A Hybrid Fruit Fly Optimization Algorithm for the Realistic Hybrid Flowshop Rescheduling Problem in Steelmaking Systems

Jun-Qing Li, Quan-Ke Pan, *Member, IEEE*, and Kun Mao

***Abstract*—In this study, we propose a hybrid fruit fly optimization algorithm (HFOA) to solve the hybrid flowshop rescheduling problem with flexible processing time in steelmaking casting systems. First, machine breakdown and processing variation disruptions are considered simultaneously in the rescheduling problem. Second, each solution is represented by a fruit fly with a well-designed solution representation. Third, two novel decoding heuristics considering the problem characteristics, which can significantly improve the solution quality, are developed. Several routing and scheduling neighborhood structures are proposed to balance the exploration and exploitation abilities. Finally, we propose an effective HFOA with well-designed smell and vision search procedures. In addition, an iterated greedy (IG) local search is embedded in the proposed algorithm to further enhance its exploitation ability. The proposed algorithm is tested on sets of instances generated from industrial data. Through comprehensive computational comparisons and statistical analyses, the performance of the proposed HFOA algorithm is favorably compared against several algorithms in terms of both solution quality and efficiency.**

***Note to Practitioners*—The steelmaking rescheduling process is critical to the effective operation of iron and steel production. This study models the steelmaking rescheduling problem with flexible processing time as a complex hybrid flowshop in which two types of disruptions, machine breakdown and processing variation, are considered concurrently. A weighted sum of the five objectives, including minimization of the average sojourn time, earliness penalty, tardiness penalty, cast-break penalty, and system instability penalty, is considered in the proposed algorithm. We develop an effective hybrid fruit fly optimization algorithm (HFOA) that applies two vectors to represent individuals and presents routing and scheduling neighborhood structures. An IG-based local search procedure is embedded to enhance the exploitation ability of the proposed algorithm. Two decoding heuristics considering the problem characteristics are developed. The effectiveness of the proposed HFOA is demonstrated through comparisons to other well-known and recently developed meta-heuristics. This work can be extended to practical problems by considering other types of disruptions. In addition, the proposed HFOA can also be generalized, and to other hybrid flowshop rescheduling problems.**

***Index Terms*—Fruit fly optimization algorithm, heuristic, hybrid flowshop, iterated greedy, rescheduling.**

I. Introduction

In modern iron and steel production systems, the scheduling of steelmaking-continuous casting (SCC) plays an important role and can increase production efficiency and profit. The classical process of SCC can be divided into three stages, i.e., steelmaking, refining, and continuous casting [1]–[12]. Based on the above partition, Xuan and Tang modeled the SCC process as a complex hybrid flowshop (HFS) problem with batch production at the last stage [5]. Unlike the classical HFS problem, SCC scheduling has many special realistic requirements that should be satisfied. Therefore, the SCC scheduling problem is harder than the classical HFS problem. The mathematical programming model for the SCC scheduling problem was proposed previously [1], [2]. Since then, many heuristic and meta-heuristic algorithms have been applied to the SCC scheduling problem, including Lagrangian relaxation [4], [5], [12], the auction-based approach [6], tabu search (TS) algorithm [10], and the discrete artificial bee colony (DABC) algorithm [11]. Many hybrid algorithms have also been investigated for the SCC scheduling problem, such as the combination of ant colony optimization (ACO) and nonlinear optimization methods [7] as well as decomposition with constraint propagation [9]. Because of its complexity, most of the literature on SCC scheduling problems has focused on scheduling in a static environment, assuming deterministic production features, such as that machines are always available, jobs arrive at the predefined time point, and the processing time for each charge is deterministic.

However, in a realistic production system, many dynamic events, such as machine breakdown, new job arrival, cancella-

Manuscript received October 18, 2014; revised March 25, 2015; accepted April 18, 2015. Date of publication May 13, 2015; date of current version April 05, 2016. This paper was recommended for publication by Associate Editor R. Uzsoy and Editor L. Shi upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation of China under Grants 61104179, 61174187, 51435009, and 61374187, the Program for New Century Excellent Talents in University under Grant NCET-13-0106, the Specialized Research Fund for the Doctoral Program of Higher Education under Grant 20130042110035, the Science Foundation of Liaoning Province in China under Grant 2013020016, the Basic Scientific Research Foundation of Northeast Universityunder Grants N110208001 and N130508001, the Starting Foundation of Northeast University under Grant 29321006, Key Laboratory Basic Research Foundation of Education Department of Liaoning Province under Grant LZ2014014, and the IAPI Fundamental Research Funds under Grant 2013ZCX02. *(Corresponding author: Quan-Ke Pan.)*

J. 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. Pan is with the State Key Lab of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: panquanke@gmail.com).

K. Mao is with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China (e-mail: mao_kun@126.com).

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/TASE.2015.2425404

tion of jobs, job processing variation, and job release variation, exist, and these events that cannot be predicted will make the initial scheduling infeasible. Consequently, rescheduling, which can repair an infeasible schedule or generate an optimal schedule after a disruption, becomes essential. Aytug *et al.* [13] classified rescheduling approaches into three groups: 1) reactive approaches; 2) robust scheduling approaches; and 3) predictive-reactive approaches. Vieira *et al.* provided a framework of strategies, policies, and methods for the rescheduling of manufacturing systems [14]. Ouelhadj and Petrovic provided a review of the state-of-the-art of dynamic scheduling via different methods, such as heuristics, meta-heuristics, multi-agent systems, and other artificial intelligence techniques [15]. Allahverdi and Mittenthal addressed the problem of minimizing the makespan in a two-machine flowshop when the machines are subject to random breakdowns [16]. Rahmani and Heydari considered a two-machine flowshop with uncertain processing times and unexpected arrivals of new jobs [17]. In recent years, heuristic and meta-heuristic algorithms have been developed to solve rescheduling problems. Tang *et al.* proposed a neural network model and algorithm to solve the dynamic hybrid flowshop scheduling problem [18]. Petrovic and Duenas utilized a genetic algorithm (GA) and presented a new fuzzy logic-based decision support system for parallel machine scheduling/rescheduling in the presence of uncertain disruptions [19]. Zandieh and Gholami proposed an immune algorithm for scheduling an HFS with sequence-dependent setup times and machines with random breakdowns [20]. Yang and Li investigated a clustering particle swarm optimizer (PSO) for dynamic optimization problems [21]. Wang and Choi presented a decomposition-based approach to solve flexible flowshop (FFS) scheduling under random machine breakdown [22]. Xuan and Li investigated the dynamic HFS scheduling problem via an improved Lagrangian relaxation (LR) and a batch decomposition strategy [23]. Rahmani *et al.* investigated the rescheduling problem in flexible flowshop problems, in which three types of disruptions are considered, i.e., new job arrival, machine breakdown, and job processing time variation [24]. Katragjini *et al.* utilized an iterated greedy (IG) algorithm to solve rescheduling problems with consideration of simultaneous disruptions in flowshop scheduling problems [25]. These works demonstrate that evolutionary or meta-heuristic algorithms are promising for solving rescheduling problems. However, most of the above-mentioned studies investigated dynamic scheduling problems in classical flowshop or HFS problems without considering various specific industrial constraints and can thus not be applied directly to realistic production systems.

Compared to the extensive research on static SCC and rescheduling in classical flowshop or hybrid flowshop scheduling problems, few studies have considered SCC rescheduling in a dynamic environment. Cowling *et al.* proposed a multi-agent architecture for the integrated dynamic scheduling of a hot strip mill (HSM) and a continuous caster [26], [27]. References [26], [27] considered continuous casters, a HSM and a Slabyard but did not consider the scheduling process in the steelmaking and refining stages. Roy *et al.* developed a knowledge model for disturbances in steelmaking systems [28]. Yu and Pan investigated the operation time delay in SCC production systems and proposed a three-stage rescheduling method that includes batch splitting (BS), forward scheduling and backward scheduling [29]. Since it does not consider machine breakdown, this rescheduling method is difficult to apply to realistic production. Tang *et al.* proposed a differential evolution (PIDE) algorithm with a real-coded matrix representation for the SCC rescheduling problem, which included a two-step method for generating the initial population and a new mutation strategy [30]. A weighed sum of four objectives was considered, and two types of disruption events, including machine breakdown and early or late arrival of jobs, were previously investigated [30]. However, to adapt to the just-in-time (JIT) rule, the earliness and tardiness penalties should be considered, and an efficient problem-specific heuristic should be developed to minimize the system penalty. Furthermore, in the above-presented literature regarding SCC rescheduling problems, none have considered rescheduling in a flexible environment, e.g., adjustable processing time for each charge, which is very common in realistic production systems of steelmaking casting horizons.

In 2012, a fruit fly optimization algorithm (FOA), which mimics the food search procedure of fruit fly swarms, was developed by Pan [31]. The applications of FOA have verified that FOA is competitive with other optimization algorithms [31], [32]. The main advantages of the canonical FOA: 1) the FOA has few parameters, which makes it easy to implement; 2) similar to other intelligent algorithms, such as GA, ACO, and PSO, FOA is an evolutionary algorithm with a parallel search framework in which many heuristics, meta-heuristics and operators can be embedded; and 3) knowledge-based or problem-specific approaches can also be easily incorporated into the search framework of the FOA to further enhance its exploitation and exploration abilities which makes it easy to apply to real-world rescheduling problems. To the best of our knowledge, there is no literature solving the steelmaking rescheduling problem using FOA. Therefore, we solve the flexible SCC rescheduling problem using an improved FOA.

In this study, we consider the SCC rescheduling problem with following specific characteristics: 1) machine breakdown and processing variation disruptions are considered simultaneously in the rescheduling problem; 2) a weighted sum of the five realistic objectives, including the average sojourn time, earliness penalty, tardiness penalty, cast-break penalty, and system instability penalty, is minimized; and 3) adjustable processing times for each charge are considered. The main contributions of this study are the following: 1) each solution is represented by a fruit fly with a well-designed solution representation; 2) two decoding heuristics considering the problem characteristics are developed to significantly improve the solution quality; 3) several routing and scheduling neighborhood structures are presented to balance the exploration and exploitation abilities; 4) well-designed smell and vision search procedures are developed; and 5) an iterated greedy (IG)-based local search is embedded in the proposed algorithm to further enhance the exploitation ability.

The remainder of this paper is organized as follows. Section II briefly describes the problem. Next, two heuristics considering the problem structures are presented in Section III.

In Section IV, the canonical FOA and IG are presented. Next, Section V gives the framework of the proposed algorithm. Section VI illustrates the experimental results and makes comparisons to the performances of algorithms from the literature to demonstrate the superiority of the proposed algorithm. Finally, Section VII presents the concluding remarks and future research directions.

II. Problem Descriptions

A. Steelmaking Rescheduling Problem

The main processes of a steelmaking production system are as follows. First, at the steelmaking stage, molten iron from the iron-making process enters a converter or electric arc furnace to reduce the undesired impurity contents. Second, at the refining stage, the charge of molten steel is transferred into a refining furnace to further eliminate impurities and add the required alloy ingredients. Third, at the continuous casting stage, the liquid steel flows down from the tundish and enters a crystallizer, where it is solidified into slabs. The practical constraint is that a set of charges must be continuously processed in the same cast.

To describe the problem, we use the following assumptions.

- In each stage, there are several identical parallel machines, which can be selected by any charge that flows through the stage.
- All charges or jobs follow the same processing sequence from the first stage to the last stage.
- In the last stage, a set of jobs are grouped into a pre-defined cast to be continuously processed in the same caster, which should not be interrupted.
- In the last stage, the setup time of a new cast is considered.
- Transfer times between stages are considered.
- Each charge or job should flow through each stage and select exactly one machine in each stage.
- The disruption data are deterministic.

To bring the algorithm closer to the industrial reality, we consider minimization of the weighted sum of the following realistic objectives: 1) F_1, the average sojourn time; 2) F_2, the earliness penalty; 3) F_3, the tardiness penalty; 4) F_4, the cast-break penalty; and 5) F_5, the system instability penalty. The sojourn time of a job is the duration between the completion time of the first stage and the starting time of the last stage. The system instability is the number of changes in machine assignment until all jobs are completed.

The notations used in this paper are given here.

Indices:

i	Index of jobs, $i = 1, 2 \ldots, n$
k	Index of machines, $k = 1, 2, \ldots, m$.
j	Index of stages, $j = 1, 2, \ldots, s$.
p	Index of casts, $p = 1, 2, \ldots, l$.

Parameters:

n	Total number of jobs.
m	Total number of machines.
s	Total number of stages.
l	Total number of casts in the last stage.
$b_{i,j}$	Starting time of charge i at stage j in the initial schedule.
$e_{i,j}$	Completion time of charge i at stage j in the initial schedule.
$p_{i,j}$	Standard processing time of job i at stage j.
$p^L_{i,j}$	Minimum processing time of job i at stage j.
$p^H_{i,j}$	Maximum processing time of job i at stage j.
PM_j	Set of parallel machines at stage j.
J	Set of n jobs, $J = \{J_1, J_2, \ldots, J_{\mathrm{n}}\}$.
J_p	$= \{o_{p-1} + 1, o_{p-1} + 2, \ldots, o_p\}$. pth cast in the continuous casting stage, where $o_{p-1}+1$, $o_{p-1}+2$, and $o_p \in J$, and $s_{(o_{p-1}+i),s} = e_{(o_{p-1}+i-1),s}$, $i = 2, \ldots, \lvert J_p \rvert$.
Ω	Set of l casts, $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_l\}$.
$T_{j,j+1}$	Transfer time from stage j to stage $j+1$.
ST_p	Setup time of cast p at the last stage.
D_p	Predefined starting time of cast p in the upper planning level.
$E_{j,k}$	Starting time of the disruption that occurs at stage j on machine k.
MB^k_d	Duration of the machine breakdown event on machine k.
pt_d	Delay length of the processing variation.
W_1	Penalty coefficient for the average sojourn time.
W_2	Penalty coefficient for earliness.
W_3	Penalty coefficient for tardiness.
W_4	Penalty coefficient for cast-break.
W_5	Penalty coefficient for scheduling instability, i.e., the number of jobs processed on different machines in the initial and revised schedules.

Decision variables

$\overline{b_{i,j}}$	Starting time of charge i at stage j after disruption in the revised schedule.
$\overline{e_{i,j}}$	Completion time of charge i at stage j after disruption in the revised schedule.
$\overline{p_{i,j}}$	Processing time of job i at stage j after disruption.
$\overline{b_p}$	Starting time of cast p after disruption.
$\overline{e_p}$	Completion time of cast p after disruption.

The aim of this study is to minimize the weighted sum of the five objectives given as follows:

$$\min f = W_1F_1 + W_2F_2 + W_3F_3 + W_4F_4 + W_5F_5$$
$$F_1 = \sum_{i=1}^{n} (\overline{b_{i,s}} - \overline{e_{i,1}})/n \quad (1)$$

$$F_2 = \sum_{p=1}^{l} \max(0, D_p - \overline{b_p}) \tag{2}$$

$$F_3 = \sum_{p=1}^{l} \max(0, \overline{b_p} - D_p) \tag{3}$$

$$F_4 = \sum_{p=1}^{l} \sum_{i=2}^{|J_p|} \left(\overline{e_{(o_{p-1}+i),s}} - \overline{b_{(o_{p-1}+i-1),s}}\right) \tag{4}$$

$$F_5 = \sum_{j=1}^{s} \sum_{i=1}^{n} u_{i,j}. \tag{5}$$

B. Computational Complexity

Sun and Wang [33] showed that parallel machine earliness and tardiness scheduling with proportional weights is NP-hard. Lee verified that parallel machine scheduling problems with machine availability constraints are NP-hard [34]. Pan *et al.* concluded that, as an extension of the HFS, the steelmaking casting problem with proportional weights for three objectives, i.e., the sojourn time, earliness and tardiness penalties, is NP-hard [11]. Because our problem is the steelmaking casting problem that considers machine breakdown and processing variation disruptions in which proportional weights of five objectives are considered, it is a generalization of the problem considered previously [33], [34], and [11], and is thus also NP-hard by restriction.

III. Solution Technologies Based on Problem-Specific Characteristics

In this study, we adopt the predictive-reactive approach to solve the rescheduling problem [13]. In the first step, we generate a predictive schedule representing the desired behavior of the shop floor over the time horizon considered. Next, the initial schedule is modified during execution in response to disruption events. Any operation in progress at the start of the disruption must be discarded, and operations starting after the end of the disruption are rescheduled.

In this section, we first introduce the operation partition approach that corresponds to the start of the disruption. To minimize the system penalties, we then introduce two efficient heuristics: 1) the cast-break minimization heuristic, which includes a processing-delay strategy and a cast-break erasing strategy, and 2) the right-shifting heuristic. The cast-break minimization heuristic is used to minimize the cast-break penalty, and the right-shifting heuristic is used to minimize the earliness/tardiness penalty.

A. Operation Partition

In the rescheduling context, the operations should first be divided into different groups according to their starting time. The detailed groups are as follows.

- G_1: the first group contains all of the operations that have started their processing before the event time point and are not affected by the disruption, i.e., $G_1 = \{O_{i,j} \,|\, (e_{i,j} \leq E_{j,k}) \vee (b_{i,j} < E_{j,k} \wedge x_{i,j,k} = 0)\}$.

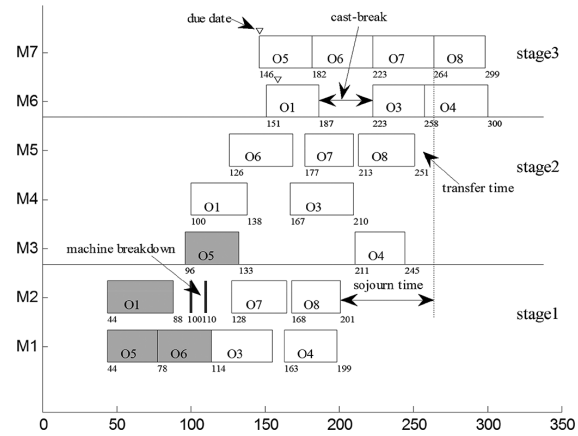


Fig. 1. Operation partition.

- G_2: the second group contains all of the operations that are affected by the machine breakdown disruption and should be erased from the current shop, i.e., $G_2 = \{\{O_{i,j} \,|\, (e_{i,j} > E_{j,k} \wedge b_{i,j} < E_{j,k} \wedge x_{i,j,k} = 1)\}, O_{i,j+1}, \ldots, O_{i,s}\}$
- G_3: the third group contains all of the operations that have not started their processing when the disruption occurs, i.e., $G_3 = \{O_{i,j} \,|\, b_{i,j} > E_{j,k}\}$.

The following example will help illustrate this complex HFS rescheduling problem. Consider an instance with eight charges and three converters in the first stage, three refining furnaces in the second stage and two casters in the last stage. Each caster processes one cast in a workday. Each cast contains four charges. Thus, $\Omega_1 = \{1, 2, 3, 4\}$, $\Omega_2 = \{5, 6, 7, 8\}$, $\mathrm{PM}_1 = \{1, 2\}$, $\mathrm{PM}_2 = \{3, 4, 5\}$, $\mathrm{PM}_3 = \{6, 7\}$. Let $T_{1,2} = 12$, $T_{2,3} = 13$, $\mathrm{ST}_1 = \mathrm{ST}_2 = 100$, and $D_1 = 158$. The Gantt chart for a solution is provided in Fig. 1.

In the Gantt chart, each charge is represented by a rectangle. The four charges filled with a gray color are those that have started their work before the given machine breakdown time point and therefore cannot be rescheduled, whereas the other operations are those that can be rescheduled. When machine breakdown occurs, charge O_2 is in process on the breakdown machine M_2 and should be deleted from the system because of the temperature constraints. Meanwhile, the following operations belonging to the charge O_2, operations $O_{2,2}$ and $O_{2,3}$, should also be deleted from the shop. All of the operations are then divided into three different groups according to the above-described partition mechanism. The first group $G_1 = \{O_{5,1}, O_{6,1}, O_{1,1}$, and $O_{5,2}\}$. The second group $G_2 = \{O_{2,1}\, O_{2,2}$ and $O_{2,3}\}$, and the third group G_3 contains all of the other operations which can be rescheduled in the system.

B. Minimize Cast-Break Via the Processing-Delay Strategy

In a realistic steelmaking casting system, the processing time of each charge at each stage is not fixed due to the consideration of many unpredictable factors. To begin the scheduling, the common heuristic method is to set the processing time of each charge to its initial standard processing time, i.e., $p_{i,j}$ for charge i at stage j. In a static environment without any disruption, each charge will be processed according to the standard processing time. However, in a dynamic environment, we should adjust the

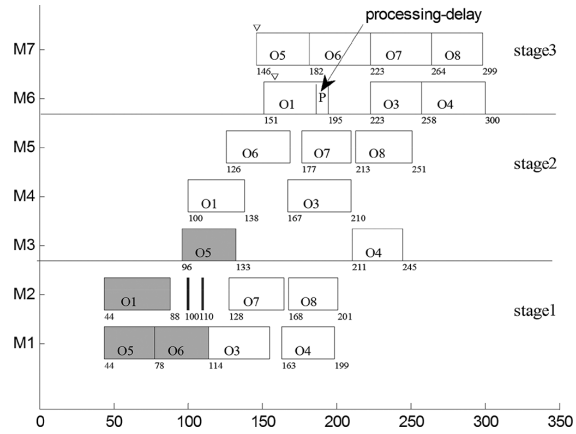


Fig. 2. Gantt chart after applying the processing-delay heuristic.

processing time of the affected charge to satisfy certain constraints. The detailed steps of the proposed processing-delay heuristic are given here.

Step 1. For each charge in the first divided group, set the processing time to the standard processing time.

Step 2. For each charge at all stages except the last stage, set the processing time to the standard processing time.

Step 3. Perform the following substeps.

a. For each cast at the last stage, find whether there exist any cast-breaks between two intermediate charges. If not satisfied, stop the heuristic; otherwise, perform step 3b.

b. For the identified cast-break, if the former charge $O_{i,s}$ has not completed its work at the disruption time, delay its processing time as long as possible, i.e., add $\min\{0.1p_{i,s}, b_{i+1,s} - e_{i,s}\}$ to the current processing time.

The above heuristic can be completed in time $O(n)$. For the example problem given in Fig. 1, the Gantt chart resulting after applying the processing-delay heuristic is given in Fig. 2. As shown in Fig. 2, the processing time of operation O_1 at the last stage has been extended to decrease the cast-break penalty. Assuming $C_1 = 10$, $C_2 = 1$, $C_3 = 10$, $C_4 = 50$, and $C_5 = 30$, the objective value is

$$\begin{aligned} F =& F_1 + F_2 + F_3 + F_4 + F_5 \\ =& 10 \times 55.875 + 1 \times 7 + 10 \times 0 + 50 \times 28 + 30 \times 0 \\ =& 1965.75. \end{aligned}$$

C. Minimize Cast-Break Via the Cast-Break Erasing Strategy

To further minimize the cast-break penalty, we propose a cast-break erasing strategy. Fig. 3 shows a Gantt chart for a steelmaking problem at the last stage. The first batch has three casting breaks, which are between operations 2 and 3, 3 and 5, and 5 and 7. If we move each operation to the right to achieve a schedule that is as compact as possible, we will decrease the

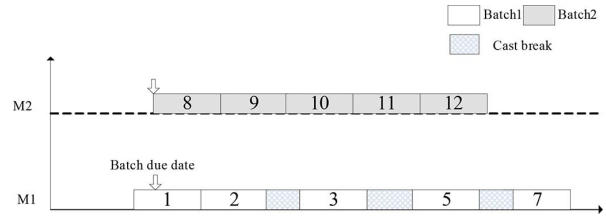


Fig. 3. Gantt chart for an example with cast-breaks.

penalty value for the cast-break and earliness. However, the average sojourn time and tardiness will increase because of the right shift. The proposed cast-break erasing strategy is given here.

1) Decide the Number of Casting Breaks h': Suppose that there are n casting breaks, which divide the current batch into $n+1$ groups. In each group q, there are G_q operations. Let S_b be the starting time of the current batch and P_b be the due date of the current batch. Let the right shift length h satisfy the following condition: $0 \le h \le \sum_{\mathrm{i}=0}^{\mathrm{n}} L_i$, where $L_0 = 0$, and L_i represents the length of the ith casting break.

If we right shift the current batch to erase h' casting breaks, where h' is an integer value, the first operation of the current batch will be shifted to the right by $\sum_{\mathrm{i}=0}^{h'} L_i$ units. We then obtain the following values.

- Decrease in the casting break penalty: $\Delta_1 = C_4 \sum_{\mathrm{i}=0}^{h'} L_i$.
- Variation of the earliness/tardiness penalty:

$$\begin{aligned} \Delta_2 = C_3\alpha\left(\sum_{\mathrm{i}=0}^{h'} L_i - \alpha(P_b - S_b, 0), 0\right) \\ - C_2\beta\left(\sum_{\mathrm{i}=0}^{h'} L_i, \alpha(P_b - S_b, 0)\right) \end{aligned} \quad (6)$$

  where $\alpha(x, y)$ and $\beta(x, y)$ represent the maximum and minimum values, respectively, of the two given variables.
- Increase in the sojourn time penalty: $\Delta_3 = C_1 \sum_{\mathrm{i}=0}^{h'} L_i (\sum_{\mathrm{j}=0}^{i} G_j)$, where $G_0 = 0$.

Then, the value of h' is the integer that yields the maximum value for the expression

$$\Delta = \Delta_1 - \Delta_2 - \Delta_3. \quad (7)$$

2) Decide the Right Shift Length: After determining the value of h', the next step is to determine the value of h, where h represents the length of the right shift. The formulas are given here.

- Decrease in the casting break penalty: $\nabla_1 = C_4 h$.
- Variation of the earliness/tardiness penalty:

$$\nabla_2 = C_3\alpha(h - \alpha(P_b - S_b, 0), 0) - C_2\beta(h, \alpha(P_b - S_b, 0)). \quad (8)$$

- Increase in the sojourn time penalty:

$$\begin{aligned} \nabla_3 =& C_1 \sum_{\mathrm{i}=0}^{h'} L_i \left(\sum_{\mathrm{j}=0}^{i} G_j\right) \\ &+ C_1 \left(\left(h - \sum_{\mathrm{i}=0}^{h'} L_i\right) \sum_{\mathrm{j}=0}^{h'+1} G_j\right). \end{aligned} \quad (9)$$

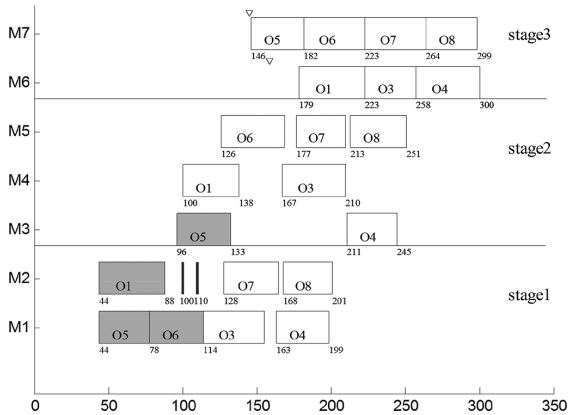


Fig. 4. Gantt chart obtained after applying the cast-break erasing strategy.

Then, the value of h is the integer that yields the maximum value for the following formula:

$$\nabla = \nabla_1 - \nabla_2 - \nabla_3 \tag{10}$$

where $\sum_{i=0}^{h'} L_i \le h \le \sum_{i=0}^{h'+1} L_i$

3) Detailed Steps: Given the Gantt chart for the considered batch, record the length of the casting breaks $L = \{L_1, L_2, \ldots, L_n\}$and the number of each group of operations $G = \{G_1, G_2, \ldots, G_n\}$.

Step 1) For each casting break from left to right, one by one, find the maximum value of h', i.e., the maximum number of casting breaks, that maximizes Δ.

Step 2) If all casting breaks have been processed, let $h = \sum_{i=0}^{h'} L_i$; otherwise, compute h such that the value of ∇ is maximized.

Step 3) The obtained objective value is

$$f_{\text{new}} = f_{\text{old}} - \nabla. \tag{11}$$

The above heuristic can be completed in time $O(n^2)$. For the given example presented in Fig. 2, we compute h' and h as follows: $h' = 1, \Delta_1 = 50 \times 28 = 1400, \Delta_2 = 10 \times 21 - 1 \times 7 = 203, \Delta_3 = 10 \times (59.375 - 55.875) = 35$ and $\Delta = \Delta_1 - \Delta_2 - \Delta_3 = 1162$.

Because only one cast-break exists in the system, we obtain $h = 36$, and $\nabla = \Delta = 1162$. We then obtain the resulting fitness as

$$f_{\text{new}} = 1965.75 - 1162 = 803.75.$$

Fig. 4 gives the Gantt chart obtained after applying the cast-break erasing strategy. The Gantt chart reveals that the schedule after applying the cast-break erasing strategy is better than the previous one.

D. Minimize the Earliness/Tardiness Penalty Via the Right-Shifting Heuristic

Here, we present a right-shifting strategy considering the disruption events to minimize the penalty of earliness, which is given as follows.

Similar to [11], consider a schedule with N batches or casts that can be rescheduled as a whole and N_E earlier casts that start earlier than the given due date times and can be rescheduled. We first rank the earlier casts in nondecreasing order of their earliness values to obtain a sequence: $\omega = (\omega(1), \omega(2), \ldots, \omega(N_E))$ with $E(1) \le E(2) \le \cdots \le E(N_E)$, where $E(1), E(2), \ldots,$ and $E(N_E)$ are the earliness values of casts $\omega(1), \omega(2), \ldots,$ and $\omega(N_E)$, respectively, and $|\omega(i)|$ represents the total operations in batch $\omega(i), 1 \le i \le N_E$.

Because repeated earliness values may exist in the above sequence, we then set a vector $\beta = (\beta(1), \beta(2), \ldots, \beta(N_U))$, where $N_U \le N_E$, to record the unique earliness values only once.

Let N_E^1 represent the length of the sequence ω without erasing any elements, i.e., $N_E^1 = \text{len}(\omega)$. Let N_E^2 denote the length of the sequence after erasing the first element with an earliness value equal to $\beta(1)$. Then, let N_E^l be the length of the sequence after erasing the first element with an earliness value less than or equal to $\beta(l-1)$.

Property 1: For the above schedule, the right shift time $q$that leads to a minimum earliness/tardiness penalty is $q = l$, where l should satisfy the following conditions: $N_E^l > (C_3 N + C_1 \sum_{i=1}^{N} |\omega(i)|/n)/(C_2 + C_3)$.

According to the property, the value of the fitness decrease is

$$\Delta\pi = \sum_{i=1}^{q} \left\{ \left(C_3 N - (C_2 + C_3) N_E^i + C_1 \times \sum_{i=1}^{N} |\omega(i)|/n)(\beta(i) - \beta(i-1) \right) \right\} \tag{12}$$

where $\beta(0) = 0$, and $N_E^0 = 0$.

Therefore, the new fitness is computed as

$$f_{\text{new}} = f_{\text{old}} - \Delta\pi \tag{13}$$

where f_{old} is the fitness value before the right-shifting, and f_{new} is the resulting fitness value.

For example, given an example steelmaking casting rescheduling problem, Fig. 5(a) gives the Gantt chart of the last stage. Among the five batches at the last stage, the first batch should maintain its starting time, whereas the following four batches can be rescheduled. Table I gives the earliness value for each cast. Therefore, we obtain $N = 4, N_E = 3, \omega = (3, 5, 4)$, and $\beta = (2, 3, 5)$. Let $C_2 = 20$, $C_3 = 5$, $C_1 = 30$, and the initial fitness value $f = 900$. We then compute N_E^l as follows:

$$\left(C_3 N + C_1 \sum_{i=1}^{N} |\omega(i)|/n \right) \Big/ (C_2 + C_3) = 1.8$$
$$l = 1, N_E^1 = \text{len}(\omega) = 3 > 1.8$$
$$l = 2, N_E^2 = \text{len}(\omega - \{3\}) = 2 > 1.8$$
$$l = 3, N_E^3 = \text{len}(\omega - \{3, 5\}) = 1 < 1.8.$$

Therefore, $q = 2$, i.e., the entire schedule with the exception of the operations belonging to the first cast (J_1 and J_2) should be right shifted twice. The first right-shift takes two time units, whereas the second right-shifting consumes one time unit. The resulting fitness value is computed as follows: $\Delta\pi = (45 - 75) \times (2 - 0) + (45 - 50) \times (3 - 2) = -65$. Therefore, the final fitness is $f = 900 - 65 = 835$.

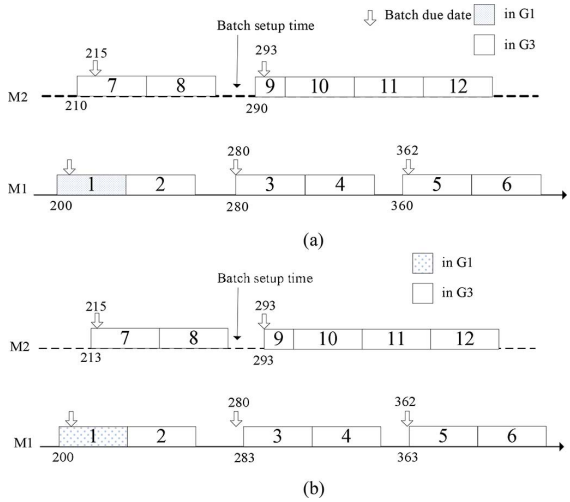


Fig. 5. Gantt chart considering the right-shifting heuristic. (a). Gantt chart without applying the right-shifting heuristic. (b). Gantt chart after applying the right-shifting heuristic.

TABLE I
EARLINESS VALUE FOR EACH CAST

No.	Cast number	Earliness value	Total operation
1	3	2	2
2	4	5	4
3	5	3	2

The above heuristic can be completed in time $O(n)$. Fig. 5(b) gives the Gantt chart for the schedule obtained after applying the proposed right-shifting approach.

IV. RELATED ALGORITHMS

A. Canonical FOA

In the canonical FOA, a certain number of fruit flies construct an initial fruit fly population. The initial population then goes through several generations, in which each fruit fly directs its search process by using two types of organs, i.e., osphresis and vision organs. The osphresis organs of fruit flies can help them find all types of scents floating in the air, whereas the vision organs help them find the food source after getting close to the food location. The main steps of the canonical FOA are as follows [31].

Step 1. Randomly initialize the positions for a swarm of fruit flies.
Step 2. For each fruit fly, perform the following steps.
Step 3. Change the current position in a random direction with a random strength.
Step 4. Estimate the distance value between its current position and the possible food source.
Step 5. Evaluate the fitness value for each fruit fly and record the best position.
Step 6. Induce the entire swarm of fruit flies to fly to the best position.
Step 7. If the stopping condition is satisfied, stop the algorithm; otherwise, return to step 2).

B. Canonical IG

The iterated greedy (IG) algorithm was proposed by Ruiz and Stützle in 2007 to solve the permutation flowshop scheduling problem [35]. Since then, IG has been utilized and extended by an increasing number of researchers, and it has been verified to be efficient for solving different scheduling problems [35]–[38]. The detailed implementation of the canonical IG is as follows.

Step 1. Randomly generate an initial solution. Set the initial parameters, such as the destruction length d, and the stop condition.
Step 2. For the current solution π, perform the following steps until the stopping condition is satisfied.
Step 3. Destruction phase. Randomly remove d operations from the current solution π, and let π' be the resulting partial solution.
Step 4. Construction phase. For each removed operation O_{i}, find the optimal position in the partial solution π' and insert O_{i} into π'.
Step 5. Check whether the newly generated solution satisfies the acceptance condition. If it is satisfied, replace the current solution with the newly generated one; otherwise, discard it.

V. PROPOSED ALGORITHM

Here, a hybrid FOA algorithm (HFOA) is proposed. The main procedures of the proposed HFOA are as follows. A solution is encoded in the form of two vectors, one of which indicates the sequence of operations at each stage and the other the choice of machines at each stage. For each vector, a number of neighborhoods are defined to allow new solutions to be generated from an existing one. During the decoding process, we apply simple forward and backward heuristics and the two problem-specific heuristics discussed in Section-III. At a given iteration of the HFOA, for each solution in the current population, the algorithm randomly generates a number of neighboring solutions using the proposed neighborhoods defined for the scheduling and routing vectors. The solution value of each of these solutions is computed, and the best of these becomes the new incumbent solution. Then, the worst solution in the current population is induced by the best fruit fly found thus far. If there is any solution that has not been updated during the last several iterations, an IG-based local search is conducted for the best solution found thus far, and the original solution is replaced with the newly generated solution.

A. Encoding

In this study, we propose a novel coding mechanism to solve the steelmaking casting rescheduling problem. Each individual is represented by two vectors. The first vector, the routing vector, contains the information of the machine assignment, whereas the second vector, the scheduling vector, represents the scheduling sequence for each operation at each stage. Given the problem shown in Fig. 1, 6 gives the two encoding vectors. The scheduling vector is illustrated in Fig. 6(a), whereas Fig. 6(b) shows the routing vector. In the scheduling

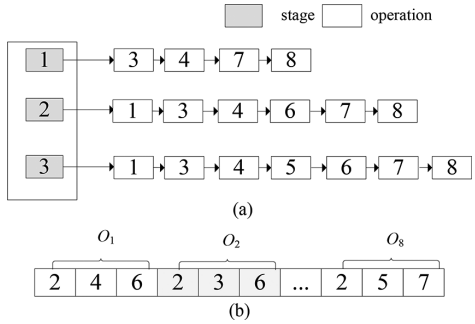


Fig. 6. Encoding representation. (a) Scheduling vector. (b) Routing vector.

vector, three stages contain all of the operations that can be rescheduled. For example, at the first stage, four operations can be rescheduled, that is, the other four operations at the first stage should be scheduled according to the initial schedule. At the second stage, six charges can be rescheduled. Then, at the last stage, all of the operations except the cancelled operation can be adjusted to minimize the objective. Moreover, the scheduling vector also indicates the scheduling sequence of the operations that should be rescheduled. For example, at the last stage, job J_5 is to be scheduled before the other three jobs, i.e., 6–8. In the routing vector, each charge or job is represented by three elements, which represent the assigned machine at the three stages, respectively. For example, for the first job O_1, the three elements are 2, 4, and 6, which means that machine M_2 is assigned to process job O_1 at the first stage, and M_4 and M_6 are selected for O_1 at the following two stages. It should be noted that, at the last stage, each charge is selected into the predefined cast, which should be processed on the determined machine. In other words, at the last stage, we cannot change the assigned machine for any operation.

B. Decoding Strategy

In this study, we consider two types of disruptions, i.e., machine breakdown and processing time variation. For the processing time variation disruption, we extend or shorten the processing time for the affected operation. For the machine breakdown, we propose a decoding process for a different group of operations. The first group contains the operations that cannot be rescheduled. Therefore, after operation partition, we should fix the starting and completion times for each operation in the first group which will maintain their current status. For the second group of operations, considering the temperature constraints in the realistic production system, the affected operation that is interrupted by a machine breakdown disruption should be deleted from the shop and subsequent operations belonging to the affected job should be ignored. In addition, in this case, the operations that are assigned to the breakdown machine have to be scheduled to start just after the restart of the machine.

To adapt to the realistic steelmaking scenario, we introduce two procedures for the decoding process, i.e., the forward heuristic and backward heuristic. The forward strategy is to schedule each charge as soon as possible to minimize several objectives, such as the maximal completion time and the tardiness penalty. However, an earlier completion time may result in a larger average sojourn time and therefore affect the quality and component of the affected charge because of the temperature constraint. Therefore, in this section, we also propose a backward strategy considering minimization of the average sojourn time. These two strategies are as follows:

1) Forward Heuristic: The forward heuristic is used to decide the starting time of each charge as quickly as possible. For the rescheduling problem, the disruption durations are known and we are given the partial Gantt chart for the operations that cannot be rescheduled, that is, we cannot change the starting time of these operations. Then, we should reschedule each operation in the third group, i.e., the operations that can be rescheduled. The detailed steps are given here.

a) Steelmaking stage: Take charge $\pi(i)$ from the scheduling vector at the first stage, one by one. Let u_k be the available time of machine m_k, where $k \in \mathrm{PM}_1$, and m_k the machine assigned to charge $\pi(i)$ in the routing vector. Then, the starting time of $\pi(i)$ without considering the disruption will be $\overline{b_{i,1}} = u_k$. If the processing of $\pi(i)$ is in conflict with the machine breakdown disruption on m_k, set the starting time of $\pi(i)$ to $\overline{b_{i,1}} = \max\{u_k, E_{j,k} + \mathrm{MB}_d^k\}$. The available time of converter k should be set to $u_k = \overline{b_{i,1}} + p_{i,1}$.

b) Refining stage: Take charge $\pi(i)$ from the scheduling vector at the second stage, one by one. Let u_k be the available time of machine m_k, where $k \in \mathrm{PM}_2$. The starting time of $\pi(i)$ without considering any disruption will be $\overline{b_{i,2}} = \max\{u_k, \overline{e_{i,1}} + T_{1,2}\}$. If the processing of $\pi(i)$ is in conflict with the machine breakdown disruption on m_k, then set the starting time to $\overline{b_{i,2}} = \max\{u_k, \overline{e_{i,1}} + T_{1,2}, E_{j,k} + \mathrm{MB}_d^k\}$. The available time of converter k should be set to $u_k = \overline{b_{i,2}} + p_{i,2}$.

c) Continuous Casting Stage: Take charge $\pi(i)$ from the scheduling vector at the last stage, one by one. Let $\gamma(i)$ be the cast including charge $\pi(i)$ and k be the predetermined machine for processing cast $\gamma(i)$. Determine the starting time for each charge $\pi(i)$ without considering the continuity of casting as follows: $\overline{b_{i,s}} = \max\{\mathrm{ST}_{\gamma(i)}, u_k, \overline{e_{i,s-1}} + T_{s-1,s}\}$, where $u_k = 0$, if $\pi(i)$ is the first charge being processed on machine k. Let $u_k = \overline{b_{i,s}} + p_{i,s}$ after $\pi(i)$ completes its processing task at the last stage on machine k.

The above heuristic can be completed in time $O(n)$. For the problem given in Section III, the Gantt chart resulting after applying the forward heuristic is given in Fig. 7. The starting times of the two casts are 158 and 146, respectively. The objective value is computed as follows:

$$\begin{aligned} F =& F_1 + F_2 + F_3 + F_4 + F_5 \\ =& 10 \times 61.75 + 1 \times 7 + 10 \times 0 + 50 \times 36 + 30 \times 0 \\ =& 2424.5. \end{aligned}$$

2) Backward Heuristic: To satisfy the casting constraints, we should adjust each charge at the last stage to be processed without gaps between other charges in the same cast. In addition, to minimize the average sojourn time, we should right shift each charge at the preceding stages to achieve a schedule that is as compact as possible. The detailed steps of the backward heuristic are given here.

a) Continuous casting stage: At the last stage, for continuous casting constraints, there is no disruption for any machine. Take charge $\pi(i)$ from the scheduling vector at the last stage one

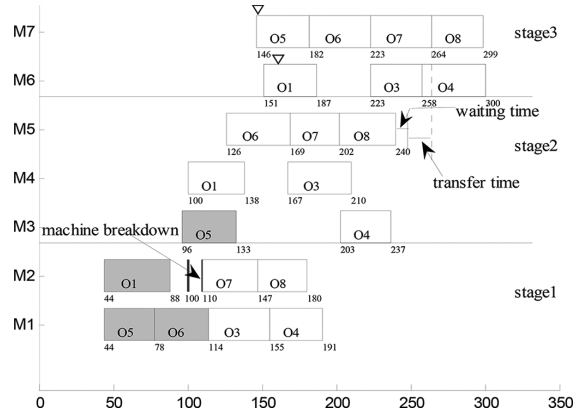


Fig. 7. Gantt chart obtained after applying the forward heuristic.

by one. Let $\gamma(i)$ be the cast including charge $\pi(i)$ and k be the predetermined machine for processing cast $\gamma(i)$. Determine the starting time for each charge $\pi(i)$ considering the continuity of casting as follows:

$$s_{i,s} = \begin{cases} (1):\ \max\left\{\mathrm{ST}_{\gamma(i)}, u_k, \overline{e_{i,s-1}} + T_{s-1,s}, \overline{b_{\gamma(i)}}\right\}, \\ \qquad -\text{if charge } i \text{ is the first job in cast } \gamma(i) \\ (2):\ \max\{u_k, \overline{e_{i,s-1}} + T_{s-1,s}\}, \text{otherwise} \end{cases} \tag{14}$$

where $u_k = 0$ if $\pi(i)$ is the first charge being processed on machine k, and $u_k = \overline{b_{i,s}} + \overline{p_{i,s}}$ after $\pi(i)$ completes its processing tasks at the last stage on machine k.

It should be noted that we cannot right shift operations that have started their work before the disruption event at the last stage. Therefore, a cast-break will occur if some operations right shift to be as compact as possible, whereas other operations in the same cast maintain their starting times. Moreover, if the next operation of charge $\pi(i)$ in the batch is cancelled, charge $\pi(i)$ should be right shifted to satisfy the continuous casting constraint. However, too much right shifting will also increase the tardiness. Therefore, during the backward procedure, we only maintain the starting time of charge $\pi(i)$ if its following operation in the same batch is canceled. To erase this type of cast-break, we use the cast-break erasing heuristic discussed in Section III.

b) Refining stage: After adjusting the starting time at the casting stage, we should determine the starting time of charge $\pi(i)$ at the refining stage without changing the assigned machine. Take charge $\pi(i)$ from the scheduling vector at the second stage one by one, from last to first. The stating time of charge $\pi(i)$ is set to $\overline{b_{i,s-1}} = \min\{u_k, \overline{b_{i,s}} - T_{s-1,s}\}$ without considering the machine breakdown event on machine k. If charge $\pi(i)$ conflicts with the machine breakdown event, we obtain the updated starting time of charge $\pi(i)$ as $\overline{b_{i,s-1}} = \min\{E_{2,k} + \mathrm{MB}_d^k, \overline{b_{i,s-1}} + p_{i,s-1}\}$.

c) Steelmaking stage: Each charge $\pi(i)$ will be rescheduled after adjusting the starting time at the refining stage. Take charge $\pi(i)$ from the scheduling vector at the first stage one by one, from last to first. If there is no conflict between $\pi(i)$ and the machine breakdown event of the assigned machine, the starting time of charge $\pi(i)$ is set to $\overline{b_{i,1}} = \min\{u_k, \overline{b_{i,2}} - T_{1,2}\}$; otherwise, the starting time of charge $\pi(i)$ is set to $\overline{b_{i,1}} = \min\{E_{1,k} + \mathrm{MB}_d^k, \overline{b_{i,1}} + p_{i,1}\}$.

The above heuristic can be completed in time $O(n)$. The Gantt chart resulting after applying the backward heuristic is given in Fig. 1, and the new objective value is

$$\begin{aligned} F =& F_1 + F_2 + F_3 + F_4 + F_5 \\ =& 10 \times 55.875 + 1 \times 7 + 10 \times 0 + 50 \times 36 + 30 \times 0 \\ =& 2365.75 \end{aligned}$$

C. Neighborhood Structures

Considering the problem characteristics of the SCC rescheduling problem, we propose two different neighborhood structure for machine assignment and operation scheduling, namely routing and scheduling neighborhood structure, respectively.

1) Routing Neighborhood Structure: The aim of the routing neighborhood structure is to give a different machine assignment to the current individual. The mutation neighborhood is commonly used in the current literature to change another available machine for a randomly selected operation [30], [39].

To consider both the exploitation and the exploration capabilities of the proposed algorithm, we present an efficient routing neighborhood structure that combines the mutation neighborhood and a developed crossover neighborhood.

Let $\mathrm{P^B}$ denote the population that contains the solutions with the best fitness value. $\mathrm{P^B}$ is initiated as empty. After each generation, the best solution s_b found thus far will update $\mathrm{P^B}$ as follows: 1) if s_b is better than the solutions in $\mathrm{P^B}$, empty $\mathrm{P^B}$ and insert s_b into $\mathrm{P^B}$ and 2) if the fitness value of s_b is the same as the solutions in $\mathrm{P^B}$, then check whether s_b contains an objective value that is different from those of the solutions in $\mathrm{P^B}$. If both of the above two conditions are satisfied, insert s_b into $\mathrm{P^B}$. For example, given a solution s_b with the fitness value $f = 13700$ and the objectives values $f_1 = 1252$, $f_2 = 10$, $f_3 = 3$, $f_4 = 15$, and $f_5 = 13$, we denote $s_\mathrm{b} = \{13700, 1252, 10, 3, 15, 13\}$. If the current $\mathrm{P^B} = \{s_1 : \{13700, 1253, 10, 2, 15, 13\}, s_2 = \{13700, 1253, 20, 1, 15, 13\}\}$, we should insert s_b into $\mathrm{P^B}$.

After performing the above-mentioned steps, $\mathrm{P^B}$ will contain the best solutions with different objective values for the considered five objectives. Therefore, both the exploitation and exploration abilities will be maintained. With population $\mathrm{P^B}$, the detailed steps are as follows.

Step 1. Set the two parameters p_l and $l_{\max}$, where p_l represents the probability to learn from the best solution found thus far, and $l_{\max}$ is the maximum learning strength.

Step 2. Evenly divide the entire evolution stage into $l_{\max}$ parts.

Step 3. During the evolution, perform the following steps.

- a) If the evolution time follows into the ith part, then set the current learning strength l_s to $(l_{\max} - i - 1)$.
- b) For each solution s_i, perform the mutation neighborhood.
- c) For each solution s_i, generate a random number r. If $r \le p_l$, perform step 3d); otherwise, perform step 3e).

d) Perform the following steps l_s times:
   i. Randomly select one stage except the last stage.
   ii. At the selected stage, randomly select one schedulable operation.
   iii. Randomly select a best solution s_{b} in P^{B}.
   iv. Set the machine assignment the same as that in s_{b} for the selected operation in s_{i}.
e) Go back to step 3).

2) Scheduling Neighborhood Structure: To solve the hybrid flowshop scheduling problem, neighborhood structures, such as insertion, swap, pairwise exchange, and multi-swap, are commonly used in the literature. For the static scheduling problem, Pan *et al.* verified that the multi-swap neighborhood can be evaluated more efficiently than the other three neighborhood structures [11]. The second best is the insertion neighborhood structure. However, in the rescheduling problem, one of the objectives is to minimize the system instability. Therefore, the above-discussed neighborhood structures should play different roles in the rescheduling problem.

In this study, considering the problem structure and the balance of the exploration and exploitation abilities, we combine the two neighborhood structures, i.e., the multi-swap and insertion neighborhoods, and present a random selection approach. In other words, in each generation, we randomly select one of the two scheduling neighborhood structures to generate neighboring solutions.

D. Population Initialization

In the SCC rescheduling problem, each solution has many evaluation criteria, and a balance should be created between the solution quality and the system stability. Therefore, to generate a population with a high level of solution quality and diversity, we propose a very simple population initialization heuristic, which is given below.

Step 1. Divide each charge into different groups at the disruption time point.
Step 2. Let counter $\mathrm{Cnt} = 1$, and perform the following steps until $\mathrm{Cnt} = \mathrm{PS}$.
Step 3. Generate a solution based on the given initial schedule in a random manner and evaluate it. If the newly generated solution is not the same as any individual in the current population, insert it into the population and let $\mathrm{Cnt} = \mathrm{Cnt} + 1$; otherwise, discard it.
Step 4. Return to step 3.

E. Smell-Based Search Procedure

In FOA, each fruit fly performs exploitation tasks by utilizing a smell-based search procedure. Therefore, the efficiency of the smell-based search procedure is crucial for FOA. In this study, the smell-based search procedure for the rescheduling problem is implemented as follows.

Step 1. Set parameter *SN*, which is the size of the neighboring space for each fruit fly to be exploited. In other words, the *SN* value decides the search strength for each fruit fly.
Step 2. For each fruit fly S_{i}, perform the following steps *SN* times.
Step 3. Generate a neighboring fruit fly around the given individual S_{i} by applying the proposed routing and scheduling neighborhood structures, respectively.
Step 4. Store the newly generated neighboring individual in a new vector named T_{i}.

F. Vision -Based Search Procedure

To enhance the search ability of the entire swarm, in canonical FOA, the vision-based search procedure is utilized to induce the entire population to a better search space. To consider the balance of the exploitation and exploration abilities, we develop an improved vision-based search procedure, which can both improve the performance of the current population and retain the diversity of the entire swarm. The detailed steps are as follows.

Step 1. For each fruit fly S_{i}, perform steps 2 and 3.
Step 2. Evaluate all of the neighboring solutions in the neighboring solution set T_{i}.
Step 3. If the best neighboring solution with the minimum fitness value (N_{i}) is better than the current individual, replace the latter with the former.
Step 4. For the entire population, perform steps 5–7.
Step 5. Sort the entire population in non-decreasing order according to their fitness values.
Step 6. For the best fruit fly found thus far, perform the insertion neighborhood several times to generate a new fruit fly named S_{j}.
Step 7. Find the worst fruit fly in the current population and replace it with S_{j}.

G. IG-Based Local Search Procedure

In this study, to enhance the search ability of the algorithm, we embed an IG-based local search in the proposed algorithm. The detailed implementation of the IG-based local search is as follows.

Step 1. Set parameter *LEN*, which is the destruction length of the IG-based local search.
Step 2. For the given fruit fly S_{i}, perform the following steps.
Step 3. In the destruction phase, randomly select *LEN* operations and delete them from the scheduling vector of S_{i}. Next, store the deleted operations in a vector denoted T_{d}.
Step 4. In the construction phase, fetch each operation in vector T_{d} one by one, and find the optimal position for the selected operation until T_{d} is empty.

Step 5. Evaluate the newly generated neighboring individual, and update the given solution S_{i} if the former is better than the latter. Moreover, update the best solution found thus far.

H. Exploration Procedure

To avoid becoming stuck in a local optimum, we develop an exploration procedure in the proposed HFOA, which is given below.

Step 1. For each fruit fly S_{i}, record the update iteration number (*UIN*).
Step 2. In each generation, set *UIN* for each fruit fly as follows: If the fruit fly is updated by a newly generated neighboring individual, set *UIN* to zero; otherwise, increase it.
Step 3. Sort the entire population in non-increasing order according to its *UIN*.
Step 4. Find the fruit fly with a *UIN* greater than L_{n}, and denote it S_{u}. If more than one fruit fly satisfies that condition, randomly select one.
Step 5. For the best fruit fly found thus far, perform the IG-based local search discussed above. Replace S_{u} with the newly generated fruit fly.

I. Flowchart of the Proposed Algorithm

The detailed flowchart of the proposed HFOA algorithm is provided in Fig. 8.

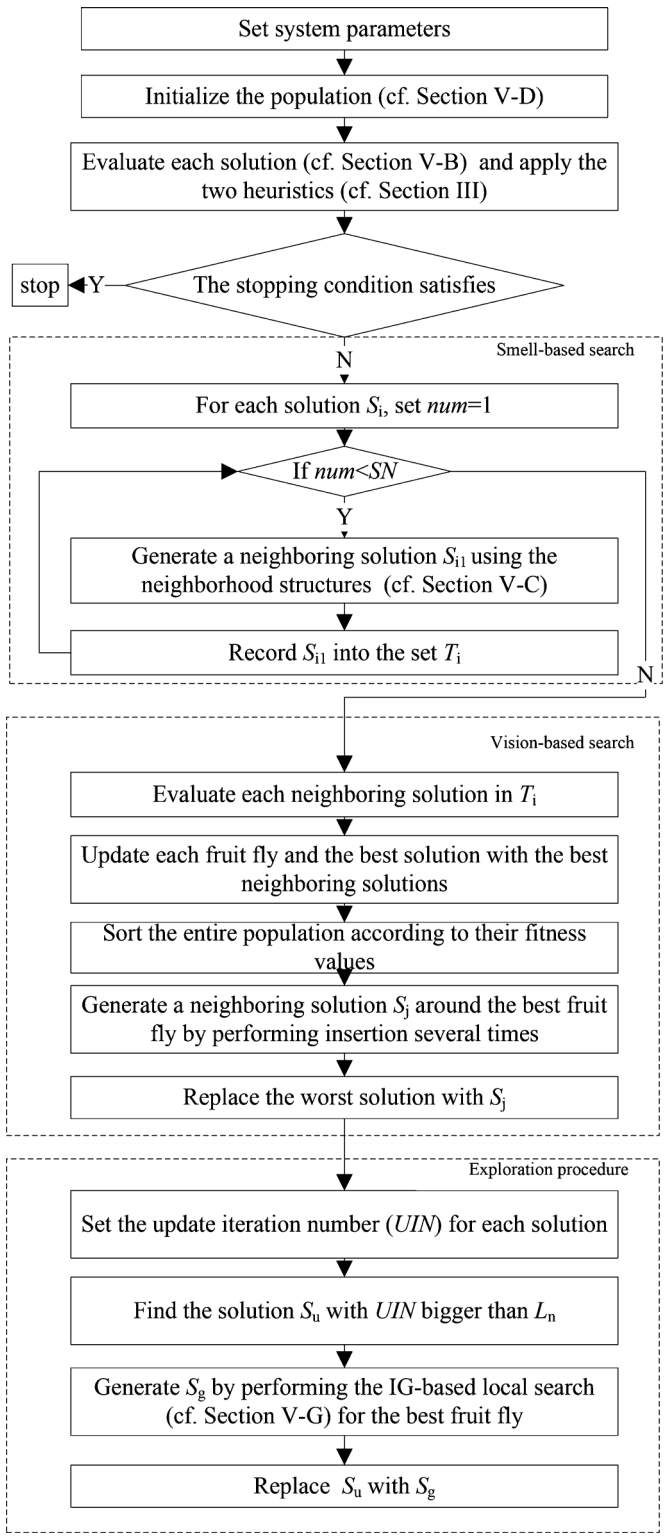


Fig. 8. Flowchart of the proposed algorithm.

VI. 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. The algorithms compared include two previously developed algorithms, i.e., DABC [11]and PIDE [30]. It should be noted that PIDE is a very recently published and efficient algorithm that is used for solving the steelmaking casting rescheduling problem, and DABC is also efficient for solving the steelmaking casting problem. To make a fair comparison on the same test instance, we code the above algorithms. All algorithms use the same maximum elapsed CPU time limit of $t = 100$ s as a termination criterion. The best results of the experiments for the 120 randomly generated problems from 30 independent runs were collected for performance comparisons. The Taguchi method of DOE [41] is utilized to test the influence of the key parameters on the performance of the proposed algorithm.

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

$$\mathrm{RPI(C)} = \frac{C_c - C_b}{C_b} \times 100 \tag{15}$$

where C_b is the best solution found by all of the compared algorithms, whereas C_c is the best solution generated by a given algorithm.

A. Experimental Instances

In this study, we generate 15 problem instances based on 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.

TABLE II
COMBINATIONS OF DIFFERENT EVENTS

Event type	T	M	V
E1	1	1	2
E2	1	1	3
E3	1	2	1
E4	1	3	1
E5	2	1	2
E6	2	1	3
E7	2	2	1
E8	2	3	1

- There are three converters, five refining furnaces, and four continuous casters in the shop. In each workday, a continuous caster can process three or four casts. Each cast generally contains 8–12 charges or jobs. Therefore, in each workday, we consider 12–16 casts and a total of approximately 120 charges in the scheduling. For each charge or job, the processing time is randomly generated in the ranges of [38], [40], [36], [50], and [38], [44] for the three stages considered, respectively;
- For each machine, the release time is not considered as a technical capability;
- The transfer times for each set of two consecutive stages are in the range of [10]–[15];
- The setup time for each cast is set to 100;
- The practical minimum and maximum processing time of job i at stage j, i.e., $p_{i,j}^{L}$ and $p_{i,j}^{H}$ are set to $0.9p_{i,j}$ and $1.1p_{i,j}$, respectively;
- The predefined starting time of the first cast on each caster in the continuous stage can be estimated by the sum of the processing time and transfer time of each charge related to the cast;
- Two levels of event time points: (1) T1, at the first stage, randomly select a machine M_i, and randomly generate a machine breakdown disruption at a time point equal to 30% of the machine completion time on M_i; (2) T2, at the penultimate stage, randomly select a machine M_i, and randomly generate a machine breakdown disruption at a time point equal to 70% of the machine completion time on M_i;
- Three levels of machine breakdown durations: 0%, 3%, and 6% of the maximal completion time of the breakdown machine;
- Three levels of processing variations: 0%, 10%, and 20% of the standard processing time of the affected charge.

Therefore, we obtain a total of 120 test instances with eight different combinations of disruptions for realistic steelmaking rescheduling problems, which are given in Table II. In Table II, the columns labeled "T", "M" and "V" represent the levels of the event time point, machine breakdown duration, and processing variation, respectively.

B. Experimental Parameters

To make a fair parameter tuning, we randomly generate a separate and independent data set, with 120 instances ranging from 20 jobs to 120 jobs. According to [40] and our preliminary experiments, the number of iterations during which the solution does not improve (L_{n}) is also set to 20. The remaining five parameters are the population size (PS), the size of the neighborhood in the smell-based search process (*SN*) for the HFOA procedure, the probability to learn from the best solution (p_l), the maximum learning strength $(l_{\max})$, and the destruction length (LEN) for the IG-based local search procedure. According to our preliminary experiments, the levels of the five parameters are presented in Table III, where n represents the number of charges in the system. The Taguchi method of DOE [41] is utilized to test the influence of these five parameters on the performance of the proposed algorithm.

TABLE III
COMBINATIONS OF KEY PARAMETER VALUES

Parameter	Level			
	1	2	3	4
PS	10	30	50	100
SN	*n*/20	*n*/10	*n*/5	*n*/2
p_l	0.1	0.3	0.5	0.7
l_{max}	6	5	4	3
LEN	1	2	3	4

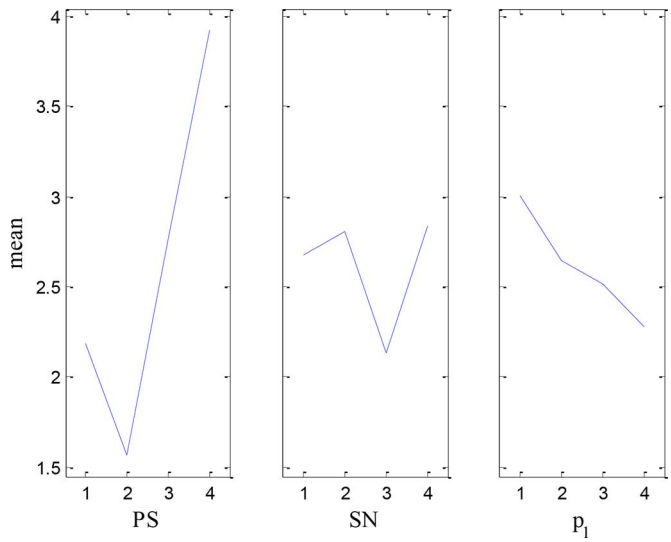


Fig. 9. Factor level trend of the first three key parameters.

For the first three parameters, an orthogonal array $\mathrm{L}_{16}(4^3)$ is performed. For each parameter combination, the proposed algorithm is run 30 times independently, and the average RPI value obtained by the proposed algorithm is then collected as the response variable (RV). Fig. 9 reports the factor level trend of the three parameters.

It can be observed from Fig. 9 that the proposed algorithm exhibits better performance under the three parameters with the following levels: PS with level 2, *SN* with level 3, and p_l with level 4. Fig. 9 also shows that parameter PS is more critical than the other two parameters in the proposed algorithm. A large value of PS means that more computational resources are consumed in the exploration procedure and that the algorithm will

lose its exploitation ability. An overly small value of PS indicates a loss of the exploration ability of the algorithm. Therefore, to balance the exploration and exploitation abilities, a suitable value for the key parameter PS is set to 30 in the proposed HFOA. According to the above-described analysis, the suitable values for the three considered parameters, namely PS, *SN* and p_l, are set to 30, $n/5$ and 0.7, respectively.

Similar experiments are also carried out to optimize the parameters for other parameters that are used for comparison purposes. Their desirable parameter settings based on the experimental results are the following: $l_{\max} = 5$ and $\text{LEN} = 3$.

The Taguchi method of DOE is also utilized to perform extensive experiments to tune the parameters of the compared algorithms. According to our preliminary experiments, the parameters for the compared algorithms are as follows: 1) for the canonical GA in Section VI-F, the population size (P_{Size}) is set to 20, 0.05 for the mutation probability (P_{m}), and 0.3 for the crossover probability (P_{c}); 2) for the canonical TS in Section VI-F, the tabu list length and the tabu tenure are both set to $(n \times m)/5$, where m represents the total number of machines in the system; 50 and 10 for t_{int} and t_{div}, respectively, which are the number of iterations of TS used in the intensification and diversification phase, respectively; 3) for DABC, the population size (PS) is set to 50, the number of employed bees or onlookers is equal to PS, and the number of trials that a food source is assumed to be abandoned (*limit*) is set to 100; and 4) for PIDE, the population size (P_{Size}) is set to 50, 0.95 for the mutation probability (P_{m}), and 0.5 for the crossover probability (P_{c}).

C. Effectiveness of the Proposed Problem-Specific Heuristics

Here, to examine the effectiveness of the proposed two heuristics, i.e., the cast-break erasing and right-shifting approaches, we make detailed comparisons of the presented heuristics in Table IV, where H-I represents the HFOA without any problem-specific heuristics discussed above, H-II the HFOA with the right-shifting heuristic, H-III the HFOA with the cast-break erasing heuristic, and H-IV the HFOA with both proposed heuristics. Each combination is independently run on the same PC mentioned above with the 120 test instances.

It can be observed from Table IV that: 1) H-IV obtained 14 improved values out of the given 15 instances, whereas H-II and H-III can obtain only one improved value each; 2) H-I is the worst one among the compared algorithms and presents approximately 78-fold greater performance than H-IV; 3) the last row in the table shows that H-IV performs the best and exhibits approximately 14-fold lower performance than the second-best algorithm H-III; and 4) in a nutshell, we can obtain better results by concurrently applying the two proposed heuristics.

We also collect the CPU times consumed by each heuristic to find the results listed in the comparison table. The average CPU times for H-I, H-II, H-III and H-IV are 34.97, 40.12, 46.43, 50.64 s, respectively. From the CPU time comparisons, we can see that the differences in the computational time is not very obvious between H-III and H-IV, but H-IV obtains an average RPI value which is approximately 150-fold lower than H-III. The comparisons in the CPU time verify the efficiency of the combination of the proposed problem-specific heuristics.

TABLE IV
COMPARISONS OF THE RPI VALUES FOR PROBLEM-SPECIFIC HEURISTICS

Instance	H-I	H-II	H-III	H-IV
Case1	5.24	0.44	3.92	0.00
Case2	5.49	4.74	1.82	0.00
Case3	5.68	4.62	0.57	0.00
Case4	5.45	1.08	1.27	0.00
Case5	15.94	7.98	0.43	0.00
Case6	9.42	0.42	0.00	0.08
Case7	5.87	2.02	2.12	0.00
Case8	17.30	1.15	7.07	0.00
Case9	7.66	4.59	0.37	0.00
Case10	10.76	3.67	0.28	0.00
Case11	1.75	0.00	0.41	0.00
Case12	8.04	7.62	0.89	0.00
Case13	8.66	8.03	0.29	0.00
Case14	7.49	7.26	1.31	0.00
Case15	2.78	1.17	1.30	0.00
Mean	7.84	3.65	1.47	0.01

To further investigate the primary effectiveness of the two heuristics, a multiple comparisons test known as the Friedman test [42], [43] is performed. The Friedman test is a non-parametric statistical test that can be used to identify whether there are significant differences between different algorithms. The *p*-value obtained by the Friedman test provides information about whether a statistical hypothesis test is significant or not. The smaller the *p*-value is, the stronger the evidence against H0 [43]. That is, a smaller *p*-value means that there is a more significant difference between the compared algorithms.

After applying the Friedman test, we obtained a *p*-value of 2.2849e-08, which is very near zero and demonstrates that the results of the different algorithms are significantly different. However, the main drawback of the Friedman test is that it only tells whether there are significant differences over the entire algorithms compared, but it cannot determine the results of a pairwise comparison. In this study, we also apply the Holm multiple comparison test [44] as a *post hoc* procedure for the pairwise comparisons. Fig. 10(a) shows the multiple comparison results, which demonstrates that H-IV is significantly different from the other three compared heuristics.

In summary, H-IV is effective for the problem under consideration, and it is necessary to take advantage of the problem-specific characteristics for developing a solution technology for the rescheduling problem.

D. Effectiveness of the Proposed Neighboring Structures

Here, to assess the efficiency of the proposed neighborhood structures, we perform three independent experiments for comparisons of the routing and scheduling neighborhood structures discussed in Section V.

1) Comparisons of Different Scheduling Neighborhood Structures: To check the efficiency of the proposed scheduling neighborhood structures, we code and test three types of

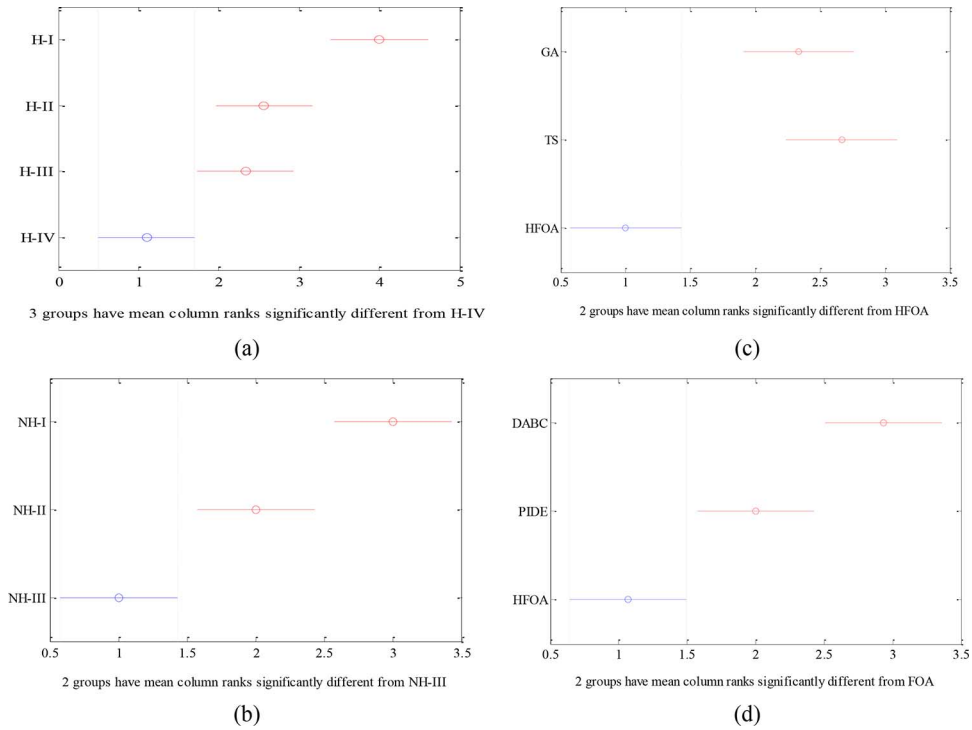


Fig. 10. Multi-comparison results.

scheduling neighborhood structures for the given considered problems, i.e., multi-swap, insertion and our proposed method. For each compared algorithm, 30 independent runs are performed for each instance. The experimental results show that our proposed method exhibits better performance than the other two compared neighborhood structures.

The advantages of the proposed scheduling neighborhood structures are as follows. First, by using the multi-swap approach, the algorithm can better perform the exploitation task. Second, by using the insertion method, the algorithm can better escape from local optima.

2) Comparisons of Different Routing Neighborhood Structures: To assess the efficiency of the proposed routing neighborhood structures, we also code and evaluate the mutation and single-point crossover (SPC) methods.

It can be concluded from the experimental results that our proposed method exhibits better performance than the other two compared neighborhood structures. The main reasons for this finding are the following: 1) during the earlier evolution stages, the proposed routing neighborhood achieves more learning from the best solution, which enhances the convergence ability of the proposed algorithm; 2) during the later evolution stages, the learning strength will decrease, which may aid the algorithm avoid becoming stuck in local optima; and 3) the application of the mutation neighborhood makes the proposed heuristic generate different solutions.

3) Combination of Different Neighboring Structures: Table V gives the comparison results of the RPI values for combinations of different neighboring approaches. The different combinations are given here:

- NH-I, the neighboring approach that only contains the proposed routing neighborhood structure;
- NH-II, the neighboring approach that only considers the scheduling neighborhood;
- NH-III, the neighboring approach that considers both the routing and scheduling neighborhoods.

TABLE V
COMPARISONS OF THE RPI VALUES FOR NEIGHBORHOOD COMBINATIONS

Instance	NH-I	NH-II	NH-III
Case1	9.73	3.78	0.00
Case2	9.48	3.23	0.00
Case3	9.93	0.62	0.00
Case4	18.21	3.89	0.00
Case5	32.88	2.07	0.00
Case6	25.17	4.40	0.00
Case7	14.48	1.94	0.00
Case8	40.36	6.28	0.00
Case9	25.62	4.68	0.00
Case10	28.13	10.34	0.00
Case11	4.04	2.18	0.00
Case12	30.23	6.23	0.00
Case13	38.64	4.62	0.00
Case14	16.78	8.54	0.00
Case15	23.03	0.35	0.00
Mean	21.78	4.21	0.00

The CPU times consumed by NH-I, NH-II, and NH-III are 15.31, 42.91, and 50.64 s, respectively. It can be observed from Table V that: 1) NH-II leads to greater benefit than NH-I, the main reason is that a new solution is generated by changing

TABLE VI
COMPARISONS OF THE RPI VALUES FOR THE ENHANCED STRATEGIES

Instance	$HFOA_{NH}$	HFOA
Case1	0.14	0.00
Case2	2.54	0.00
Case3	0.10	0.00
Case4	2.18	0.00
Case5	0.65	0.00
Case6	1.21	0.00
Case7	0.97	0.00
Case8	0.00	0.72
Case9	3.54	0.00
Case10	6.15	0.00
Case11	0.98	0.00
Case12	1.33	0.00
Case13	4.01	0.00
Case14	6.72	0.00
Case15	0.00	0.28
Mean	2.03	0.07

TABLE VII
COMPARISONS OF THE RPI VALUES FOR GA, TS, AND HFOA

Instance	GA	TS	HFOA
Case1	4.56	2.67	0.00
Case2	2.54	0.38	0.00
Case3	3.30	2.08	0.00
Case4	7.45	4.69	0.00
Case5	2.26	4.24	0.00
Case6	1.62	3.15	0.00
Case7	2.38	8.06	0.00
Case8	2.67	10.89	0.00
Case9	1.32	6.63	0.00
Case10	2.30	4.85	0.00
Case11	1.83	9.96	0.00
Case12	4.66	10.34	0.00
Case13	14.03	10.78	0.00
Case14	6.90	9.23	0.00
Case15	3.16	7.17	0.00
Mean	4.06	6.34	0.00

another available machine for a selected operation in NH-I, whereas NH-II changes the scheduling order to generate a new solution, which makes NH-II have a wider searching space than NH-I, and 2) NH-III exhibits the best performance and enables both the routing and scheduling neighborhoods. Therefore, in this study, by applying the NH-III method to combine the routing and scheduling neighborhood structures, our proposed algorithm may allow the identification of more improved results after the disruption.

The resulting p-value of the Friedman test is 3.0590e-07, which indicates that there are significant differences among the three compared algorithms. As shown in Fig. 10(b) the neighboring approach that considers both the routing and scheduling neighborhoods is significantly better than the other two approaches.

E. Effectiveness of the Enhanced Strategies in HFOA

To investigate the effectiveness of the enhanced strategies, we implement the HFOA algorithm presented in Section V and the HFOA algorithm without the enhanced IG-based local search strategies ($HFOA_{NH}$ for short). The parameters for the two compared algorithms are set to the same values as in Section VI-B. The only difference between HFOA and $HFOA_{NH}$ is that the HFOA algorithm embeds the IG-based local search strategy during the exploration procedure. The two algorithms are tested on the same PC and with the same test instances. After 30 independent runs, the average RPI results for each instance with different event types are collected for comparison, which is given in Table VI.

It can be observed from Table VI that: 1) to solve the 15 given instances, the proposed HFOA with the IG-based local search algorithm yielded 13 improved values, whereas $HFOA_{NH}$ only found improved values for Cases 8 and 15; and 2) HFOA obtained an average RPI value of 0.07, which is approximately three-fold lower than the result obtained with $HFOA_{NH}$.

The resulting p-value of the Friedman test is 0.0045, which indicates that there are significant differences between the two compared algorithms.

F. Comparisons With the Canonical Algorithms

To make a fair comparison between the proposed algorithm and the other meta-heuristics, such as GA and TS, we code GA and TS algorithms. Table VII gives the comparison results for the three compared algorithms.

For GA and TS algorithms, we use the same encoding and decoding approaches as in the HFOA algorithm. For GA, we adopt the single-point crossover, swap mutation operators as in [45]. For TS, we adopt the swap and mutation neighborhoods as in [46]. The two problem-specific heuristics discussed in Section III are also embedded in the two algorithms.

It can be concluded from Table VII that our proposed method shows better performance than the canonical GA and TS algorithms. Compared with GA and TS algorithms, the main advantages of the proposed HFOA are as follows: 1) the proposed routing and scheduling neighborhood structures enhance the exploitation and exploration capabilities; 2) the proposed smell-search and vision-search procedures increase the search ability; 3) the proposed exploration procedure enhances the exploration ability; and 4) the proposed IG-based local search procedure further enhances the exploitation ability.

The resulting p-value of the Friedman test is 8.5749e-06, which indicates that there are significant differences between the three compared algorithms. The average CPU times consumed by GA, TS, and HFOA are 82.42, 79.77, and 50.64 s, respectively. It can be concluded from the comparison results and

TABLE VIII
COMPARISONS OF THE RPI VALUES WITH A CPU TIME OF 30 S

Instance	Scale	DABC	PIDE	HFOA
Case1		27.05	21.74	**0.97**
Case2	20-jobs	18.28	6.95	**0.00**
Case3		23.14	13.82	**1.17**
Case4		53.93	6.11	**0.00**
Case5	60-jobs	15.98	5.56	**0.00**
Case6		33.21	**0.26**	9.15
Case7		10.40	6.49	**1.19**
Case8	80-jobs	6.23	16.98	**0.00**
Case9		19.73	3.80	**0.31**
Case10		28.75	16.79	**0.82**
Case11	100-jobs	12.44	5.64	**2.22**
Case12		26.84	3.74	**1.13**
Case13		95.23	**2.66**	11.37
Case14	120-jobs	102.33	34.90	**0.00**
Case15		24.03	24.49	**0.73**
Mean		33.17	11.33	**1.94**

TABLE IX
COMPARISONS OF THE RPI VALUES WITH A CPU TIME OF 100 S

Instance	Scale	DABC	PIDE	HFOA
Case1		26.00	21.59	**0.00**
Case2	20-jobs	17.86	6.95	**0.00**
Case3		21.23	12.95	**0.00**
Case4		51.77	2.88	**0.00**
Case5	60-jobs	14.47	4.50	**0.00**
Case6		32.35	0.26	**0.00**
Case7		9.61	6.01	**0.00**
Case8	80-jobs	5.29	13.75	**0.00**
Case9		18.92	2.68	**0.00**
Case10		25.53	15.66	**0.00**
Case11	100-jobs	4.94	2.66	**0.00**
Case12		26.53	2.23	**0.00**
Case13		95.06	**0.00**	1.36
Case14	120-jobs	85.93	32.79	**0.00**
Case15		23.35	22.85	**0.00**
Mean		30.59	9.85	**0.09**

Fig. 10(c) that the proposed HFOA algorithm is significantly better than the canonical GA and TS algorithms.

G. Comparisons with the Presented Efficient Algorithms

To make a fair comparison between the proposed algorithm and two other recently published algorithms for steelmaking casting problems, i.e., DABC and PIDE, we code the above-described two algorithms used for comparison purposes in solving our SCC rescheduling problems and run them on the same PC with the same stop conditions. For each compared algorithm, the decoding strategy with forward and backward heuristics as discussed in Section V-B is embedded. To make a fair and extensive comparison, all of the algorithms adopt the same population initialization method. The two problem-specific heuristics discussed in Section III are not embedded in DABC and PIDE. Each compared algorithm is run 30 independent times for each given instance. All algorithms adopt the same maximum elapsed CPU time limit of $t = 30$ and $t = 100$ s as a termination criterion. This criterion is practical in realistic production systems. The comparison results of RPI values at $t = 30$ and $t = 100$ s for the three compared algorithms are reported in Tables VIII and IX, respectively. In the two tables, we use the best solution found by any algorithm at $t = 100$ s as the reference value C_b when computing the RPI values.

It can be observed from Table VIII that: 1) at the time limit of $t = 30$, the proposed HFOA yielded 13 improved values out of the 15 given instances, whereas the second-best algorithm PIDE, obtained two improved solutions; 2) on average, the proposed HFOA obtained an RPI value of 1.94, which is almost sixfold less than that of the PIDE algorithm, which was the second-best performer, with an overall average RPI of 11.33; and 3) the comparison results show the efficiency and robustness of the proposed HFOA within a short computational time.

Table IX gives the comparison results at $t = 100$ s. The average CPU times consumed by DABC, PIDE, and HFOA are 37.10, 75.83, and 50.64 s, respectively. It can be observed from Table IX that: 1) the proposed HFOA obtained 14 improved values out of the 15 given instances, indicating that it is better than the other algorithms; 2) from the comparison of the average performance presented in the last row, we can see that the proposed HFOA obtained an RPI value of 0.09, which is almost 110-fold smaller than that of the second-best algorithm, PIDE; and 3) the comparison result with relatively longer CPU times further verifies the efficiency of the proposed HFOA.

To assess whether the observed differences from the above two tables are indeed significantly different, we carry out the Friedman test and the Holm multiple comparison test as a post hoc procedure for the pairwise comparison. The resulting p-values for Tables VIII and IX are 1.2792e-05 and 2.1146e-06, respectively. Fig. 10(d) presents the pairwise comparison results after applying the Holm multiple comparison tests for the comparison of the results presented in Table IX. It can be concluded from Fig. 10(d) that the proposed HFOA algorithm is significantly better than DABC and PIDE.

Compared with PIDE, the main advantages of HFOA are the following: 1) the two problem-specific heuristics discussed in Section III improve the solution quality; 2) the smell and vision search processes enhance the exploitation ability; 3) the exploration procedure improves the exploration ability; and 4) the IG-based local search further enhances the exploitation ability. The same advantages of HFOA are also found in the comparison with DABC. Another advantage of HFOA compared with DABC is that the proposed encoding and decoding mechanism affords HFOA the ability to search more promising locations than DABC.

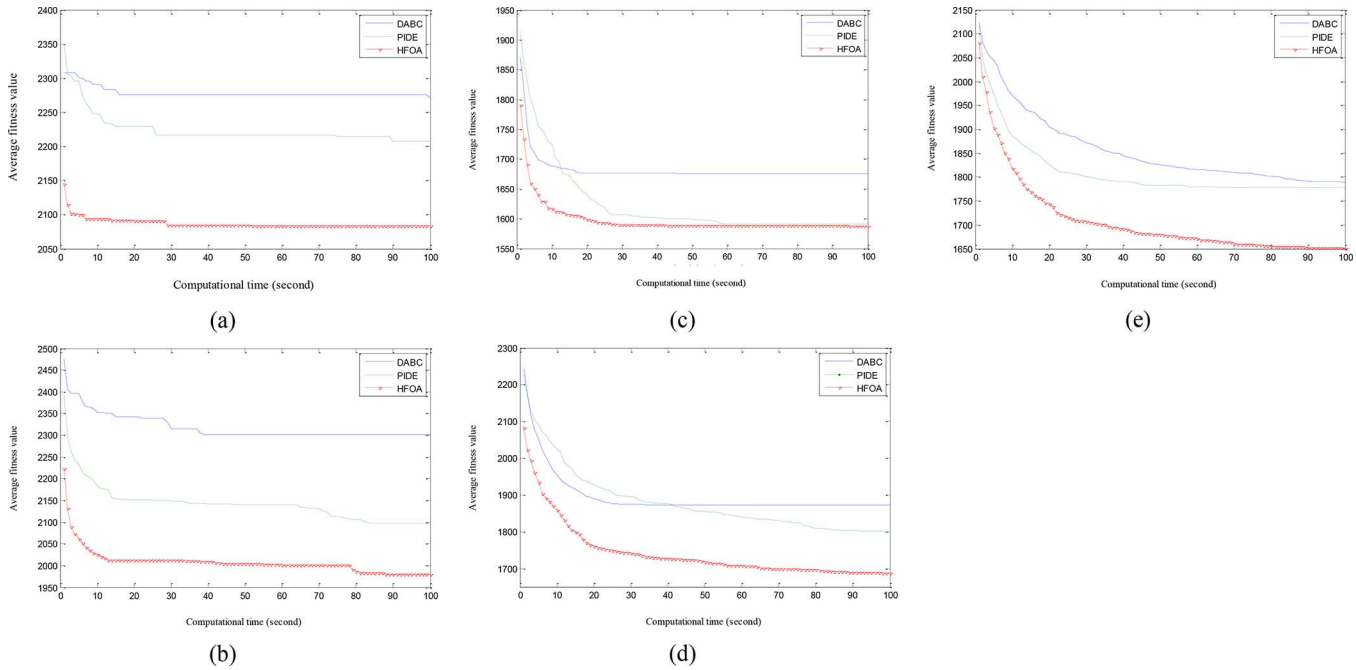


Fig. 11. Comparison of convergence curves for different scale instances: (a) 20-job instances; (b) 60-job instances; (c) 80-job instances; (d) 100-job instances; (e) 120-job instances.

To verify the convergence ability of the compared algorithms, we randomly select five instances with different scales. The convergence curves for the instances with 20, 60, 80, 100, and 120 jobs are reported in Fig. 11(a)–(e), respectively. The convergence curves for different instances indicate that the proposed HFOA algorithm shows better convergence performance than PIDE and DABC.

VII. Conclusion

In this study, a hybrid algorithm combining FOA and IG is proposed for solving the realistic steelmaking rescheduling problem with flexible processing time constraints. The characteristics of the proposed HFOA which make it suitable for this specific rescheduling problem are as follows: 1) a novel encoding mechanism is presented; 2) two decoding heuristics that consider the problem characteristics are embedded; 3) to enhance the exploitation and exploration capabilities of the proposed algorithm, several effective routing and scheduling neighborhood structures are developed; and 4) an IG-based local search procedure is utilized to enhance the exploitation ability.

The proposed algorithm is tested on steelmaking rescheduling problems. Experimental results and statistical analysis show the robustness and efficiency of the proposed algorithm. The two presented problem-specific decoding heuristics improved the performance of the HFOA algorithm by a significant margin. The HFOA algorithm also performed markedly better than the other well-known algorithms presented. Future work includes the application of the proposed algorithm to solving the hybrid flowshop rescheduling problem with other types of disruptions.

References

[1] L. X. Tang, J. Y. Liu, A. Y. Rong, and Z. H. Yang, "A mathematical programming model for scheduling steelmaking-continuous casting production," *Eur. J. Oper. Res.*, vol. 120, no. 2, pp. 423–435, 2000.

[2] J. M. Pinto and I. E. Grossmann, "A continuous time mixed integer linear programming model for short term scheduling of multistage batch plants," *Ind. Eng. Chem. Res.*, vol. 34, no. 9, pp. 3037–3051, 1995.

[3] L. X. Tang, J. Liu, A. Rong, and Z. Yang, "A review of planning and scheduling systems and methods for integrated steel production," *Eur. J. Oper. Res.*, vol. 133, no. 1, pp. 1–20, 2001.

[4] L. Tang, P. B. Luh, J. Liu, and L. Fang, "Steel-making process scheduling using Lagrangian relaxation," *Int. J. Prod. Res.*, vol. 40, no. 1, pp. 55–70, 2002.

[5] H. Xuan and L. X. Tang, "Scheduling a hybrid flowshop with batch production at the last stage," *Comp. Oper. Res.*, vol. 34, no. 9, pp. 2718–2733, 2007.

[6] V. Kumar, S. Kumar, M. K. Tiwari, and F. T. S. Chan, "Auction-based approach to resolve the scheduling problem in the steel making process," *Int. J. Prod. Res.*, vol. 44, no. 8, pp. 1503–1522, 2006.

[7] A. Atighehchian, M. Bijari, and H. Tarkesh, "A novel hybrid algorithm for scheduling steel-making continuous casting production," *Comp. Oper. Res.*, vol. 36, no. 8, pp. 250–2461, 2009.

[8] H. Missbauer, W. Hauber, and W. Stadler, "A scheduling system for the steelmaking-continuous casting process. A case study from the steelmaking industry," *Int. J. Prod. Res.*, vol. 47, no. 15, pp. 4147–4172, 2009.

[9] Y. Y. Tan and S. X. Liu, "Models and optimisation approaches for scheduling steelmaking-refining-continuous casting production under variable electricity price," *Int. J. Prod. Res.*, vol. 52, no. 4, pp. 1032–1049, 2014.

[10] L. X. Tang, J. X. Luo, and J. Y. Liu, "Modelling and a tabu search solution for the slab reallocation problem in the steel industry," *Int. J. Prod. Res.*, vol. 51, no. 14, pp. 4405–4420, 2013.

[11] 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, 2013.

[12] 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.

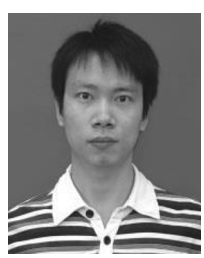
[13] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: a review and some future directions," *Eur. J. Oper. Res.*, vol. 161, no. 1, pp. 86–110, 2005.
[14] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems, a framework of strategies, policies, and methods," *J. Scheduling.*, vol. 6, no. 1, pp. 39–62, 2003.
[15] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. Scheduling.*, vol. 12, no. 4, pp. 417–431, 2009.
[16] A. Allahverdi and J. Mittenthal, "Scheduling on a two-machine flowshop subject to random breakdowns with a makespan objective function," *Eur. J. Oper. Res.*, vol. 81, no. 2, pp. 376–387, 1995.
[17] D. Rahmani and M. Heydari, "Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times," *J. Manu. Sys.*, vol. 33, no. 1, pp. 84–92, 2014.
[18] L. Tang, W. Liu, and J. Liu, "A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment," *J. Intell. Manu.*, vol. 16, no. 3, pp. 361–370, 2005.
[19] D. Petrovic and A. Duenas, "A fuzzy logic based production scheduling/rescheduling in the presence of uncertain disruptions," *Fuzzy. Set. Systems.*, vol. 157, no. 16, pp. 2273–2285, 2006.
[20] M. Zandieh and M. Gholami, "An immune algorithm for scheduling a hybrid flow shop with sequence-dependent setup times and machines with random breakdowns," *Int. J. Prod. Res.*, vol. 47, no. 24, pp. 6999–7027, 2009.
[21] S. X. Yang and C. H. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 959–974, 2010.
[22] K. Wang and S. H. Choi, "A decomposition-based approach to flexible flow shop scheduling under machine breakdown," *Int. J. Prod. Res.*, vol. 50, no. 1, pp. 215–234, 2012.
[23] H. Xuan and B. Li, "Scheduling dynamic hybrid flowshop with serial batching machines," *J. Oper. Res. Soc.*, vol. 64, no. 6, pp. 825–832, 2013.
[24] D. Rahmani, M. Heydari, A. Makui, and M. Zandieh, "A new approach to reducing the effects of stochastic disruptions in flexible flow shop problems with stability and nervousness," *Int. J. Manag. Sci. Eng. Manag.*, vol. 8, no. 3, pp. 173–178, 2013.
[25] K. Katragjini, E. Vallada, and R. Ruiz, "Flow shop rescheduling under different types of disruption," *Int. J. Prod. Res.*, vol. 51, no. 3, pp. 780–797, 2013.
[26] P. I. Cowling, D. Ouelhadj, and S. Petrovic, "A multi-agent architecture for dynamic scheduling of steel hot rolling," *J. Intell. Manu.*, vol. 14, no. 5, pp. 457–470, 2003.
[27] P. I. Cowling, D. Ouelhadj, and S. Petrovic, "Dynamic scheduling of steel casting and milling using multi-agents," *Prod. Plan. Control.*, vol. 15, no. 2, pp. 178–188, 2004.
[28] R. Roy, B. A. Adesola, and S. Thornton, "Development of a knowledge model for managing schedule disturbance in steel-making," *Int. J. Prod. Res.*, vol. 42, no. 18, pp. 3975–3994, 2004.
[29] S. P. Yu and Q. K. Pan, "A rescheduling method for operation time delay disturbance in steelmaking and continuous casting production process," *J. Iron. Steel. Res. Int.*, vol. 19, no. 12, pp. 33–41, 2012.
[30] 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, 2014.
[31] W. T. Pan, "A new fruit fly optimization algorithm: Taking the financial distress model as an example," *Knowl-Based. Syst.*, vol. 26, pp. 69–74, 2012.
[32] L. Wang, X. L. Zheng, and S. Y. Wang, "A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem," *Knowl-Based. Syst.*, vol. 48, pp. 17–23, 2013.
[33] H. Sun and G. Wang, "Parallel machine earliness and tardiness scheduling with proportional weights," *Comput. Oper. Res.*, vol. 30, no. 5, pp. 801–808, 2003.
[34] C. Y. Lee, "Machine scheduling with an availability constraint," *J. Global Opt.*, vol. 9, no. 3–4, pp. 395–416, 1996.
[35] 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.
[36] Q. K. Pan and R. Ruiz, "An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem," *OMEGA-Int. J. Manage. Sci.*, vol. 44, no. 1, pp. 41–50, 2014.
[37] C. García-Martínez, F. J. Rodriguez, and M. Lozano, "Tabu-enhanced iterated greedy algorithm: A case study in the quadratic multiple knapsack problem," *Eur. J. Oper. Res.*, vol. 232, no. 3, pp. 454–463, 2014.
[38] M. F. Tasgetiren, Q. K. Pan, P. N. Suganthan, and O. Buyukdagli, "A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem," *Comp. Oper. Res.*, vol. 40, no. 7, pp. 1729–1743, 2013.
[39] J. Q. Li and Q. K. Pan, "Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity," *Appl. Soft. Comput.*, vol. 12, no. 9, pp. 2896–2912, 2012.
[40] Q. K. Pan, M. Tasgetiren, P. N. Suganthan, and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Inform. Sci.*, vol. 181, no. 12, pp. 2455–2468, 2011.
[41] D. C. Montgomery, *Design and Analysis of Experiments*. Hoboken, NJ, USA: Wiley, 2005.
[42] W. J. Conover, *Practical Nonparametric Statistics*. New York, NY, USA: Wiley, 1980.
[43] 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.
[44] S. Holm, "A simple sequentially rejective multiple test procedure," *Scand. J. Stat.*, vol. 6, no. 2, pp. 65–70, 1979.
[45] 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.
[46] X. Wang and L. Tang, "A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers," *Comput. Oper. Res.*, vol. 36, no. 3, pp. 907–918, 2009.

Jun-Qing Li received the M.S. degree of computer science and technology from Shandong Economic University, Shandong, China, in 2004. He is currently working toward the Ph.D. degree in system engineering at Northeastern University, Shenyang, China.

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

Quan-Ke Pan (M'15) received the B.Sc. and Ph.D. degree Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1993 and 2003, respectively.

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



Kun Mao received the B.S. and M.S. degrees from Northeastern University, Shenyang, China, in 2008 and 2010, respectively, where he is currently working toward the Ph.D. degree in control theory and control engineering.

His current research interests include discrete optimization and scheduling.