Received January 21, 2020, accepted February 10, 2020, date of publication February 14, 2020, date of current version February 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2974014

A Discrete Particle Swarm Optimization Algorithm With Adaptive Inertia Weight for Solving Multiobjective Flexible Job-shop Scheduling Problem

XIAO-LIN GU, MING HUANG, AND XU LIANG
Software Technology Institute, Dalian Jiaotong University, Dalian 116021, China
Corresponding author: Xiao-Lin Gu (guxiaolin60@126.com)

This work was supported in part by the Liaoning Natural Science Foundation of China under Grant 20170540131.

ABSTRACT A discrete particle swarm optimization algorithm with adaptive inertia weight (DPSO-AIW) is proposed to solve the multiobjective Flexible Job-shop Scheduling Problem. The algorithm uses a two-layer coding structure to encode the chromosomes, namely operation sequence (OS) and machine assignment (MA). The initial population combined random selection of OS and the global selection based on operation (GSO) of MA. In order to obtain the Pareto optimal solution, non-dominated fronts are obtained by rapid non-dominated sorting. In the evolution process, the discrete particle swarm optimization algorithm is used to directly solve the values of the next generation chromosomes in the discrete domain, and the population diversity is enhanced by adaptively adjusting the variation of the inertia weight ω, and the Pareto optimal solution obtained in the process is stored in the Pareto optimal solution set (POS). Finally, numerical simulation based on two sets of international standard instances and comparisons with some existing algorithms are carried out. The comparative results demonstrate the effectiveness and practicability of the proposed DPSO-AIW in solving the multiobjective Flexible Job-shop Scheduling Problem.

INDEX TERMS Discrete particle swarm optimization, global selection based on operation, multiobjective FJSP, Pareto optimality.

I. INTRODUCTION

Flexible Job-shop Scheduling Problem (FJSP) is an extension of the Job-shop Scheduling Problem (JSP). FJSP allows multiple operations of different jobs to be processed on different machines, which changes the uniqueness of the equipment, and selecting the processing machine according to the load conditions of such resources as machines, etc., and thus the flexibility of processing is enhanced and more in line with the actual enterprise. Therefore, theoretical research on FJSP has great significance for solving the actual workshop problem of the combination optimization type of the enterprise. In the actual production process, production cost, processing time and customer satisfaction are all issues to be considered in the production scheduling optimization problem. A single scheduling target is difficult to reflect the real situation of the scheduling workshop. Nowadays, many scholars have also shifted their research direction to solving multiobjective FJSP. Chen *et al.* [1] proposed a multiobjective FJSP problem based on NSGA-II with closed relative variation. Ju *et al.* [2] proposed an improved NSGA for FJSP. In the paper, the adaptive mutation operator and elite retention strategy were introduced. The simulation experiment shows that the non-dominated sorting method can get the Pareto optimal solution quickly and correctly by dividing the whole population into three parts. Piroozfard *et al.* [3] proposed an improved multiobjective evolutionary algorithm for solving the newly extended dual-objective problem. Li *et al.* [4] proposed a Pareto-based hybrid local search (PLS) algorithm for solving multiobjective FJSP. Zhang et al [5] studied two-archive multiobjective artificial bee colony algorithm (TMABC-FS). Two new operators are employed to enhance the search capability of different kinds of bees, and two archives are proposed for obtaining a group of non-dominated feature subsets with

The associate editor coordinating the review of this manuscript and approving it for publication was Corrado Mencar.

good distribution and convergence. Based on the existing literature, this paper designs a discrete particle swarm optimization with adaptive inertia weight (DPSO-AIW) to solve FJSP. The FJSP model with makespan, workload of bottleneck machine and the total workload of machines is established. The algorithm uses a two-layer structure to encode chromosomes, and the evolution process uses discrete particle swarm optimization to solve the next generation chromosomes directly in the discrete domain. The location update uses the crossover operation in the genetic algorithm, and the value of the inertia weight ω is adaptively adjusted to enhance the diversity of the population.

The paper provides the following contributions:

(1) The FJSP model with the maximal completion time of machines, workload of bottleneck machine and the total workload of machines is established in the paper. The initial population is generated by the hybrid method, namely OS random selection and MA global selection by operation (GSO). The GSO refers to the method by which an operation selects machines in its optional machine set. The schedulings generated in this way are all feasible. When the GSO is selected, the machine with the minimum global workload is selected in the optional machine set to process. It not only ensures that the generated scheduling is feasible, but also reduces the workload of the machine as much as possible, shortens the optimization time and guarantees the quality of the initial solution. The workload of the machine is reduced as much as possible, the optimization time is shortened, and the quality of the initial solution is guaranteed. The OS random selection increase the diversity of the population and prevent the loss of the optimal solution. This method to generate the initial population greatly improves the quality of the solution and the speed of obtaining the optimal solution.

(2) The evolution process uses discrete particle swarm optimization to solve the next generation chromosomes directly in the discrete domain. The location update uses the mutation and crossover operations in the genetic algorithm. During the crossover process, the OS adopts precedence operation crossover (POX) and linear order crossover(LOX), and the MA adopts a Improved single-point crossover method. This crossover method ensures both the search capability is improved rapidly and the generated solution is always feasible. The genetic algorithm's mutation operation increases the diversity of the population. The local search ability of the algorithm is enhanced to prevent the algorithm from falling into immature convergence.

(3) Select adaptive adjustment of inertia weight to improve the diversity of the population. The inertia weight is updated by the global optimal value of the particles in the population and the exponential function of the current value. The value inertia weight ω is adaptively adjusted by using the global optimal of the particles in the population and the exponential function of the current value.

(4) The rapid non-dominated sorting method is used to determine the fronts of each candidate solution, and the specified solution is stored in the Pareto optimal solution set according to fronts order.

TABLE 1. 2×4 T-FJSP.

job	*operation*	M_1	M_2	M_3	M_4
J_1	O_{11}	2	5	3	2
	O_{12}	5	4	1	2
	O_{13}	8	3	4	1
J_2	O_{21}	7	3	1	6
	O_{22}	3	1	4	6

TABLE 2. 2×4 P-FJSP.

job	*operation*	M_1	M_2	M_3	M_4
J_1	O_{11}	2	5	3	2
	O_{12}	5	4	1	2
	O_{13}	8	3	4	1
J_2	O_{21}	7	3	1	6
	O_{22}	3	1	4	6

The rest of this paper is organized as follows: Section II introduces the formulation of FJSP, and Section III describes the basic particle swarm optimization. Section IV introduces the detailed implementation of the DPSO-AIW algorithm including encoding and decoding, initialization the population, the method to update PSO location, calculation of adaptive inertia weight, constructing PSO optimal solution set and the flow of DPSO-AIW algorithm. Section V is an analysis of the computational complexity of the DPSO-AIW algorithm. Section VI shows the results of computational studies using the DPSO-AIW algorithm and its comparisons with other algorithms. The parameters sensitivity analysis is in Section VII. Section VIII is the conclusion and direction of future research.

II. FORMULATION OF FJSP

The FJSP is commonly described as follows. There are n jobs $J = (J_1, J_2, \ldots, J_n)$ to be processed on m machines $M = (M_1, M_2, \ldots M_m)$. A job contains one or more operations, and O_{ij} represents the jth operation of job i. Each operation can be processed on different machines, but the processing time varies from machine to machine. The processing time of O_{ij} performed on machine k is t_{ijk} that is greater than 0. C_{ij} represents the completion time of the operation O_{ij}. The processing sequence of the operation is given in advance. FJSP can be divided into Total-FJSP(T-FJSP) and Partial-FJSP(P-FJSP), which are shown in Table 1 and Table 2 respectively. In T-FJSP, every operation of all jobs can be processed on any machine. In P-FJSP, the operation can only be processed on some machines, that is, the real subset of the machine set.

We consider the multiobjective FJSP with the following three objectives to be minimized: (1)the maximal completion

time of machines, that is F_1; (2) the bottleneck machine workload, that is F_2, which considers the workload balance among all machines to prevent too much work been assigned to a single machine; and (3) the total workload of machines, that is F_3, which is of interest in assigning the machine with relatively short processing time to improve economic efficiency.

Mathematically, the FJSP can be formulated as follows:

$$F_1 = min(\max(C_{ij}))(1 \le i \le n, 1 \le j \le n_i) \quad (1)$$

$$F_2 = min(\max(\sum_{i=1}^{n} \sum_{j=1}^{n_i} t_{ijk}))(1 \le k \le m) \quad (2)$$

$$F_3 = min(\max(\sum_{i=1}^{n} \sum_{j=1}^{n_i} \sum_{k=1}^{m} t_{ijk} \cdot x_{ijk})) \quad (3)$$

$$x_{ijk} = \begin{cases} 1 \ if \ operation \ O_{ij} \text{ is processed} \\ \quad \text{in machine } k \\ 0 \ otherwise \end{cases} \quad (4)$$

$$C_{ij} \ge 0 \quad (5)$$

$$C_{ij} - C_{ij-1} \ge t_{ijk} \cdot x_{ijk} (j = 1, 2, \ldots, n_i; \\ i = 1, 2, \ldots, n; k = 1, 2, \ldots, m) \quad (6)$$

$$\sum x_{ijk} = 1 \quad (7)$$

where Equation 6 ensures that the operations belong to the same job and satisfy the precedence constraints, and Equation 7 shows that one machine must be selected from the set of available machines for each operation.

FJSP processing also satisfies the following constraints and assumptions:

(1) All machines are available at time 0, and each job can be processed at time 0;

(2) At a certain time, one machine can only process one operation at a time. Once the operation is completed, the machine can be used for other operations.

(3) Once the processing begins, it cannot be interrupted;

(4) The operations of different jobs are not constrained in sequence, but there are successive constraints between the operations of the same job;

(5) The setting time of the machine and the transportation time of the operation are ignored.

III. BASIC PARTICLE SWARM OPTIMIZATION

The particle swarm optimization (PSO) algorithm is an emerging optimization algorithm based on the theory of swarm intelligence proposed by Kennedy in 1995, who was inspired by the foraging behavior of birds and fish in nature. PSO algorithm attracts the attention of many scholars because of its simple parameters, easy implementation and powerful global optimization ability. It has been widely used in the fields of function optimization, image processing, fuzzy system control and scheduling optimization.

The PSO algorithm was originally used to solve the continuous optimization problem. However, many practical engineering application problems are discrete. So using the basic idea of the PSO algorithm and changing it into a discrete form for solving large-scale discrete problems such as combinatorial optimization has become a hot issue for many scholars. Liu *et al.* [6] proposed a hybrid PSO algorithm with Pareto archives set for the FJSP problem with three targets; Chen *et al.*[7] designed an extended process coding and automatic scheduling decoding mechanism. In the paper, a multiobjective particle swarm optimization algorithm for flexible production scheduling is designed for particle maximum and minimum, convergence speed and corresponding boundary conditions. Zhang*et al.* [8] proposed a hybrid particle swarm optimization algorithm to study the multiobjective FJSP based on Pareto-dominance. Based on the complementary strengths of PSO and Variable Neighborhood Search (VNS) algorithm, Pan *et al.*[9] proposed four hybrid algorithms: PSO-VNS algorithm, Enhanced PSO(EPSO) algorithm, PSO and VNS in Turn (PVT) algorithm, and PSO and VNS Cooperative algorithm (PVC). Song and Tang [10] designed a nowait algorithm of grading for the nowait constraint between two sequential operations of a job to optimize the hybrid flow shop scheduling problem. Huang *et al.*[11] combined the multiobjective particle swarm optimization with the variable neighborhood search method to effectively solve the FJSP problem.

The basic particle swarm optimization algorithm is based on the study of the sociological behavior of birds foraging. In such problems, the algorithm first uses a set of ''particles'' to represent a set of candidate solutions to the problem. Then, each particle in the population remembers and follows the current optimal particle to search in the solution space.

Suppose that a group of M particles fly at a certain velocity in the D-dimensional search space. The state attribute of particle i is set as follows:

The current position of the particle:$x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$;

The current velocity of the particle:$v_i = (v_{i1}, v_{i2}, \ldots, v_{id})$;

The best position that particle i has experienced: $pB_i^t = \left(pB_{i1}^t, pB_{i2}^t, \ldots pB_{id}^t\right)$;

gB^t is the global optimal position experienced by the population, which is the location where the maximum fitness value is generated. The formula for the ith particle to update its position and flight velocity at the $t+1$th generation is shown in equation 8 and equation 9.

$$v_i^{t+1} = \omega \times v_i^t + c_1 r_1 \left(pB_i^t - x_i^t\right) + c_2 r_2 \left(gB^t - x_i^t\right) \quad (8)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (9)$$

In equation 8, ω represents the inertia weight; v_i^t represents the current velocity; c_1 and c_2 are the acceleration constant; r_1 and r_2 are random numbers between(0,1). They are introduced in the formula to simulate a slight unpredictable part of the behavior of the group in nature, and determine to what extent the particle remains on the original route from gB^t and pB_i^t. This is also a way to balance exploration and exploitation; pB_i^t refers to the individual optimal position of the ith particle; gB^t represents the current optimal position.

Equation 8 consists of three parts: the first part $\omega \times v_i^t$ is the previous velocity of the particle, indicating the current state of the particle; the second part $c_1 r_1 \left(pB_i^t - x_i^t\right)$ is usually called the self-cognitive part of the particle, that is, the influence

TABLE 3. The GSO process.

(a)

	M_1	M_2	M_3	M_4
O_{11}	2	5	**4**	2
O_{12}	5	4	**2**	2
O_{13}	8	3	**5**	1
O_{21}	7	3	1	6
O_{22}	3	1	**5**	6

(b)

	M_1	M_2	M_3	M_4
O_{11}	2	5	4	2
O_{12}	**7**	4	2	2
O_{13}	**10**	3	5	1
O_{21}	**9**	3	1	6
O_{22}	**5**	1	5	6

(c)

	M_1	M_2	M_3	M_4
O_{11}	2	5	**6**	**2**
O_{12}	7	4	2	3
O_{13}	10	3	**7**	**1**
O_{21}	9	3	**3**	**6**
O_{22}	5	1	**7**	**6**

(d)

	M_1	M_2	M_3	M_4
O_{11}	2	**6**	6	2
O_{12}	7	**5**	2	2
O_{13}	10	**4**	7	1
O_{21}	9	**4**	3	6
O_{22}	5	1	7	6

(e)

	M_1	M_2	M_3	M_4
O_{11}	2	6	6	**3**
O_{12}	7	5	2	**3**
O_{13}	10	4	7	1
O_{21}	9	4	3	**7**
O_{22}	5	1	7	**7**

of the particle's flight experience on its own flight, which makes the particle have a strong global search ability; the third part $c_2r_2\left(gB^t - x_i^t\right)$ is the social learning part of the particle, which embodies the information sharing between the particles, that is, the influence of particles' learning from a population on their flight.

IV. DPSO-AIW ALGORITHM

A. ENCODING AND DECODING

Each chromosome of FJSP is represented by two-layer of coding. The first one is operation sequencing (OS): the order assigned to the machine is sorted. The number in the sequence is the index of the job: the order in which each job appears in the OS represents the processing order of the various operations of the job. The second is the machine assignment (MA): assigning each operation to a set of capable machines, which means assigning each operation to a selectable machine and calculating its start time and end time [12]. The MA selection refers to the method by which an operation selects machines in its optional machine set. The schedulings generated in this way are feasible.

The decoding uses a plug-in greedy decoding algorithm to ensure that the active scheduling is generated after the chromosome is decoded [13]. According to the OS coding of the chromosome, the order of the operation on the sequence is decoded. First, the first operation on the sequence is arranged for processing, and then the second track is taken and inserted into the processing time of the corresponding machine at the best feasible processing time. In this way, all operations in the sequence are placed in their best possible position (as early as possible).

B. INITALIZATION THE POPULATION

Kacem *et al.*[14] first proposed the approach by localization (AL). Pezzella *et al.* [15] used three scheduling rules to initialize the sequence of operations. Gao *et al.* [16] proposed a method of GLR machine selection: global selection (GS), local selection (LS) and random selection (RS). Based on the above literature research, this paper proposes a method to combine the random selection of OS and the GSO of MA. The specific implementation process is as follows:

(1) The OS adopts random selection: the OS part adopts the operation-based coding mode selection in JSP, and for each operation, randomly selects and generates an OS;

(2) The MA uses GSO selection: since the OS is randomly coded, each gene in the MA is a processing machine selected for the operation in the OS. So our goal is to select the machine with the global minimum workload for processing in the optional machine set. The method is as follow:

For each operation, select the machine with the minimum workload in its optional processing machine set, and add the workload value to the machine workload of the other operations in the same column. For example, for the 2×4 T-FJSP in Table 1, the OS is randomly generated: 2 1 1 2 1. First take 2, indicating the operation O_{21}. Select the machine M_3 with the minimum workload in the optional machine set, the value is 1, and increase the remaining values of the M_2 column by 1 on the basis of the original value, see Table 3 (a). Take the remaining operations sequentially and get the MA: 3 1 3 2 4. The specific processing is shown in Table 3 (b) - (e). Values of the selected operation are with borders, while values representing the updated machine workload are in bold. Figure 1 is the Gantt chart of the 2×4 T-FJSP.

C. METHOD TO UPDATE PSO LOCATION

The equation to update velocity and position evolution in the basic PSO algorithm has three main parts: the influence of the current velocity of the particle, the "cognitive" part and the "social" part. These three parts work together to obtain the position of the next generation of particles. But the basic PSO algorithm is suitable for continuous problems, and we require discrete problems, so we imitate the optimization mechanism of the basic PSO algorithm, update the formula and define the position of discrete particle swarms according

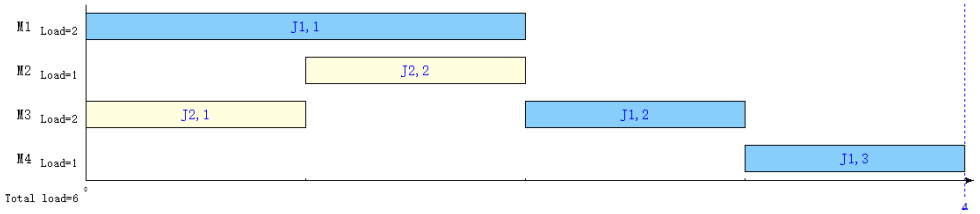


FIGURE 1. Gantt chart of the 2 × 4 T-FJSP in table 1.

to the method in equation 10.

$$x_i^{t+1} = c_2 \otimes p\left\{c_1 \otimes q\left[\omega \otimes m\left(x_i^t\right), pB_i^t\right], gB^t\right\} \quad (10)$$

where ω, c_1 and c_2 are the same as those in the basic particle swarm optimization algorithm. ω is the inertia weight, c_1 is the cognitive coefficient, c_2 is the social coefficient, and $\otimes$ is used to represent the optimization operation. Equation 10 consists of three parts.

The first part, equation 11 refers to the part affected by the current state. $m\left(x_i^t\right)$ represents the velocity of current particle. Mutation probability P_m is the random number in the range of (0, 0.1), and when P_m is less than ω, execute the operation $m\left(x_i^t\right)$, or otherwise keep the original particles unchanged. For the inertia weight calculation method, see Section D. Here $m\left(x_i^t\right)$ refers to the mutation operation of the chromosome.

$$M_i^t = \omega \otimes m\left(x_i^t\right) = \begin{cases} m\left(x_i^t\right) & P_m < \omega \\ x_i^k & otherwise \end{cases} \quad (11)$$

The mutation operation is as follow:

The OS select a chromosome according to the mutation probability, and select one of the operations randomly. Since the job has a sequence order constraint, firstly, determine the position of the precursor operation and the subsequent operation of the operation, and then randomly select a position to insert the operation between these two positions, which ensures that the resulting schedule is a feasible solution.

The MA select a parent chromosome for the mutation according to the mutation probability, and select one of the processing operations. Since each operation can be processed on multiple machines, each operation has a collection of optional processing machines. Randomly select a machine in the processing machine set to complete the mutation.

The second part, equation 12 indicates the learning part of the chromosome to itself, and $q\left(M_i^t, pB_i^t\right)$ indicates the adjustment of the chromosome according to its optimal position pB_i^t. Crossover probability P_{c_1} is the random number in the range of (0.6, 0.9), and when P_{c_1} is less than c_1, the crossover operation is performed. Otherwise, the original particle remains unchanged. $q\left(M_i^t, pB_i^t\right)$ is implemented through the crossover operation in the genetic algorithm. For the two sets of codes in FJSP, the OS adopts precedence operation crossover (POX)[15], and the MA adopts improved single-point crossover(ISX) [17].

$$Q_i^t = c_1 \otimes q\left(M_i^t, pB_i^t\right) = \begin{cases} q\left(M_i^t, pB_i^t\right) & P_{c_1} < c_1 \\ M_i^t & otherwise \end{cases} \quad (12)$$

The third part, equation 13 embodies the adjustment of particles according to the global optimal position, which shows the cooperation between particles. $p\left(Q_i^t, gB^t\right)$ is the crossover operation between Q_i^t and gB^t. Perform a crossover operation when the crossover probability P_{c_2} is less than c_2, and P_{c_2} is in the range of (0.6,0.9). Or otherwise, keep the original particles unchanged. Crossover operation process: OS uses LOX crossover, and MA uses improved single-point crossover(ISX).

$$P_i^t = c_2 \otimes p\left(Q_i^t, gB^t\right) = \begin{cases} p\left(Q_i^t, gB^t\right) & P_{c_2} < c_2 \\ Q_i^t & otherwise \end{cases} \quad (13)$$

The parent chromosomes after the coding are recorded as P_1, P_2, …, P_n, and the offspring chromosomes obtained after the crossover are recorded as C_1, C_2, …, C_n (n is the population size).

The steps for POX are as follows:

Step 1: P_1 and P_2 are sequentially taken out from the parent chromosomes. Copy all the operations included in job J_1 in P_1 to C_1 in the original order, and copy all the operations contained in job J_2 in P_2 to C_1 in the original order;

Step 2: Copy all the operations included in job J_2 in P_1 to C_2 in the original order, and copy all the operations of the job J_1 in P_2 to C_2 in the original order;

Step 3: Repeat Step 1–Step 2 in the parent chromosomes until n offspring chromosomes C_1, C_2, …, C_n are obtained;

The steps for LOX are as follows:

Step 1: randomly generate two intersection positions in the two parents P_1 and P_2, and exchange the fragments in the two intersection positions;

Step 2: The gene in the original parent, which is exchanged from another parent, is deleted;

Step 3: Copy the remaining genes from the original parent to the children in turn from the first position.

The method of ISX is as follow:

Divide all the chromosomes involved in the crossover into $\frac{n}{2}$ groups, and perform single-point crossover for the two parent chromosomes in each group: randomly select one crossover point, and exchange the machines assigned by the operations included in the two parents before the crossover point, which ensures that the chromosomes obtained after the crossover are all feasible schedulings.

D. CALCULATION OF ADAPTIVE INERTIA WEIGHT

The search process of PSO algorithm is complex and nonlinear. The inertia weight ω is an important parameter of

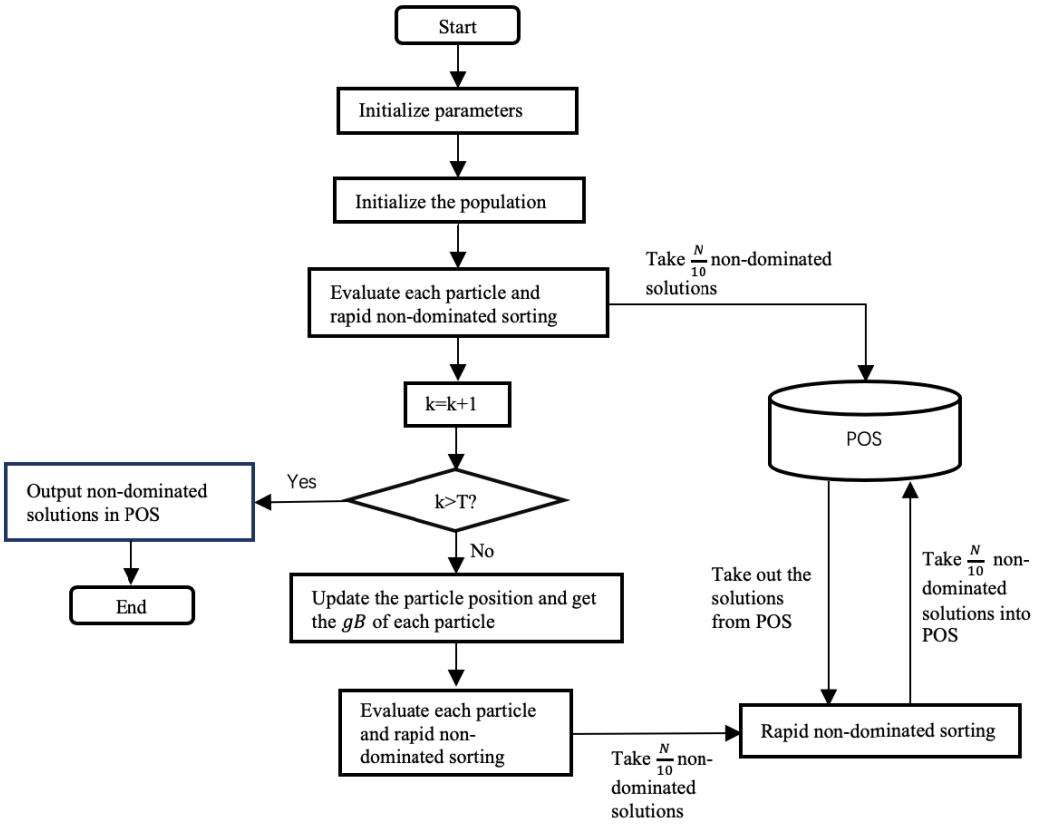


FIGURE 2. The flow chart of DPSO-AIW algorithm.

the algorithm, which is used to balance the global and local search ability of the algorithm. When the inertia weight is large, it is beneficial to global search, while a smaller weight value can accelerate the convergence of the algorithm and prevent the algorithm from falling into local optimum. Therefore, this paper chooses the method of adaptively adjusting the inertia weight to improve the diversity of the population. The inertia weight is updated by the global optimal value of the particles in the population and the exponential function of the current value. See the equation 14 and 15. At each iteration, the inertia weight is related to the current value of the particle, that is, when the relative change between the current value of the particle and the global optimal value is large, the inertia weight increases, and vice versa.

$$\omega(t+1) = \omega_{start} + (\omega_{end} + \omega_{start}) \times (\omega_{end} + \omega_{start})^{\frac{e^{m_i(t)}-1}{e^{m_i(t)}+1}} \quad (14)$$

$$m_i(t) = \frac{gB^t - x_i^t}{gB^t + x_i^t} \quad (15)$$

where t represents the current number of iterations, $\boldsymbol{\omega_{start}}$ is the initial inertia weight, and $\boldsymbol{\omega_{end}}$ is the inertia weight when the iterations reach to the maximum.

E. DPSO-AIW ALGORITHM FLOW

Based on the above analysis and discussion, the DPSO-AIW algorithm for solving the multiobjective FJSP problem is as follows:

Step 1: Set initial parameters: iteration number T, population size N, inertia weight ω, cognitive coefficient c_1 and social coefficient c_2, cycle variable t, and let $t = 1$;

Step 2: Initialize the population P. The OS is randomly generated, and the MA is generated by the GSO method. Evaluate each particle, perform rapid non-dominated sorting on the population, and find out $\frac{N}{10}$ non-dominated solutions in the current population and store in the optimal solution set (POS), the size of the POS is $\frac{N}{10}$, which is one tenth of the population size;

Step 3: Evaluate the fitness value of each particle, and perform rapid non-dominated sorting on the population to find out $\frac{N}{10}$ non-dominated solutions in the current population and compare them with $\frac{N}{10}$ non-dominated solutions in the POS, that is select $\frac{N}{10}$ non-dominated solutions from the $\frac{N}{5}$ non-dominated solutions and update the POS. The process of updating a POS is as follows:

According to the quick sorting method, the front F_i of the non-dominated level is obtained [18]. Let the number of solutions in each front be n_i, and if ($\frac{N}{10} - \sum n_{i-1} \leq n_i$), ($\frac{N}{10} - \sum n_{i-1}$) individuals are randomly selected in the front F_i to store in POS;

Step 4: $t = t + 1$;

Step 5: Determine whether the stopping criterion ($t > T$) is satisfied. if yes, go to Step 7; otherwise, go to Step 6;

Step 6: Update the particle position. The particle position is updated according to equation 7. Go to Step 3;

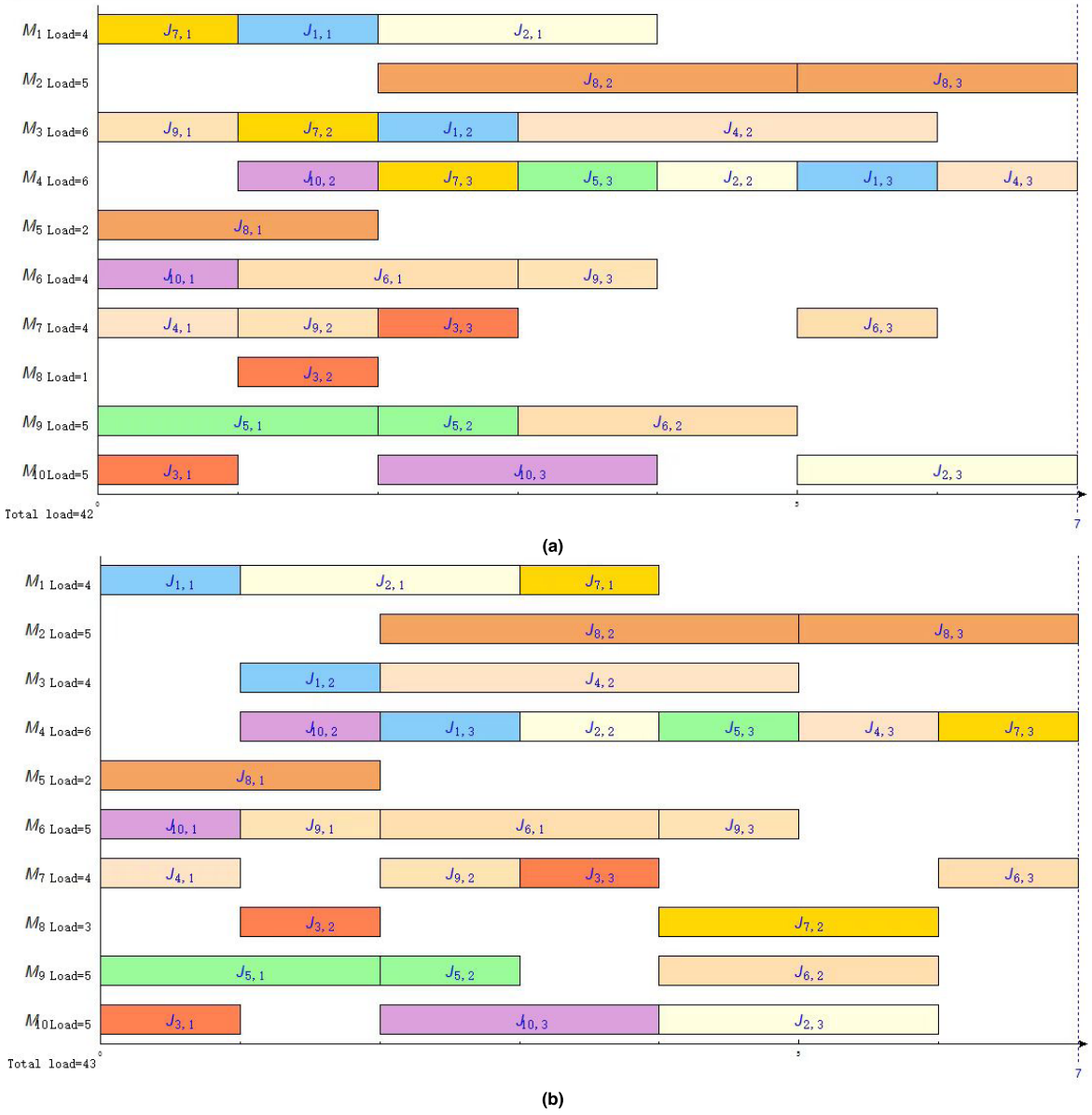


FIGURE 3. Gantt charts of the 10×10 instance obtained by the DPSO-AIW algorithm, (a) and (b) are Gantt charts generated by two different Pareto optimal solutions respectively. The solution of (a) is (7,6,42) and the solution of (b) is (7,6,43).

Step 7: Output non-dominated solutions in the POS.

The flow chart of DPSO-AIW is shown in Figure 2.

V. COMPUTATIONAL COMPLEXITY ANALYSIS

Suppose the number of machines is M, the population size is P, L is the length of the OS and MA in chromosome, the number of iterations is T and the number of objectives is B. The computational complexity of the DPSO-AIW algorithm mainly comes from the following parts: Initialization of the population, mutation and crossover operation, decoding and fast non-dominated sorting. During the initialization of the population, the OS is randomly generated, and the worst computational complexity of one iteration is $O(L \times P)$. The MA is generated using the GSO method, and its worst computational complexity is $O(2 \times M \times L \times P)$. During the mutation operation, the OS mutates randomly, the worst computational complexity of the OS is $O(L \times P)$. The mutation process of the MA is affected by the number of optional machines, the worst computational complexity of the MA is $O(M \times P)$. The crossover operation performed twice. In the first crossover operation, the OS uses POX crossover, the worst computational complexity is $O(4 \times L \times P)$, the MA uses ISX crossover, the worst computational complexity is $O(2 \times L \times P)$. In the second crossover operation, the OS adopts LOX crossover, the worst computational complexity is $O(4 \times L \times P)$, the MA adopts ISX crossover, the worst computational complexity remains $O(2 \times L \times P)$. During the decoding process, each operation in the OS performs the following two steps:

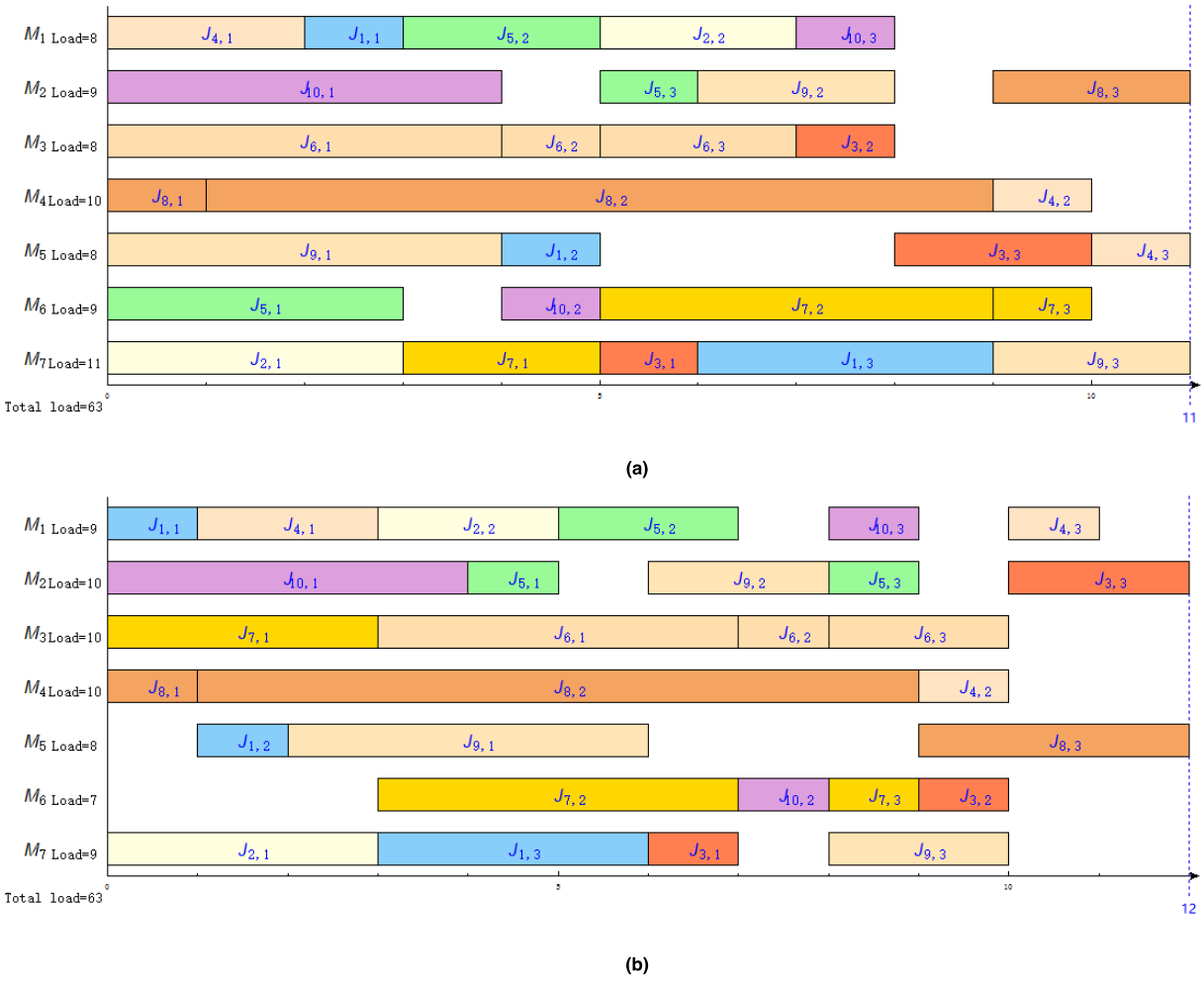


FIGURE 4. Gantt charts of the 10 × 7 instance obtained by the DPSO-AIW algorithm, (a) and (b) are Gantt charts generated by two different Pareto optimal solutions respectively. The solution of (a) is (11,11,63), and the solution of (b) is (12,10,63).

Step 1: Select the corresponding machine in the MA;

Step 2: Search the processing time in its processing schedule and decode to generate a feasible scheduling.

The worst computational complexity of the entire decoding process is $O(M \times L^2)$. The computational complexity of fast non-dominated sorting is $O(M \times B^2)$ [18]. So the worst computational complexity of DPSO-AIW algorithm is:

$$O(L \times P) + O(2 \times M \times L \times P) + O(L \times P) + O(M \times P) + [O(4 \times L \times P) + O(2 \times L \times P)] \times 2 + O(M \times L^2) + O(M \times B^2) \approx O\{[(14+2 \times M) \times L + M] \times P + O[M \times (L^2 + B^2)]$$

It can be seen from the above equation that the computational complexity of the DPSO-AIW algorithm is mainly related to the population size, the number of machines, the number of objectives, chromosome length and the iterations.

VI. SIMULATION AND ANALYSIS

To test the performance of the proposed DPSO-AIW algorithm, numerical simulations were performed using two sets of benchmark studies, including five Kacem instances [14], [19] and three BRdata instances [20]. The algorithm is written in Python programming language version 3.7.3 and run on a MacBook Pro with a 2.7GHz processor and 8 GB RAM of APPLE inc. 2015.

It is generally believed that the difficulty of solving multi-objective FJSP is closely related to the scale of the problem. Therefore, for each instance, the iteration T is 10 × n ×m. The population size P is n × m. ω_{start} is 0.9, ω_{end} is 0.4. c_1 equals to c_2, and the value is 0.75.

The first set of data were tested using the five Kacem instances: 4×5 instance, 10×7 instance, 8×8 instance, 10×10 instance and 15×10 instance. The 8 × 8 instance is a P-FJSP, and the remaining four instances are all T-FJSP. Figure 3 and Figure 4 show the Gantt charts of the two different Pareto optimal solutions obtained by the DPSO-AIW algorithm for solving the 10×10 instance and 10×7 instance respectively. When solving the Pareto optimal solution, we compare the DPSO-AIW algorithm with the existing methods MOPSO +LS [21], PSO-IN [22] and P-EDA [23]. The comparison results are shown in Table 4. The results

TABLE 4. Results of KACEM instances.

n×m	Objective	MOPSO+LS	PSO-IN	P-EDA	DPSO-AIW
4×5	F_1	16 16	-	11 11 12 13	**11** 12 11
	F_2	8 7	-	10 9 8 7	**7** 7 8
	F_3	32 33	-	32 34 32 33	**30** 30 33
10×7	F_1	15 15 16	-	**11** 11 12	12 12 11
	F_2	10 11 12	-	**11** 10 12	10 10 11
	F_3	62 61 60	-	**61** 62 60	63 63 63
8×8	F_1	14 15 16 16 17	15 15 16 16	14 15 16 16	**14** 15 14
	F_2	12 12 13 11 11	11 12 11 13	12 12 13 11	**11** 12 12
	F_3	77 75 73 78 77	82 75 77 73	77 75 73 77	**76** 75 78
10×10	F_1	7 7 8 8	7 7 8 8	7 7 8 8	**7** 7 7
	F_2	6 5 5 7	6 6 5 7	6 5 7 5	**6** 6 5
	F_3	42 43 42 41	42 42 42 41	42 43 41 42	**42** 43 43
15×10	F_1	12	11 13 13 14	11 11	**11** 12 12
	F_2	11	11 11 12 11	11 10	**11** 11 12
	F_3	91	91 93 92 92	91 93	**91** 93 95

TABLE 5. Results of MK01 instance.

MOGA				P-EDA				DPSO-AIW			
NO.	F_1	F_2	F_3	NO.	F_1	F_2	F_3	NO.	F_1	F_2	F_3
1	40	36	169	1	40	36	169	1	**40**	**36**	**168**
2	42	39	158	2	40	37	165	2	40	36	169
3	43	40	155	3	41	37	163	3	40	38	169
4	44	40	154	4	41	38	160	4	41	37	160
				5	42	36	165	5	41	37	165
				6	42	39	158	6	42	36	161
				7	42	40	157	7	42	39	169
				8	43	40	155				

TABLE 6. Results of MK02 instance.

MOGA				P-EDA				DPSO-AIW			
NO.	F_1	F_2	F_3	NO.	F_1	F_2	F_3	NO.	F_1	F_2	F_3
1	26	26	151	1	26	26	151	1	26	26	152
2	27	27	146	2	27	27	145	2	**27**	**27**	**145**
3	29	27	145	3	28	28	144	3	27	29	150
4	29	29	143	4	29	29	143	4	28	30	145
5	31	31	141	5	30	30	142	5	29	29	148
6	33	33	140	6	31	26	150	6	30	29	149
				7	31	31	141	7	31	30	148

of the compared algorithms come directly from the literature. It can be seen from Table 4 that for the 4×5 instance, the Pareto optimal solution (11 7 30) obtained by the DPSO-AIW algorithm is better than the non-dominated solution in the other three algorithms; for the 10×7 instance, P-EDA obtained better solution that dominated other algorithms, for the 8×8 instance, the Pareto optimal solution (14 11 76) obtained by the DPSO-AIW algorithm is better than the non-dominated solution in the other three algorithms. For the 10×10 instance and the 10×15 instance,

TABLE 7. Results of MK03 instance.

MOGA				P-EDA				DPSO-AIW			
NO.	F_1	F_2	F_3	NO.	F_1	F_2	F_3	NO.	F_1	F_2	F_3
1	204	133	884	1	204	204	850	1	**204**	**168**	**850**
2	204	135	882	2	210	210	848	2	204	170	856
3	204	144	871	3	213	213	844	3	206	168	847
4	204	199	855	4	221	221	842	4	208	168	850
5	213	199	850	5	222	222	838	5	210	168	862
6	214	210	849	6	231	231	834	6	210	170	863
7	221	199	847	7	240	240	832	7	210	172	863
8	222	199	848	8	249	249	830	8	210	176	870

TABLE 8. The factor levels of parameters.

Parameters	Factor Level		
	1	2	3
P	$\frac{n \times m}{2}$	$n \times m$	$n \times m \times 2$
ω_{start}	0.85	0.9	0.95
ω_{end}	0.3	0.4	0.5
(c_1, c_2)	0.6	0.75	0.9

TABLE 9. The orthogonal array.

Number	Factor Level				
	P	ω_{start}	ω_{end}	(c_1, c_2)	*Ave*
1	1	1	1	1	11.6
2	1	2	2	2	12.1
3	1	3	3	3	12.4
4	2	1	2	3	11.7
5	2	2	3	1	12.8
6	2	3	1	2	12.4
7	3	1	3	2	11.5
8	3	2	1	3	11.6
9	3	3	2	1	12.4

the DPSO-AIW algorithm obtains the same Pareto optimal solution (7 6 42) and (11 11 91) as the other three methods. The better solution for each objective of DPSO-AIW is shown in bold in Table 4. "-" indicates that no value was given in the original paper. The data show that the DPSO-AIW algorithm is effective and feasible for solving Kacem instances.

The second set of data were tested using the BRdata instances. Select the MK01, MK02, and MK03 issues in the BRdata instances. Comparison of the Pareto optimal solution obtained by the DPSO-AIW algorithm with that obtained by the MOGA [24] and P-EDA algorithms can be seen in Table 5 - Table 7, where the Pareto optimal solution obtained by the DPSO-AIW algorithm is shown in bold, and dominant solutions obtained by other algorithms are underlined. For the MK01 instance in Table 5, the non-dominated solution (40 36 168) obtained by DPSO-AIW dominate all the non-dominated solutions obtained by the other two algorithms. For the MK02 instance in Table 6, the P-EDA and DPSO-AIW obtained the same non-dominated solution (27 27 145), which dominates the non-dominated solutions obtained by the MAGO algorithm. For the MK03 instance in Table 7, the non-dominated solution (204 168 850) obtained by DPSO-AIW dominates all non-dominated solutions obtained by the other two algorithms.

TABLE 10. The mean value and standard deviation of each factor level.

Factor level	P	ω_{start}	ω_{end}	(c_1, c_2)
1	12.03	11.6	11.87	12.27
2	12.3	11.8	12.07	11.97
3	11.83	12.4	11.9	11.77
δ	0.1926	0.3399	0.0881	0.2055

It can be seen that DPSO-AIW is superior to MOGA and P-EDA in solving BRdata instances. In three instances, DPSO-AIW can achieve better Pareto optimal solutions that dominate the solutions obtained by the MOGA algorithm and the P-EDA algorithm. According to the above comparison based on Kacem instances and BRdata instances, it can be concluded that DPSO-AIW algorithm is an effective algorithm for solving multiobjective FJSP problem.

VII. PARAMETER SENSITIVITY ANALYSIS

In order to determine the impact of important parameters on the algorithm, a sensitivity analysis is performed in this section. The 4×5 instance of Kacem is used to assess performance of the DPSO-AIW with different parameter combinations. In this paper, four different factor levels P, ω_{start}, ω_{end} and (c_1, c_2) are considered, and experiments are designed using the orthogonal table of Taguchi method [25]. The optional value of each factor level is presented in Table 8, and the design structure of the orthogonal array is shown in Table 9. Each experiment runs ten times independently. *Ave* denotes the average makespan of ten experiments, and δ is the standard deviation. The mean value of each factor level is presented in Table 10 according to the results in Table 9. In Table 10, δ denotes the standard deviation for each mean value and reflects the significance of each parameter. For comparisons the values of δ, the parameters ω_{start} ranks first, the parameter (c_1, c_2) ranks second, the parameter P ranks third, and the parameter ω_{end} ranks fourth. Therefore, the parameter ω_{start} is the most significant factor in DPSO-AIW algorithm.

VIII. CONCLUSION

In this paper, DPSO-AIW is proposed to solve the multiobjective FJSP, the optimization model is established, and the DPSO-AIW algorithm is applied to the optimization model verification. The simulation data are Kacem standard instances and BRdata standard instances respectively. The simulation results are compared with the results obtained by other algorithms, and the practicability and effectiveness of the algorithm for solving multiobjective FJSP are proved. After a lot of experiments, we found that our algorithm can quickly find better solutions for cases with larger flexibility ratio (flex.)[26] which is defined as the average number of alternate machines per operation, while cases with relatively smaller flex. require more iterations and a larger initial population. It takes more time to get better results, which is what we will do next.

This paper solves the three objectives of the FJSP problem, and there are more goals for complex flexible workshops. Faced with many uncertain factors, how to design a simplified and efficient simulation model applied to the actual workshop is the future research direction. In addition, the application of particle swarm optimization to other combinatorial optimization problems requires further research.

ACKNOWLEDGMENT

The authors would like to appreciate the editors and reviewers for their helpful comments and suggestions to improve the quality of this paper.

REFERENCES

[1] H. H. Chen, Z. Q. Jiang, L. Zuo, and Y. R. Zhang, "Multi-objective flexible job-shop scheduling problem based on NSGA-II with close relative variation," *Trans. Chin. Soc. Agricult. Mach.*, vol. 46, no. 4, pp. 344–350, 2015.

[2] L. Y. Ju, J. J. Yang, J. B. Zhang, L. L. Guo, and S. B. Li, "Improved NSGA for the multi-objective flexible job-shop scheduling problem," *Comput. Eng. Appl.*, vol. 55, no. 13, pp. 260–265 and 270, 2019.

[3] H. Piroozfard, K. Y. Wong, and W. P. Wong, "Minimizing total carbon footprint and total late work criterion in flexible job shop scheduling by using an improved multi-objective genetic algorithm," *Resour., Conservation Recycling*, vol. 128, pp. 267–283, Jan. 2018.

[4] J.-Q. Li, Q.-K. Pan, and J. Chen, "A hybrid Pareto-based local search algorithm for multi-objective flexible job shop scheduling problems," *Int. J. Prod. Res.*, vol. 50, no. 4, pp. 1063–1078, Feb. 2012.

[5] Y. Zhang, S. Cheng, Y. Shi, D.-W. Gong, and X. Zhao, "Cost-sensitive feature selection using two-archive multi-objective artificial bee colony algorithm," *Expert Syst. Appl.*, vol. 137, pp. 46–58, Dec. 2019.

[6] L. Q. Liu, X. L. Zhang, L. M. Xie, and S. H. Wen, "Hybrid Pareto-based particle swarm optimization for multi-objective flexible job shop scheduling problem," *J. North Univ. China (Natural Sci. Ed.)*, vol. 34, no. 02, pp. 134–139, 2013.

[7] M. Chen, Y. L. Hu, and J. F. Jiu, "Multi-objective flexible job shop scheduling problem based on particle swarm optimization," *Mechatronics*, vol. 23, no. 1, pp. 11–15 and 60, 2017.

[8] J. Zhang, W. L. Wang, X. L. Xu, and J. Jie, "Hybrid particle-swarm optimization for multi-objective flexible job-shop scheduling problem," *Control Theory Appl.*, vol. 29, no. 6, pp. 715–722, 2012.

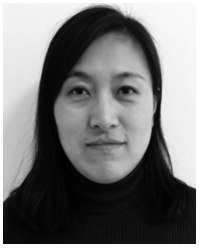
[9] Q. K. Pan, W. H. Wang, J. Y. Zhu, "Modified discrete particle swarm optimization algorithm for no-wait flow shop problem," *Comput. Integr. Manuf. Syst.*, vol. 13, no. 6, pp. 1127–1130 and 1136, 2007.

[10] J. W. Song and J. F. Tang, "No-wait hybrid flow shop scheduling method based on discrete particle swarm optimization," *J. Syst. Simul.*, vol. 22, no. 10, pp. 2257–2261, 2010.

[11] S. Huang, N. Tian, Y. Wang, and Z. Ji, "Multi-objective flexible job-shop scheduling problem using modified discrete particle swarm optimization," *SpringerPlus*, vol. 5, no. 1, p. 1432, Dec. 2016.

[12] J.-Q. Li, Q.-K. Pan, P. N. Suganthan, and T. J. Chua, "A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 52, nos. 5–8, pp. 683–697, Feb. 2011.

[13] X. Gu, M. Huang, and X. Liang, "An improved genetic algorithm with adaptive variable neighborhood search for FJSP," *Algorithms*, vol. 12, no. 11, p. 243, Nov. 2019.
[14] I. Kacem, S. Hammadi, and P. Borne, "Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic," *Math. Comput. Simul.*, vol. 60, nos. 3–5, pp. 245–276, Sep. 2002.
[15] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, 2008.
[16] L. Gao, G.H. Zhang, X. J. Wang, *Flexible Job Shop Scheduling Algorithm and Its Application*. Wuhan, China: Huazhong Univ. of Science and Technology Press, 2012, pp. 35–38.
[17] J. Q. Wang, F. L. Wang, and H. X. Zhu, "Single-point crossover multi-offspring genetic algorithm," *J. Biomath.*, vol. 30, no. 2, pp. 305–311, 2015.
[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[19] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 32, no. 1, pp. 1–13, Feb. 2002.
[20] *Flexible Job Shop Problem*. Accessed: Oct. 10, 2019. [Online]. Available: http://www.idsia.ch/~monaldo/fjsp.html
[21] G. Moslehi and M. Mahnam, "A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search," *Int. J. Prod. Econ.*, vol. 129, no. 1, pp. 14–22, Jan. 2011.
[22] Y. J. Zhong, H. C. Yang, R. Mo, and H. B. Sun, "Optimization method of flexible job-shop scheduling problem based on niching and particle swarm optimization algorithms," *Comput. Integr. Manuf.Systems*, vol. 21, no. 2, pp. 3231–3238, 2015.
[23] L. Wang, S. Wang, and M. Liu, "A Pareto-based estimation of distribution algorithm for the multi-objective flexible job-shop scheduling problem," *Int. J. Prod. Res.*, vol. 51, no. 12, pp. 3574–3592, Jun. 2013.
[24] X. Wang, L. Gao, C. Zhang, and X. Shao, "A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 51, nos. 5–8, pp. 757–767, Nov. 2010.
[25] S. Gaonkar, N. Karanjavkar, and S. N. Kadam, "Taguchi method," *Int. J. Sci. Res. Sci. Eng. Technol.*, vol. 2, no. 2, pp. 1990–2395, 2016.
[26] T. F. Abdelmaguid, "A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times," *Appl. Math. Comput.*, vol. 260, pp. 188–203, Jun. 2015.



XIAO-LIN GU was born in Dalian, Liaoning, China, in 1978. She received the B.S. degree in computer science from Northeastern University, Shenyang, China, in 2001, and the M.S. degree in computer application technology from Dalian Jiaotong University, Dalian, China, in 2006, where she is currently pursuing the Ph.D. degree in mechanical engineering.

Since 2001, she has been serving as an Assistant Professor with Dalian Jiaotong University. Since 2006, she has been serving as a Lecturer. Her main research interests include job shop scheduling, optimization algorithms, and computer-integrated manufacturing systems.

MING HUANG received the B.S. degree in computer science from the Wuhan Institute of Geodesy and Geomatics, Wuhan, China, in 1983, the M.S. degree in computer science from Jilin University, Changchun, China, in 1989, and the Ph.D. degree from the Dalian University of Technology, Dalian, China, in 1999. He is currently a Professor and a Doctoral Supervisor with Dalian Jiaotong University. His main research interests are computer integrated manufacturing, scheduling algorithms, and optimization algorithm research.

XU LIANG graduated from the Department of Computer Science, Sichuan University, Chengdu, China, in 1997. She received the B.S. degree in traffic engineering from the Dalian Railway Institute, in 2001, and the Ph.D. degree in mechanical engineering from Dalian Jiaotong University, in 2005. She is currently a Professor and a Doctoral Supervisor with Dalian Jiaotong University. Her research interests include system optimization theory and applications, and manufacturing informatization.