An improved imperialist competitive algorithm for hybrid flowshop rescheduling in steelmaking-refining-continuous casting process

Kunkun Peng¹, Xudong Deng¹, Chunjiang Zhang², Quan-Ke Pan³, Liang Ren¹ and Xinfu Pang⁴

Abstract
Steelmaking-refining-Continuous Casting (SCC) is a key process in iron and steel production. SCC scheduling is to determine an optimal schedule for the SCC process, which is a worldwide and important problem. High-quality SCC scheduling methods will help to allocate production resources effectively and increase the productivity. However, dynamic events (e.g. machine breakdown) may happen in the realistic SCC process, which will make the SCC schedule inexecutable or not optimal. In this case, SCC rescheduling is essential in order to obtain a new optimal schedule suitable for the current production environments. The SCC rescheduling can be modeled as hybrid flowshop rescheduling. In this paper, an Improved Imperialist Competitive Algorithm (IICA) is proposed to address the SCC rescheduling. In the proposed IICA, an empire initialization is first devised for constructing an initial population with diversity and certain quality. Moreover, multiswap-based local search and imperialist competition are designed to improve the exploitation ability of the IICA, while revolution and restart strategy are devised to enhance the exploration ability of the IICA. Comparison experiments with three kinds of ICA have shown the efficiency of the IICA.

Keywords
Imperialist competitive algorithm, steelmaking-refining-continuous casting, iron and steel production, hybrid flowshop, rescheduling

Date received: 17 July 2020; accepted: 31 August 2020

Introduction
Iron and steel production supplies important materials for many industries. It has special characteristics such as high temperature, high weight, and high energy consumption. Steelmaking-refining-Continuous Casting (SCC) is the bottleneck of the iron and steel production process, therefore, scheduling the SCC process, that is, SCC scheduling, is a key and complicated problem. The SCC scheduling is NP-hard, and can be modeled as hybrid flowshop scheduling. With respect to the hybrid flowshop scheduling, researchers can refer to Ribas et al., Meng et al., and Zhang et al. Powerful SCC scheduling methods will help to allocate production resources efficiently and improve the productivity.

Unfortunately, a number of dynamic events (e.g. machine breakdown, new order arrival, order cancellation) may happen in the realistic SCC process, which will make original SCC scheduling plans infeasible or not optimal. In this case, SCC rescheduling is essential to construct a new optimal schedule, which is suitable for current production environments. The SCC rescheduling can be modeled as hybrid flowshop rescheduling. That is, the SCC rescheduling can be thought as the realistic application of the hybrid flowshop rescheduling. Compared with fruitful research on the SCC scheduling, less attempts have been made on the SCC rescheduling.
rescheduling. Existing SCC rescheduling methods include heuristics, Lagrangian relaxation, and metaheuristics. Among the existing metaheuristics-based methods, Tang et al. designed a differential evolution with incremental mechanism, while Long et al. developed a hybrid algorithm which combined genetic algorithm and variable neighborhood search, where continuous caster breakdown was considered as the dynamic event. Li et al. presented a hybrid fruit fly optimization algorithm, where processing times in the continuous casting stage were considered to be controllable. Peng et al. devised an improved artificial bee colony.

Imperialist Competitive Algorithm (ICA) is a recent and effective metaheuristic, which has been employed to successfully address many NP-hard problems, such as assembly flowshop scheduling, energy-efficient flexible job shop scheduling, and distributed shop scheduling. The ICA begins with an initial population of solutions (named countries). Countries are classified as imperialists or colonies. Each colony is assigned to exactly an imperialist. Each imperialist attempts to assimilate its colonies and also tries to possess more colonies from other imperialists. The basic ICA cannot be implemented directly to address the SCC rescheduling, since it is originally presented for continuous optimization problems. In this paper, an improved ICA is designed to solve the SCC rescheduling, named IICA.

The remainder of this paper is organized as follows. Section “SCC rescheduling problem” describes the SCC rescheduling problem. Section “Introduction of Basic ICA” describes the basic ICA. Section “Improved imperialist competitive algorithm” illustrates the details of the proposed IICA. Section “Experiment results” lists experiment comparison results. Finally, section “Conclusions” concludes this paper.

SCC rescheduling problem

In general, the SCC process contains three sequential production phases, i.e., steelmaking stage, refining stage, and continuous casting stage. In some cases, several refining stages may be contained in the SCC process, called multirefining. Each production stage has its own responsibility. More specifically, in the steelmaking stage, hot molten iron is transferred into molten steel and undesired impurity ingredients are lowered to certain levels. This is achieved by the machine named converter. The refining stage has the responsibility of further eliminating the impurity ingredients, and adding the required alloy ingredients. This is produced by the machine named refining furnace. In the continuous casting stage, liquid steel is casted into solid slabs, which is achieved by the machine named continuous caster. In the steelmaking and refining stage stages, charge is the basic production unit, while in the continuous casting stage, cast is the basic production unit.

Introduction of Basic ICA

The basic ICA begins with empire initialization, which generates an initial population of $N$ solutions (named countries). Among the $N$ countries, the $N_{\text{Imp}}$ best countries (named imperialists) are utilized to construct the empires (each imperialist corresponds to each empire), while the remaining $N-N_{\text{Imp}}$ countries serve as colonies of the $N_{\text{Imp}}$ empires. That is, an imperialist and its corresponding colonies together form an empire, while each colony belongs to and only belongs to an empire, and an empire may contain several colonies. The number of colonies allocated to imperialist $i$ is determined by the fitness of the imperialist, denoted as $N_{C_i}$. The fitter the imperialist $i$ is, the bigger $N_{C_i}$ is. Then $N_{C_i}$ colonies are selected randomly for imperialist $i$. When the Empire initialization is finished, a cycle of Assimilation-Exchange of imperialist and colony-Imperialist competition-Elimination of powerless imperialists continues...
until stopping conditions are reached. More specifically, the basic ICA is displayed as follows.

Framework of the basic ICA

//Empire initialization
Generate $N$ countries, select $N_{\text{Imp}}$ best countries as imperialists to construct the empires, and allocate the remaining $N-N_{\text{Imp}}$ countries to the $N_{\text{Imp}}$ empires as colonies.

while termination condition has not been met

//Assimilation
For each colony, improve it by moving towards its relevant imperialist.

//Exchange imperialist and colony
For each imperialist, if the best one among its relevant colonies is better than it, exchange the best colony and the imperialist.

//Imperialist competition
Select one of the weakest colonies of the weakest empire, make a competition among all the empires to select an imperialist, the selected imperialist will possess the colony.

//Elimination of powerless imperialists
For each imperialist, the imperialist collapses if it lost all the colonies.
end while

Improved imperialist competitive algorithm

The section introduces the details of the designed IICA. First, encoding and decoding is shown. Then, empire initialization, multiswap-based local search, assimilation, revolution, exchange of imperialist and colony, imperialist competition, and restart strategy, are illustrated sequentially. Finally, the framework of the IICA is described.

Encoding and decoding

Charge permutation is employed as the encoding. More specifically, the encoding method of Peng et al. is utilized in the IICA. After the encoding method is decided, a decoding method should be determined, which will decode a solution to a schedule. The decoding method in Peng et al. is modified for the IICA. The modification is that, the steps relevant to controllable processing times are removed. More specifically, the decoding method of the IICA includes forward decoding and backward decoding. The forward decoding is the same as that of Peng et al., while the backward decoding is some different, where the difference is mainly that, the charge left-shifting strategy and processing time adding strategy in Peng et al. are not implemented in the IICA.

Empire initialization

An initial population with certain quality may be beneficial to enhance the performance of the IICA. In this paper, an empire initialization is specially devised for obtaining a population with diversity and certain quality. Let $N_h$ denote the number of countries obtained by heuristic instead of random generation, the empire initialization can be given as follows.

Step 1. Generate two countries $C_1$ and $C_2$ by sequencing charges in non-decreasing order based on the starting times at refining and continuous casting stage, respectively.
Step 2. Set $i = 3$.
Step 3. If $i \leq N_h$, randomly initialize country $C_i$ to be $C_1$ and $C_2$ with equal probability, and then execute multiswap or Insert operator to $C_i$ with equal probability, go to Step 4; Otherwise, go to Step 5.
Step 4. Set $i = i + 1$, and go to Step 3.
Step 5. Generate the remaining $N-N_h$ countries randomly.
Step 6. Sort all the countries, and select the $N_{\text{Imp}}$ best countries to be the imperialists, which correspond to

![Figure 1. Illustration of a SCC process.](Image)
$N_{imp}$ empires. The number of colonies allocated to the $j^{th}$ empire (say $NC_j$) is determined by Formula (1).

$$NC_j = \frac{f^{-\alpha}}{\sum_{k=1}^{N_{imp}} k^{-\alpha}}$$

where $\alpha$ is a positive real number\textsuperscript{23} and set to be $1.5$ in this paper.

**Step 7.** For each imperialist (say $j$), select $NC_j$ countries randomly and allocate them as colonies of the empire $j$.

**Multiswap-based local search**

To improve the exploitation ability of the IICA, a multiswap-based local search is developed to further improve the imperialists. Let $Q$ be the allowed maximum number, the multiswap-based local search is illustrated as follows.

**Step 1.** Set $i = 1$.

**Step 2.** Set $q = 1$.

**Step 3.** If $q \leq Q$, execute a multiswap operator to the imperialist $I_i$ if the obtained solution is better than $I_i$, update $I_i$; Otherwise, go to Step 5.

**Step 4.** Set $q = q + 1$, go to Step 3.

**Step 5.** Set $i = i + 1$, if $i \leq N_{imp}$, go to Step 2; Otherwise, stop.

**Assimilation**

The assimilation is to improve each colony by moving toward its corresponding imperialist. Let $NI_k$ denote the number of successive iterations that colony $C_k$ has not been improved, the assimilation is devised as follows.

**Step 1.** For each colony $C_k$ ($k = 1,2,\ldots,N-N_{imp}$), execute Partially Mapped Crossover (PMX) to $C_k$ and its corresponding imperialist, then two new solutions are obtained, denoted as $P_1$ and $P_2$.

**Step 2.** Select randomly a solution from $P_1$ and $P_2$ with equal probability, say $P_t$ ($t = 1$ or 2).

**Step 3.** If $P_t$ is better than $C_k$, update $C_k$ by $P_t$, and set $NI_k = 0$; Otherwise, go to Step 4.

**Step 4.** If $f(P_t) = f(C_k)$, update $C_k$ by $P_t$, set $NI_k = NI_k + 1$, where $f(P_t)$ and $f(C_k)$ denote the objective value of $P_t$ and $C_k$ respectively; Otherwise, go to Step 5.

**Step 5.** Set $NI_k = NI_k + 1$, accept $P_t$ as $C_k$ with the probability $\exp\left(-\frac{(f(P_t) - f(C_k))}{TP}\right)$, where

$$TP = \frac{\sum_{s=1}^{M} \sum_{t=1}^{S} PT_{ps}}{10} \cdot T, M \text{ and } S \text{ denote the number of charges and stages respectively, } PT_{ps} \text{ denotes the processing time of charge } p \text{ at the } s^{th} \text{ stage, } T \text{ is set to be } 0.4.$$

**Exchange imperialist and colony**

For each empire (say $j$), if the best one among its colonies (say colony $C_k$) is better than imperialist $I_j$, exchange $I_j$ and $C_k$. The operator is to make sure that the imperialist is better than the colonies in the empire, which can further guide the evolution of its corresponding colonies to more promising area.

**Imperialist competition**

In the basic ICA, the imperialist competition is executed in each iteration. However, we believe that it is not necessary to employ the imperialist competition in each iteration. This is because, a colony may not be improved in one iteration, however, it may be a promising solution and can be further improved in the following several iterations. Therefore, an imperialist...
competition is devised in this paper, that is, execute the imperialist competition for \( R \) iterations instead of one iteration. Let \( \text{Iter} \) be the iteration number, the steps of the imperialist competition can be listed as follows.

**Step 1.** If \( \text{Iter} \) is multiples of \( R \), that is, \( \text{Iter}\%R = 0 \), go to Step 2; Otherwise, stop.

**Step 2.** If the weakest empire (say \( j \)) has lost all of its colonies, select an imperialist randomly (say \( p \)), allocate imperialist \( I_j \) to \( I_p \) as a colony, stop; Otherwise, go to Step 3.

**Step 3.** Select randomly one colony from empire \( j \), select an imperialist randomly (say \( p \)), allocate the colony to \( I_p \).

**Step 4.** If empire \( j \) has lost all of its colonies, allocate the imperialist \( I_j \) to \( I_p \) as a colony.

**Restart strategy**

In the evolutionary process of the IICA, some colony may encounter such a situation that has not been improved during the last several successive iterations. This may mean that the colony has been trapped in the local optimum. To escape the local optimum, a restart strategy is devised in this paper. Let \( N_{\text{Imax}} \) be the allowed number of successive iterations that a colony has not been improved, \( LS \) be the size of local search, the restart strategy is illustrated as follows:

**Step 1.** For colony \( C_k \) (\( k = 1, ..., N - N_{\text{Imp}} \)), if \( N_k > N_{\text{Imax}} \), go to Step 2; Otherwise, stop.

**Step 2.** Construct randomly \( LS \) solutions (say \( S_1, S_2, ..., S_{\text{LS}} \)), and for each solution \( S_p \) (\( p = 1, 2, ..., LS \)), set \( S_p \) to be the best solution found so far \( G_b \). execute several multiswap to \( S_p \).

**Step 3.** Find the best one among the obtained \( LS \) solutions (denoted by \( S_b \)), set \( C_k = S_b \). If \( S_b \) is better than \( G_b \), update \( G_b \).

**Framework of the proposed IICA**

Let \( N \) and \( N_{\text{Imp}} \) be the number of countries and imperialists, respectively, \( Rr \) be the revolution rate, \( N_{\text{Imax}} \) be the allowed number of successive non-improving iterations, \( R \) be the parameter in the imperialist competition, \( G_b \) be the best solution found so far, and \( \text{Iter} \) be the iteration number. The framework of the IICA is stated as follows.

**Experiment results**

The main concern of applying the metaheuristics to solve the combinatorial optimization problems is the solution quality. In this paper, to show the effectiveness of the presented IICA, three kinds of ICA in the literature are employed for comparison. The three kinds of ICA are listed as follows, (1) the basic ICA \(^{18}\) (ICA), (2) an improved ICA devised by Shokrollahpour et al. \(^{19}\) (ICAS), and (3) hybrid ICA proposed by Ghasemishabankareh et al. \(^{24}\) (HICA). Note that the PMX operator is employed for the ICA and HICA, since the corresponding assimilation operation is not suitable for the SCC rescheduling. More specifically, for each colony, execute the PMX to it and its corresponding imperialist, which generates two solutions, select randomly a solution from the two solutions with equal probability. In the ICAS, two assimilation operations are tested, that is, (1) assimilation operation 1: the same assimilation operator employed for the ICA and HICA stated above; and (2) assimilation operation 2: the assimilation operation in the corresponding reference. Moreover, for the ICAS utilizing each assimilation operation, two revolution cases are tested, that is, (1) revolution strategy 1: the 20% worst colonies are implemented the revolution; and (2) revolution strategy 2: the 30% worst colonies are implemented the revolution. That is, four versions of ICAS are tested in this paper, which are named ICAS1, ICAS2, ICAS3, ICAS4, respectively. Specifically, the ICAS1 is equipped with assimilation operation 1 and revolution strategy 1, while the ICAS2 is equipped with assimilation operation 1 and revolution strategy 2. The ICAS3 is equipped with assimilation operation 2 and revolution strategy 1, while ICAS4 is equipped with assimilation operation 2 and revolution strategy 2.

We code the above seven compared algorithms (i.e. IICA, ICA, HICA, ICAS1, ICAS2, ICAS3, and ICAS4) in C++ + + , and run them on a 2.13 GHZ computer with 2G RAM using Windows 7 Operation System. The elapsed CPU time is set to be termination condition. Three termination conditions are used in this paper, that is, 10, 20, and 30 s. For each termination condition, each compared algorithm is run 10 times independently, the average result of the ten runs is calculated, and further served as the performance of the algorithm. Relative percentage increase (RPI) is calculated for comparing these algorithms, which is calculated by Formula (2).

\[
RPI = \frac{f_{\text{ca}} - f_{\text{lb}}}{f_{\text{lb}}} \cdot 100\%
\]

where \( f_{\text{ca}} \) denotes the average objective function value of IICA, ICA, ICAS, or HICA, and \( f_{\text{lb}} \) denotes the best one among \( f_{\text{ca}} \) (\( ca \) denotes IICA, ICA, ICAS, HICA).

**Test instances**

Thirty test instances are randomly generated based on the test instance generation method in Pan et al. \(^{4}\) and are further used to compare the above algorithms. More specifically, the SCC process contains steelmaking stage, refining stage, and continuous casting stage. The three stages contain three converts, five refining furnaces, and four continuous casters, respectively. Each continuous caster produces 3 to 4 casts in each workday, where each cast processes 8 to 12 charges. The processing time of each charge in the steelmaking
Framework of the IICA

Step 1. Set \( \text{Iter} = 1 \).

Step 2. **Empire initialization.** Generate \( N_0 \) countries by the heuristic presented in section “Empire initialization,” and generate \( N-N_0 \) countries randomly. Select \( N_{\text{Imp}} \) best countries as imperialists to construct the empire, and allocate randomly the remaining \( N-N_{\text{Imp}} \) countries to the \( N_{\text{Imp}} \) imperialists as colonies according to Formula (1). If an imperialist (Say \( I_j \)) has not been allocated any colony, select an imperialist randomly (Say \( I_j \)) and allocate \( I_j \) as a colony. Find the best one among the \( N \) countries, utilize it to initialize \( G_0 \).

Step 3. **Multiswap-based local search.** For each imperialist, execute a multiswap operator to it, if a better solution can be obtained, utilize it to replace the imperialist, continue the step until a predefined number is reached. If the updated imperialist is better than \( G_0 \), update \( G_0 \).

Step 4. **Assimilation.** For each colony, execute PMX to the colony and its corresponding imperialist, then two new solutions are obtained, selected one from the two solutions, and further determine whether it can be accepted as the colony.

Step 5. **Exchange imperialist and colony.** For each empire, if the best one among its colonies is better than the imperialist, exchange the colony and the imperialist. If a solution better than \( G_b \) is found, update \( G_b \) by the solution.

Step 6. **Revolution.** For each imperialist, execute the revolution strategy described in section “Revolution.” For each colony, execute the revolution strategy described in section “Revolution.”

Step 7. **Exchange imperialist and colony.** For each empire, if the best one among its colonies is better than the imperialist, exchange the colony and the imperialist. If a solution better than \( G_b \) is found, update \( G_b \) by the solution.

Step 8. **Imperialist competition.** If \( \text{Iter} \) is multiples of \( R \), select randomly one colony of the weakest empire (say \( j \)), select an imperialist randomly (say \( p \)), allocate the colony to imperialist \( I_p \). If empire \( j \) has lost all of its colonies, allocate the imperialist \( I_j \) to \( I_p \) as a colony.

Step 9. **Restart strategy.** For each colony, if it has not been improved during the last \( N_{\text{max}} \) iterations, construct \( LS \) solutions, which are copies of \( G_b \). Execute several multiswap to each solution. Utilize the best one of the obtained \( LS \) solutions update the colony. If a solution better than \( G_b \) is found, update \( G_b \) by the solution.

Step 10. Set \( \text{Iter} = \text{Iter} + 1 \). If the termination condition (e.g. allowed elapsed CPU time) has been met, return \( G_b \). Otherwise go to Step 3.

Stage is 38, while the processing times of each charge in the refining stage and continuous casting stage are in [36,50] and [38,44], respectively. The transferring time between the steelmaking stage and refining stage is in [10,15], which is the same as that between the refining stage and continuous casting stage.

With respect to the dynamic event, machine breakdown is considered, for each test instance, two machine breakdown cases are generated randomly, that is (1) converter breakdown: (i) breakdown converter: a converter is selected randomly (say \( c \)); (ii) rescheduling time, the rescheduling time is equal to \( RT_1 + RT_2 \), where \( RT_1 \) is 10% of the completion time on the converter \( c \), and \( RT_2 \) is the starting time of the first charge processed on the converter \( c \); (iii) breakdown duration: breakdown duration is equal to 6% of the machine completion time of the converter \( c \). (2) refining furnace breakdown: (i) breakdown refining furnace: a refining furnace is selected randomly (say \( rf \)); (ii) rescheduling time, the rescheduling time is equal to \( RT_1 + RT_2 \), where \( RT_2 \) is 30% of the completion time of the refining furnace \( rf \) and \( RT_2 \) is the starting time of the first charge processed on the refining furnace \( rf \); (iii) breakdown duration: breakdown duration is equal to 6% of the machine completion time of the refining furnace \( rf \).

Therefore, under each termination condition, for each machine breakdown case of each test instance, each compared algorithm is run 10 times independently, the average result of the 10 runs is calculated, and RPI can be further calculated. Then two RPIs can be obtained for two machine breakdown cases of each test instance. The average value of two RPIs is employed as the average RPI of each compared algorithm.

**Comparison results**

In this section, the IICA is compared with the ICA, HICA, ICA\(_{S1}\), ICA\(_{S2}\), ICA\(_{S3}\), and ICA\(_{S4}\). In the IICA, the main parameters stated in section “Framework of the proposed IICA,” that is \( N, N_{\text{Imp}}, R, N_{\text{max}}, \) and \( R \) are set to be 80, 0.3, 500, and 20, respectively. To make the comparison fair, the same termination condition is used, that is, allowed CPU time. CPU time 10 s is employed first. Table 1 lists the average RPI comparison results of the seven algorithms when CPU time = 10 s. From Table 1, it can be noticed that the IICA and HICA perform better than ICA, ICA\(_{S1}\), ICA\(_{S2}\), ICA\(_{S3}\), and ICA\(_{S4}\), and the IICA is worse than the HICA. More specifically, the average RPIs of IICA and HICA are 1.56% and 0.65%, respectively, while the average RPIs of the ICA, ICA\(_{S1}\), ICA\(_{S2}\), ICA\(_{S3}\), and ICA\(_{S4}\) are 25.13%, 27.17%, 26.72%, 32.06%, and 32.06%, respectively.

Moreover, the termination condition is set to be longer CPU time, that is, 20 s. Table 2 gives the corresponding average RPI comparison values of these compared algorithms. Table 2 shows that the IICA performs better than other algorithms, that is, the HICA, ICA, ICA\(_{S1}\), ICA\(_{S2}\), ICA\(_{S3}\), and ICA\(_{S4}\). Precisely, the average RPI of IICA is 0.69%, while the average RPIs
IICA is an effective method for addressing the SCC rescheduling problem. Experimental results and analysis have shown that the IICA outperforms the HICA. In conclusion, the above experimental results have shown that the IICA always outperforms the remaining algorithms. Moreover, under shorter CPU time (i.e. 10 s), the performance of the IICA is worse than that of the HICA, while under longer CPU time (i.e. 20 and 30 s), the IICA outperforms the HICA. In conclusion, the above experimental results and analysis have shown that the IICA is an effective method for addressing the SCC rescheduling problem.

### Conclusions

In the paper, we present an improved ICA for the SCC rescheduling, which can be recognized as the realistic application of the hybrid flowshop rescheduling. An
empire initialization is devised to construct the initial countries with diversity and certain quality. Meanwhile, multiswap-based local search and imperialist competition are designed to enhance the intensification of the IICA. Moreover, revolution and restart strategy are devised to improve the diversification. In the future work, the IICA can be further enhanced by designing more effective operators based on problem characteristics or attempting to combine with other techniques. Meanwhile, the IICA can be employed for addressing other production scheduling problems, for example, welding shop scheduling, energy-efficient scheduling, distributed scheduling, integrated process planning and scheduling. Furthermore, the existing rescheduling methods for the hybrid flowshop can be modified and adopted for the SCC rescheduling.

Authors’ Note
Kunkun Peng is also affiliated with State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan, China.

Declaration of conflicting interests
The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding
The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This project is supported by National Natural Science Foundation of China (Grant No. 51705177, 51905199, 61773269).

ORCID iD
Chunjiang Zhang https://orcid.org/0000-0002-2284-7291

References
1. Cui H and Luo X. An improved Lagrangian relaxation approach to scheduling steelmaking-continuous casting process. Comput Chem Eng 2017; 106: 133–146.
2. García-Menéndez D, Morán-Palacios H, Ortega-Fernández F, et al. Scheduling in continuous steelmaking casting: a systematic review. ISIJ Int 2020; 60(6): 1097–1107.
3. Peng K, Pan Q and Zhang B. An improved artificial bee colony algorithm for steelmaking-refining-continuous casting scheduling problem. Chin J Chem Eng 2018; 26(8): 1727–1735.
4. Pan Q, Wang L, Mao K, et al. An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. IEEE Trans Autom Sci Eng 2013; 10(2): 307–322.
5. Jiang S, Zheng Z and Liu M. A preference-inspired multi-objective soft scheduling algorithm for the practical steelmaking-continuous casting production. Comput Ind Eng 2018; 115: 582–594.
6. Jiang S, Liu M and Hao J. A two-phase soft optimization method for the uncertain scheduling problem in the steelmaking industry. IEEE Trans Syst Man Cybern-Syst 2017; 47(3): 416–431.
7. Ribas I, Leisten R and Framiñan JM. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Comput Oper Res 2010; 37(8): 1439–1454.
8. Meng L, Zhang C, Shao X, et al. Mathematical modeling and optimisation of energy-conscious hybrid flow shop scheduling problem with unrelated parallel machines. Int J Prod Res 2019; 57(4): 1119–1145.
9. Zhang B, Pan Q, Gao L, et al. A three-stage multiobjective approach based on decomposition for an energy-efficient hybrid flow shop scheduling problem. IEEE Trans Syst Man Cybern-Syst 2019; Epub ahead of print May 2019. DOI: 10.1109/tsmc.2019.2916088.
10. Tang L, Zhao Y and Liu J. An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production. IEEE Trans Evol Comput 2014; 18(2): 209–225.
11. Li J, Pan Q and Mao K. A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems. IEEE Trans Autom Sci Eng 2016; 13(2): 932–949.
12. Peng K, Pan Q, Gao L, et al. An improved artificial bee colony algorithm for real-world hybrid flowshop rescheduling in steelmaking-refining-continuous casting process. Comput Ind Eng 2018; 122: 235–250.
13. Peng K, Pan Q, Gao L, et al. A multi-start variable neighbourhood descent algorithm for hybrid flowshop rescheduling. Swarm Evol Comput 2019; 45: 92–112.
14. Sun L, Ying Y, Luan F, et al. Rescheduling optimization of steelmaking-continuous casting process based on the Lagrangian heuristic algorithm. J Ind Manag Optim 2017; 13(3): 1431–1448.
15. Yu S, Chai T and Tang Y. An effective heuristic rescheduling method for steelmaking and continuous casting production process with multirefining modes. IEEE Trans Syst Man Cybern-Syst 2016; 46(12): 1675–1688.
16. Mao K, Pan Q, Peng X, et al. An effective Lagrangian relaxation approach for rescheduling a steelmaking-continuous casting process. Control Eng Practice 2014; 30: 67–77.
17. Long J, Zheng Z and Gao X. Dynamic scheduling in steelmaking-continuous casting production for continuous caster breakdown. Int J Prod Res 2017; 55(11): 3197–3216.
18. Atashpaz-Gargari E and Lucas C. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: 2007 IEEE congress on evolutionary computation (CEC 2007). Singapore, 25–28 September 2007, pp. 4661–4667. New York: IEEE.
19. Shokrollaeepour E, Zandieh M and Dorri B. A novel imperialist competitive algorithm for bi-criteria scheduling of the assembly flowshop problem. Int J Prod Res 2011; 49(11): 3087–3103.

Table 3. Continued

| Instance | ICA | HICA | ICA51 | ICA52 | ICA53 | ICA54 | IICA |
|----------|-----|------|-------|-------|-------|-------|------|
| 27       | 19.07 | 0.00  | 14.08 | 13.41 | 21.66 | 21.66 | 0.16 |
| 28       | 33.19 | 1.07  | 24.80 | 20.80 | 39.78 | 39.78 | 0.00 |
| 29       | 22.32 | 1.55  | 18.94 | 17.42 | 26.04 | 26.04 | 0.00 |
| 30       | 15.34 | 0.21  | 13.10 | 12.58 | 18.32 | 18.32 | 0.05 |
| Average  | 27.04 | 1.19  | 22.75 | 21.64 | 34.10 | 34.10 | 0.58 |

1. Cui H and Luo X. An improved Lagrangian relaxation approach to scheduling steelmaking-continuous casting process. Comput Chem Eng 2017; 106: 133–146.
2. García-Menéndez D, Morán-Palacios H, Ortega-Fernández F, et al. Scheduling in continuous steelmaking casting: a systematic review. ISIJ Int 2020; 60(6): 1097–1107.
3. Peng K, Pan Q and Zhang B. An improved artificial bee colony algorithm for steelmaking-refining-continuous casting scheduling problem. Chin J Chem Eng 2018; 26(8): 1727–1735.
20. Lei D, Li M and Wang L. A two-phase meta-heuristic for multiobjective flexible job shop scheduling problem with total energy consumption threshold. *IEEE T Cybern* 2019; 49(3): 1097–1109.
21. Lei D, Yuan Y, Cai J, et al. An imperialist competitive algorithm with memory for distributed unrelated parallel machines scheduling. *Int J Prod Res* 2020; 58(2): 597–614.
22. Lian K, Zhang C, Gao L, et al. Integrated process planning and scheduling using an imperialist competitive algorithm. *Int J Prod Res* 2012; 50(15): 4326–4343.
23. Lu Z and Huang W. Iterated tabu search for identifying community structure in complex networks. *Phys Rev E* 2009; 80(2): 026130.
24. Ghasemishabankareh B, Shahsavari-Pour N, Basiri M, et al. A hybrid imperialist competitive algorithm for the flexible job shop problem. In: *Australasian conference on artificial life and computational intelligence*, 2 February 2016, pp. 221–233. Cham: Springer.
25. Wen L, Li X and Gao L. A transfer convolutional neural network for fault diagnosis based on ResNet-50. *Neural Comput Appl* 2020; 32: 6111–6124.
26. Wen L, Ye X and Gao L. A new automatic machine learning based hyperparameter optimization for workpiece quality prediction. *Meas Control*. Epub ahead of print July 2020. DOI: 10.1177/0020294020932347.
27. Li X, Lu C, Gao L, et al. An effective multiobjective algorithm for energy-efficient scheduling in a real-life welding shop. *IEEE Trans Ind Inform* 2018; 14(12): 5400–5409.
28. Lu C, Gao L, Li X, et al. A multi-objective approach to welding shop scheduling for makespan, noise pollution and energy consumption. *J Clean Prod* 2018; 196: 773–787.
29. Zhang B, Pan Q, Gao L, et al. A multiobjective evolutionary algorithm based on decomposition for hybrid flowshop green scheduling problem. *Comput Ind Eng* 2019; 136: 325–344.
30. Meng L, Zhang C, Shao X, et al. MILP models for energy-aware flexible job shop scheduling problem. *J Clean Prod* 2019; 210: 710–723.
31. Pan Q, Gao L, Wang L, et al. Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Syst Appl* 2019; 124: 309–324.
32. Ruiz R, Pan Q and Naderi B. Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega-Int J Manage Sci* 2019; 83: 213–222.
33. Wang G, Gao L, Li X, et al. Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm. *Swarm Evol Comput* 2020; 57: 100716.
34. Li X, Gao L, Pan Q, et al. An effective hybrid genetic algorithm and variable neighborhood search for integrated process planning and scheduling in a packaging machine workshop. *IEEE Trans Syst Man Cybern-Syst* 2019; 49(10): 1933–1945.