

Article

An improved artificial bee colony algorithm for steelmaking–refining–continuous casting scheduling problem☆

Kunkun Peng, Quanke Pan*, Biao Zhang

State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

ARTICLE INFO

Article history:

Received 11 October 2017

Received in revised form 1 June 2018

Accepted 11 June 2018

Available online 19 June 2018

Keywords:

Artificial bee colony

Steelmaking–refining–continuous casting

Hybrid flowshop scheduling

Variable neighborhood search

ABSTRACT

Steelmaking–refining–Continuous Casting (SCC) scheduling is a worldwide problem, which is NP-hard. Effective SCC scheduling algorithms can help to enhance productivity, and thus make significant monetary savings. This paper develops an Improved Artificial Bee Colony (IABC) algorithm for the SCC scheduling. In the proposed IABC, charge permutation is employed to represent the solutions. In the population initialization, several solutions with certain quality are produced by a heuristic while others are generated randomly. Two variable neighborhood search neighborhood operators are devised to generate new high-quality solutions for the employed bee and onlooker bee phases, respectively. Meanwhile, in order to enhance the exploitation ability, a control parameter is introduced to conduct the search of onlooker bee phase. Moreover, to enhance the exploration ability, the new generated solutions are accepted with a control acceptance criterion. In the scout bee phase, the solution corresponding to a scout bee is updated by performing three swap operators and three insert operators with equal probability. Computational comparisons against several recent algorithms and a state-of-the-art SCC scheduling algorithm have demonstrated the strength and superiority of the IABC.

© 2018 The Chemical Industry and Engineering Society of China, and Chemical Industry Press. All rights reserved.

1. Introduction

Iron and steel production is a worldwide problem, providing raw materials for a series of industries, such as petro-chemical, construction, and machinery manufacturing [1]. In the production, Steelmaking–refining–Continuous Casting (SCC) processing is the bottleneck [2], which processes hot metal to steel with a well-defined chemical composition and solidifies the steel into slabs. Effective SCC scheduling methods are crucial to improve production productivity, resulting in significant monetary savings. Moreover, the problem is well known to be NP-hard and considered as one of the most difficult industrial scheduling problems [3, 4]. The problem has attracted much interest in recent years [1, 4, 5].

The classical SCC process can be divided into three stages, *i.e.* steelmaking, refining, and continuous casting, each of them has multiple parallel machines. In the literature, the SCC scheduling problem is usually modeled as a Hybrid Flowshop Scheduling (HFS) problem [6–8]. A variety of SCC scheduling methods have been proposed, which mainly can be classified as three categories: heuristics, mathematical methods and metaheuristics. In terms of heuristics, Yu and Chai [9] proposed a series of scheduling sequence methods and equipment assignment

methods, which can be applied when devising SCC scheduling methods. To further deal with the SCC scheduling problem considering dynamic events, Yu and Pan [10] presented a rescheduling heuristic for the SCC scheduling problem with operation time delay. Subsequently, Yu *et al.* [11] proposed an effective heuristic for SCC scheduling simultaneously considering multirefining modes and job start-time delay. With respect to the mathematical methods, Mao *et al.* [12], Cui and Luo [1] devised Lagrangian relaxation approaches for the SCC scheduling problem, while Mao *et al.* [13] further developed a Lagrangian relaxation approach for the SCC scheduling problem considering dynamic events. In terms of metaheuristics, population-based metaheuristics have been widely employed. Pan *et al.* [2] presented a mixed integer HFS model and then proposed an effective Artificial Bee Colony (ABC) algorithm. Subsequently, Pan [5] modeled the SCC scheduling problem as a combination of two coupled sub-problems, *i.e.* a charge scheduling problem and a cast scheduling problem, which correspond to HFS and parallel machine scheduling respectively. They devised a co-evolutionary ABC with two sub-swarms, where each sub-swarm corresponds to a sub-problem. Li *et al.* [8] designed an effective Fruit Fly Optimization (FOA) algorithm for the SCC scheduling, and Li *et al.* [14] further presented a hybrid FOA for the SCC scheduling with dynamic events. Tang *et al.* [15] presented an improved Differential Evolution (DE) algorithm for the SCC scheduling with dynamic events, where a real-coded matrix encoding mechanism, a specially-designed population initialization, and a new mutation strategy were devised. Jiang *et al.* [16] proposed a bi-layer optimization approach for the SCC scheduling with controllable

☆ Supported by the National Natural Science Foundation of China (51705177, 51575212), the Program for New Century Excellent Talents in University (NCET-13-0106), and the Program for HUST Academic Frontier Youth Team.

* Corresponding author.

E-mail addresses: pengkunkun@126.com, (K. Peng), panquanke@hust.edu.cn, (Q. Pan), zhangbiao1218@gmail.com (B. Zhang).

processing times, where a hybrid DE combined with a variable neighborhood decomposition search is designed for a parallel machine scheduling problem in the last stage and an iterative backward list scheduling algorithm is designed for HFS in the former stages. Hao *et al.* [17] designed a particle swarm optimization based two-layered approach to deal with the SCC scheduling problem considering dynamic events, while Long *et al.* [18] proposed a hybrid genetic algorithm integrated with a general variable neighbourhood search to solve the SCC scheduling problem considering continuous caster breakdown.

ABC is a recent swarm intelligence optimization algorithm, which has been successfully applied for solving both continuous and combinatorial optimization problems [19–21]. In the work for combinatorial optimization problems, ABC has been widely studied to solve kinds of scheduling problems, e.g. permutation flow shop scheduling [22], job shop scheduling [23, 24], flexible job-shop scheduling [25, 26], hybrid flow shop scheduling with limited buffers [27], lot-streaming flow shop scheduling [28], blocking lot-streaming flow shop scheduling [29], single machine scheduling [30], and parallel batch-processing machine scheduling [31]. In this paper, an Improved ABC (IABC) algorithm is developed for the SCC scheduling problem. A heuristic is first devised to generate initial population with certain quality. Unlike the previous ABC devised for the SCC scheduling problem, to enhance the exploitation ability of IABC, two Variable Neighborhood Search (VNS) neighborhood operators are designed to generate new high-quality solutions for the employed bee and onlooker bee phases, respectively. Meanwhile, a control parameter is applied to conduct the search of onlooker bee phase with the aim of further improving exploitation. Moreover, in order to enhance the exploration ability, a control acceptance criterion is employed to decide whether a new solution can be accepted. Furthermore, in the scout bee phase, three swap operators and three insert operators are performed with equal probability to the original solution corresponding to a scout bee and then replace the original one. Finally, the IABC is compared with some recent metaheuristics and a state-of-the-art SCC scheduling algorithm, the comparison results show that the IABC is effective.

The rest of this paper is organized as follows. Section 2 describes the SCC scheduling problem. In Section 3, the basic ABC algorithm is introduced. In Section 4, the proposed IABC is developed. In Section 5, test experiments and algorithm comparisons are given. Finally, the conclusions are summarized in Section 6.

2. The SCC Scheduling Problem

In the SCC scheduling problem, steelmaking, refining and continuous casting stages are responsible for different functions respectively [2]. More specifically, in the steelmaking stage, impurities such as nitrogen, silicon, sulfur and excess carbon are removed from the raw iron, while alloying elements such as manganese, nickel, chromium and vanadium are added to produce different grades of steel. In the refining stage, the impurities are further eliminated and the required alloy ingredients are added into molten steel to further improve the quality of the molten steel. Finally, the molten steel is solidified into slabs in the continuous casting stage [1, 2]. In the steelmaking and refining stages, charge is the basic production unit, corresponding to job in the flowshop scheduling, while in the continuous casting stage, cast is the basic unit. A cast is a set of charges which need to be continuously processed using the same crystallizer on the same caster and is determined in advance at planning level. This is due to the technological constraints.

The SCC scheduling problem is to determine an optimal schedule with minimal sojourn time, earliness of cast starting penalty, tardiness of cast starting penalty and so on [2], where the sojourn time of a charge is the duration between the completion time in the steelmaking stage and the starting time in the continuous casting stage. The optimal schedule specifies each charge is to be processed at what time and on which machine in the steelmaking stage, refining stage, and continuous casting stage [7]. A SCC process is a very complicated manufacturing operation, which can be characterized by high-temperature high-weight

material flow. Therefore, unlike general industrial scheduling problems, SCC scheduling problem has to meet a list of special requirements and constraints, some of which are described as follows [2].

- (1) Each charge is processed at most on one machine at a time, and each machine processes at most one charge at a time.
- (2) Each charge has to wait for processing if no machine can be available when it arrives the refining and continuous casting stages. However, wait will cause temperature drop and then reheating cost.
- (3) Multi-stage (Say $S-2$ stages) refining modes, i.e. there are $S-2$ stages for refining, usually exist in the refining, where each refining can be considered as a stage. With the steelmaking and continuous casting stages, in summary, there are S stages in the SCC scheduling problem.
- (4) Transfer times between two successive stages have to be considered.
- (5) The charges in a cast are prohibited to be interrupted in the continuous casting stage.
- (6) A cast should be started at the given time. Starting before the given time (called *earliness of cast starting*) and starting after the given time (called *tardiness of cast starting*) will result in inventory cost or delay cost.

Based on the problem description, the SCC scheduling problem with S stages can be formulated as a HFS model, which is described as follows [2], where the intermediate $S-2$ stages are utilized for refining.

Parameters:

M_i	Set of parallel machines at the i th stage, $i \in \{1, 2, \dots, S\}$.
$\Omega = \{1, 2, \dots, n\}$	Set of n charges.
$\Omega_l = \{z_{l-1} + 1, z_{l-1} + 2, \dots, z_l\}$	The l th cast in the continuous casting stage, where $z_{l-1} + 1, z_{l-1} + 2, \dots, z_l \in \Omega$, $z_{l-1} < z_l$, and $z_0 = 0$.
N	Total number of casts in the SCC scheduling problem. $\Omega = \Omega_1 \cup \Omega_2 \cup \dots \cup \Omega_N$, where $\Omega_{l_1} \cap \Omega_{l_2} = \phi$, $\forall l_1, l_2 \in \{1, 2, \dots, N\} \wedge l_1 \neq l_2$.
$B_k = \{\Omega_{b_{k-1}+1}, \Omega_{b_{k-1}+2}, \dots, \Omega_{b_k}\}$	Set of casts to be processed on the k th continuous caster, where $b_{k-1} + 1, b_{k-1} + 2, \dots, b_k \in \{1, 2, \dots, N\}$, $b_{k-1} < b_k$, and $b_0 = 0$.
PT_{ij}	Processing time of charge j at stage i , $j \in \Omega$ and $i \in \{1, 2, \dots, S\}$.
$TT_{i, i+1}$	Transfer time from stage i to $i+1$, $i \in \{1, 2, \dots, S-1\}$.
ST_l	Setup time of cast Ω_l , $l \in \{1, 2, \dots, N\}$.
RT_{ik}	Release time of machine k at stage i .
D_l	Starting time of cast Ω_l .
w_1	Penalty coefficient for average sojourn time.
w_2	Penalty coefficient for earliness of cast starting.
w_3	Penalty coefficient for tardiness of cast starting.
V	A very large positive number.

Decision variables:

s_{ij}	Starting time of charge j at stage i .
x_{ijk}	If charge j is assigned to machine k at stage i , $x_{i,j,k} = 1$; otherwise $x_{i,j,k} = 0$.
y_{i,j_1,j_2}	If charge j_1 is preceding charge j_2 to be processed at stage i , $y_{i,j_1,j_2} = 1$; if charges j_1 and j_2 start at the same time, $y_{i,j_1,j_2} = \frac{1}{2}$; otherwise $y_{i,j_1,j_2} = 0$. That is, $y_{i,j_1,j_2} = \begin{cases} 1 & s_{i,j_1} < s_{i,j_2} \\ \frac{1}{2} & s_{i,j_1} = s_{i,j_2} \\ 0 & \text{otherwise} \end{cases}$.

$$\min f = w_1 f_1 + w_2 f_2 + w_3 f_3 \quad (1)$$

$$f_1 = \sum_{j=1}^n (s_{sj} - (s_{1j} + PT_{1j})) / n \quad (2)$$

$$f_2 = \sum_{l=1}^N \max(0, D_l - s_{S, z_{l-1}+1}) \quad (3)$$

$$f_3 = \sum_{l=1}^N \max(0, s_{S, z_{l-1}+1} - D_l) \quad (4)$$

s.t.

$$\sum_{k \in M_i} x_{ijk} = 1, \forall j \in \Omega, i \in \{1, 2, \dots, S\} \quad (5)$$

$$s_{ij} - x_{ijk} \cdot RT_{ik} \geq 0, \forall j \in \Omega, k \in M_i, i \in \{1, 2, \dots, S-1\} \quad (6)$$

$$s_{i+1, j} - (s_{ij} + PT_{ij} + TT_{i, i+1}) \geq 0, \forall j \in \Omega, i \in \{1, 2, \dots, S-1\} \quad (7)$$

$$y_{i, j_1, j_2} + y_{i, j_2, j_1} = 1, \forall j_1, j_2 \in \Omega, i \in \{1, 2, \dots, S-1\} \quad (8)$$

$$s_{i, j_2} - (s_{i, j_1} + p_{i, j_1}) + V \cdot (3 - y_{i, j_1, j_2} - x_{i, j_1, k} - x_{i, j_2, k}) \geq 0, \forall j_1, j_2 \in \Omega, k \in M_i, i \in \{1, 2, \dots, S-1\} \quad (9)$$

$$s_{Sj} - x_{Sjk} \cdot RT_{Sk} - ST_{b_{k-1}+1} \geq 0, j = z_{b_{k-1}} + 1 \wedge j \in \Omega, k \in M_S \quad (10)$$

$$s_{Sj} + PT_{Sj} = s_{S, j+1}, \forall j, j+1 \in \Omega_l, l = 1, 2, \dots, N \quad (11)$$

$$s_{S, z_l+1} \geq s_{S, z_l} + PT_{S, z_l} + ST_{l+1}, \forall \Omega_l, \Omega_{l+1} \in B_k \quad (12)$$

$$x_{ijk} \in \{0, 1\}, j \in \Omega, k \in M_i, i \in \{1, 2, \dots, S\} \quad (13)$$

$$y_{i, j_1, j_2} \in \{0, 1/2, 1\}, j_1, j_2 \in \Omega, i \in \{1, 2, \dots, S\} \quad (14)$$

Formula (1) is the objective function, i.e. the weighted sum of the penalty for average sojourn time, earliness of cast starting, and tardiness of cast starting, which is denoted by f . Formulas (2)–(4) represent the average sojourn time, earliness of cast starting, and tardiness of cast starting respectively, where are denoted f_1 , f_2 and f_3 respectively. Formula (5) illustrates that a charge has to be processed in all stages, and at each stage a charge has to be processed by exactly one machine. Formula (6) describes that the starting time of a charge on a machine at the steelmaking and refining stages has to be greater than the release time of the machine. Formula (7) describes that for two consecutive operations of a charge, the latter operation can be started only after the former operation has been finished and has been transferred to the corresponding machine. Formulas (8) and (9) give machine capacity constraints, i.e., for two charges processed on the same machine, the latter charge can be started only after the former one has been finished. Formula (10) specifies the condition in which the first cast on a caster can be started. Formula (11) represents the casting continuity constraint for the charges in a cast. Formula (12) emphasizes that the setup time is needed between two consecutive casts on the same caster. Formulas (13) and (14) provide the value ranges of decision variables.

3. The Basic ABC Algorithm

The ABC algorithm is a recent swarm intelligence metaheuristic originally proposed for continuous function optimization problems [21, 29]. The ABC simulates the intelligent foraging behavior of honey-bee swarm. It starts from an initial population generated randomly, and then executes a cycle of employed bee phase, onlooker bee phase, and scout bee phase until the termination condition is reached. In the employed bee phase, each employed bee finds a new food source in its neighborhood and shares the information with the onlookers. In the onlooker bee phase, each onlooker bee selects a food source based on the probability selection and finds a new food source by utilizing

the same method as the employed bee phase. In the scout bee phase, for each scout bee, i.e. an employed bee corresponding to a food source has not been improved within a given number of successive iterations, replace the food source with one generated randomly. The ABC can be described as follows.

Procedure of the basic ABC

Initialize all the food sources randomly

Evaluate population

while termination condition is not satisfied

//Employed bee phase

For each employed bee x_i , find a new food source x_{new} in the neighborhood of its present position x_i . If $f(x_{new}) < f(x_i)$, $x_i = x_{new}$.

//Onlooker bee phase

For each onlooker bee, select a food source based on roulette wheel selection, and then find a new food source x_{new} in the neighborhood of its present position x_i . If $f(x_{new}) < f(x_i)$, $x_i = x_{new}$.

//Scout bee phase

For each scout bee, replace the food source with a food source generated randomly.

Update the best solution found so far

end while

4. The Proposed IABC

In this section, the proposed IABC is introduced in detail. First, the encoding and decoding scheme is described. Secondly, population initialization, employed bee and onlooker bee phases, two VNS neighborhood operators, and scout bee phase, are designed sequentially. Finally, the framework of the IABC is given.

4.1. Encoding and decoding

Charge permutation-based representation is applied to encode the solutions, which corresponds to the processing sequence of the charges. To evaluate the solutions, a decoding scheme is needed. In this paper, the decoding scheme in [2] is utilized, containing two sequential steps: (1) Forward decoding: at steelmaking and refining stages, select each charge one by one according to the sequence in the charge permutation, assign each selected charge to the first available machine, while at the continuous casting stage, for each cast, process the charges sequentially; (2) Backward decoding: in the continuous casting stage, for each charge except the last one in a cast, if there is a cast break between it and its following charge, shift the charge right to erase the cast break, and then in the steelmaking and refining stages, for each machine, shift the starting time of each charge. For example, for an instance with 10 charges, a solution may be encoded as Fig. 1. According to the decoding scheme, at steelmaking and refining stages, charges 9, 6, 1, 10, 5, 7, 4, 2, 3 and 8 are sequentially selected and assigned to the first available machine, then at the continuous casting stage, the charges belonging to the same cast are processed sequentially. Subsequently, backward decoding is executed to erase the cast break.

9	6	1	10	5	7	4	2	3	8
---	---	---	----	---	---	---	---	---	---

Fig. 1. Example of an instance with 10 charges.

4.2. Population initialization

In the basic ABC, the initial solutions are produced randomly, which may result in good exploration while poor exploitation. However, population initialization is important for the ABC, since initial population with certain quality may result in a faster convergence to good solutions [2]. In Pan *et al.* [2], an initial solution (say P_1) is generated by sorting the charges in the non-decreasing order of their starting times,

which corresponds to a good solution, while other initial solutions are generated randomly. In this paper, unlike the basic ABC and Pan *et al.* [2], to achieve a good balance between exploration and exploitation, *Num* initial solutions with certain quality are produced by a heuristic, while other *M-Num* initial solutions are generated randomly. More specifically, the steps are described as follows.

- Step 1 Obtain starting time for each charge at continuous casting stage according to the planned starting time of its cast, and then sort the charges in the non-decreasing order of their starting times. Denote the obtained charge permutation P_1 . Set $m = 2$.
- Step 2 Produce P_m by performing the pairwise exchange operator to P_1 .
- Step 3 Set $m = m + 1$, if $m = \text{Num} + 1$, go to Step 4; Otherwise, go to Step 2.
- Step 4 Generate the remaining *M-Num* initial solutions by randomly sequencing the charges.

4.3. Employed bee and onlooker bee phases

In the employed bee phase, the new solutions are generated by performing a VNS neighborhood operator (called VNS_1), which is to be described in Section 4.4 in detail. In the onlooker bee phase, for each onlooker bee, to select a food source with better quality, the tournament selection with the size of four is employed to select a solution in the population, and then the new solutions are generated by performing a VNS neighborhood operator (called VNS_2), which is also to be given in Section 4.4. Moreover, a control parameter is introduced to conduct the search of onlooker bee phase with the intention of intensifying exploitation, and the onlooker bee phase is performed *r* replications before the algorithm enters the scout bee phase, which is different from that of the basic ABC, i.e. performing the onlooker bee phase only once. Furthermore, unlike the basic ABC, the new produced solution replaces the original solution only and only if it is better than the original solution. In the proposed IABC, to enhance the exploration, slightly worse solutions can also be accepted to replace the original solution, which can improve the ability of escaping from local optima. The control acceptance criterion proposed by [33] is employed due to its simplicity and effectiveness, i.e. the new produced solution P_{new} replaces the original solution P_n if and only if $f(P_{\text{new}}) \leq bf(P_n)$, where *b* is a real number being close to 1 and is set to be 1.05 in this paper.

4.4. Two VNS neighborhood operators

When devising an ABC for the SCC scheduling problem, a neighborhood operator should be first defined for solution generation, since it is the core of the ABC and lays the foundation. Specifically, it generates new solutions for the employed bee and onlooker bee phases. Different neighborhood operators have been applied to the SCC scheduling problem, such as swap operator and pairwise exchange operator, they may have different performances for different instances. Therefore, they may be complementary, and integrating them appropriately may achieve better result. VNS is a metaheuristic which systematically exploits the idea of neighborhood change, and has been successfully applied to solve a series of combinatorial optimization problems [34]. In this paper, by tailoring the framework of VNS, we develop two VNS neighborhood operators, named VNS_1 and VNS_2 , which are employed to generate new solutions for the employed bee and onlooker bee phases respectively. The aim is to avoid repeated search and explore different search paths. More specifically, VNS_1 contains three neighborhood operators, i.e. pairwise exchange, swap and partially-mapped crossover (PMX), which are called dynamically in the evolutionary process. VNS_2 contains of three neighborhood operators, i.e. pairwise exchange, swap and insert.

Let P_n be the starting solution, P_{new} be the obtained solution by performing a neighborhood operator, i.e. pairwise exchange, swap, PMX or insert, P_{lb} be the local optima of the VNS_1 or VNS_2 , which will

be returned as the new solution for the employed bee phase or onlooker bee phase, and *MI* be the maximum number of iterations. The specific steps of VNS_1 and VNS_2 are listed as follows.

Procedure of VNS_1

```

 $k=1, P_{nc}=P_n$ 
for  $i=1$  to  $MI$ 
   $P_{\text{new}}=P_{nc}$ 
  if  $k=1$ 
     $P_{\text{new}}=P_{\text{new}} \oplus$  pairwise exchange
  else if  $k=2$ 
     $P_{\text{new}}=P_{\text{new}} \oplus$  swap
  else if  $k=3$ 
     $P_{\text{new}}=P_{\text{new}} \oplus$  PMX
  end if
  if  $f(P_{\text{new}}) < f(P_{nc})$ 
     $k=1$ 
     $P_{nc}=P_{\text{new}}$ 
     $P_{\text{lb}}=P_{\text{new}}$ 
  else
     $k=k+1$ 
    if  $k>3$ 
       $k=1$ 
    end if
    if  $f(P_{\text{new}}) < f(P_{\text{lb}})$ 
       $P_{\text{lb}}=P_{\text{new}}$ 
    end if
  end if
   $i=i+1$ 
end for
Return  $P_{\text{lb}}$ 

```

Procedure of VNS_2

```

 $k=1, P_{nc}=P_n$ 
for  $i=1$  to  $MI$ 
   $P_{\text{new}}=P_{nc}$ 
  if  $k=1$ 
     $P_{\text{new}}=P_{\text{new}} \oplus$  pairwise exchange
  else if  $k=2$ 
     $P_{\text{new}}=P_{\text{new}} \oplus$  swap
  else if  $k=3$ 
     $P_{\text{new}}=P_{\text{new}} \oplus$  insert
  end if
  if  $f(P_{\text{new}}) < f(P_{nc})$ 
     $k=1$ 
     $P_{nc}=P_{\text{new}}$ 
     $P_{\text{lb}}=P_{\text{new}}$ 
  else
     $k=k+1$ 
    if  $k>3$ 
       $k=1$ 
    end if
    if  $f(P_{\text{new}}) < f(P_{\text{lb}})$ 
       $P_{\text{lb}}=P_{\text{new}}$ 
    end if
  end if
   $i=i+1$ 
end for
Return  $P_{\text{lb}}$ 

```

4.5. Scout bee phase

If a solution in the current population (say P_u) has not been improved in *L* successive iterations, unlike the basic ABC where a solution is generated randomly in the search space, in this paper, a new solution is produced by executing perturbation to P_u , and then replace P_u by the new solution. This is because, P_u has survived many iterations and it may carry better information than others in the population during the search process, its neighboring solutions could be the most promising ones [2]. The perturbation is described as follows: generate a real number *rnd* in the range of [0,1] randomly, if *rnd* < 0.5, execute swap operator three times to P_u ; Otherwise, execute insert operator three times to P_u .

4.6. The framework of the proposed IABC

Let *PS* be the population size, *Num* be the number of initial solutions generated by heuristic, *L* be the maximum number of successive non-improving iterations, *r* be the replication times of the onlooker phase, *T* be the maximum iterations of the IABC, and P_b be

the best solution found so far. The framework of the IABC can be described as follows.

- Step 1 Population initialization.** Set $t = 1$. Generate an initial solution P_1 by the heuristic in Section 4.2, generate $Num-1$ initial solutions by executing the pairwise exchange operator to P_1 , and then generate $N-Num$ initial solutions randomly. And then evaluate the solutions and find the best solution among them (say P_{cb}), let $P_b = P_{cb}$.
- Step 2 Employed bee phase.** For each employed bee P_m in the population, generate a new solution P_{new} by VNS_1 . If P_{new} is better than P_b , let $P_m = P_{new}$, $P_b = P_{new}$. If the control acceptance criterion is satisfied, let $P_m = P_{new}$.
- Step 3 Onlooker bee phase.** Set $m = 1$. Select a solution (Say S_m) in the population for the onlooker bee by the tournament selection with the size of four, and set $m = m + 1$, repeat the steps until $m = M$. For each onlooker bee S_m , generate a new solution S_{new} by VNS_2 . If S_{new} is better than P_b , let $S_m = S_{new}$, $P_b = S_{new}$. If the control acceptance criterion is satisfied, let $S_m = S_{new}$. Repeat this Step until the onlooker bee phase is performed r times.
- Step 4 Scout bee phase.** If a solution in the population has not been improved within L successive iterations, produce a new solution S_{new} by executing three swap operators or three insert operators to it, and then replace it by the new solution. If S_{new} is better than P_b , let $P_b = S_{new}$.
- Step 5** Set $t = t + 1$. If the termination condition has been met (e.g. $t > T$ or CPU time is reached), go to Step 6. Otherwise go to Step 2.
- Step 6** Return P_b as the best solution.

According to the framework of the IABC, three main phases are contained in each iteration, including the employed bee phase, onlooker bee phase and scout bee phase. For each phase, a computational complexity analysis is made. More specifically, the computational complexity of the employed bee phase is $O(PS \times MI)$, where MI denotes the maximum number of iterations of VNS_1 and VNS_2 , the computational complexity of the onlooker bee phase is $O(PS \times MI)$, and the computational complexity of the scout bee phase is $O(PS)$.

5. Computational Results

The major concern about using metaheuristics is the quality of the obtained solution. To evaluate the performance of the IABC, test experiments and comparisons are conducted in this section. 30 instances are generated randomly according to the instance generation method stated in [2], which are 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. There are 3 converters, 5 refining furnaces, and 4 continuous casters. Each continuous caster processes 3–4 casts in each workday, and each cast generally contains 8–12 charges. Therefore, there are 12–16 casts and 120 charges or so in the scheduling instances. The processing times of each charge in the steel-making, refining and continuous stages are in the ranges of $PT_{1j} = 38$, $PT_{2j} \in [36, 50]$ and $PT_{3j} \in [38, 44]$ respectively. The release time of each machine is 0. The transfer times from the steelmaking phase to the refining phase and from the refining stage to the continuous stages are in the ranges of [10, 15].

The IABC algorithm is coded in C++ and run on a 3.10 GHZ computer with 4 G RAM using Windows 7 Operation System. Four algorithms are employed for comparison, i.e. the ABC algorithm [32–34], the Genetic Algorithm in Ruiz and Maroto (GA_R) [35], the Iterated Local Search (ILS) algorithm in Dong *et al.* [36], and a state-of-the-art SCC scheduling algorithm, i.e. Fruit fly Optimisation Algorithm (FOA) in Li *et al.* [8]. For each algorithm, the encoding and decoding in this paper is adopted. 20 independent runs for each instance are carried out, average result of the 20 runs is calculated. Relative Percentage

Increase (RPI) over the best average result found by the comparison algorithms is employed to assess and compare the performance of each algorithm. Specifically, the RPI is calculated as follows.

$$RPI = \frac{f_A - f_b}{f_b} \times 100\% \quad (15)$$

where f_A is the average result for algorithm A, i.e. f_{IABC} , f_{ABC} , f_{GA_R} , f_{ILS} , and f_{FOA} denote the average result for the IABC, ABC, GA_R , ILS, and FOA respectively, and f_b is the best average result found by the compared algorithms.

5.1. Parameter setting

Four main parameters are included in the IABC, i.e. the population size (PS), the maximum number of successive iterations a solution cannot be improved (L), the maximum number of iterations for VNS_1 and VNS_2 (MI), and the control parameter for onlooker bee phase (r). The Taguchi method of design of experiment (DOE) [37] is employed to decide desirable values for the parameters. The levels of the four parameters are listed in Table 1. An orthogonal array $L_{16}(4^4)$ is selected, and the corresponding 16 parameter combinations are shown in Table 2. For each parameter combination, the IABC is run 20 times independently for a randomly selected instance, i.e. instance 14. The average objective value of the 20 runs is calculated and also given in Table 2. According to Table 2, the factor level trend of the four parameters is illustrated in Fig. 2. Meanwhile, the significance value of each parameter for the instances is given in Table 3. According to this Table, one observes that PS is the most important parameter among the four, while r , L and MI rank the second, third and fourth, respectively. According to the above analysis, the suitable parameter values are as follows: $PS = 30$, $L = 100$, $MI = 10$, and $r = 3$.

Table 1
Combinations of the six parameter values

Parameter	Factor level			
	1	2	3	4
PS	10	30	50	100
L	30	50	70	100
MI	10	20	30	40
r	2	3	4	5

Table 2
Orthogonal array and average objective value

	Factor				Average objective value
	PS	L	MI	r	
1	10	30	10	2	7286.53
2	10	50	20	3	7397.38
3	10	70	30	4	7359.93
4	10	100	40	5	7189.66
5	30	30	20	4	7221.24
6	30	50	10	5	7234.10
7	30	70	40	2	7442.86
8	30	100	30	3	7123.49
9	50	30	30	5	7449.51
10	50	50	40	4	7428.09
11	50	70	10	3	7204.21
12	50	100	20	2	7233.91
13	100	30	40	3	7447.42
14	100	50	30	2	7553.51
15	100	70	20	5	7709.47
16	100	100	10	4	7675.84

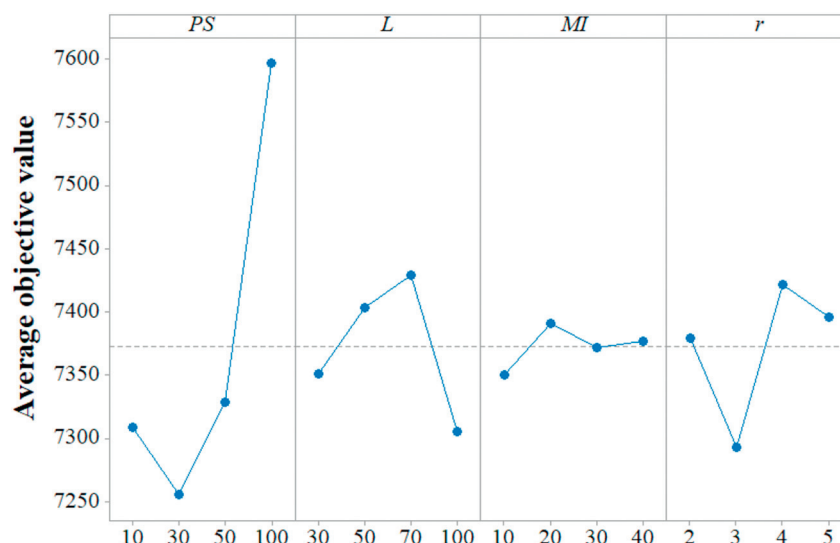


Fig. 2. Factor level trend of four parameters.

Table 3
Response value and significance rank

Level	PS	L	MI	r
1	7308	7351	7350	7379
2	7255	7403	7391	7293
3	7329	7429	7372	7421
4	7597	7306	7377	7396
Delta	341	123	40	128
Rank	1	3	4	2

5.2. Comparison with three metaheuristics

In this section, the IABC is compared with the ABC, GA_R , and ILS. To compare the four algorithms fairly, the same termination condition is used, i.e. CPU time 10 s. Moreover, for the GA_R and ILS, the parameter settings proposed in the corresponding references are adopted, while for the ABC, the parameter settings are the same as those of the IABC. Specific comparison results are reported in Table 4. From Table 4, one can observe that the IABC outperforms ABC, GA_R and ILS. Specifically, the IABC has the smallest average RPI , 0.49%, which is smaller than the average RPI of the ABC, GA_R and ILS, 13.35%, 10.24% and 2.69% respectively. Moreover, the IABC performs the best for 21 out of 30 instances. In addition, for the remaining 9 instances, the ABC also performs better than the ABC and GA_R , while it is slightly worse than the ILS. This clearly shows the effectiveness of the IABC. Furthermore, we can also find that the ILS performs better than the ABC and GA_R , while the ABC performs the worst.

In order to investigate whether the results obtained by the IABC differ from those by the compared algorithms in a statistically significant way, the Wilcoxon' rank sum tests with the 5% significance level are conducted for the IABC against ABC, IABC against GA_R , and IABC against ILS. The corresponding results (i.e. the values of h) are described in Table 5. $h = 1$ indicates that the results obtained by the former algorithm is significantly better than those by the latter one, while $h = 0$ indicates that the results by the two compared are not significantly different. It can be seen that the IABC produces 30, 30 and 20 significantly better values out of 30 instances than the ABC, GA_R , and ILS, respectively. This shows the superiority of the IABC.

To further demonstrate the strength of the IABC, the four algorithms are run with longer CPU time, i.e. 20 s. Corresponding comparison results are described in Table 6. From Table 6, it can be seen that the

IABC also performs better than the ABC, GA_R and ILS. Precisely, the IABC has the least average RPI 0.94%, which is smaller than the average RPI of the ABC, GA_R and ILS, 12.55%, 9.33% and 2.03% respectively. Additionally, the IABC reaches the best performance in 16 out of 30 instances. For the remaining 14 instances, the IABC also performs better than the ABC and GA_R , while it is slightly worse than the ILS. The Wilcoxon' rank sum tests with the 5% significance level are also conducted for the IABC against ABC, IABC against GA_R , and IABC against ILS. The corresponding results are reported in Table 7. According to Table 7, one can observe that the IABC has 30, 30 and 20 significantly better values out of 30 instances than ABC, GA_R , and ILS.

Table 4
Comparison results with CPU time 10 s

Instance	IABC	ABC	GA_R	ILS
1	3.26	11.36	8.98	0.00
2	0.71	2.22	1.30	0.00
3	0.00	9.14	8.83	0.02
4	0.00	12.62	8.83	7.34
5	0.00	19.11	17.57	13.90
6	0.00	16.60	5.19	0.92
7	0.00	13.97	9.15	0.90
8	0.00	16.19	10.10	2.22
9	0.00	8.13	6.94	3.80
10	0.00	6.54	5.10	0.84
11	0.00	13.27	11.78	5.08
12	0.00	27.02	21.71	6.78
13	0.00	12.03	10.36	8.93
14	0.00	24.99	23.93	1.92
15	0.04	6.77	5.07	0.00
16	0.00	19.56	15.53	1.65
17	0.00	18.53	11.46	3.23
18	0.81	8.78	7.00	0.00
19	0.00	0.14	0.12	0.00
20	0.00	13.46	11.86	5.47
21	0.00	1.26	0.84	0.00
22	6.22	21.28	15.96	0.00
23	3.16	20.50	14.68	0.00
24	0.00	6.76	6.19	0.88
25	0.00	8.77	7.61	2.63
26	0.26	25.67	18.18	0.00
27	0.00	20.79	15.90	7.75
28	0.14	9.54	8.04	0.00
29	0.00	11.92	6.99	6.41
30	0.21	13.58	11.95	0.00
Mean	0.49	13.35	10.24	2.69

Table 5

Rank sum test results for the compared algorithms with CPU time 20 s

Instance	IABC, ABC	IABC, GA _R	IABC, ILS
1	1	1	0
2	1	1	0
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	0
16	1	1	0
17	1	1	1
18	1	1	1
19	1	1	0
20	1	1	1
21	1	1	0
22	1	1	1
23	1	1	0
24	1	1	1
25	1	1	1
26	1	1	1
27	1	1	0
28	1	1	0
29	1	1	1
30	1	1	0

Table 7

Rank sum test results for the compared algorithms with CPU time 20 s

Instance	IABC, ABC	IABC, GA _R	IABC, ILS
1	1	1	1
2	1	1	1
3	1	1	0
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	0
16	1	1	0
17	1	1	0
18	1	1	1
19	1	1	0
20	1	1	1
21	1	1	0
22	1	1	1
23	1	1	0
24	1	1	0
25	1	1	1
26	1	1	1
27	1	1	0
28	1	1	1
29	1	1	1
30	1	1	0

5.3. Comparison with state-of-the-art SCC scheduling algorithm

In this section, the IABC is compared with a state-of-the-art SCC scheduling algorithm, i.e. FOA. Two termination conditions, i.e. CPU

time 10 s and 20 s, are tested respectively. Additionally, the parameter settings of the FOA is set the same as those in the corresponding reference. Comparison results are displayed in Table 8. According to Table

Table 6

Comparison results with CPU time 20 s

Instance	IABC	ABC	GA _R	ILS
1	5.92	12.15	10.54	0.00
2	1.44	2.74	1.84	0.00
3	0.17	8.59	7.69	0.00
4	0.00	11.18	7.51	6.52
5	0.00	17.03	14.93	10.65
6	0.00	13.59	3.37	0.55
7	0.00	11.94	6.45	0.19
8	0.00	15.10	8.31	1.96
9	0.00	7.09	5.77	1.88
10	0.00	5.43	3.81	0.58
11	0.00	11.52	9.52	4.44
12	0.00	23.44	18.18	4.54
13	0.00	11.14	9.61	8.54
14	0.00	26.49	26.40	3.89
15	3.65	9.31	7.07	0.00
16	0.43	17.80	12.69	0.00
17	0.01	15.84	9.89	0.00
18	1.74	8.88	6.98	0.00
19	0.16	0.27	0.24	0.00
20	0.00	12.15	10.86	4.03
21	0.15	1.19	0.57	0.00
22	7.80	21.57	15.52	0.00
23	3.65	16.63	13.42	0.00
24	0.00	5.43	4.85	0.07
25	0.00	10.73	9.28	3.12
26	0.27	25.02	15.51	0.00
27	0.00	19.12	13.18	7.72
28	1.91	11.40	9.85	0.00
29	0.00	11.06	5.10	2.19
30	0.92	12.59	10.95	0.00
Mean	0.94%	12.55%	9.33%	2.03%

Table 8

Comparison results with CPU time 10 s and 20 s

Instance	10 s		20 s	
	IABC	FOA	IABC	FOA
1	8.70	0.00	9.01	0.00
2	0.00	0.04	0.00	0.02
3	0.52	0.00	0.44	0.00
4	0.00	5.03	0.00	5.08
5	0.00	0.94	0.00	0.40
6	0.00	0.22	0.00	0.15
7	1.20	0.00	0.92	0.00
8	0.00	1.11	0.00	1.33
9	1.04	0.00	1.30	0.00
10	0.00	0.94	0.00	0.89
11	1.90	0.00	2.55	0.00
12	0.00	0.87	0.00	0.87
13	0.00	0.93	0.00	0.48
14	0.00	7.91	0.00	10.67
15	6.38	0.00	7.68	0.00
16	0.00	3.22	0.00	1.12
17	0.00	2.49	0.00	3.09
18	1.46	0.00	2.67	0.00
19	0.00	0.00	0.00	0.00
20	0.00	2.17	0.00	1.09
21	0.00	0.01	0.00	0.00
22	0.00	1.00	0.42	0.00
23	0.00	0.34	0.00	0.89
24	0.00	1.00	0.00	1.11
25	0.00	1.61	0.00	5.07
26	0.00	5.30	0.00	4.31
27	0.00	3.75	0.00	3.08
28	0.00	0.15	0.00	0.12
29	0.00	0.37	0.00	0.27
30	0.00	0.69	0.18	0.00
Mean	0.71%	1.34%	0.84%	1.34%

Table 9

Rank sum test results for the compared algorithms with CPU time 10 and 20 s

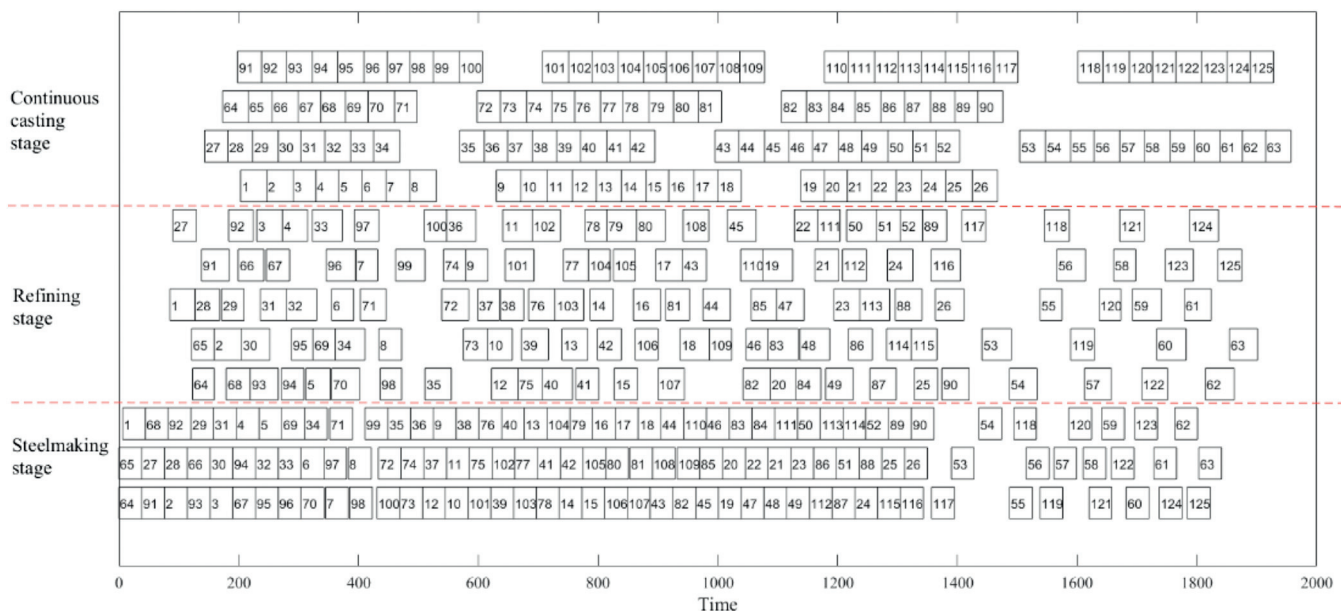
Instance	10 s	20 s
	IABC, FOA	IABC, FOA
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	0	0
8	1	1
9	0	0
10	1	1
11	0	0
12	0	0
13	1	1
14	1	1
15	0	1
16	1	0
17	1	1
18	0	0
19	1	1
20	1	1
21	0	0
22	0	0
23	1	1
24	1	1
25	1	1
26	1	1
27	1	1
28	0	0
29	1	0
30	1	0

8, it can be observed that the IABC outperforms the FOA under two termination conditions. Specifically, under CPU time 10 s and 20 s, the IABC has achieved average *RPI* of 0.71% and 0.84% respectively, while the FOA has achieved average *RPI* of 1.34% and 1.34% respectively. Moreover, under CPU time 10 s and 20 s, the IABC performs better than the FOA

for 23 and 21 out of 30 instances, respectively. The Wilcoxon' rank sum tests with the 5% significance level are also conducted for the IABC against FOA. The corresponding results are listed in Table 9. From Table 9, one can observe that compared with the FOA, the IABC produces significantly better solutions for 21 and 19 instances under CPU time 10 s and 20 s, respectively. Therefore, it is suggested that the IABC could be applied as an alternative approach to well solve the SCC scheduling problem. Furthermore, a Gantt chart for instance 1 obtained by the IABC is shown in Fig. 3. In Fig. 3, the steel-making, refining, and continuous casting stages are placed from the bottom to the top, where the three stages have 3, 5 and 4 machines respectively. More specifically, the values of f , f_1 , f_2 and f_3 are 5234.32, 92.432, 0 and 431, respectively.

6. Conclusions

In this paper, an effective IABC is developed for the SCC scheduling problem. In the IABC, a population initialization heuristic is devised, which can obtain several good initial solutions without compromising the randomness. Moreover, two VNS neighborhood operators are devised to generate high-quality solutions for the employed bee and onlooker bee phases, respectively. These solutions replace the original ones with a control acceptance criterion to enhance the exploration ability. A control parameter is devised to intensify the search of the onlooker bee phase. In the scout bee phase, the solution corresponding to a scout bee is replaced by a new solution, which is yielded by performing three swap operators and three insert operators with equal probability. Finally, comparison results with several metaheuristics and a state-of-the-art SCC scheduling algorithm have demonstrated the effectiveness of the IABC. Therefore, it could be applied as an alternative approach to well solve the SCC scheduling problem. Although this work is presented in terms of the SCC scheduling, it is suggested that, the IABC could also be applied to many other scheduling problems such as permutation flowshop scheduling and flexible job shop scheduling.

**Fig. 3.** Gantt chart of instance 1 obtained by the IABC.

References

- [1] H. Cui, X. Luo, An improved Lagrangian relaxation approach to scheduling steelmaking–continuous casting process, *Comput. Chem. Eng.* 106 (2017) 133–146.
- [2] Q. Pan, L. Wang, K. Mao, et al., An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process, *IEEE Trans. Autom. Sci. Eng.* 10 (2) (2013) 307–322.
- [3] I. Harjunkoski, I.E. Grossmann, A decomposition approach for the scheduling of a steel plant production, *Comput. Chem. Eng.* 25 (2001) 1647–1660.
- [4] D. Pacciarelli, M. Pranzo, Production scheduling in a steelmaking–continuous casting plant, *Comput. Chem. Eng.* 28 (12) (2004) 2823–2835.
- [5] Q. Pan, An effective co-evolutionary artificial bee colony algorithm for steelmaking–continuous casting scheduling, *Eur. J. Oper. Res.* 250 (3) (2016) 702–714.
- [6] H. Xuan, L. Tang, Scheduling a hybrid flowshop with batch production at the last stage, *Comput. Oper. Res.* 34 (9) (2007) 2718–2733.
- [7] L. Tang, J. Liu, A. Rong, et al., A mathematical programming model for scheduling steelmaking–continuous casting production, *Eur. J. Oper. Res.* 120 (2) (2000) 423–435.
- [8] J. Li, Q. Pan, K. Mao, et al., Solving the steelmaking casting problem using an effective fruit fly optimisation algorithm, *Knowl.-Based Syst.* 2 (2014) 28–36.
- [9] S. Yu, T. Chai, Heuristic scheduling method for steelmaking and continuous casting production process, *Control Theory A* 33 (11) (2016) 1413–1421.
- [10] S. Yu, Q. Pan, A rescheduling method for operation time delay disturbance in steelmaking and continuous casting production process, *J. Iron Steel Res. Int.* 19 (12) (2012) 33–41.
- [11] S. Yu, T. Chai, Y. Tang, An effective heuristic rescheduling method for steelmaking and continuous casting production process with multirefining modes, *IEEE Trans. Syst. Man Cybern. Syst. Hum.* 46 (12) (2016) 1675–1688.
- [12] K. Mao, Q. Pan, X. Pang, et al., A novel Lagrangian relaxation approach for a hybrid flowshop scheduling problem in the steelmaking–continuous casting process, *Eur. J. Oper. Res.* 236 (2014) 51–60.
- [13] K. Mao, Q. Pan, X. Pang, et al., An effective Lagrangian relaxation approach for rescheduling a steelmaking–continuous casting process, *Control Eng. Pract.* 30 (2014) 67–77.
- [14] J. Li, Q. Pan, K. Mao, A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems, *IEEE Trans. Autom. Sci. Eng.* 13 (2) (2016) 932–949.
- [15] L. Tang, Y. Zhao, J. Liu, An improved differential evolution algorithm for practical dynamic scheduling in steelmaking–continuous casting production, *IEEE Trans. Evol. Comput.* 18 (2) (2014) 209–225.
- [16] S. Jiang, M. Liu, J. Hao, et al., A bi-layer optimization approach for a hybrid flow shop scheduling problem involving controllable processing times in the steelmaking industry, *Comput. Ind. Eng.* 87 (2015) 518–531.
- [17] J. Hao, M. Liu, S. Jiang, et al., A soft-decision based two-layered scheduling approach for uncertain steelmaking–continuous casting process, *Eur. J. Oper. Res.* 244 (3) (2015) 966–979.
- [18] J. Long, Z. Zheng, X. Gao, Dynamic scheduling in steelmaking–continuous casting production for continuous caster breakdown, *Int. J. Prod. Res.* 55 (11) (2017) 3197–3216.
- [19] L. Cui, G. Li, Z. Zhu, et al., A novel artificial bee colony algorithm with an adaptive population size for numerical function optimization, *Inf. Sci.* 414 (2017) 53–67.
- [20] I. Khan, M.K. Maiti, A swap sequence based Artificial Bee Colony algorithm for Traveling Salesman Problem, *Swarm Evol. Comput.* (2018) 1–11.
- [21] D. Karaboga, B. Gorkemli, C. Ozturk, et al., A comprehensive survey: Artificial bee colony (ABC) algorithm and applications, *Artif. Intell. Rev.* 42 (1) (2014) 21–57.
- [22] X. Li, S. Ma, Multiobjective discrete artificial bee colony algorithm for multiobjective permutation flow shop scheduling problem with sequence dependent setup times, *IEEE Trans. Eng. Manag.* 64 (2) (2017) 149–165.
- [23] S. Sundar, P.N. Suganthan, C.T. Jin, et al., A hybrid artificial bee colony algorithm for the job-shop scheduling problem with no-wait constraint, *Soft. Comput.* 21 (5) (2017) 1193–1202.
- [24] N. Sharma, H. Sharm, A. Sharma, Beer froth artificial bee colony algorithm for job-shop scheduling problem, *Appl. Soft Comput.* 68 (2018) 507–524.
- [25] L. Wang, G. Zhou, Y. Xu, et al., An effective artificial bee colony algorithm for the flexible job-shop scheduling problem, *Int. J. Adv. Manuf. Technol.* 60 (2012) 303–315.
- [26] X. Li, Z. Peng, B. Du, et al., Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems, *Comput. Ind. Eng.* 113 (2017) 10–26.
- [27] J. Li, Q. Pan, Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm, *Inf. Sci.* 316 (2015) 487–502.
- [28] Q. Pan, M.F. Tasgetiren, P.N. Suganthan, et al., A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, *Inf. Sci.* 181 (12) (2011) 2455–2468.
- [29] D. Gong, Y. Han, J. Sun, A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems, *Knowl.-Based Syst.* 148 (2018) 115–130.
- [30] A. Yurtkuran, E. Emel, A discrete artificial bee colony algorithm for single machine scheduling problems, *Int. J. Prod. Res.* 54 (22) (2016) 6860–6878.
- [31] R. Zhang, P. Chang, S. Song, et al., A multi-objective artificial bee colony algorithm for parallel batch-processing machine scheduling in fabric dyeing processes, *Knowl.-Based Syst.* 116 (2017) 114–129.
- [32] D. Karaboga, B. Akay, A comparative study of Artificial Bee Colony algorithm, *Appl. Math. Comput.* 214 (1) (2009) 108–132.
- [33] D. Lei, X. Guo, Hybrid flow shop scheduling with not-all-machines options via local search with controlled deterioration, *Comput. Oper. Res.* 65 (2016) 76–82.
- [34] P. Hansen, N. Mladenović, J.A. Moreno Pérez, Variable neighbourhood search: Methods and applications, *Ann. Oper. Res.* 175 (1) (2010) 367–407.
- [35] R. Ruiz, C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, *Eur. J. Oper. Res.* 169 (3) (2006) 781–800.
- [36] X. Dong, H. Huang, P. Chen, An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion, *Comput. Oper. Res.* 36 (5) (2009) 1664–1669.
- [37] H. Zheng, L. Wang, X. Zheng, Teaching–learning–based optimization algorithm for multi-skill resource constrained project scheduling problem, *Soft. Comput.* 21 (6) (2017) 1537–1548.