An Improved Artificial Bee Colony Algorithm for Solving Hybrid Flexible Flowshop With Dynamic Operation Skipping

Jun-qing Li, *Member, IEEE*, Quan-ke Pan, and Pei-yong Duan

***Abstract*—In this paper, we propose an improved discrete artificial bee colony (DABC) algorithm to solve the hybrid flexible flowshop scheduling problem with dynamic operation skipping features in molten iron systems. First, each solution is represented by a two-vector-based solution representation, and a dynamic encoding mechanism is developed. Second, a flexible decoding strategy is designed. Next, a right-shift strategy considering the problem characteristics is developed, which can clearly improve the solution quality. In addition, several skipping and scheduling neighborhood structures are presented to balance the exploration and exploitation ability. Finally, an enhanced local search is embedded in the proposed algorithm to further improve the exploitation ability. The proposed algorithm is tested on sets of the instances that are generated based on the realistic production. Through comprehensive computational comparisons and statistical analysis, the highly effective performance of the proposed DABC algorithm is favorably compared against several presented algorithms, both in solution quality and efficiency.**

***Index Terms*—Artificial bee colony (ABC) algorithm, dynamic operation skipping, heuristic, hybrid flexible flowshop (HFF).**

I. Introduction

IN MODERN iron and steel production systems, the scheduling of molten iron plays an important role and can clearly increase the production efficiency and profit. The classical process of molten iron scheduling can be divided into five stages: 1) blast furnace; 2) preprocessing; 3) dephosphorization or desulphurization; 4) post-processing; and 5) iron pouring stages [1], [2]. In the first stage, the molten iron is first hybridized and processed in blast furnaces and then poured into pots carried on torpedo cars. Then, these torpedo cars are hauled by engines on a rail track to one of the two steel-making sites, where the molten iron will be processed through preprocessing, dephosphorization or desulphurization, post-processing, and iron pouring stages.

In Shanghai Baoshan Iron and Steel Complex (Baosteel), there are generally two types of molten iron, i.e., the common and specific molten irons. The specific molten iron can be further divided into three types: 1) demanganization molten iron; 2) silicon iron; and 3) pretreatment (desiliconization, dephosphorization, and desulphurization) molten iron. For the processing capacity limitation of the pre- and post-processing devices, the specific molten iron must be processed through the two stages, while the common molten iron should skip one or two of the two stages according to the capacity constraints or the due date window. In other words, the common molten iron will be dynamically selected to skip certain stages. The dynamic operation skipping is the main characteristic of the molten iron scheduling problem in Baosteel, and it can also be extended to other scheduling applications.

The hybrid flow shop (HFS) is one branch of the classical flow shop scheduling problem, which has been verified to be an NP-hard problem [3]–[20]. Many real-world manufacturing systems can be modeled as an HFS problem with certain constraints, such as the steel-making casting problems [14]. The hybrid flexible flowshop (HFF) is an extension of the classical HFS, which is more complex than the latter because of the addition need to permitting operation skipping [21]–[32]. Several meta-heuristics have been applied to solve the HFF problems, such as particle swarm optimization (PSO) [21] and genetic algorithm (GA) [31]. The molten iron scheduling problem can be viewed as an HFF problem with dynamic missing operations as follows: there are n torpedo pots (jobs) and k stages. At each stage, there are m parallel machines. All the jobs can either flow through each stage or skip certain stages. When any job arrives at any stage, it should select one, and only one, machine to be processed on. The goal is to find a nonpreemptive schedule that minimizes the average sojourn time, the penalties of tardiness and earliness, and the penalty of the skipping rate.

Manuscript received December 24, 2014; revised April 17, 2015; accepted June 4, 2015. Date of publication June 26, 2015; date of current version May 13, 2016. This work was supported in part by the National Natural Science Foundation of China under Grant 61104179, Grant 61374187, and Grant 61174187, in part by the Basic Scientific Research Foundation of Northeastern University under Grant N110208001 and N130508001, in part by the Key Laboratory Basic Research Foundation of Education Department of Liaoning Province under Grant LZ2014014, in part by the Program for New Century Excellent Talents in University (NCET-13-0106), in part by the Specialized Research Fund for the Doctoral Program of Higher Education (20130042110035), in part by the Science Foundation of Liaoning Province in China (2013020016), and in part by the IAPI Fundamental Research Funds (2013ZCX02). This paper was recommended by Associate Editor K.-C. Tan. *(Corresponding author: Quan-ke Pan.)*

J.-Q Li is with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China, and also with the School of Computer Science, Liaocheng University, Liaocheng 252059, China (e-mail: lijunqing.cn@gmail.com).

Q.-K. Pan is with the State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: panquanke_lcu@163.com).

P-Y. Duan is with the School of Computer Science, Liaocheng University, Liaocheng 252059, China (e-mail: duanpeiyong@lcu.edu.cn).

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

Digital Object Identifier 10.1109/TCYB.2015.2444383

Real-world manufacturing systems often involve uncertainty and dynamic changes, especially in make-to-order shop environments [33]. However, there is very limited available literature on HFF with dynamic changes, such as dynamic job-skip-stage, which we call HFF-D, let alone comparative studies. In real-world manufacturing systems, there are many applications that can be modeled as an HFF-D problem, such as the molten iron scheduling problems. Therefore, it is urgent to address this type of scheduling problem. The artificial bee colony (ABC) algorithm is proposed by Karaboga to optimize multivariable and multimodal continuous functions. Since 2005, ABC has been applied to solve many optimization problems. Experimental results compared with those of other algorithms verified the efficiency of the ABC algorithm [14], [34]–[43]. To the best of our knowledge, there is no literature aimed at solving the iron scheduling problem by using ABC. Therefore, in this paper, we propose an improved discrete ABC (DABC) to solve the molten iron scheduling problem in iron and steel industries.

In this paper, we develop a novel DABC algorithm to solve the HFF scheduling problem with dynamic stage-skipping for jobs. The main contributions of this paper are as follows: 1) HFF with dynamic job skipping, which is applicable in many industrial applications, is first proposed and modeled; 2) considering the problem characteristics, a dynamic encoding, and flexible decoding mechanism is developed; 3) a right-shift decoding method is proposed to minimize the earliness and tardiness penalties; 4) several skipping and scheduling neighborhood structures are presented to balance the exploration and exploitation abilities; and 5) an enhanced local search is embedded in the proposed algorithm to further improve the exploitation ability.

The rest of this paper is organized as follows. Section II briefly describes the molten iron scheduling problem. Next, the framework of the proposed algorithm is presented in Section III. Then, Section IV illustrates the experimental results and compares them to those of the presented algorithms from the literature to demonstrate the superiority of the proposed algorithm. Finally, the conclusion is given in Section V.

II. Problem Descriptions

A. Molten Iron Process

In this paper, we consider the molten iron scheduling problem in Baosteel, which is illustrated in Fig. 1.

Baosteel is equipped with four blast furnaces for iron-making, two steel-making sites for pretreatment, and two iron-pouring sites. Each blast furnace has four tapping holes, and three of them are used to pouring molten iron, while the other is used for maintenance. Each blast furnace performs roughly 12–14 pouring and provides three or four torpedo pots for each pouring. Each torpedo pot can transport approximately 270 tons of molten iron. Therefore, Baosteel produces approximately 150 torpedo pots every day. In the first steel-making site, there is one preprocessing device and two desulfurization devices. The second steel-making site has one preprocessing device, three desulfurization or dephosphorization devices, and two post-processing devices. The first

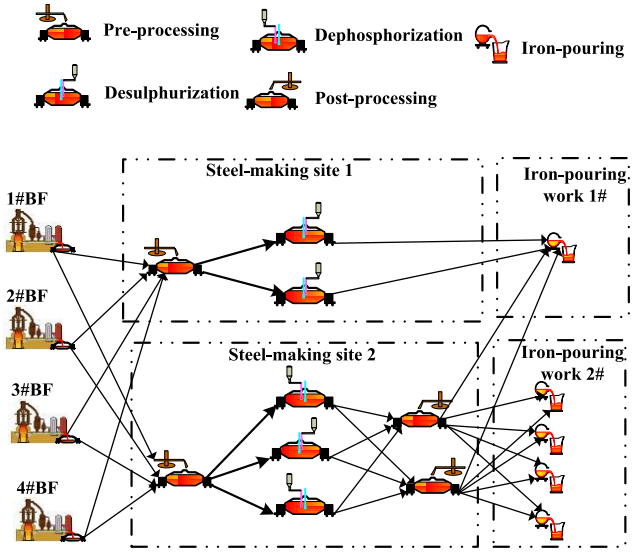

Fig. 1. Layout of the molten iron transportation track.

steel-making site has one iron pouring site, while the second steel-making site has four iron pouring sites to receive the molten iron from the torpedo pots. Generally, Baosteel dispatches molten iron as follows: molten iron drained from blast furnaces #1 and #2 is assigned to the first steel-making site, while molten iron from blast furnaces #3 and #4 is assigned to the second steel-making site. If blast furnaces #1 and #2 cannot supply enough molten iron to the first steel-making site, blast furnaces #3 and #4 also provide enough molten iron to the first steel-making site. Meanwhile, if blast furnaces #3 and #4 cannot supply enough molten iron to the second steel-making site, blast furnaces #1 and #2 also provide enough molten iron to the second steel-making site.

The characteristics of the molten iron process can be summarized as follows.

1) There are five stages in the molten iron scheduling problem, i.e., the blast furnace, preprocessing, dephosphorization or desulphurization, post-processing, and iron pouring stages.
2) In each stage, there are several identical parallel machines, which can be selected by any torpedo (job) flowed through the stage.
3) All torpedoes or jobs follow the same processing sequence, that is, from the first stage to the last stage; some torpedoes or jobs may skip some stages according to the processing capacity of the parallel machines in certain stages or the due date window.
4) The processing times of all torpedoes at all stages and the transportation times are non-negative, known, deterministic, and uninterrupted.
5) Each torpedo can be processed at most on one machine at a time, and each machine can process at most one torpedo at a time.
6) Setup times for torpedoes at each stage are relatively short, and therefore, can be ignored.
7) Transfer times between stages are considered.
8) Preemption is not permitted, that is, any torpedo should remain on the assigned machine until it completes its work.

B. Molten Iron Scheduling Problem

To make the problem closer to the industrial reality, we consider minimization of the weighted sum of four penalty values, i.e., the average sojourn time, the total earliness, the total tardiness, and the skipping rate of the critical machines. The sojourn time of a job is the duration between the completion time in the first stage and the starting time in the last stage. Minimizing sojourn times leads to minimization of waiting times, and thus, energy consumption can be reduced [14]. The skipping rate penalty is to compute the sum of the skipping rate of the devices in the pre- and post-processing stages.

Due to its complexity, we consider the molten iron scheduling problem as an HFF-D problem in a static environment. The notations used in this paper are summarized as follows.

i	Index of the jobs, $i = 1, 2, \ldots, n$.
k	Index of the machines, $k = 1, 2, \ldots, m$.
j	Index of the stages, $j = 1, 2, \ldots, 5$.
n	Total number of jobs.
m	Total number of machines.
$p_{i,j}$	Processing time of job i at stage j.
E_j	Set of parallel machines at stage j.
m_j	Number of parallel machines at stage j.
J	Set of n jobs, $J = \{J_1, J_2, \ldots, J_n\}$.
$T_{h,j}$	Transfer time from stage h to stage j.
$[d_s^i, d_e^i]$	Due time window of job i.
w_1	Penalty coefficient for the average sojourn time.
w_2	Penalty coefficient for the earliness.
w_3	Penalty coefficient for the tardiness.
w_4	Penalty coefficient for the skipping rate.
$b_{i,j}$	Starting time of job i at stage j.
$e_{i,j}$	Completion time of job i at stage j.
Ω_i	Set of stages to be accessed by job i.
$x_{i,j,k}$	If job i is processed on machine k at stage j, $x_{i,j,k} = 1$; otherwise $x_{i,j,k} = 0$.
$z_{i,h,j}$	If job i is to be processed at stage j immediately after its completion at stage h, $z_{i,h,j} = 1$; otherwise $z_{i,h,j} = 0$.
$y_{i,l,j}$	If job i is preceding job l to be processed at stage j, $y_{i,l,j} = 1$; if jobs l and i start at the same time at stage j, $y_{i,l,j} = 1/2$; otherwise $y_{i,l,j} = 0$.

By using the above notations, we give the formulation of this paper as follows:

$$\min f = w_1F_1 + w_2F_2 + w_3F_3 + w_4F_4$$

$$F_1 = \sum_{i=1}^{n} \left(b_{i,5} - e_{i,1}\right)/n \tag{1}$$

$$F_2 = \sum_{i=1}^{n} \max\left(0, d_s^i - b_{i,5}\right) \tag{2}$$

$$F_3 = \sum_{i=1}^{n} \max\left(0, b_{i,5} - d_e^i\right) \tag{3}$$

$$F_4 = \left(1 - \left(\sum_{j=2,4}\sum_{k=1}^{m_j}\sum_{i=1}^{n} x_{i,j,k}\right)\Big/ 2n\right) \times 100 \tag{4}$$

$$\sum_{\substack{j\in\Omega_i \\ k\in E_j}} x_{i,j,k} = 1, \quad \forall i \in J \tag{5}$$

$$b_{i,j} - \left(b_{i,h} + p_{i,h} + T_{h,j}\right) \cdot z_{i,h,j} \geq 0, \quad \forall i \in J, h, j \in \Omega_i \tag{6}$$

$$y_{i,l,j} + y_{l,i,j} = 1, \quad \forall i, l \in J, j \in (\Omega_i \cap \Omega_l) \tag{7}$$

$$b_{l,j} - \left(b_{i,j} + p_{i,j}\right) + U \cdot \left(3 - y_{i,l,j} - x_{i,j,k} - x_{l,j,k}\right) \geq 0, \quad \forall i, l \in J, k \in E_j, j \in (\Omega_i \cap \Omega_l) \tag{8}$$

$$x_{i,j,k} \in \{0, 1\}, \quad \forall i \in J, j \in \{1, 2, 3, 4, 5\}, k \in E_j \tag{9}$$

$$z_{i,h,j} \in \{0, 1\}, \quad \forall i \in J, h, j \in \{1, 2, 3, 4, 5\} \tag{10}$$

$$y_{i,l,j} \in \left\{0, \frac{1}{2}, 1\right\}, \quad \forall i, l \in J, j \in \{1, 2, 3, 4, 5\}. \tag{11}$$

The four functions (1)–(4) are to minimize the penalty caused by the average sojourn time, the total earliness, the total tardiness, and the skipping rate penalty. Constraints (5) make sure that each operation can select one and only one available machine in each stage which it must go through. Constraints (6) guarantee that for two consecutive operations of each job, the next one can be started only after the completion of the preceding one plus the transfer time between the two stages. Constraints (7) and (8) make sure that processing overlap of jobs on the same machine is not permitted. Constraints (9)–(11) define the value ranges for the decision variables.

C. Example Problem Instance

The following example will help in illustrating this complex HFF-D problem. Consider an instance with five torpedoes and stages. There are two blast furnaces in the first stage, two preslag-pouring devices in the second stage, two dephosphorization or desulphurization devices in the third stage, two post-slag-pouring devices in the fourth stage, and two pouring devices in the last stage. In the five torpedoes, the second and third torpedoes are of the special molten irons, while the others are of the common molten iron. That is, $n = 5$, $m = 10$, $E_1 = \{1, 2\}, E_2 = \{1, 2\}, E_3 = \{1, 2\}, E_4 = \{1, 2\}, E_5 = \{1, 2\}$. Let $w_1 = 1$, $w_2 = 1$, $w_3 = 0.5$, and $w_4 = 1$. The set of special iron $= \{2, 3\}$ and the set of common iron $= \{1, 4, 5\}$. The processing times p_{ij}, the transfer times T_{ij}, and the due date window are given as follows:

$$\left[p_{ij}\right]_{5\times5} = \begin{bmatrix} 30 & 30 & 30 & 40 & 30 \\ 30 & 30 & 30 & 40 & 30 \\ 30 & 30 & 30 & 30 & 30 \\ 30 & 30 & 30 & 40 & 30 \\ 30 & 30 & 30 & 30 & 30 \end{bmatrix}$$

$$\left[T_{ij}\right]_{5\times5} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \\ 20 & 15 & 0 & 0 & 0 \\ 30 & 30 & 10 & 0 & 0 \\ 40 & 45 & 20 & 10 & 0 \end{bmatrix}$$

$$\begin{bmatrix} d_s^i \\ d_e^i \end{bmatrix}_{5\times5} = \begin{bmatrix} 150 & 180 & 200 & 200 & 220 \\ 170 & 200 & 220 & 220 & 230 \end{bmatrix}.$$

The Gantt chart of a solution for the above problem instance is shown in Fig. 2, where each operation is represented by a rectangle labeled with the job number. For example, in the

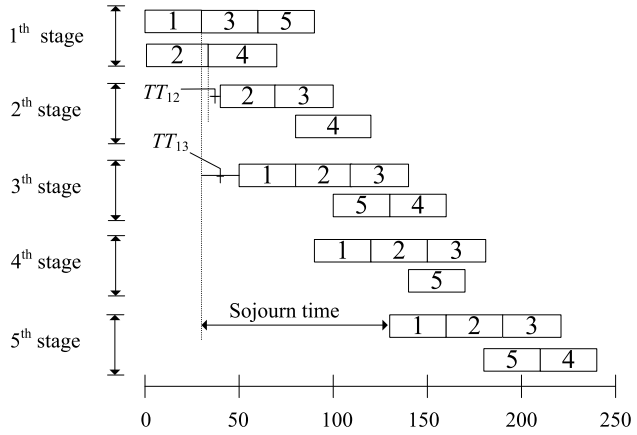

Fig. 2. Example Gantt chart.

first stage, job J_1 is processed on the first machine, and all the five jobs should be processed in the first stage. The two jobs J_1 and J_5 skip the second stage, while J_4 skips the fourth stage. The average sojourn time F_1 is 11.8. The starting times for jobs 1–5 at the last stage are 160, 190, 220, 240, and 210, respectively. So, jobs 1–3 are all on due date. The total earliness value F_2 is 10, and the total tardiness value F_3 is 20. The total number of operations being processed at stages 2 and 4 is 7. Then, F_4 is $(1-7/10)\times 100 = 30$. Therefore, the objective value is $11.8\times 1+10\times 1+20\times 0.5+30\times 1 = 61.8$.

III. Proposed Algorithm

In this section, we first introduce the problem-specific heuristics, which include the dynamic encoding mechanism, the flexible decoding strategy, and the right-shift heuristic. Then, we present the neighborhood structures, the population initialization method, and the strategies for the three types of artificial bees.

A. Dynamic Encoding Mechanism

To solve the HFS scheduling problem, the permutation-based encoding mechanism is commonly used [4], [5], [7], [11]–[17]. In the permutation-based representation, each solution is represented by a string of integers. Each integer corresponds to a job, and the sequence represents the processing order at the first stage.

In the proposed algorithm, each individual is represented by two vectors. The first vector is named the scheduling vector, and the second vector is named the skipping vector, which contains the information of the operation skipping. To consider the exploitation and exploration abilities, we propose a dynamic encoding mechanism for the scheduling vector. First, we divide the entire evolution evenly into two phases. During the first part of evolution, we apply the permutation-based encoding method for the scheduling vector, which is represented in Fig. 3(a). For the second part of evolution, we develop a novel detailed representation, which is given in Fig. 3(b). In this detailed representation, each scheduling vector is represented by five sub-vectors, in which each sub-vector contains n elements. Therefore, the length of the scheduling vector is equal to $5n$. It should be noted that, at the first stage,

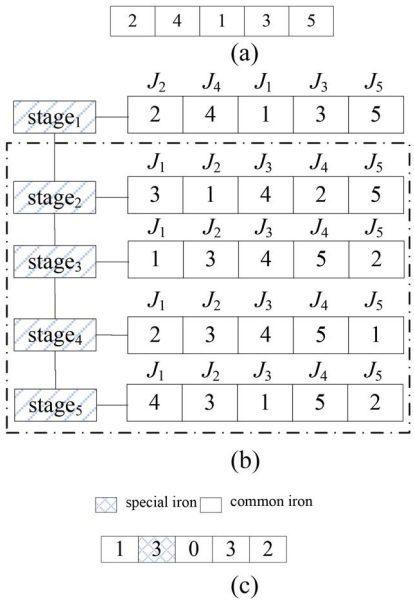

Fig. 3. Encoding representation. Scheduling vector for the (a) first and (b) second part of evolution. (c) Skipping vector.

each job and machine is available at time zero; therefore, all the jobs are scheduled according to their occurrence in the scheduling vector at the first stage. At the following stage, each job will be scheduled according to their completion time at the previous stage. However, for the situation in which several jobs with different available times are waiting for the same machine, we should schedule each waiting job according to its sequence in the scheduling vector at the corresponding stage.

In the proposed dynamic encoding mechanism, the element in the sub-vector in Fig. 3(b) has different meanings for different stages. At the first stage, each element represents the corresponding job, and the sequence of these elements corresponds to the scheduling order. For example, in Fig. 3(b), at the first stage, the first element is 2, which represents job J_2, and the last one is for J_5. All jobs are scheduled according to their sequence in the first sub-vector, i.e., the processing sequence at the first stage is J_2, J_4, J_1, J_3, and J_5. At the following stages, the element in the sub-vector gives the processing priority for the corresponding job J_i when several jobs are waiting for the same machine. For example, given that three jobs J_1, J_2, and J_5 are ready and waiting for M_2 at the second stage, we can see from the sub-vector that the processing sequence is J_2, J_1, and J_5. That is, J_2 is the first job to be processed after M_2 is available, because in the sub-vector at the second stage, the elements that correspond to J_2, J_1, and J_5 are 1, 3, and 5, respectively.

The skipping vector tells the skipping status of each operation. Therefore, the length of the skipping vector is equal to n. In the skipping vector, there are four types of status: 1) "0" represents skipping the two stages, that is, the corresponding job should skip the pre- and post-processing stages; 2) "1" means that the corresponding job will only skip the preprocessing stage and be processed in the post-processing stage; 3) "2" means that the job will be processed in the preprocessing stage while skipping the post-processing stage; and 4) "3" indicates that the job will not skip any stage.

```
//at the first stage j=1
For each machine k∈ E_1, set μ_k = 0
For i ∈ Φ_j
    Find the first available machine k* = min_{k∈E_1}{u_k}
    Arrange for job π_i to be processed on machine k*
    Let e_{i,1}=μ_{k*}+p_{i,1}, and μ_{k*} = e_{i,1}
End for

//at stages j=2,…,5
For j=2,…, 5
    Let b_{π_i^(j)} be the possible starting time of π_i^(j), which is set to
        e_{i,j-1} + Σ_{h=1}^{j} T_{h,j} · z_{i,h,j}
    Generate a job permutation π^(j) = (π_1^(j), π_2^(j), ..., π_{|Φ_j|}^(j)) according to
        ascending b_{π_i^(j)}
    For i ∈ Φ_j
        Find the first available machine k* = min_{k∈E_j}{u_k}
        If a set of jobs Ω satisfies b_{π_i^(j)} ≤ u_{k*}, where π_i^(j) ∈ Ω, then
            arrange for job π_i^(j), which has the minimum value in the
            scheduling vector at stage j among the jobs in Ω, to be
            processed on machine k*
        Let e_{i,j}=max(μ_{k*}, b_{π_i^(j)})+p_{i,j}, and μ_{k*} = e_{i,j}
    End for
End for
```

Fig. 4. Flexible decoding strategy.

Note that, in the production system, the special molten iron should not skip any stage; in other words, we should assign 3 for the special molten iron. For example, in Fig. 3(c), the second job is of the special molten iron, which should be processed through all the stages.

B. Flexible Decoding Strategy

Note that, the permutation-based encoding mechanism is easy to implement. However, the scheduling sequence at the following stages should be decided according to the completion time at the previous stage, which is not flexible and can result in many promising search space regions being ignored. For example, if more than one job with different completion times at the previous stage are waiting for the same available machine, the job with the earlier completion time will deterministically be processed first. For example, consider two jobs J_1 and J_2 with completion times of 25 and 30 at the first stage that are waiting for the same available machine M_1. At time point 35, M_1 is available, and J_1 will be processed first because it has an earlier completion time at the first stage than J_2.

To avoid ignoring more promising search space regions, we propose a flexible decoding strategy. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a solution, and $\pi^{(j)} = (\pi_1^{(j)}, \pi_2^{(j)}, \ldots, \pi_{|\Phi_j|}^{(j)})$ represent the job permutation generated in stage j by sorting jobs in increasing order of their completion times at stage $j-1$. Denoted as μ_k the idle time of machine k. The pseudocode is given in Fig. 4.

It should be noted that, at the first stage, all the jobs are processed according to their sequence in the first part of the scheduling vector. In the following stages, all the jobs are first sequenced in an ascending order according to their completion times at the previous stage. Then, if one machine becomes available while several jobs are waiting for it, all the waiting jobs will be scheduled according to their occurrence values in the corresponding sub-vectors.

For the above example, let the completion times at the first stage for the two jobs J_2 and J_3 be equal to 30 and 25, respectively. Both J_2 and J_3 are waiting for M_2 in the second stage, which is available at time point 35. Without the reference sequence, as commonly used in [4]–[17], J_3 will be processed first because it has an earlier completion time at the first stage. In this paper, according to the reference sequence in the second part, J_2 is to be processed first, which increases the flexibility and can thus widen the search area.

C. Right-Shift Strategy

In this section, we present a right-shifting strategy to minimize the penalty of earliness. Suppose $\pi^k = (\pi_1^k, \pi_2^k, \ldots, \pi_h^k)$ is the job sequence on machine k at the last stage. We first decode a block that contains operations that are processed consecutively on the same machine. Let B_i be the block that contains the operation π_i^k; in other words, if two operations are processed without any idle time, then we set them to the same block. Let $\gamma = (\gamma_{B_1,B_2}, \gamma_{B_2,B_3}, \ldots, \gamma_{B_{q-1},B_q}, \gamma_{B_q,B_{q+1}})$ be the set of idle times between two consecutive blocks and $\gamma_{B_i,B_{i+1}}$ be the idle time between B_i and B_{i+1}. Suppose there are q blocks $B = (B_1, B_2, \ldots, B_q)$ being processed on a given machine; we set $\gamma_{B_q,B_{q+1}} = \infty$. Let $\delta^k = (\delta_1, \delta_2, \ldots, \delta_h)$ be the earliness values for $\pi^k = (\pi_1^k, \pi_2^k, \ldots, \pi_h^k)$, which is computed as follows: if $b_{i,5} \in [d_s^i, d_e^i]$, set $\delta_i = 0$; otherwise, set $\delta_i = d_s^i - b_{i,5}$. Note that, $\delta_i > 0$ represents that the corresponding operation π_i^k starts earlier than its due date, $\delta_i = 0$ means π_i^k starts on its due date, and $\delta_i < 0$ means π_i^k starts after its due date. Let S_E be the set that includes all the operations with $\delta_i > 0$; S_D includes the operations with $\delta_i = 0$, and S_T the same for $\delta_i < 0$.

To right shift the operation π_i^k, we should consider all the operations following π_i^k in B_i and also the idle time $\gamma_{B_i,B_{i+1}}$. The decrease in the earliness value due to a single right shift of π_i^k is $w_2|S_E|$, the increase in the tardiness value is $w_3|S_T|$, and the increase in the average sojourn time value is $w_1|S_{B_i}|/n$. Therefore, if the condition $w_2|S_E| > w_3|S_T| + w_1|S_{B_i}|/n$ is satisfied, we can right-shift π_i^k by one time unit. Meanwhile, to right any operation, we should consider the idle time between the current and following blocks. If we can right shift to erase the idle time $\gamma_{B_i,B_{i+1}}$, then the two blocks will be combined into one block. The right-shift procedure is given in Fig. 5 with the computational time complexity ($O(n^2)$).

Given the job sequence illustrated in Fig. 6(a), there are four blocks, i.e., $B = \{\{1, 2\}\{3\}, \{4, 5, 6, 7\}, \{8\}\}$. We first compute the idle time between each pair of consecutive blocks: {5, 20, 10}. Then, we compute the earliness values $\delta^k = \{10, 0, -10, 0, 20, 10, 10, 10\}$. Let $w_1 = 10$, $w_2 = 10$, and $w_3 = 5$. For the first block, we divide all the operations into three sets, i.e., $S_E = \{1\}$, $S_T = \{\}$, and $S_D = \{2\}$.

For the first loop, we obtain the first operation with a positive earliness value π_1^k, and the condition $w_2|S_E| - w_3|S_T| - w_1|S_{B_i}|/n = 10\times1 - 10\times2/8 = 7.5 > 0$. Next, we obtain $\Delta = \min\{10, 10\} = 10$ and $\theta = \min\{10, 5\}$. Therefore, we can right

For each job π_i^k (i=1,2,…,h) and the its block S_{B_i}, While $w_2|S_E| > w_3|S_T| + w_1\frac{|S_{B_i}|}{n}$ do.
Step 1. $\Delta = \min\{\min_{i\in S_E}\{E_i\}, \min_{i\in S_D}\{d_e^i - b_{i,5}\}\}$
Step 2. $\theta = \min\{\Delta, \gamma_{B_i,B_{i+1}}\}$
Step 3. For each job i in block S_{B_i}, perform the following steps to right-shift it.
(1) $b_{i,5} = b_{i,5} + \theta$
(2) If $\delta_i > \theta$ or $\delta_i < 0$ then
$\delta_i = \delta_i - \theta$
End if
(3) If $\delta_i > 0$ and $b_{i,5} \in [d_s^i, d_e^i]$ then
$\delta_i = 0$;
$S_E = S_E - i$;
$S_D = S_D + i$
End if
(4) If $\delta_i = 0$ and $b_{i,5} > d_e^i$ then
$\delta_i = d_e^i - b_{i,5}$;
$S_D = S_D - i$;
$S_T = S_T + i$
End if
(5) If $\theta = \gamma_{B_i,B_{i+1}}$ then
$B_i = B_i + B_{i+1}$;
$\gamma_{i,i+1} = 0$;
$q = q - 1$
End if
End While

Fig. 5. Right-shift strategy.

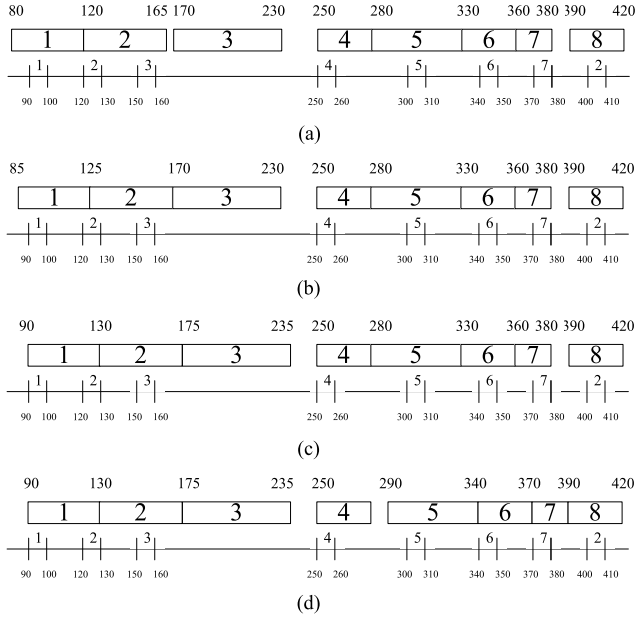

Fig. 6. Example procedure of the right-shift strategy. (a) Original Gantt chart. (b) Gantt chart after the first loop. (c) Gantt chart after the second loop. (d) Gantt chart after the third loop.

shift the first block by five time units. After the right-shifting, we obtain the new starting time for each operation in the first block, i.e., $b_{1,5} = 85$ and $b_{2,5} = 125$. Then, the updated earliness values should be $\delta^k = \{5, 0, -10, 0, 20, 10, 10, 10\}$. The new block $B_1 = \{1, 2, 3\}$, i.e., the following block should be combined into the first block because the idle time between them has been erased. Fig. 6(b) illustrates the Gantt chart after applying the first loop.

For the second loop, we first divide the new block into three different sets: 1) $S_E = \{1\}$; 2) $S_D = \{2\}$; and 3) $S_T = \{3\}$. Then, we obtain the first operation with a positive earliness value π_1^k, and the condition $w_2|S_E| - w_3|S_T| - w_1|S_{B_i}|/n = 10 \times 1 - 5 \times 1 - 10 \times 3/8 > 0$. Then, we obtain $\Delta = \min\{5, 5\} = 5$ and $\theta = \min\{5, 20\}$. Therefore, we can further right shift the first block five time unit. After the right shifting, we obtain the new starting time for each operation in the first block, i.e., $b_{1,5} = 90$, $b_{2,5} = 130$, and $b_{3,5} = 175$. Then, the updated earliness values should be $\delta^k = \{0, 0, -15, 0, 20, 10, 10, 10\}$. Then, the first block becomes as follows: $S_D = \{1, 2\}$ and $S_T = \{3\}$. Fig. 6(c) describes the Gantt chart after applying the second loop.

For the third loop, we obtain the first operation with a positive earliness value π_5^k, and we should consider {5, 6, 7} as a block, where $S_E = \{5, 6, 7\}$. Then, we obtain the condition $w_2|S_E| - w_3|S_T| - w_1|S_{B_i}|/n = 10 \times 3 = 30 > 0$. Then, we obtain $\Delta = \min\{10, \infty\} = 10$ and $\theta = \min\{10, 10\} = 10$. Therefore, we can right shift the second block {5, 6, 7} by ten time units. After the right shifting, we obtain the new starting time, i.e., $b_{5,5} = 290$, $b_{6,5} = 340$, and $b_{7,5} = 370$. Then, the updated the earliness values should be $\delta^k = \{0, 0, -15, 0, 10, 0, 0, 10\}$. Then, the updated $B = \{\{1, 2, 3\}\{4\}, \{5, 6, 7, 8\}\}$. For the last block, because the condition $w_2|S_E| - w_3|S_T| - w_1|S_{B_i}|/n = 0$, we cannot right shift any operation further. Therefore, the last Gantt chart after right shifting is in Fig. 6(d).

It can be observed that, before applying the right-shift strategy, the earliness and tardiness penalties are 600 and 50, respectively. After applying the strategy, the resulted earliness and tardiness penalties are 200 and 75, respectively. Note that, the average sojourn time penalty increases by 75. Therefore, we get a decrease of 300 for the total penalties by using the proposed right-shift strategy.

D. Neighborhood Structures

Considering the problem characteristics of the molten iron scheduling problem, we propose two levels of neighborhood structure for iron skipping and operation scheduling named the skipping and scheduling neighborhood structures, respectively.

1) Skipping Neighborhood Structure: The skipping neighborhood structure provides a different skipping plan for the current individual. To consider both exploitation and exploration capability of the proposed algorithm, we present a novel skipping neighborhood structure, which is given in Fig. 7.

2) Scheduling Neighborhood Structure: For solving the hybrid flowshop scheduling problem, the neighborhood structures, such as insertion, swap, pairwise exchange, and multiswap, are commonly used in the literature. Pan *et al.* [14] verified that the multiswap neighborhood can be evaluated more efficiently than the other three neighborhood structures. In this paper, considering the problem structure and the balance of the exploration and exploitation abilities, we combine the insertion and mutation neighborhood structures in a random manner. Fig. 8(a) gives an example of the mutation neighborhood structure, and Fig. 8(b) gives an example of the insertion neighborhood structure.

Procedure *ET_skipping_neighbor*()
Input: the skipping vector for the current solution.
Output: the skipping vector for the newly generated solution.
Step 1. Compute the earliness and tardiness penalty.
Step 2. If the earliness penalty is larger than the tardiness penalty, then randomly select a common iron and set "1", "2", and "3" with the same probability to the skipping type of the selected iron.
Step 3. If the tardiness penalty is larger than the earliness penalty, then randomly select a common iron and set "0," "1," and "2" with the same probability to the skipping type of the selected iron.
Step 4. If the tardiness penalty is equal to the earliness penalty, then randomly select a common iron and set a random iron skipping type different with the current skipping type.
End.

Fig. 7. Procedure of the skipping neighborhood structure.

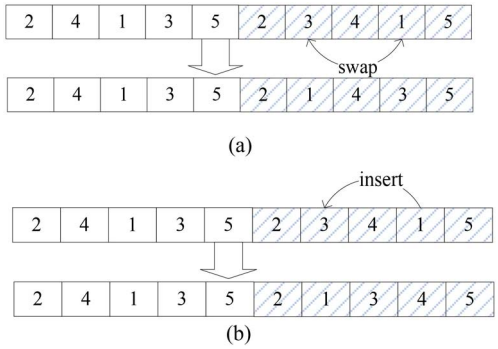

Fig. 8. Scheduling neighborhood structures. (a) Mutation and (b) insertion neighborhood structures.

E. Population Initialization

To generate a population with a high level of solution quality and diversity, we propose a very simple population initialization heuristic, which is given as follows.

Step 1: Let counter Cnt $=$ 1, and perform the following steps until Cnt $=$ PS.
Step 2: Generate a solution in a random way and evaluate it. If the new generated solution is not the same as any individual in the current population, insert it into the population and let Cnt $=$ Cnt $+$ 1; otherwise, discard it.
Step 3: Go back to step 2.

F. Employed Bee Phase

The employed bees complete the exploitation task around their given solutions. When applying the proposed DABC algorithm for solving the HFF-D problem, we set off all employed bees to perform the exploitation task around the given food source. The detailed steps for each employed bee are given as follows.

Step 1: Apply the ith employed bee to the ith food source in the current population.
Step 2: Produce NS neighboring food sources around the given solution.
Step 3: Evaluate each newly generated food source and perform the following replacement processes: 1) if its fitness value is better than the best solution found so far, then replace the latter and 2) if its fitness value is better than the current food source, then replace the latter.

G. Onlooker Bee Phase

In the canonical ABC algorithm, similar to the wheel selection in GA, an onlooker bee selects a food source with a winning probability from several candidate solutions. In this paper, we adopt a simple method for each onlooker bee as follows.

Step 1: Randomly select three solutions in the current population.
Step 2: Compare the three selected solutions; select the solution with the minimum fitness value as the current solution.
Step 3: Produce NS neighboring food sources around the given solution.
Step 4: Evaluate the newly generated food source, and update both worst solution and best one in the current population.

H. Scout Bee Phase

In the classical ABC algorithm, a solution becomes an abandoned one if it cannot be improved after a certain number of generations. Then, a scout bee replaces the abandoned solution with a randomly generated solution to maintain the population with a high level of quality. In this paper, we apply the following steps for each scout bee.

Step 1: For the best solution found thus far, perform the following steps S_t times: apply the insertion operator for the scheduling vector and the proposed ET_skipping_neighbor function for the skipping vector.
Step 2: Compare the new generated neighboring solutions, and select the best one as the current solution. If there are several solutions with the same best fitness value, then randomly select one.

I. Enhanced Local Search

To further improve the searching ability of the proposed algorithm, we develop an enhanced local search for the best solution found so far. That is, after the three artificial bee processes discussed in the above section, the enhanced local search will be applied to the best solution for enhanced searching. The detailed steps of the enhanced local search are as follows.

Step 1: For the best solution, perform the following steps until the stop condition is satisfied.
Step 2: *Destruction Phase:* Randomly generate E_d positions in the skipping vector for the current solution, where E_d is a system parameter. Delete the corresponding elements from the skipping vector, and insert these deleted elements into a vector p_d.

Step 3: *Construction Phase:* For each element i in p_d, perform the following steps until p_d is empty.
- Step 3.1: For the job at position i, change the skipping type from 0 to 3, and evaluate the resulting solutions.
- Step 3.2: Select the best skipping type for the deleted element i.
- Step 3.3: Go back to step 3.

J. Framework of the Proposed Algorithm

The detailed steps of the proposed DABC algorithm are given as follows.

Step 1: Initialization phase.
- Step 1.1: Set the system parameters.
- Step 1.2: Initialize the population (see Section III-E).

Step 2: Evaluate each solution in the population (see Sections III-B and III-C). Select the best solution as the current incumbent individual.

Step 3: If the stopping criterion is satisfied, output the best solution; otherwise, perform steps 4–8.

Step 4: Employed bee phase (see Section III-F).
- Step 4.1: Set the ith employed bee to the ith food source in the current population and perform the exploitation task by using the proposed neighborhood structures discussed in Section III-D.
- Step 4.2: Evaluate the newly generated solutions, and update both current solution and best one.

Step 5: Onlooker bees phase (see Section III-G).
- Step 5.1: For each $i = 1, 2, \ldots,$ PS, repeat the following steps.
- Step 5.2: Randomly select three solutions in the current population, and select the best one as the food source for the onlooker by using the tournament selection method.
- Step 5.3: Generate several neighboring solutions by using the proposed neighborhood structures discussed in Section III-D around the selected food source.
- Step 5.4: Evaluate the newly generated solutions, and update both worst solution and best one.

Step 6: Scout bee phase (see Section III-H). If a solution in the population has not been improved during the limit trials, abandon it and put a scout bee around it.

Step 7: Perform the enhanced local search process around the best food source found so far (see Section III-I). Replace the worst food source in the current population with the best one.

Step 8: Go back to step 3.

IV. Experimental Evaluation

This section discusses the computational experiments used 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 verify the effectiveness and efficiency of the proposed DABC algorithm, we make detailed comparisons with three presented efficient algorithms, i.e., GA [31], ABC [14], and PSO [21]. The main reasons for selecting GA, ABC, and PSO are the following: 1) the ABC algorithm in [14] is developed for solving the steel-making casting problem, which is an important realistic application of HFS. The comparison between DABC and ABC is mainly used to verify the efficiency of the proposed strategies and 2) GA and PSO are the two recently presented algorithms for solving the HFF problem, whereas the extension HFF-D problem is considered in this paper. Because the HFF-D problem is first proposed in this paper and there is no literature for it, we select and extend GA, PSO, and ABC to solve the HFF-D problem to verify the performance of the proposed DABC algorithm.

The computational times for each instance are set to 100 s. The best results of the experiments for the 15 randomly generated problems over 30 independent runs were collected for performance comparisons. The Taguchi method of design of experiments (DOE) [44] is utilized to test the influence of the key parameters on the performance of the proposed algorithm. The fitness value for each solution is computed and used to make detailed comparisons. The fitness value is to minimize the four penalty values computed as in (1)–(4). The performance measure is relative percentage increase (RPI), calculated as follows:

$$\mathrm{RPI}(f) = \frac{f_c - f_b}{f_b} \times 100 \tag{12}$$

where f_b is the fitness value of the best solution found by all the compared algorithms, while f_c is the fitness value of the best solution generated by a given algorithm.

A. Experimental Instances

In this paper, we generate 15 problem instances according to the practical situations of the iron and steel production in Baosteel complex, the largest and most advanced iron and steel enterprise in China. The technological constraints are given as follows.

1) There are four blast furnaces, two preprocessing devices, five dephosphorization or desulphurization machines, two post-processing furnaces, and five iron pouring centers in the shop. For each torpedo or job, the processing times are randomly generated in the range of [35, 40] for the common irons and [40, 45] for the specific molten irons.
2) For each machine, the release time is not considered as a technical capability.
3) The transfer times for each of the two consecutive stages are in the range [10–15].
4) The setup time for each job is not considered as a technical capability.
5) For the due date window, the start and end time points for all irons are set to $[300 \pm \delta, 15n \pm \delta]$ according to the practical production data, where n represents the total number of irons in the system and δ represents a random integer number in [0, 30].

TABLE I
COMBINATIONS OF KEY PARAMETER VALUES

Parameter	Level			
	1	2	3	4
PS	10	30	50	100
NS	3	5	10	15
S_t	5	10	15	20
E_d	$\frac{1}{4}n$	$\frac{1}{5}n$	$\frac{1}{10}n$	$\frac{1}{20}n$
L_n	10	20	30	40

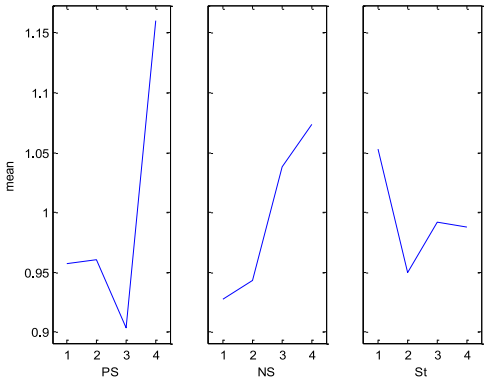

Fig. 9. Factor level trend of three key parameters.

6) The penalty coefficient values are set according to the practical experiences: $w_1 = 1, w_2 = 1, w_3 = 0.5$, and $w_4 = 1$.

B. Experimental Parameters

The five parameters include the population size (PS), the size of neighboring (SN) area for the employee and onlooker bees, the local search times for the scout bee (S_t), the destruction length (E_d) for the enhanced local search procedure, and the number of iterations during which the solution does not improve (L_n). According our preliminary experiments, the levels of the five parameters are given in Table I.

The Taguchi method of DOE is utilized to test the influence of these three parameters on the performance of the proposed algorithm. For the first three parameters, an orthogonal array $L_{16}(4^3)$ is selected. For each parameter combination, the proposed algorithm is run independently 30 times, and then the average RPI values obtained by the compared algorithms are collected as the response variable. Fig. 9 reports the factor level trend of the three parameters.

It can be observed from Fig. 9 that the proposed algorithm has better performance under the three parameters with the following levels: 1) PS with level 3; 2) SN with level 1; and 3) S_t with level 2. We can also see from Fig. 9 that the parameter PS is more critical than the other two parameters in the proposed algorithm. A large value of PS means more computational resources consumed in the exploration procedure, and the algorithm will lose exploitation ability. A too small value of PS means the loss of exploration ability of the algorithm. Therefore, to balance the exploration and exploitation ability, the suitable value for the key parameter PS is set to 50 in the proposed algorithm. According to the above analysis, the suitable values for the three considered parameters are 50, 3, and 10 for PS, SN, and S_t, respectively.

Similar experiments are also carried out to optimize other parameters that are used for comparison. Their desirable parameter settings based on the experimental results are $E_d = 1/20n$ and $L_n = 20$.

TABLE II
COMPARISON OF RPI VALUES FOR THE DYNAMIC ENCODING MECHANISM

Instance	Scale	DABC	DABC_{ND}
Case1	40	0.00	93.41
Case2	48	0.00	140.36
Case3	56	0.00	76.23
Case4	64	0.00	196.88
Case5	72	0.00	54.87
Case6	80	0.00	194.97
Case7	88	0.00	239.29
Case8	96	0.00	118.96
Case9	104	0.00	96.58
Case10	112	0.00	325.42
Case11	120	0.00	82.70
Case12	128	0.00	223.94
Case13	136	0.00	88.68
Case14	144	0.00	153.16
Case15	152	0.00	37.82
Mean		0.00	141.55

C. Effectiveness of the Dynamic Encoding Mechanism

To investigate the effectiveness of the dynamic encoding mechanism, we realize the DABC algorithm presented in Section III and the DABC algorithm using a permutation-based representation (DABC_{ND} for short). The parameters for the two compared algorithms are set the same as in Section IV-B. The only difference between DABC and DABC_{ND} is that the DABC algorithm embeds the dynamic encoding mechanism. The two compared algorithms are tested on the same PC and with the same test instances. After 30 independent runs, the average RPI results for each instance are collected for comparison, which is given in Table II.

In Table II, the first column provides the problem name and the second column lists the scale size. The following two columns report the RPI values for DABC and DABC_{ND}, respectively. It can be observed from Table II that: 1) DABC obtained 15 optimal values out of the given 15 instances, whereas DABC_{ND} obtains no optimal values; 2) from the last row in the table, we can see that DABC performs the best; and 3) in a nutshell, we can obtain better results after applying the proposed dynamic encoding mechanism.

The advantages of the proposed dynamic encoding method are as follows: 1) with the permutation-based encoding at the

TABLE III
COMPARISON OF THE RPI VALUES FOR THE FLEXIBLE DECODING METHOD

Instance	Scale	DABC	$DABC_{NF}$
Case1	40	0.47	0.00
Case2	48	0.00	7.23
Case3	56	0.00	30.62
Case4	64	0.00	26.38
Case5	72	0.00	46.64
Case6	80	0.00	48.99
Case7	88	0.00	37.93
Case8	96	0.00	55.48
Case9	104	0.00	53.45
Case10	112	0.00	64.26
Case11	120	0.00	42.43
Case12	128	0.00	64.67
Case13	136	0.00	75.96
Case14	144	0.00	65.88
Case15	152	0.00	63.73
Mean		0.03	45.58

TABLE IV
COMPARISON OF THE RPI VALUES FOR THE RIGHT-SHIFT HEURISTICS

Instance	Scale	DABC	$DABC_{NR}$
Case1	40	0.00	112.97
Case2	48	0.00	78.64
Case3	56	0.00	94.75
Case4	64	0.00	100.28
Case5	72	0.00	80.69
Case6	80	0.00	84.05
Case7	88	0.00	81.46
Case8	96	0.00	76.47
Case9	104	0.00	71.22
Case10	112	0.00	83.70
Case11	120	0.00	72.04
Case12	128	0.00	72.55
Case13	136	0.00	74.99
Case14	144	0.00	77.36
Case15	152	0.00	76.75
Mean		0.00	82.53

first part of evolution, the algorithm can locate the optimal search space quickly, and thus, can increase the searching ability; 2) detailed representation at the second part of evolution, the algorithm can perform a fine-grained search in the optimal searching locations, and therefore, increase the solution quality; 3) dynamic representation, the algorithm can balance the exploration and exploitation abilities; and 4) skipping vector, the algorithm can adjust the skipping status for each operation, and thus, widen the search space.

D. Effectiveness of the Flexible Decoding Method

In Section III, we propose a flexible decoding method to enhance the search ability. To investigate the effectiveness of the flexible decoding method, we realize the DABC algorithm presented in Section III and the DABC algorithm without using the flexible decoding method ($DABC_{NF}$ for short). The parameters for the two compared algorithms are set the same as in Section IV-B. The two compared algorithms are tested on the same PC and with the same test instances. After 30 independent runs, the average RPI results for each instance are collected for comparison, which is given in Table III.

It can be observed from Table III that: 1) DABC obtained 14 optimal values out of the given 15 instances, whereas $DABC_{NF}$ obtained only one optimal value; 2) from the last row in the table, we can see that DABC obtained an average RPI value of 0.03, which is approximately 1500 times smaller than that of $DABC_{ND}$; and 3) in conclusion, the proposed DABC algorithm performs better than $DABC_{NF}$.

E. Effectiveness of the Proposed Right-Shift Heuristics

To investigate the effectiveness of the right-shift heuristic, we realize the DABC algorithm and the DABC algorithm without using the right-shift heuristic ($DABC_{NR}$ for short). The parameters for the two compared algorithms are set the same as in Section IV-B. The two compared algorithms are tested on the same PC and with the same test instances. After 30 independent runs, the average RPI results for each instance are collected for comparison, which is given in Table IV.

It can be observed from Table IV that: 1) DABC obtained optimal values for all the 15 instances and 2) from the last row in the table, we can see that DABC obtained an average RPI value of 0.00, which is significantly better than that of $DABC_{NR}$.

F. Effectiveness of the Skipping Neighborhood Structure

In Section III, we propose a novel skipping neighborhood structure considering the earliness/tardiness penalty. To investigate the effectiveness of the neighborhood structure, we realize the skipping neighborhood structure presented in Section III and the commonly used mutation neighborhood structure [26], [39]. The mutation neighborhood structure randomly selects a common iron and changes another skipping type for it in a random manner. The parameters for the two compared algorithms are set the same as in Section IV-B. The two compared algorithms are tested on the same PC and with the same test instances. After 30 independent runs, the average RPI results for each instance are collected for comparison, which is given in Table V.

It can be observed from Table V that: 1) DABC with the proposed skipping neighborhood structure obtained 14 optimal values out of the given 15 instances, and the algorithm with the mutation neighborhood structure obtains one optimal result; 2) from the last row in the table, we can see that our method obtained an average RPI value of 0.02, which is approximately 800 times smaller than that of the algorithm with the mutation method; and 3) in a nutshell, the skipping

TABLE V
COMPARISON OF THE RPI VALUES FOR SKIPPING NEIGHBORHOOD STRUCTURES

Instance	Scale	Our method	Mutation
Case1	40	0.31	0.00
Case2	48	0.00	1.09
Case3	56	0.00	3.01
Case4	64	0.00	6.82
Case5	72	0.00	5.54
Case6	80	0.00	9.91
Case7	88	0.00	18.21
Case8	96	0.00	13.70
Case9	104	0.00	17.49
Case10	112	0.00	24.24
Case11	120	0.00	28.44
Case12	128	0.00	25.34
Case13	136	0.00	30.34
Case14	144	0.00	44.36
Case15	152	0.00	38.88
Mean		0.02	17.82

TABLE VI
COMPARISONS OF THE RPI VALUES WITH GA-I, PSO-I, AND ABC-I

Instance	Scale	DABC	GA-I	ABC-I	PSO-I
Case1	40	0.00	184.80	148.76	189.86
Case2	48	0.00	133.02	126.39	139.34
Case3	56	0.00	171.21	147.27	148.16
Case4	64	0.00	180.26	152.74	192.77
Case5	72	0.00	151.20	129.58	146.26
Case6	80	0.00	130.90	141.60	129.18
Case7	88	0.00	150.57	116.58	129.32
Case8	96	0.00	153.02	144.88	146.25
Case9	104	0.00	144.16	138.60	161.69
Case10	112	0.00	163.65	136.21	134.74
Case11	120	0.00	106.76	121.24	117.85
Case12	128	0.00	144.32	121.61	120.97
Case13	136	0.00	161.30	126.05	122.02
Case14	144	0.00	119.04	116.52	133.70
Case15	152	0.00	183.24	107.85	114.83
Mean		0.00	151.83	131.73	141.80

neighborhood structure enhances the search ability of the algorithm.

G. Comparisons With the Presented Efficient Algorithms

To make a fair comparison between the proposed algorithm and other three compared algorithms, we implement GA, PSO, and ABC with two types.

1) In the first type, each compared algorithm is coded with their own components, i.e., encoding/decoding mechanism, neighborhood structures, and local or global search approaches. Here, we named the compared algorithms in the first type GA-I, PSO-I, and ABC-I, respectively. The skipping vector is also used in GA-I, PSO-I, and ABC-I to record the skipping status for each operation.
2) In the second type, all the compared algorithms are implemented with the same components as the proposed DABC algorithm, i.e., the same encoding/decoding mechanism, right-shift heuristic, and population initialization method. Then, all the four compared algorithms perform their own evolution operators. Here, we named the compared algorithms in the second type GA-II, PSO-II, and ABC-II, respectively. Each compared algorithm is run independently 30 times for each given instance. All algorithms adopt the same maximum elapsed CPU time limit of $t = 100$ s as a termination criterion. This criterion is practical in realistic production systems.

1) Comparisons With GA-I, PSO-I, and ABC-I: The detailed implementation of GA-I, PSO-I, and ABC-I is the following.

a) For GA-I, the following components as in [31] are embedded: the permutation-based encoding, the k-way selection, the crossover operators (including right-hand-side-segment swap crossover, single point order crossover (SPOX)-1, SPOX-2, and two-point order crossover), the mutation operators (including the order shift and the swap operator), and the sequential implementation framework.
b) For PSO-I, the following components as in [21] are embedded: the smallest position value-based encoding method, the population initialization (one is generated by Nawaz-Enscore-Ham heuristic, and others are randomly generated from feasible solutions), the updated movement approach by two global best solutions, and the tabu search-based local search mechanism.
c) For ABC-I, the following components as in [14] are embedded: the permutation-based representation, the population initialization method, the employed bee, onlooker bee and scout bee phases, the neighboring solution generation operator, and the enhanced strategy.

The results for the comparisons with GA-I, PSO-I, and ABC-I are reported in Table VI. It can be observed from Table VI that: a) in comparison with the other three algorithms, the proposed DABC obtained all improved values for the given 15 instances, which is significantly better than the other compared algorithms; b) on average, the proposed DABC obtained an RPI value of 0.00, which is obviously smaller than that of the ABC-I algorithm, the second best performer with 131.73 overall average RPI; and c) the comparison results show the efficiency and robustness of the proposed DABC algorithm.

To determine whether the observed differences from the above table are indeed significantly different, we also apply the Friedman test [45], [46] and the Holm multiple comparison test [47] as a *post hoc* procedure for the pair comparison.

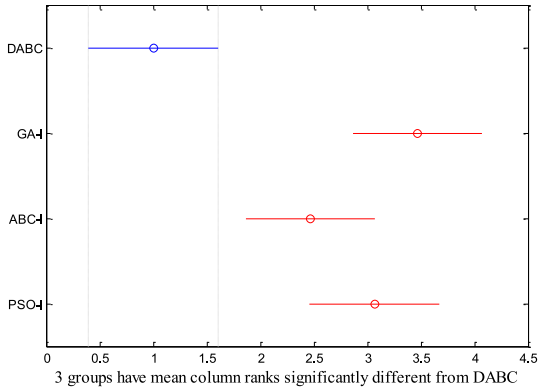

Fig. 10. Multicompare results for comparisons with GA-I, PSO-I, and ABC-I.

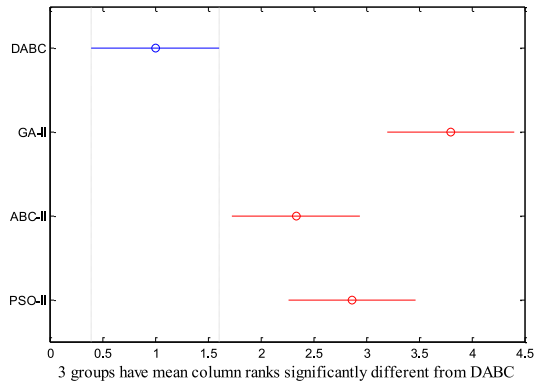

Fig. 11. Multicompare results for comparisons with GA-II, PSO-II, and ABC-II ($w_1 = 1, w_2 = 1, w_3 = 0.5,$ and $w_4 = 1$).

TABLE VII
COMPARISONS OF THE RPI VALUES WITH GA-II, PSO-II, AND ABC-II

Instance	Scale	IDABC	GA-II	DABC-II	PSO-II
Case1	40	0.00	5.10	1.33	3.15
Case2	48	0.00	7.92	1.62	1.56
Case3	56	0.00	6.63	3.05	2.98
Case4	64	0.00	4.09	2.32	4.06
Case5	72	0.00	7.36	1.23	2.79
Case6	80	0.00	8.76	1.22	6.80
Case7	88	0.00	5.92	2.90	5.61
Case8	96	0.00	6.02	2.10	8.70
Case9	104	0.00	8.05	2.03	5.97
Case10	112	0.00	10.01	3.40	5.01
Case11	120	0.00	11.02	2.35	4.02
Case12	128	0.00	11.98	3.47	7.01
Case13	136	0.00	6.03	8.89	7.02
Case14	144	0.00	11.99	8.03	10.02
Case15	152	0.00	12.01	10.11	10.05
Mean		0.00	8.19	3.60	5.65

Fig. 10 gives the pair comparison results after applying the Holm multiple comparison test. It can be concluded from Fig. 10 that the proposed DABC algorithm is significantly better than the other compared algorithms.

Compared with the presented algorithms in the first type, i.e., GA-I, ABC-I, and PSO-I, the main advantages of the proposed DABC are as follows: a) a dynamic encoding and flexible decoding mechanism considering the problem characteristics is embedded in DABC, which makes DABC more flexible to adapt to the considered production process; b) several skipping and scheduling neighborhood structures are presented to balance the exploration and exploitation abilities; c) a right-shift heuristic considering the problem structure and objective features is used to minimize the objective values; and d) an enhanced local search is embedded in DABC to further improve the exploitation ability.

2) Comparisons With GA-II, PSO-II, and ABC-II: The results for the comparisons with GA-II, PSO-II, and ABC-II are reported in Table VII. It can be observed from Table VII that: a) DABC obtained all improved values for the 15 given instances, which is significantly better than the other compared algorithms; b) on average, DABC obtained an RPI value of 0.00, whereas the second best performer with 3.60 overall average RPI; and c) the comparison results show the efficiency and robustness of the proposed DABC algorithm.

TABLE VIII
COMPARISONS OF THE RPI VALUES WITH DIFFERENT WEIGHTED VALUES ($w_1 = 0.5, w_2 = 1, w_3 = 1,$ AND $w_4 = 1$)

Instance	Scale	IDABC	GA-II	DABC-II	PSO-II
Case1	40	0.00	9.98	1.77	2.15
Case2	48	0.00	10.04	4.08	2.97
Case3	56	0.00	7.98	3.93	2.20
Case4	64	0.00	6.27	5.47	5.38
Case5	72	0.00	6.04	4.29	3.40
Case6	80	0.00	7.90	5.08	3.53
Case7	88	0.00	7.71	2.29	3.81
Case8	96	0.00	12.01	3.06	9.60
Case9	104	0.00	15.02	1.28	4.00
Case10	112	0.00	1.08	1.28	5.02
Case11	120	0.00	14.89	2.76	5.10
Case12	128	0.00	18.03	2.12	8.02
Case13	136	0.18	13.01	0.00	10.80
Case14	144	0.00	19.87	2.04	12.03
Case15	152	0.00	24.86	9.07	12.97
Mean		0.01	11.65	3.23	6.07

Fig. 11 reports the pairwise comparison results after applying the Holm multiple comparison test. It can be observed from Fig. 11 that DABC is significantly better than the other compared algorithms, even when all the algorithms are embedded with the same components, i.e., the same encoding/decoding mechanism, right-shift heuristic, and population initialization method.

To further verify the performance efficiency of the proposed DABC algorithm under different weighted values, we made a comparison with GA-II, PSO-II, and ABC-II under different type of weighted values, i.e., $w_1 = 0.5, w_2 = 1, w_3 = 1,$ and $w_4 = 1$. The comparison results in Table VIII and Fig. 12 also

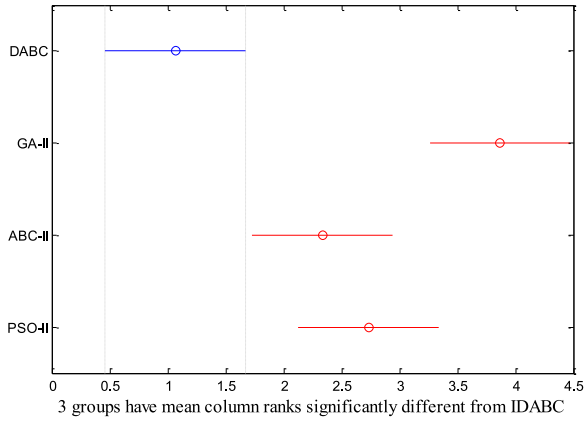

Fig. 12. Multicompare results for comparisons with GA-II, PSO-II, and ABC-II ($w_1 = 0.5, w_2 = 1, w_3 = 1,$ and $w_4 = 1$).

show the efficiency of the proposed DABC algorithm with different weighted values.

The main reasons that DABC outperforms GA-II, ABC-II, and PSO-II are as follows. Compared with ABC-II, the main advantages of DABC are the following: a) the proposed skipping and scheduling neighborhood structures balance the exploration and exploitation abilities; b) the enhanced local search method can further improve the exploitation ability; and c) the improved scout bee strategy can enhance the search ability while maintaining the exploration ability. The same advantages of DABC are also found in the comparison with GA-II and PSO-II. Other advantage of DABC compared with GA-II and PSO-II are the following: a) the proposed employed bee strategy performs the exploitation and enhance the local search ability; b) onlooker bees strategy further enhance the search ability; and c) scout bee strategy can enhance the exploration ability and avoid the algorithm being stuck in a local optima.

V. Conclusion

The HFF with dynamic operation skipping in molten iron scheduling problems is first proposed in this paper, which can be applied in many realistic applications. An improved DABC is proposed. The primary contributions of this paper are as follows: 1) a dynamic encoding mechanism is presented for the considered problem; 2) a flexible decoding method is embedded; 3) to enhance the exploitation and exploration capability of the proposed algorithm, several effective skipping and scheduling neighborhood structures are developed; 4) a problem-specific right-shift strategy is developed; and 5) an enhanced local search procedure is utilized to enhance the exploitation ability.

Comparative experimental results with three popular algorithms, namely, GA, ABC, and PSO demonstrated that the proposed DABC algorithm significantly outperforms the three compared algorithms in solving the considered problems. Experimental results and statistical analysis show the robustness and efficiency of the proposed algorithm. Future work on developing more effective and computationally more efficient algorithms for solving the considered problems is highly desirable. In addition, we should also apply the proposed algorithm to solve scheduling problems with dynamic operation skipping features in dynamic environments.

References

[1] L. X. Tang, G. S. Wang, and J. Y. Liu, "A branch-and-price algorithm to solve the molten iron allocation problem in iron and steel industry," *Comp. Oper. Res*., vol. 34, no. 10, pp. 3001–3015, 2007.

[2] D. A. Gomes, J. G. De Miranda, and M. C. De Souza, "Optimizing the scheduling of torpedo cars to feed steelmaking with hot metal," in *Proc. Int. Conf. Ironmaking Int. Symp. Iron Ore*, Sãn Luís, Brazil, 2008, pp. 1021–1028.

[3] J. N. D. Gupta, "Two-stage, hybrid flow shop scheduling problem," *J. Oper. Res. Soc*., vol. 39, no. 4, pp. 359–364, 1988.

[4] R. Ruiz and J. A. V. Rodríguez, "The hybrid flow shop scheduling problem," *Eur. J. Oper. Res.,* vol. 205, no. 1, pp. 1–18, 2010.

[5] I. Ribas, R. Leisten, and J. M. Framinan, "Review and classification of hybrid flow shop scheduling problems from a production systems and a solutions procedure perspective," *Comp. Oper. Res*., vol. 37, no. 8, pp. 1439–1454, 2010.

[6] C. Oguz and M. Ercan, "A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks," *J. Schedul*., vol. 8, no. 4, pp. 323–351, 2005.

[7] R. Ruiz and C. Maroto, "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility," *Eur. J. Oper. Res.,* vol. 169, no. 3, pp. 781–800, 2006.

[8] O. Engin, G. Ceran, and M. K. Yilmaz, "An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems," *Appl. Soft. Comput*., vol. 11, no. 3, pp. 3056–3065, 2011.

[9] K. C. Ying and S. W. Lin, "Multiprocessor task scheduling in multistage hybrid flowshops: An ant colony system approach," *Int. J. Prod. Res.*, vol. 44, no. 16, pp. 3161–3177, 2006.

[10] M. Zandieh, S. M. T. F. Ghomi, and S. M. M. Husseini, "An immune algorithm approach to hybrid flow shops scheduling with sequence dependent setup times," *Appl. Math. Comp.*, vol. 180, no. 1, pp. 111–127, 2006.

[11] L. X. Tang and X. P. Wang, "An improved particle swarm optimization algorithm for the hybrid flowshop scheduling to minimize total weighted completion time in process industry," *IEEE Trans. Control. Syst. Technol.*, vol. 18, no. 6, pp. 1303–1314, Nov. 2010.

[12] C. J. Liao, E. Tjandradjaja, and T. P. Chung, "An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem," *Appl. Soft. Comput*., vol. 12, no. 6, pp. 1755–1764, 2012.

[13] K. Mao, Q. K. Pan, X. Pang, and T. Chai, "A novel Lagrangian relaxation approach for a hybrid flowshop scheduling problem in the steelmaking-continuous casting process," *Eur. J. Oper. Res.,* vol. 236, no. 1, pp. 51–60, 2014.

[14] Q. K. Pan, L. Wang, K. Mao, J. H. Zhao, and M. Zhang, "An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process," *IEEE Trans. Autom. Sci. Eng*., vol. 10, no. 2, pp. 307–322, Apr. 2013.

[15] Q. K. Pan and Y. Dong, "An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimization," *Inf. Sci*., vol. 277, no. 1, pp. 643–655, 2014.

[16] Q. K. Pan, L. Wang, J. Q. Li, and J. H. Duan, "A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimization," *Omega*, vol. 45, no. 1, pp. 42–56, 2014.

[17] J. Q. Li, Q. K. Pan, and F. T. Wang, "A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem," *Appl. Soft. Comput*., vol. 24, no. 1, pp. 63–77, 2014.

[18] K. Wang and S. H. Choi, "A holonic approach to flexible flow shop scheduling under stochastic processing times," *Comp. Oper. Res*., vol. 43, no. 3, pp. 157–168, 2014.

[19] L. Tang, Y. Zhao, and J. Liu, "An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production," *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 209–225, Apr. 2014.

[20] M. E. Kurz and R. G. Askin, "Scheduling flexible flow lines with sequence-dependent setup times," *Eur. J. Oper. Res.,* vol. 159, no. 1, pp. 66–82, 2004.

[21] B. Naderi, S. Gohari, and M. Yazdani, "Hybrid flexible flowshop problems: Models and solution methods," *Appl. Math. Model.,* vol. 38, no. 24, pp. 5767–5780, 2014.

[22] R. J. Wittrock, "Scheduling algorithms for flexible flow lines," *IBM. J. Res. Dev.,* vol. 29, no. 4, pp. 401–412, 1985.
[23] V. J. Leon and B. Ramamoorthy, "An adaptable problem-space-based search method for flexible flow line scheduling," *IIE Trans.*, vol. 29, no. 2, pp. 115–125, 1997.
[24] M. E. Kurz and R. G. Askin, "Comparing scheduling rules for flexible flow lines," *Int. J. Prod. Econ.,* vol. 85, no. 3, pp. 371–388, 2003.
[25] D. Quadt and D. Kuhn, "A taxonomy of flexible flow line scheduling procedures," *Eur. J. Oper. Res.,* vol. 178, no. 3, pp. 686–698, 2007.
[26] R. Ruiz, F. S. Serifoglu, and T. Urlings, "Modeling realistic hybrid flexible flowshop scheduling problems," *Comp. Oper. Res*., vol. 35, no. 4, pp. 1151–1175, 2008.
[27] C. T. Tseng, C. J. Liao, and T. X. Liao, "A note on two-stage hybrid flowshop scheduling with missing operations," *Comp. Ind. Eng.,* vol. 54, no. 3, pp. 695–704, 2008.
[28] B. Naderi, M. Zandieh, and S. F. Ghomi, "A study on integrating sequence dependent setup time flexible flow lines and preventive maintenance scheduling," *J. Intell. Manuf*., vol. 20, no. 6, pp. 683–694, 2009.
[29] T. Urlings, R. Ruiz, and T. Stützle, "Shifting representation search for hybrid flexible flowline problems," *Eur. J. Oper. Res.,* vol. 207, no. 2, pp. 1086–1095, 2010.
[30] M. Zandieh and N. Karimi, "An adaptive multi-population genetic algorithm to solve the multi-objective group scheduling problem in hybrid flexible flowshop with sequence-dependent setup times," *J. Intell. Manuf*., vol. 22, no. 6, pp. 979–989, 2011.
[31] F. M. Defersha and M. Y. Chen, "Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem," *Int. J. Adv. Manuf. Technol.*, vol. 62, nos. 1–4, pp. 249–265, 2012.
[32] Z. T. Li, Q. X. Chen, N. Mao, X. M. Wang, and J. J. Liu, "Scheduling rules for two-stage flexible flow shop scheduling problem subject to tail group constraint," *Int. J. Prod. Econ.,* vol. 146, no. 2, pp. 667–678, 2013.
[33] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling," *IEEE Trans. Cybern.*, vol. 45, no. 1, pp. 1–14, Jan. 2015.
[34] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Dept. Comput. Eng., Erciyes Univ., Kayseri, Turkey, Tech. Rep. TR06, 2005.
[35] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Appl. Soft. Comput*., vol. 8, no. 1, pp. 687–697, 2008.
[36] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Appl. Math. Comput.,* vol. 214, no. 1, pp. 108–132, 2009.
[37] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Glob. Opt.,* vol. 39, no. 3, pp. 459–471, 2007.
[38] 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, 2010.
[39] J. Q. Li, Q. K. Pan, and K. Z. Gao, "Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems," *Int. J. Adv. Manuf. Technol.*, vol. 55, nos. 9–12, pp. 1159–1169, 2011.
[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] F. G. Mohammadi and M. S. Abadeh, "Image steganalysis using a bee colony based feature selection algorithm," *Eng. Appl. Artif. Intell.*, vol. 31, no. 1, pp. 35–43, 2014.
[42] T. M. Fatih, Q. K. Pan, P. N. Suganthan, and A. Oner, "A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion," *Appl. Math. Model.,* vol. 37, nos. 10–11, pp. 6758–6779, 2013.
[43] J. Q. Li and Q. K. Pan, "Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm," *Inf. Sci*., vol. 316, pp. 487–502, Sep. 2015.
[44] D. C. Montgomery, *Design and Analysis of Experiments*. New York, NY, USA: Wiley, 2005.
[45] W. J. Conover, *Practical Nonparametric Statistics*. New York, NY, USA: Wiley, 1980.
[46] J. Derrac, S. Garcia, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as methodology for comparing evolutionary intelligence algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, 2011.
[47] S. Holm, "A simple sequentially rejective multiple test procedure," *Scand. J. Stat.*, vol. 6, no. 2, pp. 65–70, 1979.

Jun-qing Li (M'10) received the master's degree in computer science and technology from Shandong Economic University, Jinan, China, in 2004.

Since 2004, he has been with the School of Computer Science, Liaocheng University, Liaocheng, China, where he became an Associate Professor in 2008. His current research interests include intelligent optimization and scheduling. He has authored over 30 refereed papers.

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.

Since 2003, he has been with the School of Computer Science, Liaocheng University, Liaocheng, China, where he was a Full Professor in 2006. He has been with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang, China, since 2011. His current research interests include intelligent optimization and scheduling. He has authored over 200 refereed papers.

Peiyong Duan received the B.Sc. degree from Shandong University, Jinan, China, in 1996, and the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 1999.

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