values are presented in Table III. The parameters of all compared algorithms are set as in the corresponding literature.

According to Table III: 1) the proposed I_{ABC} algorithm obtained 34 optimal RPI values for the 36 specified instances, whereas the VNS algorithm obtained two optimal values. In addition, according to the average RPI values in the last row of the table, the proposed algorithm outperformed the VNS algorithm; 2) the proposed algorithm outperforms the SS algorithm. The average RPI value that was obtained by the I_{ABC} algorithm is approximately 30% of that obtained by the SS algorithm, while the proposed algorithm obtained better RPI values for all 36 instances compared with the SS algorithm; and 3) for solving the specified instances, the proposed I_{ABC} algorithm obtained all of the optimal values compared with the two efficient algorithms, namely, IG and SA, which are all designed for the DFSPs. The average performances in the last row demonstrate that the proposed algorithm outperforms the two compared algorithms in solving the considered problem. Fig. 10 presents the pairwise comparison results that were obtained by applying the Holm multiple comparison tests. It is concluded from the figure that the proposed algorithm significantly outperforms the compared algorithms.

To evaluate the efficiency and effectiveness of the proposed algorithm to solve the realistic optimization problems, we also performed detailed experimental comparisons of the proposed algorithm and the other two efficient algorithms for similar problems, namely, IG and VNS. The instances are described in Section VI-A. Inside the two compared algorithms, the components that are provided in the proposed algorithm are

TABLE III
COMPARISONS WITH THE OTHER EFFICIENT ALGORITHMS (FITNESS)

Inst	scale	fitness				
		VNS	SS	SA	IG	I_{ABC}
Case1	2-20-5	0.58	0.43	0.36	0.37	**0.13**
Case2	2-20-10	0.51	0.35	0.34	0.34	**0.11**
Case3	2-20-20	0.28	0.32	0.31	0.31	**0.12**
Case4	2-50-5	0.50	0.37	0.17	0.24	**0.09**
Case5	2-50-10	0.13	0.25	0.09	0.13	**0.07**
Case6	2-50-20	**0.00**	1.32	0.44	0.47	0.44
Case7	3-20-5	0.93	0.63	0.58	0.60	**0.08**
Case8	3-20-10	1.09	0.78	0.75	0.74	**0.37**
Case9	3-20-20	1.21	0.91	0.88	0.88	**0.38**
Case10	3-50-5	0.74	0.53	0.30	0.39	**0.17**
Case11	3-50-10	0.64	0.45	0.21	0.26	**0.15**
Case12	3-50-20	**0.19**	0.53	0.25	0.27	0.20
Case13	4-20-5	0.73	0.47	0.47	0.46	**0.01**
Case14	4-20-10	1.32	0.94	0.91	0.90	**0.17**
Case15	4-20-20	1.76	1.41	1.39	1.38	**0.74**
Case16	4-50-5	0.80	0.57	0.36	0.46	**0.13**
Case17	4-50-10	1.04	0.76	0.49	0.55	**0.29**
Case18	4-50-20	0.51	0.62	0.35	0.38	**0.24**
Case19	5-20-5	0.65	0.43	0.44	0.43	**0.00**
Case20	5-20-10	1.40	0.93	0.90	0.88	**0.14**
Case21	5-20-20	2.89	2.21	2.11	2.13	**1.04**
Case22	5-50-5	0.71	0.49	0.36	0.42	**0.07**
Case23	5-50-10	1.10	0.75	0.54	0.62	**0.23**
Case24	5-50-20	0.87	0.81	0.48	0.51	**0.35**
Case25	6-20-5	0.60	0.40	0.41	0.41	**0.00**
Case26	6-20-10	1.28	0.89	0.88	0.87	**0.07**
Case27	6-20-20	2.77	2.17	2.14	2.19	**0.70**
Case28	6-50-5	0.72	0.48	0.36	0.39	**0.02**
Case29	6-50-10	1.08	0.69	0.52	0.54	**0.12**
Case30	6-50-20	1.21	0.99	0.61	0.66	**0.36**
Case31	7-20-5	0.54	0.39	0.41	0.40	**0.00**
Case32	7-20-10	1.31	0.83	0.82	0.85	**0.13**
Case33	7-20-20	2.78	1.81	1.81	2.01	**0.76**
Case34	7-50-5	0.68	0.45	0.35	0.35	**0.02**
Case35	7-50-10	1.25	0.81	0.58	0.64	**0.24**
Case36	7-50-20	1.20	0.85	0.52	0.59	**0.32**
Mean		1.00	0.78	0.64	0.67	**0.24**

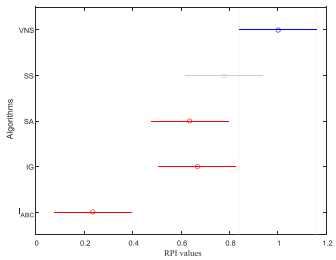


Fig. 10. Multicompare results for I_{ABC}, IG, SA, SS, and VNS.

embedded, except the batch assignment and scheduling heuristics, the scout bee heuristic, and the local search heuristics. Table IV lists the detailed comparison results. In the table, the first column corresponds to the instance name. The second column corresponds to the problem scale, where the total number of jobs is listed. The following three columns correspond to the best RPI values for each instance that were obtained by the three compared algorithms, namely, I_{ABC}, IG, and VNS. The last three columns list the average RPI values that were realized by the three algorithms.

It is concluded from the table that the proposed algorithm performs competitively and outperforms the other two efficient algorithms in terms of both the best and average performances.

TABLE IV
COMPARISONS RESULTS FOR THE REALISTIC INSTANCES

Inst	scale	best	best RPI			average RPI		
			I_{ABC}	IG	VNS	I_{ABC}	IG	VNS
Inst1	20	457.79	**0.00**	8.63	17.14	**6.66**	13.28	24.89
Inst2	30	674.86	**0.00**	3.60	12.52	**2.47**	8.77	21.72
Inst3	50	1011.72	**0.00**	7.50	19.66	**5.66**	12.37	26.45
Inst4	80	1782.67	**0.00**	6.33	18.88	**2.24**	10.01	22.58
Inst5	100	2202.61	**0.00**	9.76	20.93	**3.28**	17.45	26.86
Inst6	120	2835.63	**0.00**	10.64	18.56	**1.83**	19.03	23.07
Inst7	150	3614.60	**0.00**	14.89	22.52	**1.33**	21.98	28.41
Inst8	200	5759.23	**0.00**	18.73	22.69	**1.45**	23.23	27.86
Inst9	300	10814.20	**0.00**	23.53	26.33	**1.40**	27.71	30.96
Inst10	500	32767.90	**0.00**	32.69	33.40	**1.87**	37.15	38.01
Mean			**0.00**	13.63	21.26	**2.82**	19.10	27.08

The results also demonstrate the efficiency and effectiveness of the proposed heuristics in the proposed algorithm.

H. Experimental Analysis

According to the comparison results, the proposed algorithm substantially outperforms the efficient algorithms for related or similar scheduling problems. In addition, there are unpublished algorithms for solving the DFS scheduling problem with parallel batching and deteriorating job constraints; and the other compared algorithms may perform competitively in solving the problems that are considered in their references. However, the comparison results demonstrate the efficiency and effectiveness of the proposed algorithm.

The main advantages of the proposed algorithm are as follows: 1) two types of problem-specific structures are developed, namely, the batch assignment and right-shifting heuristics, which can group jobs into suitable batches and schedule them; 2) five types of local search operators are designed for the DFS and the parallel batching stages, which can improve the exploitation performance of the proposed algorithm; and 3) a novel scout bee heuristic that considers the useful information that is collected by the global and local best solutions is investigated, which can substantially enhance the exploration and searching performances.

VII. CONCLUSION

In this article, a type of realistic scheduling problem with parallel batching and deteriorating jobs in the DFSP environment is considered and solved via an improved ABC algorithm. The problem consists of two stages: in the first stage, a classic DFSP with deteriorating jobs is considered, whereas in the second stage, parallel batching with deteriorating jobs is investigated. The objective of the problem is to minimize the makespan of the system.

The main contributions of this article include the following: 1) considering the problem structures, two types of problem-specific heuristics, including the batch assignment and right-shifting methods, are developed, where a right-shift operator is embedded to substantially improve the makespan; 2) encoding and decoding approaches are developed that consider both the DFSP constraints and the parallel batching features that make the solutions feasible; 3) five types of local search operators are designed for the DFS and the parallel batching stages that enhance the local search performance of the proposed algorithm; and 4) a novel scout bee heuristic that considers the useful information that is collected by the global and local best solutions is investigated, which can substantially enhance searching performance. Detailed comparisons with well-known instances demonstrate the efficiency of the proposed heuristics and the effectiveness of the proposed algorithm in solving the specified complex optimization problems.

Future work will focus on the following aspects: 1) applying the proposed algorithm to other types of distributed scheduling problems, such as distributed hybrid flow-shop scheduling and rescheduling problems under parallel batching and deteriorating job constraints; 2) considering other objectives in the problem, such as minimization of the energy consumption and the workload balance; and 3) imposing other constraints on the problem, such as resource constraints and machine availability constraints.

REFERENCES

[1] Q.-K. Pan, L. Gao, and L. Wang, "A multi-objective hot-rolling scheduling problem in the compact strip production," *Appl. Math. Model.*, vol. 73, pp. 327–334, Sep. 2019.

[2] 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 Trans. Cybern.*, vol. 46, no. 6, pp. 1311–1324, Jun. 2016.

[3] J.-Q. Li, H.-Y. Sang, Y.-Y. Han, C.-G. Wang, and K.-Z. Gao, "Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions," *J. Clean. Prod.*, vol. 181, pp. 584–598, Apr. 2018.

[4] X. Li, T. Jiang, and R. Ruiz, "Heuristics for periodical batch job scheduling in a MapReduce computing framework," *Inf. Sci.*, vol. 326, pp. 119–133, Jan. 2016.

[5] B. Naderi and R. Ruiz, "The distributed permutation flowshop scheduling problem," *Comput. Oper. Res.*, vol. 37, no. 4, pp. 754–768, 2010.

[6] S. Hatami, R. Ruiz, and C. Andrés-Romano, "The distributed assembly permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 51, no. 17, pp. 5292–5308, 2013.

[7] J. Pei, P. M. Pardalos, X. Liu, W. Fan, and S. Yang, "Serial batching scheduling of deteriorating jobs in a two-stage supply chain to minimize the makespan," *Eur. J. Oper. Res.*, vol. 244, no. 1, pp. 13–25, 2015.

[8] D. Karaboga and C. Ozturk, "A novel clustering approach: Artificial bee colony (ABC) algorithm," *Appl. Soft. Comput.*, vol. 11, no. 1, pp. 652–657, 2011.

[9] S.-W. Lin, K.-C. Ying, and C.-Y. Huang, "Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm," *Int. J. Prod. Res.*, vol. 16, no. 51, pp. 5029–5038, 2013.

[10] V. Fernandez-Viagas and J. M. Framinan, "A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 53, no. 4, pp. 1111–1123, 2015.

[11] B. Naderi and R. Ruiz, "A scatter search algorithm for the distributed permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 239, no. 2, pp. 323–334, 2014.

[12] Y. Xu, L. Wang, S. Wang, and M. Liu, "An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem," *Eng. Optim.*, vol. 46, no. 9, pp. 1269–1283, 2014.

[13] H. Bargaoui, O. B. Driss, and K. Ghédira, "A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion," *Comput. Ind. Eng.*, vol. 111, pp. 239–250, Sep. 2017.

[14] J. Gao, R. Chen, and W. Deng, "An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 51, no. 3, pp. 641–651, 2013.

[15] S.-Y. Wang, L. Wang, M. Liu, and Y. Xu, "An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem," *Int. J. Prod. Econ.*, vol. 145, no. 1, pp. 387–396, 2013.

[16] K. Wang, Y. Huang, and H. Qin, "A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown," *J. Oper. Res. Soc.*, vol. 67, no. 1, pp. 68–82, 2016.

[17] A. P. Rifai, H.-T. Nguyen, and S. Z. M. Dawal, "Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling," *Appl. Soft. Comput.*, vol. 40, pp. 42–57, Mar. 2016.

[18] S.-W. Lin and C.-K. Ying, "Minimizing makespan for solving the distributed no-wait flowshop scheduling problem," *Comput. Ind. Eng.*, vol. 99, pp. 202–209, Sep. 2016.

[19] K.-C. Ying, S.-W. Lin, C.-Y. Cheng, and C.-D. He, "Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems," *Comput. Ind. Eng.*, vol. 110, pp. 413–423, Aug. 2017.

[20] J. Deng and L. Wang, "A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem," *Swarm. Evol. Comput*, vol. 32, pp. 121–131, Feb. 2017.

[21] J. Lin, Z.-J. Wang, and X. Li, "A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem," *Swarm. Evol. Comput.*, vol. 36, pp. 124–135, Oct. 2017.

[22] S. Hatami, R. Ruiz, and C. Andrés-Romano, "Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times," *Int. J. Prod. Econ.*, vol. 169, pp. 76–88, Nov. 2015.

[23] G. Zhang, K. Xing, and F. Cao, "Scheduling distributed flowshops with flexible assembly and set-up time to minimise makespan," *Int. J. Prod. Res.*, vol. 56, no. 9, pp. 3226–3244, 2018.

[24] E. M. Gonzalez-Neira, D. Ferone, S. Hatami, and A. A. Juan, "A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times," *Simul. Model. Pract. Theory*, vol. 79, pp. 23–36, Dec. 2017.

[25] Y.-Y. Lu, "Research on no-idle permutation flowshop scheduling with time-dependent learning effect and deteriorating jobs," *Appl. Math. Model.*, vol. 40, no. 4, pp. 3447–3450, 2016.

[26] M. A. Bajestani and J. C. Beck, "A two-stage coupled algorithm for an integrated maintenance planning and flowshop scheduling problem with deteriorating machines," *J. Schedul.*, vol. 18, no. 5, pp. 471–486, 2015.

[27] M. Cheng, P. R. Tadikamalla, J. Shang, and S. Zhang, "Bicriteria hierarchical optimization of two-machine flow shop scheduling problem with time-dependent deteriorating jobs," *Eur. J. Oper. Res.*, vol. 234, no. 3, pp. 650–657, 2014.

[28] W.-C. Lee, W.-C. Yeh, and Y.-H. Chung, "Total tardiness minimization in permutation flowshop with deterioration consideration," *Appl. Math. Model.*, vol. 38, no. 13, pp. 3081–3092, 2014.

[29] H. Wang, M. Huang, and J. Wang, "An effective metaheuristic algorithm for flowshop scheduling with deteriorating jobs," *J. Intell. Manuf.*, vol. 30. no. 7, pp. 2733–2742, 2019.

[30] Y. Fu, J. Ding, H. Wang, and J. Wang, "Two-objective stochastic flowshop scheduling with deteriorating and learning effect in industry 4.0-based manufacturing system," *Appl. Soft. Comput.*, vol. 68, pp. 847–855, 2017.

[31] J. Pei, X. Liu, W. Fan, P. M. Pardalos, and S. Lu, "A hybrid BA-VNS algorithm for coordinated serial-batching scheduling with deteriorating jobs, financial budget, and resource constraint in multiple manufacturers," *Omega*, vol. 82, pp. 55–69, Jan. 2019.

[32] S. Lu, X. Liu, J. Pei, M. T. Thai, and P. M. Pardalos, "A hybrid ABC-TS algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity," *Appl. Soft. Comput.*, vol 66, pp. 168–182, May 2018.

[33] A. Ham, J. W. Fowler, and E. Cakici, "Constraint programming approach for scheduling jobs with release times, non-identical sizes, and incompatible families on parallel batching machines," *IEEE Trans. Semicond. Manuf.*, vol. 30, no. 4, pp. 500–507, Nov. 2017.

[34] Y. Gao, J. Yuan, C. T. Ng, and T. C. E. Cheng, "A further study on two-agent parallel-batch scheduling with release dates and deteriorating jobs to minimize the makespan," *Eur. J. Oper. Res.*, vol. 273, no. 1, pp. 74–91, 2019.

[35] J. E. C. Arroyo, J. Y.-T. Leung, and R. G. Tavares, "An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times," *Eng. Appl. Artif. Intell.*, vol. 77, pp. 239–254, Jan. 2019.

[36] S. Zhou, X. Li, N. Du, Y. Pang, and H. Chen, "A multi-objective differential evolution algorithm for parallel batch processing machine scheduling considering electricity consumption cost," *Comput. Oper. Res.*, vol. 96, pp. 55–68, Aug. 2018.

[37] L. Tang, X. Zhao, J. Liu, and J. Y.-T. Leung, "Competitive two-agent scheduling with deteriorating jobs on a single parallel-batching machine," *Eur. J. Oper. Res.*, vol. 263, no. 2, pp. 401–411, 2017.

[38] S. Li, "Parallel batch scheduling with inclusive processing set restrictions and non-identical capacities to minimize makespan," *Eur. J. Oper. Res.*, vol. 260, no. 1, pp. 12–20, 2017.

[39] R. Zhang, P.-C. Chang, S. Song, and C. Wu, "A multi-objective artificial bee colony algorithm for parallel batch-processing machine scheduling in fabric dyeing processes," *Knowl. Based Syst.*, vol. 116, pp. 114–129, Jan. 2017.

[40] W.-F. Gao, S.-Y. Liu, and L.-L. Huang, "A novel artificial bee colony algorithm based on modified search equation and orthogonal learning," *IEEE Trans. Cybern.*, vol. 43, no. 3, pp. 1011–1024, Jun. 2013.

[41] J.-Q. Li, S.-C. Bai, P.-Y. Duan, H.-Y. Sang, Y.-Y. Han, and Z.-X. Zheng, "An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system," *Int. J. Prod. Res.*, 2019. doi: 10.1080/00207543.2019.1571687.

[42] Q.-K. Pan, M. F. Tasgetiren, P. N. Suganthan, and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Inf. Sci.*, vol. 181, no. 12, pp. 2455–2468, 2011.

[43] J.-Q. Li *et al.*, "A hybrid artificial bee colony for optimizing a reverse logistics network system," *Soft. Comput.*, vol. 21, no. 20, pp. 6001–6018, 2017.

[44] H. Liu, B. Xu, D. Lu, and G. Zhang, "A path planning approach for crowd evacuation in buildings based on improved artificial bee colony algorithm," *Appl. Soft. Comput.*, vol. 68, pp. 360–376, Jul. 2018.

[45] D. Gong, Y. Han, and J. Sun, "A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems." *Knowl. Based Syst.*, vol. 148, pp. 115–130, May 2018.

[46] J. Zhou and X. Yao, "A hybrid artificial bee colony algorithm for optimal selection of QoS-based cloud manufacturing service composition," *Int. J. Adv. Manuf. Technol.*, vol. 88, nos. 9–12, pp. 3371–3387, 2017.

[47] J.-Q. Li, Q.-K. Pan, and M. F. Tasgetiren, "A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities," *Appl. Math. Model.*, vol. 38, no. 3, pp. 1111–1132, 2014.

[48] P.-Y. Duan, J.-Q. Li, Y. Wang, H.-Y. Sang, and B. X. Jia, "Solving chiller loading optimization problems using an improved teaching-learning-based optimization algorithm," *Optim. Control Appl. Meth.*, vol. 39, no. 1, pp. 65–77, 2018.

[49] Z.-X. Zheng, J.-Q. Li, and P.-Y. Duan, "Optimal chiller loading by improved artificial fish swarm algorithm for energy saving," *Math. Comput. Simul.*, vol. 155, pp. 227–243, Jan. 2019. doi: 10.1016/j.matcom.2018.04.013.

[50] Z.-X. Zheng and J.-Q. Li, "Optimal chiller loading by improved invasive weed optimization algorithm for reducing energy consumption," *Energy Build.*, vol. 161, pp. 80–88, Feb. 2018.

[51] K. Gao, F. Yang, M. Zhou, Q. Pan, and P. N. Suganthan, "Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm," *IEEE Trans. Cybern.*, vol. 49, no. 5, pp. 1944–1955, May 2019. doi: 10.1109/TCYB.2018.2817240.

[52] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.* vol. 1, no. 1, pp. 3–18, 2011.

[53] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian J. Stat.* vol. 6, no. 2, pp. 65–70, 1979.

[54] D. Montgomery, *Design and Analysis of Experiments*, 5th ed. Hoboken, NJ, USA: Wiley, 2001, p. 684.

[55] J. Brown *et al.*, "Experimental analysis of optimization algorithms: Tuning and beyond," *Int. J. Behav. Nutrition Phys. Activity*, vol. 6, no. 3, pp. 1–7, 2014.

[56] J.-Q. Li, Q.-K. Pan, and K. Mao, "A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 932–949, Apr. 2016.

[57] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 177, no. 3, pp. 2033–2049, 2007.

[58] I. Ribas, R. Companys, and X. Tort-Martorell, "An iterated greedy algorithm for the flowshop scheduling problem with blocking," *Omega*, vol. 39, no. 3, pp. 293–301, 2011.

[59] R. Ruiz, Q.-K. Pan, and B. Naderi, "Iterated Greedy methods for the distributed permutation flowshop scheduling problem," *Omega*, vol. 83, pp. 213–222, Mar. 2019.

[60] B. Menéndez, M. Bustillo, E. G. Pardo, and A. Duarte, "General variable neighborhood search for the order batching and sequencing problem," *Eur. J. Oper. Res.*, vol. 263, no. 1, pp. 82–93, 2017.

Jun-Qing Li (M'10) received the master's degree in computer science and technology from Shandong Economic University, Shandong, China, in 2004, and the Ph.D. degree from Northeastern University, Shenyang, China, in 2016.

Since 2004, he has been with the School of Computer, Liaocheng University, Liaocheng, China. Since 2017, he has been with the School of Information Science and Engineering, Shandong Normal University, Jinan, China, where he became a Professor in 2017. He has authored over 30 refereed papers. His current research interests include intelligent optimization and scheduling.

Mei-Xian Song received the B.S. degree from the Institute of Information Science and Engineering, Qufu Normal University, Rizhao, China. She is currently pursuing the M.S. degree with Liaocheng University, Liaocheng, China.

Her current research interest includes intelligent optimization and control.

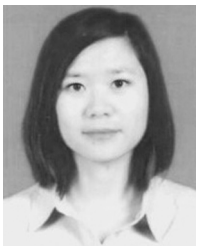
Ling Wang received the B.Sc. degree in automation and the Ph.D. degree in control theory and control engineering from Tsinghua University, Beijing, China, in 1995 and 1999, respectively.

Since 1999, he has been with the Department of Automation, Tsinghua University, where he became a Full Professor in 2008. His current research interests include intelligent optimization and production scheduling. He has authored five academic books and over 260 refereed papers.

Prof. Wang was a recipient of the National Natural Science Fund for Distinguished Young Scholars of China, the National Natural Science Award (Second Place) in 2014, the Science and Technology Award of Beijing City in 2008, and the Natural Science Award (First Place in 2003 and Second Place in 2007) nominated by the Ministry of Education of China. He is currently the Editor-in-Chief of the *International Journal of Automation and Control* and an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.

Pei-Yong Duan received the B.Sc. degree from the Shandong University of Technology (merged into Shandong University in 2000), Shandong, China, in 1996, and the Ph.D. degree from Shanghai Jiaotong University, Shanghai, China, in 1999.

From 1999 to 2014, he was with the School of Information and Electrical Engineering, Shandong Jianzhu University, where he was appointed as an Associate Professor in 1999 and a Full Professor in 2002. Since 2017, he has been with the School of Information Science and Engineering, Shandong Normal University. He has authored over 60 refereed papers. His current research interests include discrete optimization and scheduling.



Yu-Yan Han received the M.S. degree from the School of Computer Science, Liaocheng University, Liaocheng, China, in 2012, and the Ph.D. degree in control theory and control engineering from the China University of Mining and Technology, Xuzhou, China, in 2016.

Since 2016, she has been a Lecturer with the School of Computer Science, Liaocheng University. She has authored over 30 refereed papers. Her current research interests include evolutionary computation, multiobjective optimization, and flow-shop scheduling.

Hong-Yan Sang received the Ph.D. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2013.

Since 2013, she has been with the School of Computer, Liaocheng University, Liaocheng, China, where she was appointed Associate Professor in 2016. She has authored over 40 refereed papers. Her current research interests include discrete optimization and scheduling.

Quan-Ke Pan received the B.Sc. and Ph.D. degrees from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1993 and 2003, respectively.

From 2003 to 2011, he was with the School of Computer Science Department, Liaocheng University, Liaocheng, China, where he became a Full Professor in 2006. From 2011 to 2014, he was with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang, China. From 2014 to 2015, he was with the State Key Laboratory of Digital Manufacturing and Equipment Technology, Huazhong University of Science and Technology, Wuhan, China. He has been with the School of Mechatronic Engineering and Automation, Shanghai University, Shanghai, China, since 2015. He has authored one academic book and over 200 refereed papers. His current research interests include intelligent optimization and scheduling algorithms.

Prof. Pan acts as an Editorial Board Member for several journals, including *Operations Research Perspective* and *Swarm and Evolutionary Computation*.