An Improved Discrete Migrating Birds Optimization Algorithm for the No-Wait Flow Shop Scheduling Problem

SUJUN ZHANG, XINGSHEH GU, AND FUNA ZHOU

1School of Mechanical and Electrical Engineering, Henan Institute of Science and Technology, Xinxiang 453003, China
2Key Laboratory of Advanced Control and Optimization for Chemical Process, Ministry of Education, East China University of Science and Technology, Shanghai 200237, China
3Department of Electrical Automation, Shanghai Maritime University, Shanghai 200135, China

Corresponding author: Xing Sheng Gu (xsgu@ecust.edu.cn)

This work was supported in part by the National Nature Science Foundation of China under Grant 61573144, Grant 61973209, and Grant 61973120, and in part by the Scientific and Technological Project of Henan province under Grant 202102110282 and Grant 202102110281.

ABSTRACT

An improved discrete migrating birds optimization (IDMBO) algorithm is presented in this work to solve the no-wait flow shop scheduling problem (NWFFSP) with makespan criterion. In the algorithm, all of the solutions in population are treated as birds aligned in a V formation named the leader and followers. To guarantee the quality and the diversity of initial population, the leader is provided by the standard deviation heuristic (SDH) algorithm, and the rest (the followers) are generated randomly. Given that IDMBO is a neighborhood-based search algorithm, the quality of algorithm depends heavily on the neighborhood structures, where the two variants of the hybrid multi-neighborhood strategy, which are multiply neighborhood structures embedded in variable neighborhood search (VNS) in different forms, are performed to generate the neighborhood solutions for leader and followers, respectively. Furthermore, the population reset mechanism is performed after a given number iterations without improving the current solution. The local search (LS) method can further ameliorate the quality of solutions. The computational study is conducted to analyze the efficiency of IDMBO algorithm on benchmarks designed by Reeve and Taillard. And the comparison results are shown that the presented algorithm is superior to several existing algorithms for NWFFSP.

INDEX TERMS

Improved discrete migrating birds optimization algorithm, no-wait flow shop scheduling problem, hybrid multi-neighborhood strategy, population reset mechanism.

I. INTRODUCTION

The no-wait flow shop scheduling problem (NWFFSP) is an important branch of the flow shop scheduling problem, in which all jobs are processed successively in the same order on all machines; that is, the job is processed with no interruption until the processing is completed. Therefore, the job must be delayed on the first machine [1]. This problem has extensive industrial application background not only in traditional industries, such as steel rolling [2], chemical processing [3], and food industry [4], but also in modern industries, such as flexible manufacturing system and robotic cells, which all can be modeled as NWFFSP. Otherwise, it is a complex combinational optimization problem and has been proven to be NP-hard with more than two machines [5]. Solving the problem using exact algorithms is difficult when its scale is large. Thus, heuristic and meta-heuristic algorithms have been investigated to determine high-quality solution in reasonable computational time, especially for large-scale and complex optimization problems.

Heuristics are suitable for small-sized optimization problems or used to generate the initial solution in meta-heuristic algorithm, and they are usually simple and require less computing time. Many remarkable heuristic and meta-heuristic algorithms have been proposed in the literature to optimize NWFFSP with the minimization of makespan, total flow time, and other objectives. Aldowaisan and Allahverdi [6], Laha and Chakraborty [7], Ye et al. [8],
and Nailwal et al. [9] proposed heuristic and constructive heuristic algorithms to solve NWFSSP with the makespan criterion. In recent years, meta-heuristics, such as genetic algorithm (GA) by Seng et al. [10], differential evolution (DE) algorithm by Qian et al. [11], and variable neighborhood search (VNS) by Komaki and Malakooti [12], have been studied to solve the aforementioned problem. Recently, Grabowski and Pempera [13] presented local search (LS) algorithm and Pan [14] used a discrete particle swarm optimization (DPSO\textsubscript{VND}). Andre and Santosa [15] proposed the hybrid ant colony optimization (ACO) and LS (ACO–LS) algorithm to solve the same problem. Moreover, Deng et al. [16] proposed an effective co-evolutionary quantum GA to balance the exploration and exploitation of algorithm, where store-with-diversity and competitive co-evolution were designed; Meanwhile, Lin and Ying [17] provided two meta-heuristic algorithms, and the computational results showed that these algorithms outperformed existing ones. In particular, these algorithms can solve very hard and large NWFSSP to optimality. Then, Ding and Song [18] proposed an improved iterated greedy algorithm with a tabu-based reconstruction strategy (TMIIG) to solve NWFSSP with the makespan criterion, in which the VNS and acceptance criteria of new neighborhood solution are adopted to balance the global and LS abilities. What’s more, the results updated the upper bound of some instances. Shao et al. [19] presented a hybrid discrete optimization algorithm based on teaching-probabilistic learning mechanism under the frame of teaching and learning, which involves neighborhood search and probabilistic learning, as well as population reconstruction. Furthermore, it has comparable performance with some efficient algorithms in solving NWFSSP. In addition, Zhao et al. [20] proposed a discrete Water Wave Optimization (DWWO) algorithm to solve the same problem with minimization of makespan. Gao et al. [21] developed an effective heuristic algorithm in which two constructive heuristics and four composite heuristics are presented for the total flow time criterion. Moreover, for the NWFSSP with optimization the total flow time, Xia and Li [22] proposed iterative search method, and Laha et al. [23] suggested to penalty-shift-insertion-based algorithm. Besides, Akhshabi et al. and Gao et al. performed HPSO algorithm [24] and discrete harmony search algorithm [25], respectively; and Chaudhry et al. [26] developed the GA algorithm. Ying et al. [27] proposed a self-adaptive ruin-and-recreate algorithm. Additionally, Gao et al. [28] presented enhanced migrating birds optimization (MOBO) algorithm. Moreover, Qi et al. [29] proposed the fast local neighborhood search algorithm. However, Lin et al. [30] provided a cloud theory-based iterated greedy (CTIG) algorithm to minimize makespan and total flow time for NWFSSP. Enumeration algorithm [31] and improved exact methods [32] were presented to address NWFSSP based on the total tardiness criterion. Furthermore, a parato-based estimation of distribution algorithm is proposed for solving multi-objective distributed NWFSSP with sequence-dependent setup time [33].

In addition, the MBO algorithm [34], which can be applied in various fields, such as for solving credit card fraud detection [35], for flow shop scheduling problem [36–38], has attracted many researcher’s attention and obtained remarkable optimization results. So, there is still a lot of research space for its application on the others scheduling problem. Thus, this work proposes an improved discrete MBO (IDMBO) algorithm for the NWFSSP with minimization makespan. The specific steps of the algorithm are provided in the following sections.

The remaining contents are organized as follows. Section 2 provides the formulation of NWFSSP. The MBO algorithm and IDMBO algorithm are presented in section 3 and section 4, respectively. Section 5 presents the experiment design, parameter selection, and algorithm simulation result analysis. Finally, Section 6 draws some useful conclusions.

## II. FORMULATION OF NWFSSP

NWFSSP can be described as follows. Each of the n jobs from the set \( J = \{1, 2, \cdots, n\} \) is processed in the same order on all machines \( M = \{1, 2, \cdots, m\} \) without interruption; that is, the processing should not be interrupted once each job begins to be processed. The operation \( O_{j,k} \) denotes the job \( j (j = 1, 2, \cdots, n) \) processed on machine \( k (k = 1, 2, \cdots, m) \), and the corresponding processing time is \( p_{j,k} \). The setup times are ignored or included in the processing time \( p_{j,k} \). The machining process flows indicate that all jobs are available at the beginning of the processing, whereas all machines are continuously available. At any moment, each machine can process one job at most, and every job can be processed by one machine at most. In view of the no-wait restriction, any two adjacent operations on two machines must not have a waiting time. That is, the job must be delayed on the first machine when necessary. In this work, the optimization goal is to find a feasible permutation \( \pi \) of \( n \) jobs to minimize maximum completion time (i.e., makespan), which is equal to the finishing time of the last job on the last machine.

Given that the scheduling is \( \pi = \{\pi_1, \pi_2, \cdots, \pi_n\} \), it shows the sequence of jobs to be processed on all machines. Let \( E(\pi_{j-1}, \pi_j) \) be the minimum delay on the first machine between the beginning time of two jobs \( \pi_{j-1} \) and \( \pi_j \) (\( \pi_{j-1} \) and \( \pi_j \) conform to the no-wait restriction.) when \( \pi_j \) is directly processed after job \( \pi_{j-1} \). Subsequently, the completion time \( C(\pi_j, m) \) of job \( \pi_j \) on the last machine \( m \) can be computed by the following formula:

\[
C(\pi_1, m) = \sum_{k=1}^{m} p_{\pi_1,k} \quad (1)
\]

\[
C(\pi_j, m) = \sum_{i=2}^{j} E(\pi_{i-1}, \pi_i) + \sum_{k=1}^{m} p_{\pi_j,k} \quad j = 2, 3, \cdots, n
\]

And \( E(\pi_{j-1}, \pi_j) \) can be computed as follows:

\[
E(\pi_{j-1}, \pi_j)
\]
\[ C_{\text{max}} = C(\pi_0, m) \] (4)

Let \( \Pi \) denotes the set of all feasible scheduling of jobs on machine, and the goal is to find the one \( \pi^* \) from \( \Pi \) such that:

\[ C_{\text{max}}(\pi^*) \leq C_{\text{max}}(\pi), \quad \pi \in \Pi \] (5)

III. MBO ALGORITHM

Duman et al. [34] proposed a new nature-inspired metaheuristic algorithm called MBO, which is a population-based intelligent optimization algorithm, to solve the quadratic assignment problem and was usually adapted to address production scheduling problem. The optimization procedure begins with the first solution (leader bird) improved by its neighbors and progresses on two lines (following birds) to the tails, which are improved by their neighborhood solutions and the best unused in the front. After repeating the aforementioned procedure given several times (tours), the leader bird moves to the end line, and a bird that follows in the corresponding line takes its place. That is, the leader bird is updated. The new loop begins until the termination condition is met.

Figure 1 shows the algorithm flowchart, where the solutions denote the birds in the flock of the MBO algorithm, and the notations are detailed as follows. The parameters include the number of the solutions (\( N \)), the number of iterations before changing the ‘leader’ solution (\( G \)), and the maximum number of iterations (\( T_{\text{max}} \)). Neighbors to be considered (\( \beta \)) and the number of neighbors that are shared with the next solution (\( \chi \)) are also the parameters to be considered.

IV. IDMBO ALGORITHM FOR THE NWFSSP

The IDMBO algorithm is presented to solve NWFSSP with minimizing the makespan. In what follows, we proceed with the detailed description of the algorithm, and the specific steps of the algorithm are given subsequently.

A. ENCODE AND INITIALIZE POPULATION

The encoding scheme plays important role during the designing of the algorithm. In this work, permutation-based job representation is used to represent the individuals (birds) of the population, that is, the solutions are coded with as vectors of size \( n \).

\[ = p_{\pi_{j-1}, 1} + \max \left[ 0, \max_{k=2, \ldots, m} \left\{ \sum_{h=2}^{k-1} p_{\pi_{j-1}, h} - \sum_{h=1}^{k-1} p_{\pi_j, h} \right\} \right] \] (3)
The one with minimum makespan value is selected as the current sequence.

Step 3. Take out the job \( \pi_j, j = 3, 4, \cdots, n \) and insert it into every slots of the current partial sequence that has already scheduled. Then, find the best partial sequence for the next loop.

Step 4. Repeat Step 3 until the complete scheduling is constructed.

B. HYBRID MULTI-NEIGHBORHOOD STRATEGY

MBO algorithm is a neighborhood-based search algorithm and has the unique sharing mechanism among individuals that is specific to this algorithm. The leader improves by evaluating a number of its own neighbors, but, the followers improve by evaluating a number of its own neighbors and a number of the best unused neighbors from its previous solution. The best neighbor solution will replace the leader or the followers if it is better. Hence, the choice of the neighborhood structure will directly affect the effectiveness of the algorithm.

The job movement in the sequence (solution) is the most direct and effective pattern for generating the neighborhood solution for the incumbent sequence (solution). The insertion and swapping operators are highly effective for solving the flow shop scheduling problem. In this section, the effective combination, which includes the other two neighborhood structures and the two aforementioned structures, is used to generate the neighboring solution for the leader and followers in the flock. These structures are detailed as follows, given seven jobs in the permutation.

1) INSERT NEIGHBORHOOD
Two jobs from different slots \( r_1 \) and \( r_2 \) in permutation (solution) are randomly selected, and one \( r_1 \) is removed and inserted into another slot \( r_2 \). Figure 2 shows the insert operator.

2) SWAP NEIGHBORHOOD
Two jobs from different slots \( r_1 \) and \( r_2 \) in the permutation (solution) are randomly selected and exchanged. Figure 3 shows the swap operator.

3) TURN NEIGHBORHOOD
Two different slots \( r_1 \) and \( r_2 \) are randomly selected, and the order of jobs between the two slots is reversed. Figure 4 shows the turn operator.

4) SEQUENCE OF EXCHANGE NEIGHBORHOOD
Two different slots \( r_1 \) and \( r_2 \) are randomly selected, and the order of jobs between the two slots, including themselves, is reversed. Figure 5 shows the sequence of the exchange operator.

C. LOCAL SEARCH

To further enhance the exploitation of IDMBO, the local search algorithm is employed to the one selected randomly from the best three solutions of the population because they carried more useful information for the overall searching process. If the local search algorithm is executed on all individual of the population, it will take a lot of computational time and easily lead to the diversity of population destroyed because all of the solutions improve towards the same location in search space. Whereas, the local search algorithm is only performed on the best solution of the population, it is likely
**Procedure** The Variant 1 of Hybrid Multi-Neighborhood Strategy

```
Input: The original permutation \( \pi \)
output: The acquired neighborhood solution \( \pi \)
for \( k = 1:4 \)
switch (k)
case 1 employ swap operator to \( \pi \), and get \( \pi' \)
    if \( C_{\text{max}}(\pi') < C_{\text{max}}(\pi) \)
        \( k = 1; \pi = \pi' \);
    else
        \( k = k + 1 \);

case 2 employ insertion to \( \pi \), and get \( \pi' \)
    if \( C_{\text{max}}(\pi') < C_{\text{max}}(\pi) \)
        \( k = 2; \pi = \pi' \);
    else
        \( k = k + 1 \);

case 3 employ turn operator to \( \pi \), and get \( \pi' \)
    if \( C_{\text{max}}(\pi') < C_{\text{max}}(\pi) \)
        \( k = 3; \pi = \pi' \);
    else
        \( k = k + 1 \);

case 4 employ sequence of exchange operator to \( \pi \), and get \( \pi' \)
    if \( C_{\text{max}}(\pi') < C_{\text{max}}(\pi) \)
        \( k = 4; \pi = \pi' \);
    else
        \( k = k + 1 \);
end
```

that the best solution is not able to further improve and stick to the suboptimum. In this section, the VNS is embedded as local search algorithm in this work, which contains insertion and pair-wise swap two neighborhood structures, because the insert and pair-wise swap are the effective patterns to generate neighborhood solution for NWFSSP [18]. Two neighborhood structures in VNS are executed sequentially. When one cannot improve the solution, the other is replaced until neither can improve the solution.

**D. POPULATION RESET SCHEME**

MBO has strong exploitation ability and is prone to premature convergence. To help MBO escape from the suboptimum, the reset scheme is employed to population. For every individual in the population, the age is used to represent the updating process that is survives. The age variable initializes as zero. During the iteration, if the individual cannot be improved, then the age is increased by 1 and compared with the given limit. When the age of the individual exceeded to the limit, that is, when the individual becomes old enough, a new solution is produced by performing the destruction and reconstruction of the iterated greedy (IG) algorithm [40] to the individual and put it into the population instead of the original, as well as by setting the age by zero to escape the local optimal and without affecting the optimization of the algorithm. If the individual is improved, the age resets to zero. In this way, the algorithm will escape the local optimal and continue to search for remarkable solution.

**E. SPEED-UP METHODS TO CALCULATE MAKESPAN**

As can be seen from the description of algorithm, the hybrid multi-neighborhood strategy used to produce the neighborhood solution, the IG algorithm and the local search algorithm are used to generate the new permutation during searching the optimal solution. However, the multiply calculation makespan of new solution will consume significant amount of CPU time. But since they are based on the insertion, swap, turn neighborhood and sequence of exchange neighborhood which just changed the part of original solution. When compute makespan of the new solution, only need to consider the impact of the changes on makespan. This can greatly reduce the CPU time. Insert operator is used to all of the structures above mentioned, so it has the greatest impact on the CPU time consuming, and the second is swap operator. The speed-up methods of four neighborhoods are givens as follows. For the sake of simplicity, only the case of \( r_1 < r_2 \) is given, and \( r_1 > r_2 \) is omit.

Where \( \pi' \) and \( \pi \) are the new and the original sequences, respectively. But \( r_1 \) and \( r_2 \) are two different slots in sequence, and where \( T(\pi_n) \) and \( T(\pi'_n) \) are given as follows:

\[
T(\pi_n) = \sum_{k=1}^{m} p(\pi_n, k) \tag{6}
\]

\[
T(\pi'_n) = \sum_{k=1}^{m} p(\pi'_n, k) \tag{7}
\]

Speed-up method to calculate makespan for insert operator:

if \( r_1 = 1, r_2 = n \)
\[
C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_{r_1}, \pi_{r_1+1}) + E(\pi'_{r_2-1}, \pi'_{r_2}) - T(\pi_{r_2}) + T(\pi'_{r_2}) \tag{8}
\]

if \( r_1 = 1, 1 < r_2 < n \)
\[
C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_{r_1}, \pi_{r_1+1}) - E(\pi_{r_2}, \pi_{r_2+1}) + E(\pi'_{r_2-1}, \pi'_{r_2}) + E(\pi'_{r_2}, \pi'_{r_2+1}) \tag{9}
\]

if \( 1 < r_1 < n, 1 < r_2 < n \)
\[
C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_{r_1-1}, \pi_{r_1}) - E(\pi_{r_2}, \pi_{r_2+1}) + E(\pi'_{r_2-1}, \pi'_{r_2}) + E(\pi'_{r_2}, \pi'_{r_2+1}) \tag{10}
\]

if \( 1 < r_1 < n, r_2 = n \)
\[
C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_{r_1-1}, \pi_{r_1}) - E(\pi_{r_1}, \pi_{r_1+1}) + E(\pi'_{r_2-1}, \pi'_{r_2}) + E(\pi'_{r_2}, \pi'_{r_2+1}) + T(\pi_{r_2}) - T(\pi'_{r_2}) \tag{11}
\]

Speed-up method to calculate makespan for swap operator:

if \( r_1 = 1, r_2 = n \)
\[
C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_{r_1}, \pi_{r_1+1}) - E(\pi_{r_2-1}, \pi_{r_2})
\]
if $r_1 = 1$, $r_2 = 2$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_1, \pi_1) - E(\pi_2, \pi_2) + T(\pi_2) - T(\pi_2) \quad (12) \]

if $r_1 = 1, 2 < r_2 < n$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_1, \pi_1) - E(\pi_2, \pi_2) - E(\pi_2, \pi_2) + E(\pi_1, \pi_1') + E(\pi_2', \pi_2') \quad (13) \]

if $1 < r_1 < n, 1 < r_2 < n, r_2 = r_1 + 1$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_1, \pi_1) - E(\pi_2, \pi_2) - E(\pi_2, \pi_2) + E(\pi_1, \pi_1') + E(\pi_2', \pi_2') \quad (14) \]

if $1 < r_1 < n, 1 < r_2 < n, r_2 \neq r_1 + 1$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_1, \pi_1) - E(\pi_2, \pi_2) - E(\pi_2, \pi_2) + E(\pi_1, \pi_1') + E(\pi_2', \pi_2') \quad (15) \]

if $1 < r_1 < n, r_2 = n$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_1, \pi_1) - E(\pi_2, \pi_2) - E(\pi_2, \pi_2) + E(\pi_1, \pi_1') + E(\pi_2', \pi_2') \quad (16) \]

if $r_1 = n - 1, r_2 = n$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_1, \pi_1) - E(\pi_2, \pi_2) + E(\pi_1, \pi_1') + E(\pi_2', \pi_2') \quad (17) \]

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - E(\pi_1, \pi_1) - E(\pi_2, \pi_2) + E(\pi_1, \pi_1') + E(\pi_2', \pi_2') + T(\pi_2) - T(\pi_2) \quad (18) \]

Speed-up method to calculate makespan for turn operator:

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - \sum_{j=r_1+1}^{r_2} E(\pi_j, \pi_j) + \sum_{j=r_1+1}^{r_2} E(\pi_j', \pi_j') \quad (19) \]

Speed-up method to calculate makespan for sequence of exchange neighborhood:

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - \sum_{j=2}^{r_2+1} E(\pi_j, \pi_j) + \sum_{j=2}^{r_2+1} E(\pi_j', \pi_j') \quad (20) \]

if $1 < r_1 < n, 1 < r_2 < n$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - \sum_{j=r_1+1}^{r_2+1} E(\pi_j, \pi_j) + \sum_{j=r_1+1}^{r_2+1} E(\pi_j', \pi_j') \quad (21) \]

if $1 < r_1 < n, r_2 = n$

\[ C_{\text{max}}(\pi') = C_{\text{max}}(\pi) - \sum_{j=r_1+1}^{r_2} E(\pi_j, \pi_j) + \sum_{j=r_1+1}^{r_2} E(\pi_j', \pi_j') - T(\pi_2) + T(\pi_2) \quad (22) \]

\[ C_{\text{max}}(\pi') = \sum_{j=2}^{n} E(\pi_j, \pi_j') + T(\pi_2') \quad (23) \]

F. PROCEDURE OF IMPROVED DISCRETE MBO

Step 1: Set the parameters, and initialize the population, let $g = 1$.

Step 2: Leader improvement. The leader improved by $\beta$ neighboring solutions generated by performing Variant 1 of the hybrid multi-neighborhood strategy. In addition, the unused neighboring solutions are arranged in ascending order according to the makespan and then assigned to two assemblies $\Lambda_l$ and $\Lambda_r$.

Step 3: Improvement of followers in the left line. Each individual is improved by $\beta - \chi$ neighboring solutions generated by using Variant 2 of the hybrid multi-neighborhood strategy and $\chi$ solutions from assemble $\Lambda_l$. Then, the $\chi$ unused neighboring solutions are refilled to $\Lambda_l$.

Step 4: Improvement of followers in the right line. Each individual is improved by $\beta - \chi$ neighboring solutions generated by using Variant 2 of the hybrid multi-neighborhood strategy and $\chi$ solutions from assemble $\Lambda_r$. Then, the $\chi$ unused neighboring solutions are refilled to $\Lambda_r$.

Step 5: If $g < G$, then go to Step 2; otherwise, go to Step 6.

Step 6: Implement the LS to the one selected randomly from the three best individuals of the population.

Step 7: Update the age of every individual in the population.

Step 8: Age check. Check the age of each solution. Perform reset mechanism on the solution if its age is larger than the given limit.

Step 9: Update the best so far.

Step 10: Update the leader.

Step 11: Termination check. If the termination condition is not met, then go to Step 2; otherwise, stop and output the best so far.

V. EXPERIMENT AND RESULT ANALYSIS

A. EXPERIMENT SITUATION

All the experiments are executed on the MATLAB platform in a computer with 64-bit Windows 8 operating system, six-core...
Intel® Core™ i5-9600KF processor, 3.7 GHz CPU, and 8.00 GB RAM.

To test the performance of the IDMBO algorithm for the NWFSSP with the makespan criterion, we discuss the algorithm on 141 instances, which contains two subset: (i) 21 benchmarks with seven different sizes, namely, Rec01 and Rec03-Rec41 by Reeves [41]; (ii) 120 benchmarks range from $20 \times 5$ to $500 \times 20$ by Taillard [42]. To demonstrate the performance and effectiveness of IDMBO algorithm, it is compared with not only several existing methods but also its three variants based on these benchmarks.

### B. PARAMETERS FINE TUNING

Parameter selection greatly influences all intelligent optimization algorithms. Seven parameters, namely, population size ($N$), the neighbor solutions to be considered ($\beta$), the size of neighbors to be shared with the following birds ($\chi$), the number of tours ($G$), the maximum generation ($T_{\text{max}}$), destruction size ($d$) in the destruction and construction of IG, and the given limit of age ($\text{limit}$), are needed to be designed in IDMBO algorithm. Three of these parameters can be referenced in the literature [34]. When parameters $N$, $\beta$, and $\chi$ are set as 51, 3, and 1, respectively, it shows the better performance. Furthermore, parameters $G$, $d$, $\text{limit}$, and $T_{\text{max}}$ are set to three reasonable levels {5, 10, 15}, {4, 6, 8}, {10, 20, 30}, and {300, 400, 500}, respectively, listed in Table 1. The above four parameters are named four factors, and everyone has three levels. And it designed via Taguchi method of designed of experiment (DOE) [43] to determine the suitable parameter settings for the IDMBO algorithm (Table 1). It needs only $3^2 = 9$ arrays, according to the orthogonal experiment, unlike $3^4 = 81$ in the factorial experiments. Therefore, the computational time is greatly reduced without compromising the quality of parameter selection.

#### TABLE 1. Levels of the parameters for IDMBO algorithm.

| FACTORS | LEVEL1 | LEVEL2 | LEVEL3 |
|---------|--------|--------|--------|
| $G$ (1) | 5      | 10     | 15     |
| $d$ (2) | 4      | 6      | 8      |
| $\text{limit}$ (3) | 10 | 20 | 30 |
| $T_{\text{max}}$ (4) | 300 | 400 | 500 |

We tested the IDMBO algorithm on a benchmark problem by Reeves [41], namely, Rec27 ($30 \times 15$). Table 2 provides the orthogonal table $L_9(3^4)$, which includes nine group parameter set samples. The average value(AVG) of makespan of 10 times independent run for each group parameters are listed in the last column of Table 2, and the factor level curve of each parameter is presented in Figure 6. Furthermore, $k_i$ in column $j$ denotes the mean value of three groups, including factors at level $i$, where $i \in \{1, 2, 3\}$ and $j \in \{1, 2, 3, 4\}$, where the smallest $k_i$ values in each column are given in boldface, indicating that selecting the level value of the parameter can get the smaller makespan. That is, the algorithm can work well under this selection. In the last row of Table 2, the $SD$ values of the parameters are provided in descending order in parentheses after $SD$, where the $SD$ is the standard deviation of $k_1, k_2, k_3$ value of each column. And it shows the extent to which the sample deviates from the mean. It can compute according to the below formula (24).

$$SD = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (C_{ki} - C_{k_{\text{Avg}}})^2} \quad (24)$$

where $n$ is 3, and $C_{ki}$ is the value in the row $i$, and $C_{k_{\text{Avg}}}$ is the average of the $C_{ki}$, $i = 1, 2, 3$.

The larger $SD$ value indicates that the parameter has greater influence on the performance of the algorithm. By contrast, the smaller $SD$ value has less influence. That is, it displays the relative importance for the IDMBO algorithm to solve NWFFSP. It can be observed that the parameter $d$ has the greatest influence. Otherwise, $G$ has the least influence from Table 2. And it can be seen that the algorithm can achieve remarkable optimization when parameters $G$, $d$, $\text{limit}$, and $T_{\text{max}}$ are set to 5, 8, 10, and 500, respectively, from Table 2 and Figure 6. Therefore, these parameter values are selected for the IDMBO algorithm on the latter experiments.

![Figure 6. Factors level curve of IDMBO.](image)

![Table 2. Orthogonal parameter table L_9(3^4) and AVG values.](table)

| TEST | FACTOR | AVG |
|------|--------|-----|
| $G$  | 4(1)   | 3467.6 |
| 5(1) | 4(1)   | 3461.7 |
| 6(2) | 4(1)   | 3459.9 |
| 7(3) | 4(1)   | 3472.3 |
| 8(2) | 4(1)   | 3470.6 |
| 9(2) | 4(1)   | 3453.8 |
| $d$  | 4(1)   | 3483  |
| 5(1) | 4(1)   | 3483  |
| 6(2) | 4(1)   | 3483  |
| 7(3) | 4(1)   | 3483  |
| $\text{limit}$ | 10(1) | 5(1)7(2) | 1.69(3) |
| $T_{\text{max}}$ | 300(1) | 3457.2 |

99386 VOLUME 8, 2020
C. SIMULATION EXPERIMENTS OF IDMBO AND COMPAISONS

In this section, experiments are carried out on 141 with different size instances to test the performance of the algorithm described in Section IV. The instances contain two sets, (i) 21 instances designed by Reeves [41]: Rec01 to Rec41 that are divided into seven subsets, (ii) 120 instances designed by Taillard [42]: Ta001 to Ta120 that range from 20 jobs and 5 machines to 500 jobs and 20 machines. Each considered algorithm is executed on Reeves and Taillard benchmarks in 5 independent replications for each instance. Furthermore, three indicators $\Delta_{\text{min}}$, $\Delta_{\text{Avg}}$, and $SD$ are employed to evaluate the quality of experiment results and comparison performance of the compared algorithms, where $\Delta_{\text{min}}$ and $\Delta_{\text{Avg}}$ refer to the best and average relative percentage errors, respectively, to the lower bound found so far ($C_{\text{opt}}$). Given that the theoretical optimal solution of NWFSSP is constantly updated, the best known makespan $C_{\text{opt}}$ for every instances found in literature [19] that preceded in literature [18] is employed. $\Delta_{\text{min}}$ and $\Delta_{\text{Avg}}$ can be computed by following Equations (25) and (26), whereas the $SD$ is given as follows by Formula (27), which indicates the robustness of the algorithms.

\[
\begin{align*}
\Delta_{\text{min}} &= \frac{C_{\text{min}} - C_{\text{opt}}}{C_{\text{opt}}} \times 100 \\
\Delta_{\text{Avg}} &= \frac{C_{\text{Avg}} - C_{\text{opt}}}{C_{\text{opt}}} \times 100 \\
SD &= \sqrt{\frac{\sum_{i=1}^{10} (C_i - C_{\text{Avg}})^2}{n}}
\end{align*}
\]

For the makespan criterion, $C_{\text{min}}$ and $C_{\text{Avg}}$ are the minimum and mean values, respectively, in 5 independent runs for the relevant instance. Nevertheless, $SD$ denotes standard deviation, in which $C_i$ indicates the result produced in the $i$th run by a specific algorithm.

1) EFFECTIVENESS OF ALGORITHM COMPONENTS

To test the contribution of proposed algorithm components, three variants of IDMBO are verified. Table 3 lists the computational results ($\Delta_{\text{Avg}}$ and $SD$) by every variants of IDMBO ant itself for Taillard’s instances. We test IDMBO_NL algorithm to demonstrate the effect of the local search in the IDMBO algorithm, which removes the local search from IDMBO. And the IDMBO_NR algorithm is also tested to demonstrate the effect of the population reset mechanism, which removes the population reset component when the individual is old enough. Besides, the IDMBO and IDMBO3 are tested together with them. In proposed IDMBO algorithm, four neighborhood structures are included in hybrid multi-neighborhood strategy, but the slight difference between the sequence of exchange and turn neighborhood depends whether the exchange contains slots r1 and r2. Hence, the IDMBO3 algorithm, where contains three neighborhood structures in hybrid multi-neighborhood strategy taking out the turn as one variant of IDMBO algorithm is tested together with the above two variants.

From Table 3, it can be seen that the IDMBO is superior to its three variants, which indicates that the reset mechanism, local search and the hybrid multi-neighborhood strategy including four neighborhood structures all can improve the performance of IDMBO to some extent. In particular, the IDMBO is much better than the variant IDMBO_NR, which indicates that the population reset mechanism greatly improves the performance of the algorithm. In addition, the performance of IDMBO_NL outperforms the IDMBO_NR. Its reason may be that the iterated greedy algorithm (IG) as population reset mechanism can enhance the global searching ability of IDMBO algorithm and avoid falling into local optimal. However, from the indexes $\Delta_{\text{Avg}}$ and $SD$ of IDMBO and IDMBO3, IDMBO is slightly better than IDMBO3. It shows that IDMBO can enhance the performance and robust better than IDMBO3. The difference in detail between IDMBO and IDMBO3 is given below. In conclusion, the proposed IDMBO algorithm can balance the local search and the global search ability.

2) COMPARISON FOR REEVE’S INSTANCES

In the following experiments, the proposed IDMBO algorithm is compared with existing algorithms for solving the NWFSSP with the makespan criterion on all of 21 benchmarks designed by Reeves. The compared algorithms include discrete teaching-probabilistic learning (DTPL), and hybrid discrete teaching-probabilistic learning (HDTP), which is from the literature [19], a discrete water wave optimization (DWWO) algorithm [20] and the two algorithms IDMBO and IDMBO3 in this work. The experimental results contain three indexes, namely $\Delta_{\text{min}}$, $\Delta_{\text{Avg}}$, and $SD$ of the five algorithms in Table 4 and the two algorithms in Table 5, are recorded

| Table 3. Computational results of IDMBO and the three variants. |
|---------------------------------------------------------------|
| n=m | IDMBO_N | IDMBO_NL | IDMBO3 | IDMBO |
| R | $\Delta_{\text{avg}}$ | SD | $\Delta_{\text{avg}}$ | SD | $\Delta_{\text{avg}}$ | SD | $\Delta_{\text{avg}}$ | SD |
| 20×5 | 4.15 | 0.14 | 1.70 | 0.12 | 1.25 | 0.11 | 1.09 |
| 20×10 | 4.05 | 0.16 | 1.59 | 0.12 | 1.45 | 0.12 | 1.41 |
| 20×20 | 3.62 | 0.13 | 0.53 | 0.14 | 2.29 | 0.13 | 0.27 |
| 50×5 | 6.39 | 1.47 | 7.87 | 0.44 | 2.83 | 0.13 | 1.09 |
| 50×10 | 6.21 | 1.55 | 10.82 | 1.40 | 13.35 | 0.89 | 6.14 |
| 50×20 | 4.38 | 0.93 | 12.1 | 1.01 | 18.75 | 0.24 | 7.26 |
| 100×5 | 6.7 | 2.45 | 19.65 | 2.21 | 24.26 | 2.19 | 23.39 |
| 100×10 | 5.97 | 1.93 | 40.85 | 1.94 | 34 | 1.93 | 12.68 |
| 100×20 | 5.46 | 1.72 | 21.02 | 1.37 | 23.18 | 1.32 | 13.79 |
| 200×10 | 6.19 | 2.82 | 30.47 | 2.92 | 26.16 | 2.76 | 24.17 |
| 200×20 | 6.27 | 2.82 | 25.8 | 2.80 | 24.7 | 2.67 | 23.78 |
| 500×20 | 7.46 | 4.76 | 66.72 | 3.88 | 41.79 | 3.83 | 37.67 |
to validate their effectiveness, where the best results of each instance for the same index are given in boldface. However, the results of DTPL and HDTPL are obtained from the original paper. The results of DWWO are run on the same platform as the algorithms in this work with the detail setting original paper. To ensure consistency of comparative indices from the original paper, indexes \( \Delta_{\text{min}} \), \( \Delta_{\text{Avg}} \), and \( \text{SD} \) are described by Formulas (25), (26), and (27) divided by 100, respectively (Table 4). To further compare the performance differences between IDMBO3 and IDMBO, indexes \( \Delta_{\text{min}} \), \( \Delta_{\text{Avg}} \), and \( \text{SD} \) for the two aforementioned algorithms are compared (Table 5).

Table 4 shows that IDMBO and IDMBO3 algorithms are superior to DTPL and HDTPL algorithms and nearly same to the DWWO in performance for the same instance, except for a few benchmarks. The results generated by IDMBO and IDMBO3 are slightly smaller than those by HDTPL and DWWO but are significantly smaller than those by DTPL in terms of \( \Delta_{\text{min}} \), \( \Delta_{\text{Avg}} \), and \( \text{SD} \) values for most instances, especially for the larger scale instances. In addition, the difference
is observed among IDMBO, IDMBO3 and DWWO algorithms, IDMBO has strong robustness because it has smaller $SD$ compared with other algorithms. Moreover, the overall relative percentage error of the proposed algorithm is 0.058, which is also less than 0.069 and 0.095 obtained by DWWO and IDMBO3. However, the overall average percentage
TABLE 6. Computational results of IDMBO and the compared algorithms.

| n x m | \( \Delta_{\text{min}} \) | \( \Delta_{\text{avg}} \) | SD | time | \( \Delta_{\text{min}} \) | \( \Delta_{\text{avg}} \) | SD | time | \( \Delta_{\text{min}} \) | \( \Delta_{\text{avg}} \) | SD | time |
|-------|----------------|----------------|-----|------|----------------|----------------|-----|------|----------------|----------------|-----|------|
| 20 x 5 | 1.03 | 1.42 | 4.60 | 4.49 | 0.00 | 0.49 | 8.7 | 3.51 | 0.00 | 0.00 | 5.25 | 22.16 | 0.01 | 0.10 | **1.58** | 130.29 |
| 20 x 10 | 0.74 | 1.27 | 8.38 | 7.96 | 0.00 | 0.61 | 11.98 | 6.21 | 0.25 | 0.42 | 4.7 | 38.87 | 0.07 | **0.12** | 1.50 | 234.47 |
| 20 x 20 | 0.84 | 1.07 | 5.91 | 15.15 | 0.00 | 0.25 | 11.96 | 11.78 | 0.17 | 0.45 | 3.60 | 74.36 | 0.12 | **0.14** | 0.99 | 453.32 |
| 50 x 5 | 4.20 | 4.47 | 6.36 | 23.32 | 2.16 | 2.69 | 11.33 | **9.55** | 0.39 | 0.73 | 6.3 | 328.13 | 0.40 | 0.75 | **5.94** | 303.09 |
| 50 x 10 | 3.32 | 3.51 | 6.56 | 43.32 | 2.03 | 2.42 | 11.89 | **17.12** | 0.32 | 0.53 | **6.46** | 576.68 | **0.32** | 0.54 | 6.80 | 567.73 |
| 50 x 20 | 2.31 | 2.51 | **8.09** | 83.75 | 1.49 | 2.29 | 33.22 | **32.50** | 0.00 | 0.25 | 33.62 | 1080.13 | **0.00** | **0.15** | 15.12 | 1104.14 |
| 100 x 5 | 5.98 | 6.08 | **6.4** | 101.93 | 2.95 | 3.37 | 23.62 | **33.44** | 2.52 | 2.94 | 22.40 | 2743.4 | **1.76** | **2.19** | 23.39 | 602.23 |
| 100 x 10 | 4.38 | 4.48 | **6.52** | 189.38 | 2.56 | 3.14 | 32.86 | **36.67** | 1.51 | 1.71 | 34.23 | 4770.75 | 1.53 | **2.41** | 29.8 | 1129.14 |
| 100 x 20 | 3.63 | 3.80 | **10.7** | 369.08 | 2.60 | 3.05 | 33.81 | **69.64** | 0.80 | 1.08 | 25.38 | 8966.54 | 0.88 | **1.13** | 12.68 | 2225.95 |
| 200 x 10 | 5.02 | 5.08 | **7.41** | 963.75 | 3.38 | 3.87 | 50.19 | **79** | 2.50 | **2.83** | 42.31 | 39885.83 | **2.37** | 2.87 | **46.44** | 2485.69 |
| 200 x 20 | 4.64 | 4.66 | **3.74** | 1835.92 | 2.64 | 3.18 | 72.38 | **149.28** | 2.16 | **2.25** | 13.79 | 72509.83 | 2.48 | **2.66** | 31.72 | 4414.46 |
| 500 x 20 | 4.87 | 4.89 | **7.68** | 19989 | 4.63 | 4.75 | 38.82 | **419.90** | 4.59 | 4.62 | 31.24 | 194827.46 | **3.76** | **3.86** | 41.89 | 11527.31 |
| Average | 3.41 | 3.60 | **6.86** | 1968.92 | 2.04 | 2.51 | 28.4 | **72.38** | 1.27 | 1.48 | 19.11 | **27151.60** | **1.14** | **1.41** | 18.15 | 2098.15 |

\( \Delta_{\text{min}}, \Delta_{\text{avg}} \) of IDMBO are zero as same to the DWWO and IDMBO in Table 4. The results in terms of \( \Delta_{\text{min}}, \Delta_{\text{avg}}, \text{and SD} \) are shown in Table 5 to clearly see the difference between the performance of IDMBO and IDMBO3. The optimization of IDMBO is better than IDMBO3, especially for some larger size instances. The average values of \( \Delta_{\text{min}}, \Delta_{\text{avg}} \) and SD are 0.41, 0.74, and 5.93 by IDMBO algorithm, but are equal to 0.60, 0.81, and 9.66 with the IDMBO3 algorithm for all instances, respectively.

Figures 7 and 8 are the Gantt charts for Rec 27 and Rec 29. From these figures, it can be seen that the optimal results obtained by IDMBO.

3) COMPARISON FOR TAILLARD’S INSTANCES
To further demonstrate the performance of IDMBO algorithm to resolve the NWFFSP with makespan criterion on solving the large-scale problem, the compared algorithms are tested together with IDMBO on Taillard’s instances. The optimal solutions are taken from DWWO [20] proposed by Zhao. The computational results including four indexes, namely \( \Delta_{\text{min}}, \Delta_{\text{avg}}, \text{SD and time} \), for four algorithms are summarized in Table 6. The indexes \( \Delta_{\text{min}}, \Delta_{\text{avg}}, \text{and SD} \) are provided by Equations (25), (26), and (27), and the index time is the consuming CPU time. The best results of each instance are given in boldface. The three existing compared algorithms, whose abbreviations including DPSO\text{VND} [14], TMIIG [18] and DWWO [20]. The DPSO\text{VND} and DWWO algorithms are also the population-based algorithm which employs the job-permutation-based representation as the IDMBO algorithm in this work. However, the TMIIG is also carried out to compare with the IDMBO to make clear the difference between the adaptable algorithm and the population-based algorithm. In order to guarantee the consistency of the operating environment, the fairness of the algorithm, the results with the contrast algorithms are re-implement as all the details given by the original paper.

Table 6 summarizes the computation results for the aforementioned algorithms. Figure 9 and Figure 10 show the convergence curves of IDMBO and the compared algorithms for the large-scale instance Ta61 (n = 100) and Ta91 (n = 200), respectively. Form Figure 9, it show that IDMBO has the faster convergence speed in the middle and late stages evolution, and it also has the smallest makespan value compared with DPSO\text{VND}, TMIIG and DWWO. It can be seen from Table 6 that the IDMBO outperforms the DPSO\text{VND} and TMIIG in terms of \( \Delta_{\text{min}}, \Delta_{\text{avg}} \), and it can achieve a slightly better result than DWWO for most instances. In total, the average \( \Delta_{\text{min}} \) value obtained by the IDMBO algorithm is...
1.14, which is less than 1.27,2.04,3.41 of DWWO,TMIIG and DPSO$_{VND}$, and its average $\Delta_{\text{Avg}}$ value is 1.41, which is also less than 1.48,2.51,3.60 obtained by DWWO,TMIIG and DPSO$_{VND}$, and its average $SD$ value is 18.15, which is also less than 19.11, 28.40 obtained by DWWO and TMIIG, but larger than 6.86 acquired by DPSO$_{VND}$, and the average consuming CPU time is 2098.15, which is significantly less than 27151.60 consuming by DWWO.

For consuming CPU time, it takes longer CPU time to execute IDMBO algorithm than the DPSO$_{VND}$ and TMIIG for all instances. However, as the scale of problem increases, the time to carry out IDMBO is significantly less than the time to carry out the DWAO algorithm except for on solving small instances($n=20$). It may be because the TMIIG is an adaptable algorithm, while the IDMBO, DPSO$_{VND}$ and DWWO are population-based algorithms with 51, $n$ and $n/3$ individuals in population, respectively. As the problem size increases, the population size and the cost of CPU time of DPSO$_{VND}$ and DWWO algorithms will tend to rapid increase. It is worth mentioning that the better results can be obtained by IDMBO algorithm compared with by other compared algorithms for the larger scale instances especially the instance ($n = 500$). Figure 10 shows that it also has the smallest makespan value but the convergence speed of IDMBO is slightly slower. It may be because in DPSO$_{VND}$ and DWWO algorithms, all individuals in the initial population are constructed by heuristics, while in IDMBO algorithm the only one individual is generated by SDH. As we all know, for the meta-heuristic, the better initial solutions can improve the searching speed and avoids the blindness of search. With the above analysis, it can be concluded that the proposed IDMBO algorithm can achieve the good balance between exploration and exploitation.

VI. CONCLUSION

In the work, an improved discrete migrating birds optimization (IDMBO) algorithm is presented to solve NWFSSP with makespan criterion. For initialization the population, the SDH is performed to generate the first solution (the leader (bird), and the others (the followers) are generated randomly, which can guarantee the quality and diversity of the population. Because the MBO algorithm is the neighborhood search algorithm, the quality of the neighborhood solution is very important. The different formations of hybrid multi-neighborhood strategy, which includes four neighborhood structures based on insertion and swap operators, are used to produce the neighborhood of the leader and follower birds, respectively. Otherwise, the IDMBO algorithm is easily trapped into the local optima, and the resetting mechanism will be performed when the individual exceeding the given generations is not updated. Furthermore, the LS algorithm can further enhance the LS capability of the algorithm. Finally, the simulation results and comparison based on 141 instances by Reeve and Talliard demonstrate the effectiveness and efficiency of the proposed IDMBO algorithm.

REFERENCES

[1] H. Samarghandi and T. Y. ElMekkawy, “A meta-heuristic approach for solving the no-wait flow-shop problem,” Int. J. Prod. Res., vol. 50, no. 24, pp. 7313–7326, 2012.
[2] T. Aldowaisan and A. Allahverdi, “Minimizing total tardiness in no-wait flowshops,” Found Comput. Decis. Sci., vol. 37, no. 3, pp. 62–149, 2012.
[3] C. Rajendran, “A no-wait flowshop scheduling heuristic to minimize makespan,” J. Oper. Res. Soc., vol. 45, no. 4, pp. 472–478, Dec. 1994.
[4] N. G. Hull and C. Sriskandarajah, “A survey of machine scheduling problems with blocking and no-wait in process,” Oper. Res., vol. 44, no. 3, pp. 510–525, 1996.
[5] H. Röck, “The three-machine no-wait flow shop is NP-complete,” J. Acm, vol. 31, no. 2, pp. 336–345, 1984.
[6] T. Aldowaisan and A. Allahverdi, “New heuristics for no-wait flowshops to minimize makespan,” Comput. Oper. Res., vol. 30, no. 8, pp. 1219–1231, 2003.
[7] D. Laha and U. K. Chakraborty, “A constructive heuristic for minimizing makespan in no-wait flow shop scheduling,” Int. J. Adv. Manuf. Tech., vol. 41, nos. 1–2, pp. 97–109, 2009.
[8] H. Ye, W. Li, and E. Miao, “An effective heuristic for no-wait flow shop production to minimize makespan,” J. Manuf. Syst., vol. 40, pp. 2–7, 2016.
[9] K. K. Nailwal, D. Gupta, and K. Jeet, “Heuristics for no-wait flow shop scheduling problem,” J. Oper. Res. Soc., vol. 45, no. 4, pp. 472–478, Dec. 1994.
[10] H. Röck, “The three-machine no-wait flow shop is NP-complete,” J. Acm, vol. 31, no. 2, pp. 336–345, 1984.
[31] B. Qian, L. Wang, R. Hu, D. X. Huang, and X. Wang, “A DE-based approach to no-wait flow-shop scheduling,” Comput. Ind. Eng., vol. 57, no. 3, pp. 787–805, 2009.

[32] M. Komaki and B. Malakooti, “General variable neighborhood search algorithm to minimize makespan of the distributed no-wait flow shop scheduling problem,” Prod. Eng., vol. 11, no. 3, pp. 315–329, 2017.

[33] J. Grabowski and J. Pempera, “Some local search algorithms for no-wait flow-shop problem with makespan criterion,” Comput. Oper. Res., vol. 32, no. 8, pp. 2197–2212, 2005.

[34] Q. K. Pan, M. F. Taşgetiren, and Y.-C. Liang, “A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem,” Comput. Oper. Res., vol. 35, no. 9, pp. 2807–2839, 2008.

[35] O. Andre and B. Santos, “ACO-LS algorithm for solving no-wait flow shop scheduling problem,” in Proc. Soft. Comput., Berlin, Heidelberg, 2015, pp. 89–97.

[36] G. L. Deng, M. Wei, Q. T. Su, and M. Zhao, “An effective co-evolutionary quantum genetic algorithm for the no-wait flow shop scheduling problem,” Adv. Mech. Eng., vol. 7, no. 12, pp. 1–10, 2015.

[37] S. W. Lin and K.-C. Ying, “Optimization of makespan for no-wait flow-shop scheduling problems using efficient heuristics,” Omega, vol. 64, pp. 115–125, Oct. 2016.

[38] J. Y. Ding, S. J. Song, J. N. D. Gupta, R. Zhang, and R. M. Chiong, “An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem,” Appl. Soft. Comput., vol. 30, pp. 604–613, 2015.

[39] W. Shao, D. Pi, and Z. Shao, “A hybrid discrete optimization algorithm based on teaching-probabilistic learning mechanism for no-wait flow shop scheduling,” Knowl.-Based Syst., vol. 107, pp. 219–234, Sep. 2016.

[40] F. Zhao, H. Liu, Y. Zhang, W. Ma, and C. Zhang, “A discrete water wave optimization algorithm for no-wait flow-shop scheduling problem,” Expert Syst. Appl., vol. 91, pp. 347–363, Jan. 2018.

[41] K. Z. Gao, Q. K. Pan, P. N. Suganthan, and J. Q. Li, “Effective heuristics for the no-wait flow shop scheduling problem with total flow time minimization,” Int. J. Adv. Manuf. Tech., vol. 66, pp. 9–12, Dec. 2013.

[42] Z. Xia and X. Li, “Iterative search method for total flowtime minimization no-wait flow-shop problem,” Int. J. Mach. Learn. Cyber., vol. 6, no. 5, pp. 747–761, 2015.

[43] D. Laha, J. N. D. Gupta, and S. U. Sapkal, “A penalty-shift-insertion-based algorithm to minimize total flow time in no-wait flow shops,” J. Oper. Res. Soc., vol. 65, no. 10, pp. 1611–1624, 2014.

[44] M. Akshabi, R. Takvakoli-Moghaddam, and F. Rahnamay-Roodposhti, “A hybrid particle swarm optimization algorithm for a no-wait flow shop scheduling problem with total flow time constraint,” Int. J. Adv. Manuf. Tech., vol. 70, nos. 5–8, pp. 1181–1188, 2014.

[45] K. Z. Gao, Q. K. Pan, and J. Q. Li, “Discrete harmony search algorithm for the no-wait flow shop scheduling problem with total flow time criterion,” Int. J. Adv. Manuf. Tech., vol. 56, nos. 5–8, pp. 683–692, 2011.

[46] I. A. Chaudhry, R. Ahmed, and A. M. Khan, “Genetic algorithm to minimize flowtime in a no-wait flowshop scheduling problem,” IOP Conf. Ser., Mater. Sci. Eng., vol. 65, no. 1, 2014, Art. no. 012007.

[47] K. C. Ying, S. W. Lin, and W. J. Wu, “Self-adaptive ruin-and-recreate algorithm for minimizing total flow time in no-wait flowshops,” Comput. Ind. Eng., vol. 101, pp. 167–176, 2016.

[48] K. Z. Gao, P. N. Suganthan, and T. J. Chua, “An enhanced migrating birds optimization algorithm for no-wait flow shop scheduling problem,” in Proc. IEEE SCIS, Dec. 2013, pp. 9–13.

[49] X. M. Qi, H. T. Wang, H. H. Zhu, J. Zhang, F. L. Chen, and J. Yang, “Fast local neighborhood search algorithm for the no-wait flow shop scheduling with total flow time minimization,” Int. J. Prod. Res., vol. 54, no. 16, pp. 4957–4972, 2016.

[50] S. Lin, C. Lu, and K. Ying, “Minimizing the sum of makespan and total weighted tardiness in a no-flow shop flow,” IEEE Access, vol. 6, pp. 78666–78677, 2018.

[51] H. Samarghandi and M. Behroozi, “An enumeration algorithm for the no-wait flow shop problem with due date constraints,” IFAC-Papers OnLine, vol. 49, no. 12, pp. 1803–1808, 2016.

[52] K. Ying, C. Lu, and S. Lin, “Improved exact methods for solving no-wait flowshop scheduling problems with due date constraints,” IEEE Access, vol. 6, pp. 30702–30713, 2018.

[53] W. S. Shao, D. C. Pi, and Z. S. Shao, “A Pareto-based estimation of distribution algorithm for solving multiobjective distributed no-wait flow shop scheduling problem with sequence-dependent setup time,” IEEE Trans. Autom. Sci. Eng., vol. 99, no. 3, pp. 1344–1360, Jul. 2019.

[54] B. Zhang, Q. K. Pan, L. Gao, X. L. Zhang, H. Y. Sang, and J. Q. Li, “An effective modified migrating birds optimization for hybrid flowshop scheduling problem with lot streaming,” Appl. Soft. Comput., vol. 52, pp. 14–27, Aug. 2017.

[55] Y. Han, J.-Q. Li, D. Gong, and H. Sang, “Multi-objective migrating birds optimization algorithm for stochastic lot-streaming flow shop scheduling with blocking,” IEEE Access, vol. 7, pp. 5946–5962, 2019.

[56] M. Nawaz, J. E. E. Enscore, Jr., and I. Ham, “A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem,” Omega, vol. 11, no. 1, pp. 91–95, 1983.

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

[58] C. R. Reeves, “A genetic algorithm for flowshop sequencing,” Comput. Oper. Res., vol. 22, no. 1, pp. 5–13, Jan. 1995.

[59] E. Taillard, “Benchmarks for basic scheduling problems,” Eur. J. Oper. Res., vol. 64, no. 2, pp. 278–285, 1993.

[60] J. J. Shuster, “Design and analysis of experiments,” Methods Mol. Biol., vol. 404, no. 3, pp. 235, Dec. 2007.

SUIJUN ZHANG received the B.S. degree from the Henan Institute of Science and Technology, in 2004, and the M.S. degree from Henan University, in 2007. She is currently a Lecturer with the School of Mechanical and Electrical Engineering, Henan Institute of Science and Technology. Her research interests include scheduling and optimization for process industry, intelligent optimization, and industrial process control.

XINGSHENG GU received the B.S. degree from the Nanjing Institute of Chemical Technology, in 1982, and the M.S. and Ph.D. degrees from the East China University of Chemical Technology, in 1988 and 1993, respectively. He is currently a Professor with the School of Information Science and Engineering, East China University of Science and Technology. His research interests include planning and scheduling for process industry, modeling, control, and optimization for industry processes, intelligent optimization, faults detection and diagnosis, and so on.

FUNA ZHOU received the Ph.D. degree from Shanghai Maritime University, in 2009. She is currently a Professor and a Ph.D. Supervisor with the School of Logistic and Engineering, Shanghai Maritime University. Her research interests include deep learning-based fault diagnosis and prognosis maintenance.