IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 2, 2018, accepted April 25, 2018, date of publication May 7, 2018, date of current version June 5, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2833552

Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases

TIANHUA JIANG[1] AND CHAO ZHANG[2]
[1]School of Transportation, Ludong University, Yantai 264025, China
[2]Department of Computer Science and Technology, Henan Institute of Science and Technology, Xinxiang 453003, China

Corresponding author: Tianhua Jiang (jth1127@163.com)

This work was supported in part by the Training Foundation of Shandong Natural Science Foundation of China under Grant ZR2016GP02, in part by the Project of Shandong Province Higher Educational Science and Technology Program under Grant J17KA199, and in part by the Special Research and Promotion Program of Henan Province under Grant 182102210257.

ABSTRACT Grey wolf optimization (GWO) algorithm is a new population-oriented intelligence algorithm, which is originally proposed to solve continuous optimization problems inspired from the social hierarchy and hunting behaviors of grey wolves. It has been proved that GWO can provide competitive results compared with some well-known meta-heuristics. This paper aims to employ the GWO to deal with two combinatorial optimization problems in the manufacturing field: job shop and flexible job shop scheduling cases. The effectiveness of GWO algorithm on the two problems can give an idea about its possible application on solving other scheduling problems. For the discrete characteristics of the scheduling solutions, we developed a kind of discrete GWO algorithm with the objective of minimizing the maximum completion time (makespan). In the proposed algorithm, searching operator is designed based on the crossover operation to maintain the algorithm work directly in a discrete domain. Then an adaptive mutation method is introduced to keep the population diversity and avoid premature convergence. In addition, a variable neighborhood search method is embedded to further enhance the exploration. To evaluate the effectiveness, the discrete GWO algorithm is compared with other published algorithms in the literature for the two scheduling cases. Experimental results demonstrate that our algorithm outperforms other algorithms for the scheduling problems under study.

INDEX TERMS Job shop, flexible job shop, makespan, discrete grey wolf optimization, genetic operator, variable neighborhood search.

I. INTRODUCTION

Swarm intelligence optimization algorithms imitate the evolutionary processes in nature on the basis of the collective intelligence of swarms. In general, these algorithms start with a population of individuals (solutions), which attempt to survive and reproduce in a given environment. During the evolutionary process, the collective intelligence is built up by the interaction between individuals with each other and with the environment. After a number of generations, the expected solutions will be obtained for the problem. Two classical swarm intelligence algorithms are Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), which are respectively inspired by the behaviors of birds and ants. With the continuous exploration and understanding of the world, more and more swarm intelligence optimization algorithms are gradually developed involving Artificial Bee Colony (ABC) Algorithm [1], Fruit Fly Optimization Algorithm (FFOA) [2], Cat Swarm Optimization (CSO) [3], Bat Algorithm (BA) [4], Grey Wolf Optimization (GWO) [5], Migrating birds optimization (MBO) [6], and so on. Due to their inherent advantages, these algorithms have been successfully applied to solve various optimization problems in different research areas.

Grey wolf optimization (GWO) is a novel swarm-based intelligence algorithm on the basis of the social hierarchy and hunting mechanism of natural grey wolves. It has been proved that GWO is competitive to other well-known meta-heuristic algorithms such as differential evolutionary algorithm and particle swarm optimization [5]. Due that GWO is a very efficient intelligence algorithm, it has been successfully used to

deal with economic dispatch problem [7], [8], the capacitated vehicle routing problem [9], power dispatch problem [10], system reliability optimization [11], and so on. However, as far as authors' knowledge, there are few literature involving the application of GWO on the workshop scheduling problems in the manufacturing field. Lu *et al.* [12] proposed a multi-objective discrete grey wolf optimization (MODGWO) for a real-world scheduling case in a welding workshop with the objective of optimizing production efficiency and machine load. Komaki and Kayvanfar [13] designed a hybrid multi-objective grey wolf optimization (HMOGWO) to solve the multi-objective dynamic welding scheduling problem. Asadzadeh [14] developed a grey wolf optimizer to solve the two-stage assembly flow shop scheduling problem considering the release time of jobs. The application of GWO in the aspect of production scheduling problems should be paid more attention.

As one of the most important problems in the manufacturing area, production scheduling concerns with optimizing the allocation of manufacturing resources in order to achieve the optimal production targets. The job shop scheduling (JSP) and flexible job shop scheduling problems (FJSP) are both viewed as one of the most difficult combinatorial optimization problems. For the JSP, it is an important topic in production management, which concentrates on determining the job permutation on each machine subject to the processing constraints in order to achieve some expected targets. As an extension of the JSP, FJSP provides a closer approximation to the real-world production, where each operation is allowed to be processed on more than one machine from its alternative machine set. Compared with the JSP, the complexity lies on the addition of the machine assignment for operations. Due to the NP-hard characteristics of the two problems, it is difficult to acquire the global optimum by exact algorithms even for a small-size instance. Under this situation, various intelligence algorithms have been proposed for solving those problems.

For the JSP, Asadzadeh [14] proposed an agent-based local search genetic algorithm. In his study, a multi-agent system was presented to realize the algorithm. Zhao *et al.* [15] developed a hybrid differential evolution and estimation of distribution algorithm combining the merits of estimation of distribution algorithm and differential evolution algorithm. Baykasoglu *et al.* [16] tested the performance of the teaching-learning based optimization algorithm on the job shop scheduling problem. Wang and Duan [17] presented a hybrid biogeography-based optimization algorithm for solving the job-shop scheduling problem. Chassaing *et al.* [18] introduced an efficient GRASP×ELS approach to solve some JSPs with a number of hard constraints. Zhao *et al.* [19] proposed a chemotaxis-enhanced bacterial foraging optimization based on a new chemotaxis for dealing with the job shop scheduling problem. Peng *et al.* [20] developed an algorithm introducing a tabu search into the path relinking to solve the job shop scheduling problem. Sels *et al.* [21] proposed a genetic algorithm and a scatter search to deal with the job shop scheduling problem. The scatter search splits the population in a diverse and high-quality set between individuals in a controlled way. Seo and Kim [22] developed an ant colony optimization algorithm with parameterised search space for solving the job shop scheduling problem to minimize the makespan.

For the FJSP, Nouiri *et al.* [23] used the particle swarm optimization algorithm to minimize the makespan. Rahmati and Zandieh [24] developed a biogeography-based optimization algorithm for solving the flexible job shop scheduling problem aiming to minimize the maximum completion time. Karthikeyan *et al.* [25] proposed a hybrid discrete firefly algorithm to solve the multi-objective flexible job shop scheduling problem, where three objectives-the maximum completion time, the workload of the critical machine and the total workload of all machines are considered simultaneously. Gao *et al.* [26] developed an effective discrete harmony search to solve the FJSP aiming to minimize the weighted combination of two minimization criteria namely, the maximum of the completion time (makespan) and the mean of earliness and tardiness. Wu and Wu [27] proposed an elitist quantum-inspired evolutionary algorithm with the objective of minimizing the maximum completion time (makespan). Henchiri and Ennigrou [28] presented a multi-agent model based on the hybridization of the tabu search and particle swarm optimization in order to solve the flexible job shop. Xu *et al.* [29] proposed an effective teaching-learning-based optimization algorithm aiming to solve the flexible job-shop problem with fuzzy processing time. Kaplanoglu [30] presented an object-oriented approach for the multi-objective FJSP along with simulated annealing optimization. Chang *et al.* [31] developed a hybrid Taguchi-genetic algorithm for solving the flexible job shop scheduling problem with makespan minimization. Zandieh *et al.* [32] proposed an improved imperialist competitive algorithm for the FJSP scheduling problem under condition-based maintenance with the objective of minimizing the makespan. Shen *et al.* [33] addressed the flexible job shop scheduling problem with sequence-dependent setup times and developed a tabu search algorithm with specific neighborhood functions and a diversification structure.

Although JSP and FJSP are hot topics in the production management area, many intelligence algorithms usually trap into local optimum and could not get the global optimum. Therefore, the research on these two problems is still important in the production scheduling field. As far as we know, the GWO algorithm is seldom adopted for solving the JSP and FJSP. As mentioned above, the original GWO is proposed for dealing with the continuous optimization problems. However, JSP and FJSP are typical discrete combinatorial optimization problems. Therefore, the GWO need to be redesigned beginning with a discrete encoding method according to the characteristics of the problems. Then, a discrete individual updating approach was subsequently proposed to maintain the algorithm work directly in a discrete domain. In addition, a type of adaptive mutation method was used to keep the diversity of the population and avoid the premature

convergence. To further enhance the performance of the proposed algorithm, a variable neighborhood search was embedded into the algorithm and performed on the decision-layer individuals. Finally, the proposed discrete GWO algorithm was tested with the benchmark cases available in the existing literature.

The rest of this paper is organized as follows: the JSP and the FJSP scheduling problems are described in Section II. The overview of the original GWO algorithm is described in Section III. The implementation of the discrete GWO algorithm for JSP and FJSP is addressed in Section IV. In Section V, we report the experimental results, summarize the findings of this study and present the research directions in the further work.

II. PREPARATION SCHEDULING STATEMENT

In this study, we aimed to minimize the makespan in the classical JSP and FJSP. The descriptions of the two problems are shown in the following sub-sections.

A. JOB SHOP SCHEDULING PROBLEM

In the JSP, n jobs and m machines are considered in the workshop, where each job is composed of m operations to be processed. The required machine and the fixed processing time of each operation are fixed and known. In addition, no preemption is allowed once a job is started. Each machine can only process a job at the same time. The problem is aiming to determine the operation sequence on each machine with the objective of optimizing the makespan, which can be represented by (1), where C_i is the completion time of Job i and $C_{\max}$ defines the makespan.

$$\min C_{\max} = \min \max_{1 \le i \le n} (C_i) \tag{1}$$

B. FLEXIBLE JOB SHOP SCHEDULING PROBLEM

For the FJSP, due to the addition of the machine assignment, it can be divided into two sub-problems, i.e., machine assignment and job permutation. In the workshop, there are n jobs and m machines to be scheduled. Each job consists of a sequence of precedence constrained operations, whose processing times depend on the assigned machines. In addition, some constraints should be also considered. Machine can process only one operation simultaneously. Job cannot be interrupted once it starts. Each operation must be processed after its predecessor is completed. The problem is aiming to assign each operation to an appropriate machine and to determine the operation sequences on all machines. The objective is to minimize the makespan, which can also be represented by (1).

III. OVERVIEW OF GREY WOLF OPTIMIZATION

The GWO can be viewed as a typical population-based intelligence algorithm, which mimics the hierarchy structure and hunting mechanism of wolf pack in nature. Like other swarm algorithms, it also starts with a predefined size of population, in which wolves are hierarchically classified into four types according to their fitness values [5]. The best wolf is denoted as the alpha (α), the second and third best ones are named beta (β) and delta (δ), respectively. All other individuals are regarded as omega (ω), whose search behaviors are mainly guided by α, β and δ.

During the hunting process, wolves tend to encircle the victim at first, which can be formulated by (2)-(5).

$$\boldsymbol{D} = |\boldsymbol{C} \cdot \boldsymbol{X}_p(t) - \boldsymbol{X}(t)| \tag{2}$$

$$\boldsymbol{X}(t+1) = \boldsymbol{X}_p(t) - \boldsymbol{A} \cdot \boldsymbol{D} \tag{3}$$

$$\boldsymbol{A} = 2\boldsymbol{a}\boldsymbol{r}_1 - \boldsymbol{a} \tag{4}$$

$$\boldsymbol{C} = 2\boldsymbol{r}_2 \tag{5}$$

t is the current iteration number, $\boldsymbol{X}_p$ indicates the position vector of the prey, and $\boldsymbol{X}$ denotes the current position of an individual wolf. $\boldsymbol{A}$ and $\boldsymbol{C}$ are coefficient vectors, by which wolves can reach different situations around the prey. $\boldsymbol{D}$ is distance from prey location. $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are random vectors inside [0,1]. The elements of $\boldsymbol{a}$ are linearly decreased from 2 to 0 over the course of iterations and used to coordinate the exploration and exploitation ability.

Although the hunting process is mainly guided by α, β and δ, the actual position of the prey is unknown in an abstract search space. To imitate the hunting process, it is assumed that α, β and δ can guesstimate the possible location of the prey. Therefore, the three best individuals on the decision level are saved and guide the others to update their positions in each iteration, which can be formulated by (6)-(8). The detailed description of the algorithm can be found in [5].

$$\begin{aligned} \boldsymbol{D}_\alpha &= |\boldsymbol{C}_1 \cdot \boldsymbol{X}_\alpha(t) - \boldsymbol{X}(t)|, \\ \boldsymbol{D}_\beta &= |\boldsymbol{C}_2 \cdot \boldsymbol{X}_\beta(t) - \boldsymbol{X}(t)|, \\ \boldsymbol{D}_\delta &= |\boldsymbol{C}_3 \cdot \boldsymbol{X}_\delta(t) - \boldsymbol{X}(t)| \end{aligned} \tag{6}$$

$$\begin{aligned} \boldsymbol{X}_1 &= \boldsymbol{X}_\alpha(t) - \boldsymbol{A}_1 \cdot \boldsymbol{D}_\alpha, \\ \boldsymbol{X}_2 &= \boldsymbol{X}_\beta(t) - \boldsymbol{A}_2 \cdot \boldsymbol{D}_\beta, \\ \boldsymbol{X}_3 &= \boldsymbol{X}_\delta(t) - \boldsymbol{A}_3 \cdot \boldsymbol{D}_\delta \end{aligned} \tag{7}$$

$$\boldsymbol{X}(t+1) = \frac{\boldsymbol{X}_1 + \boldsymbol{X}_2 + \boldsymbol{X}_3}{3} \tag{8}$$

IV. IMPLEMENT OF DISCRETE GWO ALGORITHM

A. ENCODING AND DECODING

As mentioned before, the original GWO is proposed for solving the continuous optimization problems. However, the JSP is a typical discrete combinatorial problem. Therefore, we firstly developed a discrete encoding scheme according to the characteristics of the JSP. Here, an operation-based encoding method was employed to represent the scheduling solutions. Taking a 3×3 JSP (3 jobs, 3 machines) shown in Table 1 for example, the encoding method is illustrated in Fig.1, where each element corresponds to a code of the candidate job. The same values represent the different operations of the same job. For example, the second '1' means the 2nd operation of Job 1.

By scanning the permutation with job index from left to right, a schedule scheme can be decoded. Seen from Fig.1, the first '3' corresponds to the first operation of Job 3, which

TABLE 1. The correspondence between jobs and machines of 3 × 3 JSP.

	Jobs	Operations		
		Operation 1	Operation 2	Operation 3
Machine order	Job 1	Machine 3	Machine 1	Machine 2
	Job 2	Machine 2	Machine 3	Machine 1
	Job 3	Machine 1	Machine 2	Machine 3
Processing time	Job 1	11	7	6
	Job 2	7	4	8
	Job 3	8	9	8

FIGURE 1. An example of encoding for JSP.

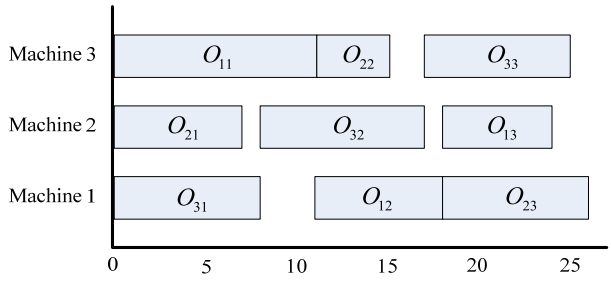

FIGURE 2. Decoding of the scheduling scheme.

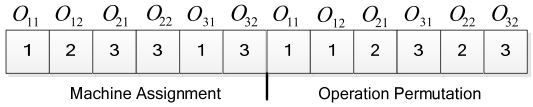

FIGURE 3. An example of encoding for FJSP.

will be processed on Machine 1; the first '1' corresponds to the first operation of Job 1, which will be processed on Machine 3; the first '2' corresponds to the first operation of Job 2, which will be processed on Machine 2; the second '2' corresponds to the second operation of Job 2, which will be processed on Machine 3; the second '1' corresponds to the second operation of Job 1, which will be processed on Machine 1; the second '3' corresponds to the second operation of Job 3, which will be processed on Machine 2; the third '3' corresponds to the third operation of Job 3, which will be processed on Machine 3; the third '2' corresponds to the third operation of Job 2, which will be processed on Machine 1; the third '1' corresponds to the third operation of Job 1, which will be processed on Machine 2.

After the scanning, the operation sequence can be represented by $(O^1_{31}, O^3_{11}, O^2_{21}, O^3_{22}, O^1_{12}, O^2_{32}, O^3_{33}, O^1_{23}, O^2_{13})$, where O^2_{13} represents the third operation of Job 1 on Machine 2, and so on. For the feasible schedule, it must meet the process constraints $O_{11} \rightarrow O_{12} \rightarrow O_{13}$, $O_{21} \rightarrow O_{22} \rightarrow O_{23}$ and $O_{31} \rightarrow O_{32} \rightarrow O_{33}$. In addition, the operation sequences on Machine 1 is $O_{31} \rightarrow O_{12} \rightarrow O_{23}$, Machine 2 is $O_{21} \rightarrow O_{32} \rightarrow O_{13}$and Machine 3 is $O_{11} \rightarrow O_{22} \rightarrow O_{33}$. So the first operations which can be processed on the Machine 1, 2, 3 are O_{31}, O_{21}, O_{11} respectively. By considering the process constraint, we scheduled O_{31} on Machine 1, O_{21} on Machine 2 and O_{11} on Machine 3 one after another at the earliest allowable time. Then the operations which can be processed on Machine 1, 2, 3 are O_{12}, O_{32}, O_{22}, respectively. By considering the process constraint, we scheduled O_{12}, O_{32}, O_{22} on machine 1, 2, 3 one after another at the earliest allowable time. And then the operations which can be processed on Machine 1, 2, 3 are O_{23}, O_{13}, O_{33}, respectively. By considering the process constraint, we scheduled O_{23}, O_{13}, O_{33} on machine 1, 2, 3 one after another at the earliest allowable time. Then the corresponding schedule can be shown in Fig. 2.

As the extension version of the JSP, FJSP can be divided into two sub-problems: machine assignment and operation permutation. Thus, a two-phase discrete encoding mechanism was adopted in Fig. 3. The former segment aims to assign an appropriate machine to each operation, and the latter tries to get an operation permutation on each machine. Fig. 2 shows the encoding method of a 3 × 3 FJSP (3 jobs, 3 machines), where each job contains two operations to be processed. In Fig. 3, the value of each element in the first segment defines the machine assignment, which are stored in a fixed order. In the second component, the elements with the same value represent the operations of the same job, e.g., the second '2' means the 2nd operation of Job 2.

By scanning the operation permutation from left to right and reading the machine assignment of each operation, a schedule scheme can be decoded. Seen from Fig.3, the first '1' in the operation permutation section corresponds to the first operation of Job 1, which will be processed on Machine 1; the second '1' in the operation permutation section corresponds to the second operation of Job 1, which will be processed on Machine 2; the first '2' in the operation permutation section corresponds to the first operation of Job 2, which will be processed on Machine 3; the first '3' in the operation permutation section corresponds to the first operation of Job 3, which will be processed on Machine 1; the second '2' in the operation permutation section corresponds to the second operation of Job 2, which will be processed on Machine 3; the second '3' in the operation permutation section corresponds to the second operation of Job 3, which will be processed on Machine 3. After the scanning, the operation sequence can be represented by $(O^1_{11}, O^2_{12}, O^3_{21}, O^1_{31}, O^3_{22}, O^3_{32})$. The decoding process can be implemented by the mentioned method above.

B. POPULATION INITIALIZATION

For a swarm intelligence algorithm, the initial solutions are crucial for the convergence speed and solution quality. Thus, it is very important to generate an effective initial population. For the JSP, five priority rules were employed to generate

good initial solutions as follows: Most Work Remaining (MWR), Most Operation Remaining (MOR), Shortest Processing Time (SPT), Longest Processing Time (LPT) and Random Rule (RR). They were randomly selected to generate initial solutions.

MWR: The job with the most amount of work remaining has the high priority.

MOR: The job with the most number of operations remaining has the high priority.

SPT: The job with the shortest processing time has the high priority.

LPT: The job with the longest processing time has the high priority.

RR: Jobs are randomly selected for the permutation.

For the FJSP, the population initialization can be composed by machine assignment and operation scheduling phases. For the machine assignment phase, the global selection (GS), local selection (LS) and random selection (RS) were combined to get the initial machine assignment [35]. For the operation permutation component, a predefined number of operation sequence schemes were generated at random for each obtained machine assignment. Then the combination of the two components with the best fitness value was treated as an initial solution in the population. This procedure will be repeated until all the initial scheduling solutions are generated.

C. DISCRETE SEARCH OPERATOR FOR THE PREY

The GWO cannot be directly employed to deal with combinatorial optimization problems by (2)-(8). To ensure the feasibility of the scheduling solutions, a modified discrete search operator is developed based on the crossover operation of genetic algorithm, which can be formulated as (9). $\bar{X}_k$ represents the discrete scheduling solution corresponding to kth wolf; $\bar{X}_\alpha, \bar{X}_\beta, \bar{X}_\delta$ define the scheduling solutions of α, β and δ; f defines the proposed discrete crossover operation, *rand* is a random number inside [0, 1].

$$\bar{X}_k(t+1) = \begin{cases} f(\bar{X}_k(t), \bar{X}_\alpha(t)), & rand \le \frac{1}{3} \\ f(\bar{X}_k(t), \bar{X}_\beta(t)), & \frac{1}{3} < rand < \frac{2}{3} \\ f(\bar{X}_k(t), \bar{X}_\delta(t)), & rand \ge \frac{2}{3} \end{cases} \quad (9)$$

For the operation permutation in JSP and FJSP, the precedence preserving order-based crossover (POX) [36] was adopted to realize the crossover operation in this study, which can be described by the following procedure and illustrated by Fig. 4.

(1) Create two subsets SUB_1 and SUB_2.

(2) Randomly select jobs into the subset SUB_1, the other jobs are filled into SUB_2.

(3) Copy the selected jobs in SUB_1 from Parent 1 to Child 1 and from Parent 2 to Child 2, and keep their position unchanged.

(4) Copy the jobs in SUB_2 from Parent 2 to Child 1 and from Parent 1 to Child 2, and keep their position unchanged.

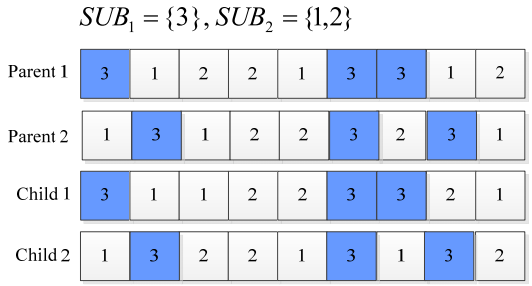

FIGURE 4. POX crossover operation.

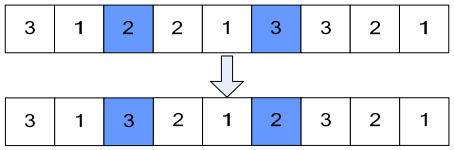

FIGURE 5. Swap mutation.

For the machine assignment in the FJSP, a two-point crossover was used in this study, that is, two positions were randomly selected from parent individuals, and the elements between them in parents were exchanged with each other to get two children.

D. ADAPTIVE MUTATION OPERATOR

In the above section, the discrete updating of individuals was implemented according to (9). Although it maintained the partial characteristics the original GWO, each current individual was just updated based on the information of the three best wolves α, β and δ, which tends to result in the loss of population diversity and make the algorithm appear premature convergence.

To overcome this drawback, an adaptive mutation operator was developed in our algorithm, where the mutation rate can be formulated by (10). p_k denotes the mutation rate of the kth individual; *fit* defines the fitness and equals to $M/C_{\max}$, where M is a constant. $fit_{\max}$ and $fit_{\min}$ mean the maximum and minimum values of fitness in the current generation. Seen from (10), the mutation rate of each wolf varies according to the fitness during the evolution process. If the individual has a better fitness, a slight mutation will be imposed on it to avoid its moving away from the global optimum, and vice versa.

$$p_k(t) = 1 - \frac{fit_{\max}(t) - fit_k(t)}{fit_{\max}(t) - fit_{\min}(t)} \quad (10)$$

The mutation operators for the operation permutation section are shown as below and illustrated in Figs. 5-7.

Swap: Randomly choose two operations e_1 and e_2 corresponding to different jobs, and exchange their positions.

Insert: Randomly choose two operations e_1 and e_2, and insert e_2 into the front of e_1.

Inverse: Randomly choose two operations e_1 and e_2, and inverse the order between them.

In addition, a mutation operator for the machine assignment in the FJSP was developed as below.

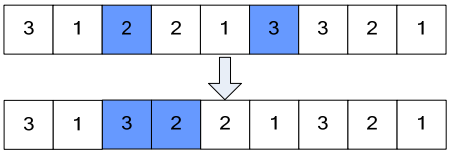

FIGURE 6. Insert mutation.

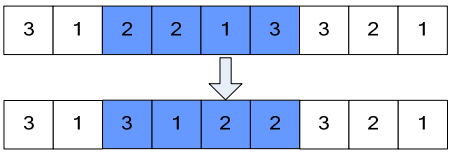

FIGURE 7. Inverse mutation.

(1) Set $k \leftarrow 1$
(2) Perform the procedure below until $k > popsize$.
  if $rand < p_k$ then
    if $p_k < 0.5$ then $\bar{X}_k := \text{Swap}(\bar{X}_k)$
    elseif $0.5 \le p_k \le 0.8$ then $\bar{X}_k := \text{Insert}(\bar{X}_k)$
    else $\bar{X}_k := \text{Inverse}(\bar{X}_k)$
    endif
  endif
  $k \leftarrow k+1$
(3) Terminate the mutation.

FIGURE 8. The mutation process for JSP.

(1) Set $k \leftarrow 1$.
(2) Perform the procedure below until $k > popsize$.
  if $rand < p_k$ then
    if $p_k < 0.5$ then $\bar{X}_k := \text{Swap} + \text{Assignment}(\bar{X}_k)$
    elseif $0.5 \le p_k \le 0.8$ then $\bar{X}_k := \text{Insert} + \text{Assignment}(\bar{X}_k)$
    else $\bar{X}_k := \text{Inverse} + \text{Assignment}(\bar{X}_k)$
    endif
  endif
  $k \leftarrow k+1$
(3) Terminate the mutation.

FIGURE 9. The mutation process for FJSP.

Assignment: Randomly choose an operation with more than one alternative machine. Choose a different machine at random from the alternative machine set to replace the original one.

Because the FJSP consists of two sub-problems, the mutation operators can be represented as *Swap* + *Assignment*, *Insert* + *Assignment* and *Inverse* + *Assignment*, which were selected according to the values of mutation rate.

The pseudo-codes of the mutation processes for the JSP and FJSP can be respectively shown in Figs. 8 and 9.

E. VARIABLE NEIGHBORHOOD SEARCH

In GWO, the search process is guided by the three best individuals (α, β and δ) towards the potential optimum, which means that the quality of α, β and δ is crucial for the performance of the algorithm. Therefore, a variable neighborhood search (VNS) was performed on the three best individuals by considering their important effects. It is well-known that the variable neighborhood search can pinpoint the optimal solution by changing the neighborhood structures. The basic framework of VNS is shown in Fig.10. To implement the procedure, the mutation operators above are taken as the neighborhood structures in the VNS. In addition, the local search is shown in Fig.11.

(1) Select the neighborhood structure set NB_l, $l = 1,2,\cdots,l_{\max}$; get an initial solution $\bar{X}$ and set a stopping condition $q_{\max}$ and $q \leftarrow 1$.
(2) Perform the following procedures until the stop condition $q > q_{\max}$ is met.
  1) Set $l \leftarrow 1$.
  2) Perform the steps below until $l > l_{\max}$.
    a. Generate a point $\bar{X}'$ randomly from the l th neighborhood of $\bar{X}$.
    b. Apply some local search method with the initial solution $\bar{X}'$ to get the obtained local optimum $\bar{X}''$.
    c. If the local optimum $\bar{X}''$ is better than the incumbent, then $\bar{X} \leftarrow \bar{X}''$, set $l \leftarrow 1$ and continue the search with NB_l; otherwise, set $l \leftarrow l+1$.
  3) $q \leftarrow q+1$.
(3) Output the result.

FIGURE 10. Basic framework of VNS.

(1) Get the initial solution $\bar{X}'$, define a stopping condition $\rho_{\max}$ and set $\rho \leftarrow 1$.
(2) Perform the procedure below until $\rho > \rho_{\max}$.
  $\bar{X}'' :=$ Randomly perform a neighborhood structure to $\bar{X}'$
  if $C_{\max}(\bar{X}'') < C_{\max}(\bar{X}')$ then
    $\bar{X}' \leftarrow \bar{X}''$
  endif
  $\rho \leftarrow \rho+1$
(3) Set $\bar{X}'' \leftarrow \bar{X}'$ and output the result.

FIGURE 11. The framework of local search.

F. PROCEDURE OF THE DISCRETE GWO

With the above design, the procedure of the discrete GWO is illustrated in Fig. 12.

V. RESULTS AND DISCUSSION

In this section, our proposed discrete GWO was evaluated by using various problems, which cover 39 and 15 benchmark instances respectively belonged to JSP and FJSP. We implemented the algorithm in FORTRAN on a VMWare Workstation with 2GB main memory under WinXP.

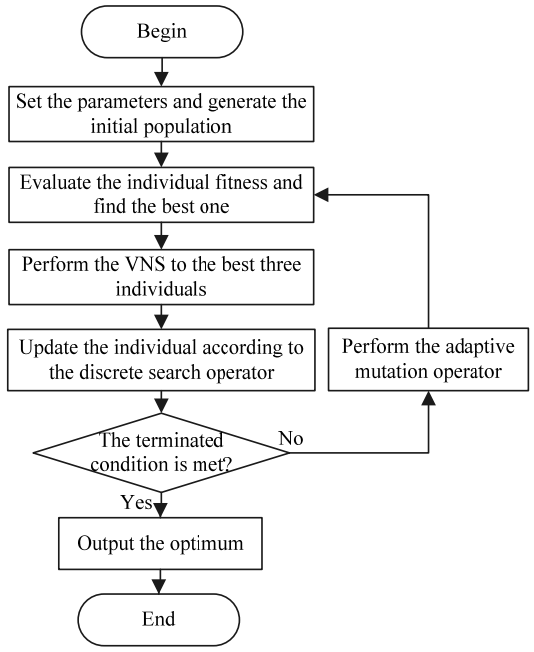

FIGURE 12. The framework of the discrete GWO.

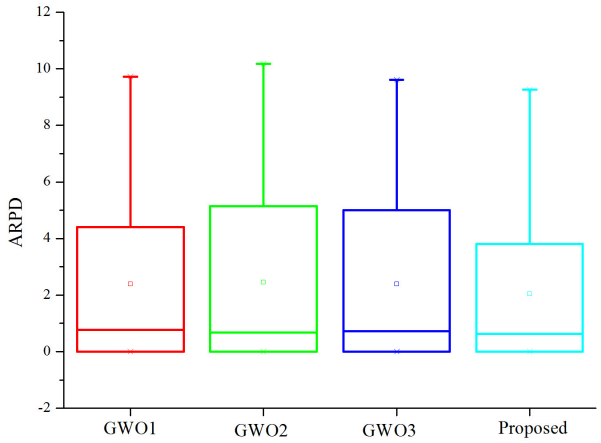

FIGURE 13. Box plot for algorithms in JSP.

A. EFFECTIVENESS FOR SOLVING THE JSP

We first evaluated the effectiveness of our algorithm for solving the JSP. The parameters for the discrete GWO algorithm were defined as follows: population size 200, the maximum iteration $5 \times m \times n$, $q_{\max} = 10$ and $\rho_{\max} = 30$.

To implement the adaptive mutation, three operators were adopted according the mutation rate and selected according to the mutation rate. We first test the effectiveness of this adaptive mutation method in Table 2. GWO1 represents the algorithm where only the *Swap* mutation is used. GWO2 represents the algorithm where only the *Insert* mutation is used. GWO3 represents the algorithm where only the *Inverse* mutation is used. 'Time' is the average run time (in seconds) of 10 replications for each instance. 'Mean' defines the average value of 'ARPD' and 'Time'. 'ARPD' defines the average relative percentage deviation (RPD) of 10 replications for each instance, which can be obtained as follows:

$$ARPD = \sum_{r=1}^{R} \frac{100 \times (Al_r - BKS)}{BKS} \Big/ R \qquad (11)$$

where R is the number of runs, BKS is the best known solutions in the current literature, Al_r is the obtained value in the rth run by the algorithm for each instance. The bold values correspond to better results.

It can be seen from Table 2 that the proposed GWO can yield better ARPD values than other algorithms. Fig.13 shows the box plot for the experimental results in Table 2. It can be seen from the figure that the proposed GWO is statistically better than the other algorithms. In addition, the average CPU times consumed by the proposed GWO algorithms are only more than those of GWO1.

TABLE 2. Comparison between the ARPD results of different algorithms.

Instance	GWO1		GWO2		GWO3		Proposed	
	ARPD	Time	ARPD	Time	ARPD	Time	ARPD	Time
FT06	**0.00**	6.7	**0.00**	7.6	**0.00**	7.2	**0.00**	6.6
FT10	7.42	58.8	5.14	67.0	5.00	63.9	**3.80**	62.2
FT20	2.54	73.4	3.26	81.8	4.14	78.0	**1.69**	76.1
LA01	**0.00**	14.9	**0.00**	17.2	**0.00**	15.9	**0.00**	13.8
LA02	1.47	14.3	**0.37**	16.5	1.30	15.7	1.16	14.5
LA03	2.98	14.7	2.48	16.3	2.70	16.3	**1.91**	14.3
LA04	0.65	14.5	**0.00**	15.8	0.58	16.6	0.63	12.9
LA05	**0.00**	14.6	**0.00**	16.8	**0.00**	15.8	**0.00**	13.6
LA06	**0.00**	37.8	**0.00**	42.2	**0.00**	41.6	**0.00**	36.1
LA07	**0.00**	37.5	**0.00**	41.8	**0.00**	40.9	**0.00**	34.9
LA08	**0.00**	38.1	**0.00**	435	**0.00**	42.1	**0.00**	38.1
LA09	**0.00**	37.4	**0.00**	42.5	**0.00**	40.2	**0.00**	35.9
LA10	**0.00**	36.4	**0.00**	41.6	**0.00**	39.1	**0.00**	34.0
LA11	**0.00**	75.5	**0.00**	82.7	**0.00**	81.5	**0.00**	71.3
LA12	**0.00**	74.7	**0.00**	81.4	**0.00**	79.9	**0.00**	72.0
LA13	**0.00**	74.6	**0.00**	82.9	**0.00**	81.6	**0.00**	71.4
LA14	**0.00**	71.6	**0.00**	79.2	**0.00**	77.2	**0.00**	67.5
LA15	**0.00**	76.6	**0.00**	80.3	**0.00**	80.5	**0.00**	74.6
LA16	3.92	60.9	4.38	66.6	3.81	64.5	**3.17**	61.1
LA17	1.15	61.2	1.33	63.6	1.41	64.9	**1.10**	60.3
LA18	2.62	60.6	2.50	64.4	**1.98**	64.6	2.24	58.9
LA19	**1.26**	61.4	2.80	64.3	1.81	73.4	1.34	61.2
LA20	6.79	60.0	6.05	64.6	7.51	64.9	**5.60**	60.8
LA21	**6.17**	155.9	7.03	161.2	6.73	164.2	6.41	153.4
LA22	**6.95**	153.3	7.60	162.4	7.64	161.8	7.69	154.3
LA23	0.68	154.1	0.67	164.3	0.72	161.8	**0.47**	150.9
LA24	6.22	152.3	8.04	165.1	7.52	163.8	**5.60**	155.8
LA25	**4.40**	152.0	6.41	167.8	5.72	163.2	5.06	160.5
LA26	4.15	305.8	4.75	323.4	2.83	317.0	**3.48**	324.5
LA27	8.45	305.5	7.22	328.6	7.61	320.0	**6.19**	314.8
LA28	6.25	311.0	**5.46**	327.7	6.71	328.4	6.12	325.0
LA29	9.72	311.3	10.17	328.0	9.61	326.8	**9.27**	313.2
LA30	3.06	308.6	1.80	334.0	2.35	326.0	**1.22**	316.4
LA31	**0.00**	829.2	**0.00**	895.7	**0.00**	888.7	**0.00**	877.2
LA32	**0.00**	847.0	**0.00**	893.0	**0.00**	871.9	**0.00**	837.2
LA33	**0.00**	831.6	**0.00**	896.4	**0.00**	864.7	**0.00**	856.5
LA34	0.77	842.6	0.84	895.8	**0.02**	885.4	0.29	833.2
LA35	0.30	873.0	0.28	897.3	0.11	868.4	**0.00**	842.6
LA36	5.62	382.8	7.24	403.5	5.62	393.0	**5.36**	387.5
Mean	2.40	204.9	2.46	228.6	2.40	214.7	2.05	206.5

TABLE 3. Comparison between the results of different algorithms.

Instance	BKS	PaGA	LSGA	TLBO	NS-HDE/EDA	BFO	Proposed
FT06	**55**	**55**	**55**	**55**	**55**	**55**	**55**
FT10	**930**	997	976	938	937	1107	940
FT20	**1165**	1196	1209	**1165**	1178	1245	1178
LA01	**666**	**666**	**666**	**666**	**666**	772	**666**
LA02	**655**	**655**	**655**	**655**	**655**	661	**655**
LA03	**597**	617	**597**	**597**	**597**	621	**597**
LA04	**590**	607	**590**	607	**590**	594	**590**
LA05	**593**	**593**	**593**	**593**	**593**	**593**	**593**
LA06	**926**	**926**	**926**	**926**	**926**	987	**926**
LA07	**890**	**890**	**890**	**890**	**890**	902	**890**
LA08	**863**	**863**	**863**	864	**863**	**863**	**863**
LA09	**951**	**951**	**951**	**951**	**951**	998	**951**
LA10	**958**	**958**	**958**	**958**	**958**	990	**958**
LA11	**1222**	**1222**	**1222**	**1222**	**1222**	1320	**1222**
LA12	**1039**	**1039**	**1039**	-	**1039**	1051	**1039**
LA13	**1150**	**1150**	**1150**	-	**1150**	1227	**1150**
LA14	**1292**	**1292**	**1292**	-	**1292**	**1292**	**1292**
LA15	**1207**	1273	**1207**	-	**1207**	1231	**1207**
LA16	**945**	994	959	946	956	965	956
LA17	**784**	793	792	-	**784**	801	790
LA18	**848**	860	857	-	855	977	859
LA19	**842**	873	860	-	852	871	845
LA20	**902**	912	907	-	907	951	937
LA21	**1046**	1146	1114	1091	1058	-	1090
LA22	**927**	1007	989	-	952	-	970
LA23	**1032**	1033	1035	-	1038	-	**1032**
LA24	**935**	1012	1032	-	973	-	982
LA25	**977**	1067	1047	-	1000	-	1008
LA26	**1218**	1323	1307	-	1229	-	1239
LA27	**1235**	1359	1350	1256	1287	-	1290
LA28	**1216**	1369	1312	-	1275	-	1263
LA29	**1152**	1322	1311	-	1220	-	1244
LA30	**1355**	1437	1451	-	1371	-	**1355**
LA31	**1784**	1844	**1784**	**1784**	**1784**	-	**1784**
LA32	**1850**	1907	**1850**	-	**1850**	-	**1850**
LA33	**1719**	-	1745	-	**1719**	-	**1719**
LA34	**1721**	-	1784	-	**1721**	-	**1721**
LA35	**1888**	-	1958	-	**1888**	-	**1888**
LA36	**1268**	-	-	1332	1315	-	1311

Furthermore, the proposed GWO was compared with other published algorithms such as LSGA [14], NS-HDE/EDA [15], TLBO [16], BFO [19] and PaGA [37]. The computational results are given in Tables 3, in which the instance names, the problem sizes ($n \times m$), the best-known solutions (BKS), the best results obtained by existing algorithms and our proposed algorithm are listed. In Table 3, the boldface means the BKS, and the symbol '-' means that the value was not listed in the corresponding literature. Seen from Table 3, our proposed GWO obtained the best-known results for 23 out of 39 instances. In the meantime, the number of the best-known results obtained by our algorithm is more than or equal to all other algorithms under the same instances.

Table 4 shows the number of solved instances (NI) and the $ARPD'$ of all compared algorithms. The value of $ARPD'$ can be calculated as follows:

$$ARPD' = \sum_{l=1}^{NI} \frac{100 \times (Best - BKS)}{BKS} \Big/ NI \qquad (12)$$

TABLE 4. The comparison of $ARPD'$ for JSP.

Algorithm	*NI*	Others (%)	Proposed (%)	Improvement
PaGA	35	3.67	1.28	2.39
LSGA	38	2.64	1.18	1.46
TLBO	19	0.78	0.81	-0.03
NS-HDE/EDA	39	0.95	1.23	-0.28
BFO	23	4.81	0.42	4.39

TABLE 5. Computational data for the Brandimarte instances.

Instance	GWO1		GWO2		GWO3		Proposed	
	ARPD	Time	ARPD	Time	ARPD	Time	ARPD	Time
Kacem01	**0.00**	3.7	**0.00**	3.9	**0.00**	3.7	**0.00**	3.9
Kacem02	1.43	28.2	**0.00**	32.0	1.43	35.2	**0.00**	27.6
Kacem03	3.64	33.4	1.82	37.6	3.64	41.6	**0.00**	32.6
Kacem04	11.43	57.8	**5.72**	59.0	8.58	58.8	**5.72**	58.5
Kacem05	29.09	169.0	29.09	200.8	30.91	192.8	**23.63**	176.0
MK01	**6.15**	61.4	7.69	69.6	7.69	67.8	6.66	64.6
MK02	25.83	66.2	26.67	73.2	25.83	71.0	**25.0**	70.0
MK03	**0.00**	366.8	**0.00**	387.6	**0.00**	392.7	**0.00**	377.6
MK04	40.94	207.8	40.42	210.4	39.58	228.6	**35.67**	218.2
MK05	6.43	121.2	**5.36**	135.2	5.83	136.4	**5.36**	131.4
MK06	124.85	445.4	130.30	491.0	130.91	463.8	**120.45**	480.7
MK07	14.29	214.2	13.38	226.0	13.53	221.2	**11.58**	213.4
MK08	**0.00**	1009.5	**0.00**	1083.2	**0.00**	1064.1	**0.00**	1026.2
MK09	12.71	1096.6	11.84	1147.7	12.11	1163.1	**9.43**	1123.8
MK10	55.63	1653.3	52.24	1849.7	53.33	1898.9	**51.15**	1744.3
Mean	22.16	369.0	21.64	400.5	22.22	402.6	19.64	383.3

where *Best* means the best result found by algorithms from 10 replications for each instance. Table 4 shows the $ARPD'$ of the proposed algorithm and other published algorithms (Others). The column defined 'Improvement' shows the reduction of $ARPD'$ obtained by our algorithm with respect to other algorithms. Seen from Table 4, the proposed GWO yields the better $ARPD'$, except for TLBO and NS-HDE/EDA. However, the number of BKS values obtained by our algorithm is more than NS-HDE/EDA and equal to TLBO. Based on the comparison results, it can be concluded that our algorithm has good effectiveness for minimizing the makespan in JSP.

B. EFFECTIVENESS FOR SOLVING THE FJSP

In this section, we verified the performance of our proposed algorithm for solving the FJSP. The parameters for the proposed discrete GWO algorithm are defined as follows: population size is 200, the maximum iteration is $10 \times m \times n$, $q_{\max} = 10$ and $\rho_{\max} = 30$.

The effectiveness of the adaptive mutation method is also first tested in Table 5. GWO1 represents the algorithm where only the *Swap* + *Assignment* mutation is used. GWO2 represents the algorithm where only the *Insert* + *Assignment*

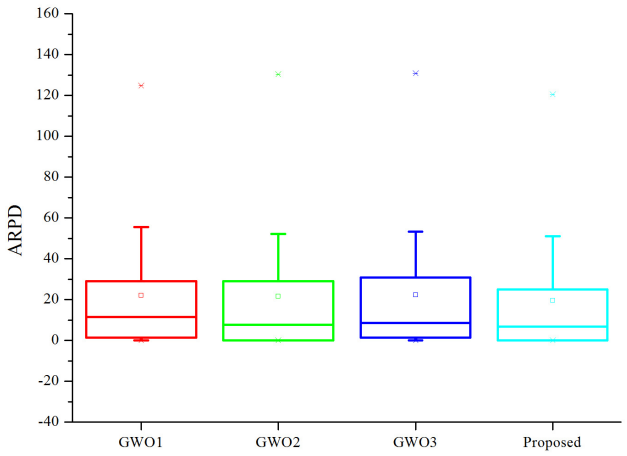

FIGURE 14. Box plot for algorithms in FJSP.

TABLE 6. Computational data for the Kacem instances.

Instance	BKS	GENACE	HA	edPSO	Proposed
Kacem01	**11**	**11**	**11**	**11**	**11**
Kacem02	**14**	-	15	17	**14**
Kacem03	**11**	12	13	-	**11**
Kacem04	**7**	**7**	**7**	8	**7**
Kacem05	**11**	12	12	-	13

TABLE 7. Computational data for the Brandimarte instances.

Instance	BKS	MATSPSO	HA	edPSO	Proposed
MK01	**36**	39	42	41	40
MK02	**24**	27	28	26	29
MK03	**204**	207	**204**	207	**204**
MK04	**48**	65	75	65	64
MK05	**168**	174	179	171	175
MK06	**33**	72	69	61	69
MK07	**133**	154	149	173	147
MK08	**523**	**523**	555	**523**	**523**
MK09	**299**	340	342	307	322
MK10	**165**	299	242	312	249

mutation is used. GWO3 represents the algorithm where only the *Inverse* + *Assignment* mutation is used. It is clear that the proposed GWO can yield better ARPD values than other algorithms. Fig.14 shows the box plot for the experimental results in Table 5. It can be seen from the figure that the proposed GWO is statistically better than the other algorithms.

In order to test the performance of our algorithm, it was compared with other published algorithms, such as edPSO [23], MATSPSO [28], GENACE [38] and HA [39]. A summary of computational results is given in Tables 6 and 7. Tables also listed the instance names, the problem sizes, the best-known solutions (BKS) and the results obtained by published algorithms and our algorithm. In Tables 6 and 7, the boldface represents the BKS, and the symbol '-' means that the value was not available in the literature. Seen from the data, our discrete GWO obtained more BKS values than other algorithms except for TLBO.

TABLE 8. The comparison of $ARPD'$ for FJSP.

Algorithm	NI	Others (%)	Proposed (%)	Improvement
GENACE	4	4.55	4.55	0.00
MATSPSO	10	29.02	24.17	4.85
HA	15	21.26	17.72	3.54
edPSO	13	20.22	19.05	1.17

In addition, the comparison of $ARPD'$ obtained by proposed algorithm and other algorithms is also listed in Table 8. It can be seen from the comparison results that the proposed algorithm yields an improvement compared with other algorithms, except for the GENACE and TLBO. However, our algorithm can obtain more BKS values than GENACE. Based on the comparison results, it can be concluded that our algorithm has good effectiveness for the FJSP.

VI. CONCLUSIONS

GWO was developed as a new swarm-based intelligence algorithm for dealing with optimization problems. It has been successfully used to solve continuous problems. Here, the original GWO algorithm was modified and tested on JSP and FJSP with typical discrete characteristics. As far as we know, these two scheduling problems are amongst the most complicated combinatorial optimization problems in the manufacturing field. Therefore, the performance of GWO algorithm on these two problems can provide an idea about its possible application on solving other production scheduling problems.

Extensive experiments were conducted to demonstrate that our proposed discrete GWO algorithm has some potential when compared to some existing published algorithms in the literature. According to the comparison results, our discrete GWO is capable of providing better solutions for some instances, similar or comparable results for others. It seems that the searching mechanism of GWO algorithm has some potential for solving combinatorial optimization problems and it deserves further exploration in our future work.

REFERENCES

[1] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, Apr. 2007.

[2] W.-T. Pan, "A new fruit fly optimization algorithm: Taking the financial distress model as an example," *Knowl.-Based Syst.*, vol. 26, pp. 69–74, Feb. 2012.

[3] S.-C. Chu and P.-W. Tsai, "Computational intelligence based on the behavior of cats," *Int. J. Innov. Comput., Inf. Control*, vol. 3, no. 1, pp. 163–173, Feb. 2007.

[4] X.-S. Yang and A. H. Gandomi, "Bat algorithm: A novel approach for global engineering optimization," *Eng. Comput.*, vol. 29, no. 5, pp. 464–483, 2012.

[5] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[6] E. Duman, M. Uysal, and A. F. Alkaya, "Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem," *Inf. Sci.*, vol. 217, pp. 65–77, Dec. 2012.

[7] M. Pradhan, P. K. Roy, and T. Pal, "Grey wolf optimization applied to economic load dispatch problems," *Int. J. Elect. Power Energy Syst.*, vol. 83, pp. 325–334, Dec. 2016.

[8] H. M. Song, M. H. Sulaiman, and M. R. Mohamed, "An application of grey wolf optimizer for solving combined economic emission dispatch problems," *Int. Rev. Model. Simul.*, vol. 7, no. 5, pp. 838–844, May 2014.
[9] L. Korayem, M. Khorsid, and S. S. Kassem, "Using grey wolf algorithm to solve the capacitated vehicle routing problem," *Mater. Sci. Eng.*, vol. 83, no. 1, pp. 12–14, Jan. 2015.
[10] M. H. Sulaiman, Z. Mustaffa, M. R. Mohamed, and O. Aliman, "Using the gray wolf optimizer for solving optimal reactive power dispatch problem," *Appl. Soft Comput.*, vol. 32, pp. 286–292, Jul. 2015.
[11] A. Kumar, S. Pant, and M. Ram, "System reliability optimization using gray wolf optimizer algorithm," *Quality Reliab. Eng. Int.*, vol. 33, no. 7, pp. 1327–1335, Nov. 2017.
[12] C. Lu, S. Xiao, X. Li, and L. Gao, "An effective multi-objective discrete grey wolf optimizer for a real-world scheduling problem in welding production," *Adv. Eng. Softw.*, vol. 99, pp. 161–176, Sep. 2016.
[13] G. M. Komaki and V. Kayvanfar, "Grey wolf optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time," *J. Comput. Sci.*, vol. 8, pp. 109–120, May 2015.
[14] L. Asadzadeh, "A local search genetic algorithm for the job shop scheduling problem with intelligent agents," *Comput. Ind. Eng.*, vol. 85, pp. 376–383, Jul. 2015.
[15] F. Zhao, Z. Shao, J. Wang, and C. Zhang, "A hybrid differential evolution and estimation of distribution algorithm based on neighbourhood search for job shop scheduling problems," *Int. J. Prod. Res.*, vol. 54, no. 4, pp. 1039–1060, Apr. 2016.
[16] A. Baykasoğlu, A. Hamzadayi, and S. Y. Köse, "Testing the performance of teaching–learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases," *Inf. Sci.*, vol. 276, pp. 204–218, Aug. 2014.
[17] X. Wang and H. Duan, "A hybrid biogeography-based optimization algorithm for job shop scheduling problem," *Comput. Ind. Eng.*, vol. 73, pp. 96–114, Jul. 2014.
[18] M. Chassaing, J. Fontanel, P. Lacomme, L. Ren, N. Tchernev, and P. Villechenon, "A GRASP × ELS approach for the job-shop with a web service paradigm packaging," *Expert Syst. Appl.*, vol. 41, no. 2, pp. 544–562, Feb. 2014.
[19] F. Zhao, X. Jiang, C. Zhang, and J. Wang, "A chemotaxis-enhanced bacterial foraging algorithm and its application in job shop scheduling problem," *Int. J. Comput. Integr. Manuf.*, vol. 28, no. 10, pp. 1106–1121, Aug. 2015.
[20] B. Peng, Z. Lü, and T. C. E. Cheng, "A tabu search/path relinking algorithm to solve the job shop scheduling problem," *Comput. Oper. Res.*, vol. 53, pp. 154–164, Jan. 2015.
[21] V. Sels, K. Craeymeersch, and M. Vanhoucke, "A hybrid single and dual population search procedure for the job shop scheduling problem," *Eur. J. Oper. Res.*, vol. 215, no. 3, pp. 512–523, Dec. 2011.
[22] M. Seo and D. Kim, "Ant colony optimisation with parameterised search space for the job shop scheduling problem," *Int. J. Prod. Res.*, vol. 48, no. 4, pp. 1143–1154, Apr. 2010.
[23] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *J. Intell. Manuf.*, vol. 29, no. 3, pp. 603–615, Mar. 2018.
[24] S. H. A. Rahmati and M. Zandieh, "A new biogeography-based optimization (BBO) algorithm for the flexible job shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 58, nos. 9–12, pp. 1115–1129, Feb. 2012.
[25] S. Karthikeyan, P. Asokan, S. Nickolas, and T. Page, "A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems," *Int. J. Bio-Inspired Comput.*, vol. 7, no. 6, pp. 386–401, 2015.
[26] K. Z. Gao, P. N. Suganthan, Q. K. Pan, T. J. Chua, T. X. Cai, and C. S. Chong, "Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives," *J. Intell. Manuf.*, vol. 27, no. 2, pp. 363–374, Apr. 2016.
[27] X. Wu and S. Wu, "An elitist quantum-inspired evolutionary algorithm for the flexible job-shop scheduling problem," *J. Intell. Manuf.*, vol. 28, no. 6, pp. 1441–1457, Aug. 2017.
[28] A. Henchiri and M. Ennigrou, "Particle swarm optimization combined with tabu search in a multi-agent model for flexible job shop problem," in *Proc. ICSI*, 2013, pp. 385–394.
[29] Y. Xu, L. Wang, S.-Y. Wang, and M. Liu, "An effective teaching–learning-based optimization algorithm for the flexible job-shop scheduling problem with fuzzy processing time," *Neurocomputing*, vol. 148, pp. 260–268, Jan. 2015.
[30] V. Kaplanoğlu, "An object-oriented approach for multi-objective flexible job-shop scheduling problem," *Expert Syst. Appl.*, vol. 45, pp. 71–84, Mar. 2016.
[31] H.-C. Chang, Y.-P. Chen, T.-K. Liu, and J.-H. Chou, "Solving the flexible job shop scheduling problem with Makespan optimization by using a hybrid Taguchi-genetic algorithm," *IEEE Access*, vol. 3, pp. 1740–1754, Oct. 2015.
[32] M. Zandieh, A. R. Khatami, and S. H. A. Rahmati, "Flexible job shop scheduling under condition-based maintenance: Improved version of imperialist competitive algorithm," *Appl. Soft Comput.*, vol. 58, pp. 449–464, Sep. 2017.
[33] L. Shen, S. Dauzère-Pérès, and J. S. Neufeld "Solving the flexible job shop scheduling problem with sequence-dependent setup times," *Eur. J. Oper. Res.*, vol. 265, no. 2, pp. 503–516, Mar. 2018.
[34] B. Giffler and G. L. Thompson, "Algorithms for solving production-scheduling problems," *Oper. Res.*, vol. 8, no. 4, pp. 487–503, Aug. 1960.
[35] G. Zhang, L. Gao, and Y. Shi, "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 3563–3573, Apr. 2011.
[36] Y. Demir and S. K. Işleyen, "An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations," *Int. J. Prod. Res.*, vol. 52, no. 13, pp. 3905–3921, Feb. 2014.
[37] L. Asadzadeh and K. Zamanifar, "An agent-based parallel approach for the job shop scheduling problem with genetic algorithms," *Math. Comput. Model.*, vol. 52, nos. 11–12, pp. 1957–1965, Dec. 2010.
[38] N. B. Ho, J. C. Tay, and E. M.-K. Lai, "An effective architecture for learning and evolving flexible job-shop schedules," *Eur. J. Oper. Res.*, vol. 179, no. 2, pp. 316–333, Jun. 2007.
[39] M. Ziaee, "A heuristic algorithm for solving flexible job shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 71, nos. 1–4, pp. 519–528, Mar. 2014.

TIANHUA JIANG was born in Weihai, China, in 1983. He received the B.S. degree in automation from the University of Jinan, Jinan, China, in 2007, the M.S. degree in control science and engineering from the Sichuan University of Science and Engineering, Zigong, China, in 2010, and the Ph.D. degree with the MOE Key Laboratory of Measurement and Control of Complex Systems of Engineering, School of Automation, Southeast University, China, in 2015.

Since 2015, he has been a Lecturer with the School of Transportation, Ludong University, Yantai, China. His research interests include production scheduling and intelligence algorithm. His recent publications have appeared in some peer-reviewed journals such as the *Journal of Intelligent & Fuzzy Systems*, the *International Journal of Industrial Engineering: Theory, Applications and Practice*, and *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*.

CHAO ZHANG was born in Xinxiang, China, in 1983. He received the B.S. degree in automation from the Zhongyuan University of Technology, Zhengzhou, China, in 2007, and the M.S. degree in control science and engineering from the University of Science and Technology Beijing, Beijing, China, in 2010. He is currently pursuing the Ph.D. degree with the MOE Key Laboratory of Measurement and Control of Complex Systems of Engineering, School of Automation, Southeast University, China.

Since 2014, he has been a Lecturer with the Department of Computer Science and Technology, Henan Institute of Science and Technology, Xinxiang, China. His research interests include adaptive control and intelligence algorithm.