Journal of Systems Engineering and Electronics
Vol. 32, No. 2, April 2021, pp.272 – 285

An improved multi-objective optimization algorithm for solving flexible job shop scheduling problem with variable batches

WU Xiuli[1,*], PENG Junjian[1], XIE Zirun[1], ZHAO Ning[1], and WU Shaomin[2]

1. School of Mechanic Engineering, University of Science and Technology Beijing, Beijing 100083, China;
2. Kent Business School, University of Kent, Kent CT2 7FS, UK

Abstract: In order to solve the flexible job shop scheduling problem with variable batches, we propose an improved multi-objective optimization algorithm, which combines the idea of inverse scheduling. First, a flexible job shop problem with the variable batches scheduling model is formulated. Second, we propose a batch optimization algorithm with inverse scheduling in which the batch size is adjusted by the dynamic feedback batch adjusting method. Moreover, in order to increase the diversity of the population, two methods are developed. One is the threshold to control the neighborhood updating, and the other is the dynamic clustering algorithm to update the population. Finally, a group of experiments are carried out. The results show that the improved multi-objective optimization algorithm can ensure the diversity of Pareto solutions effectively, and has effective performance in solving the flexible job shop scheduling problem with variable batches.

Keywords: flexible job shop, variable batch, inverse scheduling, multi-objective evolutionary algorithm based on decomposition, a batch optimization algorithm with inverse scheduling.

DOI: 10.23919/JSEE.2021.000024

1. Introduction

Optimization of batch scheduling is one of the important research topics in batch scheduling [1]. This paper aims to investigate the flexible job shop scheduling problem with variable batches (FJSP-VB). Compared to the classic flexible job shop scheduling problem [2], the FJSP-VB needs to determine the batch size for each job besides machine assignment and machine scheduling. The FJSP-VB is therefore more complex than the FJSP and it is challenging to solve it.

According to different job shop types, batch scheduling can be categorized into the parallel machine batch scheduling problem [3,4], the flow shop batch scheduling problem [5−8], the job shop batch scheduling problem [9], and the flexible job shop batch scheduling problem [10−17]. Since Reiter [18] first introduced the concept of "batch flow" in the context of the job shop scheduling problem in 1966, the scheduling problem considering batch production has attracted increasing interest from authors. However, there are some challenges on batch scheduling problems in the FJSP context that needs investigation [10].

There are some studies on the FJSP that considers batches. For example, Zhang et al. [1] proposed a competitive cooperative bird migration optimization algorithm for the FJSP-VB. They designed a chromosome with a three-dimensional (3D) structure to optimize the batch size. For the same problem, Novas [10] used the constraint programming and defined the maximum batch number of jobs in the constrained programming (CP) model, according to which they further generated the solution space. Although the batch size can be set to zero so that the batch size can be reduced, there are still a large solution space and challenges for the optimization algorithm. Defersha et al. [13] used the genetic algorithm to study the FJSP, considering batches. Although the variable batch problem was considered, the batch number was assumed known. This may limit the overall optimization of the problem. Ham et al. [14] proposed the CP model and the mixed integer programming (MIP) model with efficient inequality to solve the FJSP with parallel batch processors. Zhou et al. [15] studied the multi-objective FJSP with batch and dynamic factors in the aero-engine blade manufacturing factory. They improved the hyper-heuristics with multi-agent. Bozek et al. [16] used the tabu search to solve the FJSP-VB. Jia et al. [17] designed a multi-objective ant colony optimization algorithm to solve the FJSP with batches. They proposed a local optimization method to improve the ant colony op-

Manuscript received December 02, 2020.
*Corresponding author.
This paper was supported by the National Key R&D Plan (2020YFB1712902) and the National Natural Science Foundation of China (52075036).

timization algorithm.

Existing literature usually assumes that the batch size of each job is known in advance, under which the optimization result is simply a scheduling solution of the known batches. It is hard to obtain the globally optimal solution for the FJSP-VB. Therefore, this paper proposes an improved multi-objective optimization algorithm (IMOA). The main contributions of this work are as follows.

(i) A model of the FJSP-VB to optimize the makespan and the transportation time is formulated.

(ii) An IMOA is proposed to search the global optimal solution for the FJSP-VB.

(iii) A batch optimization algorithm based on inverse scheduling (BOA-IS) is proposed to optimize the batch size.

(iv) Two strategies for updating the neighborhood are proposed to increase the diversity of the population.

The remainder of the paper is organized as follows. Section 2 describes the FJSP-VB. Section 3 proposes the IMOA. Section 4 provides numerical experiments. Section 5 concludes the paper and proposes future work.

2. FJSP-VB

2.1 Problem description

The FJSP-VB can be described as follows. There are n jobs to be processed on m machines. Each job is composed of a number of operations and each operation is performed on available machines. The processing time of an operation on its available machines may be different. The demand amount of each job can be divided into multiple sub-batches to process. Operations belonging to the same sub-batch of a job should follow the processing constraints. Operations in different sub-batches of the same job need not be subject to the constraints. Some time is required for jobs to transfer from one machine to another. The task of the FJSP-VB is to assign the available machines for each operation in each sub-batch and then determine the processing sequence on each machine. The objective is to optimize the makespan and the transferring time.

In order to simplify the problem, some assumptions are made as follows.

(i) The initial available time of all machines is 0, and all jobs can be processed at $t = 0$;

(ii) The amount of the demand of each job is known in advance;

(iii) Each sub-batch is independent during scheduling;

(iv) The processing time of the operation on different available machines is known;

(v) Machine maintenance, emergency orders or machine breakdown is not considered in this study.

2.2 Model formulation

2.2.1 Notations

Some notations are given in Table 1.

Table 1 The notations

Notation	Definition
m	Amount of the machines
n	Amount of the jobs
l	Machines index, $l = 1, 2, \cdots, m$
i	Jobs index, $i = 1, 2, \cdots, n$
$\boldsymbol{J_i}$	Sub-batches of job i
j	Sub-batch index of job i, $j = 1, 2, \cdots, \lvert\boldsymbol{J}_{ii}\rvert$
$\boldsymbol{K_i}$	Operations of job i
k	Operation index, $k = 1, 2, \cdots, \lvert\boldsymbol{K}_i\rvert$
O_{ijk}	The kth operation of the jth sub-batch of job i
t_{ijkl}	Processing time of operation O_{ijk} on machine l
t_{ijk}	Processing time of operation O_{ijk}
b_i	Batch size of job i
b_{ij}	Batch size of the jth sub-batch of job i
b'_{ij}	Batch size of the jth sub-batch of job i after adjustment
n_{ij}	Actual added batch size of the jth sub-batch of job i
n'_{ij}	Virtual added batch size of the jth sub-batch of job i
$C_{\max}$	The makespan
S_{ijk}	Start time of operation O_{ijk}
C_{ijk}	Ending time of operation O_{ijk}
T	Total transportation time
$T_{l_1l_2}$	Transferring time from machine l_1 to machine l_2; $T_{l_1l_2}=0$, if $l_1 = l_2$
x_{ijkl}	$x_{ijkl}=1$ if the operation O_{ijk} is processed by machine l; otherwise $x_{ijkl}=0$
x_{ijklt}	At time t if $x_{ijkl}=1$, $x_{ijklt}=1$; otherwise, $x_{ijklt}=0$
y_{ij}	If the jth sub-batch of job i needs to increase the batch size, $y_{ij}=1$; otherwise $y_{ij}=0$

2.2.2 Formulation of FJSP-VB

The following mathematical model can then be given.

$$\min C_{\max} = \min(\max C_{ijk}) \tag{1}$$

$$\min T = \min\left(\sum_{i=1}^{n}\sum_{j=1}^{J_i}\sum_{k=2}^{K}\sum_{l_1=1}^{m}\sum_{l_2=1}^{m} x_{ijkl_2}x_{ij(k-1)l_1}T_{l_1l_2}\right) \tag{2}$$

s.t.

$$C_{ijk}=S_{ijk}+\sum_{l=1}^{m} t_{ijkl}x_{ijkl}, \quad \forall i,j,k \tag{3}$$

$$\sum_{l=1}^{M} x_{ijkl} = 1, \quad \forall i,j,k,l \tag{4}$$

$$S_{ij(k-1)}\sum_{l_1=1}^{m}x_{ij(k-1)l_1}+\sum_{l_1}^{m}t_{ij(k-1)l_1}x_{ij(k-1)l_1}+$$

$$T_{l_1l_2}\leqslant S_{ijk}\sum_{l_2=1}^{m}x_{ijkl_2},\quad \forall i,j,k \tag{5}$$

$$\sum_{j=1}^{J_i}b_{ij}=b_i,\quad \forall i,j \tag{6}$$

$$x_{ijklt}+x_{ghflt}\leqslant 1,\quad \forall i,j,k,l,g,h,f,l,t \tag{7}$$

$$S_{ijk}\geqslant 0,\quad \forall i,j,k \tag{8}$$

$$0<b_{ij}\leqslant b_i,\quad \forall i,j \tag{9}$$

$$J_i\leqslant b_i,\quad \forall i \tag{10}$$

$$x_{ijklt}\in\{0,1\},\quad \forall i,j,k,l,t \tag{11}$$

$$y_{ij}\in\{0,1\},\quad \forall i,j \tag{12}$$

$$x_{ijkl}\in\{0,1\},\quad \forall i,j,k,l \tag{13}$$

Equations (1) and (2) are objective functions. Equation (3) indicates that the completion time of each operation equals the sum of the start time and processing time. Equation (4) indicates that each operation can only be processed on one machine. Equation (5) indicates that the start time of an operation is greater than the sum of the completion time of its previous operation and the transferring time. Equation (6) indicates that the sum of the sub-batches for one job is equal to the demand amount. Equation (7) indicates that the overlap between operations is not allowed. Equation (8) indicates that the start processing time of all job batches can start from zero. Equation (9) indicates that the size of each sub-batch of one job is not greater than its demand amount. Equation (10) indicates that the number of sub-batches of one job is not greater than its demand amount. Equations (11)–(13) define the value range of the variable, respectively.

3. IMOA

3.1 IMOA flowchart

In the existing literature, little application of the multi-objective evolutionary algorithm based on decomposition (MOEA/D) algorithm in the flexible job shop batch scheduling problem can be found [19]. MOEA/D has some drawbacks in solving the multi-objective optimization problem. When the Pareto front has a heavy tail, the performance of the MOEA/D algorithm is greatly influenced. Hence, some improved versions of the MOEA/D have been proposed. For example, Dong et al. [20] proposed a chain segmentation based strategy. The approximate Pareto front shape is derived from the initial population, the shape of the front is cut equally to generate the weight vector, and the distribution performance of the algorithm is improved according to the improved weight vector. Jiang et al. [21] proposed a hybrid multi-objective evolutionary algorithm based on decomposition and solved the Pareto front problem with the hybrid algorithm. In order to solve the homogenization problem of neighbors, some MOEA/D based improved algorithms have been proposed [22,23].

In order to avoid being trapped into local optimums during searching and to improve the poor performance of MOEA/D in solving multi-objective combinatorial optimization problems, we propose two methods as follows:

(i) When updating the neighborhood, a threshold is set to limit the updating individuals. For example, there are 10 neighbors and eight neighbors that can be considered to replace the original individuals. If the threshold is 0.5, only four neighbors can be used to update the population. Thus, the diversity of the population is ensured by avoiding generating too many similar or same individuals.

(ii) A clustering algorithm is employed to dynamically adjust the diversity of the neighbors. When the number of individuals in the same cluster exceeds the threshold, some individuals in other clusters are used to replace some of the individuals to avoid premature.

The pseudocode of the IMOA is shown below.

1. Initialize parameters and population;
2. Set iter= 0;
3. WHILE iter < Max_iterate
4. FOR (p = 1, 2,$\cdots$, popsize)
5. Select two individuals in the neighborhood of p: x^a, x^b, and generate new individuals x^c
6. Update neighborhood individuals
7. Use BOA-IS to optimize the size of batches
8. Update neighborhood individuals based on the threshold
9. Update external population *EP*
10. Dynamic adjustment of the neighborhood individual diversity by the clustering algorithm
11. END FOR
12. iter = iter + 1
13. IF iter = Max_interate
14. IF The number of sub-batch of job > 1
15. IF Currently optimized solution $\succ$ Previous optimized solution
16. Increase the batches size of jobs
17. iter = 0
18. ELSE
19. Output the previous optimization solution

```
            BREAK
          END IF
        ELSE
          Increase the batch number of jobs;
          iter = 0
        END IF
      END IF
END WHILE
```

The detailed steps are as follows:

Step 1 Initialize parameters, including population size popsize, crossover probability *pc*, mutation probability *pm*, neighborhood weight vector number *T*, maximum iteration number Max_iterate; initialize the population Pop and the external population *EP*; initialize the weight vector $\boldsymbol{w}$ and ideal point *z*.

Step 2 Assign a weight vector to each sub-problem and set the iteration number iter = 0.

Step 3 For each individual *p* in the population, randomly select two individuals in its neighborhood. A random number between 0 and 1 is generated. If *pc* is larger than the random number, the two individuals are crossed to generate an offspring; otherwise, the crossover operation step will be skipped.

Step 4 Randomly generate a random number between 0 and 1. If *pm* is larger than the random number, the two selected individuals will be mutated to generate two offspring individuals; otherwise, the mutation operation step will be skipped.

Step 5 Execute the BOA-IS algorithm to adjust the batch size of sub-batch of each job.

Step 6 Update the ideal solution *z*.

Step 7 Update the neighborhood solution with the threshold strategy. The neighborhood individuals are randomly updated according to the similarity with a predefined ratio.

Step 8 Update the external population *EP*. Compare the offspring individuals with the individuals in the external population, and remove the individuals dominated by the offspring individuals in the external population. If the offspring individuals are not dominated by any of the individuals in the external population, they are added to the external population.

Step 9 Adjust the diversity of neighborhood individuals dynamically with the clustering algorithm. The neighborhood individuals of the current individual *x* are selected for the cluster analysis. If the number of the individuals with a very high similarity in one cluster exceeds a predefined threshold, update those individuals partially.

Step 10 If the terminal condition is met, check whether the batch number of a job is greater than 1 or not. If so, execute Step 11. Otherwise, return to Step 3. If the terminal condition is not met, return to Step 3 directly.

Step 11 Compare the solution with the solution of the previous iteration. If the solution in this iteration is better, increase the batch number of jobs on the critical path of the non-dominated solution, and initialize the batch information, and return to Step 3. Otherwise, output the solution of the previous iteration and then end the searching.

3.2 Details of the IMOA

3.2.1 Coding

To propose an evolutionary algorithm, the first task is to represent the problem into a chromosome. An effective chromosome should include not only the scheduling information, but also the batching information of each job. We therefore propose an encoding method for the FJSP-VB. A chromosome is composed of three parts. The first part is the batch number for each job, the second part is the size for each batch size, and the last part is the scheduling information. For example, if there is an instance which has three jobs, each job has three operations, and the demand amount of each job is 10. The chromosome is encoded, as shown in Fig. 1. There are three genes (i.e., "2 1 1") in the first part, which means that job 1 is divided into two sub-batches, and job 2 and job 3 are both one sub-batch respectively. There are four genes (i.e.,"4 6 10 10") in the second part, which indicates that the numbers of the two sub-batches of job 1 are 4 and 6 respectively, the size of the sub-batch of job 2 and job 3 is 10. In the third part of batch scheduling information, taking the second occurrence of "2-1" as an example, it represents the second operation of the first sub-batch of job 2.

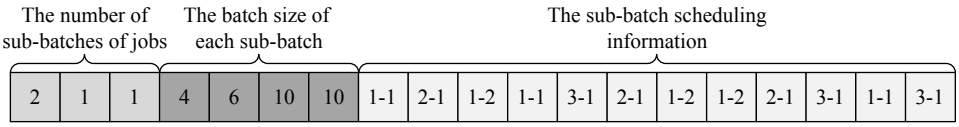

Fig. 1 An example for encoding

3.2.2 BOA-IS for decoding

Reverse scheduling is a scheduling optimization method by adjusting parameters to further optimize the original feasible scheduling solution. Up to now, the study on reverse scheduling is still in its early stage [24]. In this study, we introduce the idea of reverse scheduling in designing the scheduling algorithm. Since the processing

parameters such as processing time, production batch size of job, and transferring time between machines, of the FJSP-VB are not allowed to be changed, the only parameter for the inverse scheduling method to adjust is the sub-batch size of each job, which is also a decision variable. During adjusting, it is necessary to ensure that the amount of the demand of each job cannot change. Therefore, when the batch b_{ij} of a certain sub-batch j of job i is increased by n_{ij}, the batch size of other sub-batches of the job should be reduced accordingly. To calculate the size of each sub-batch of job i, a virtual variable n'_{ij} indicating the quantity of the sub-batch that the current sub-batch has increased or decreased is introduced based on the calculation of n_{ij}. That is, $n'_{ij} > 0$ means that the sub-batch size of b_{ij} is increased, and $n'_{ij} < 0$ means that the sub-batch size of b_{ij} is decreased. Thus the total batch size of job i becomes $b_i + n'_{ij}$. In order to keep the original batch size unchanged, we need to scale down the sub-batch size. The sub-batch size is $\frac{b_i}{b_i+n'_{ij}}$ reduction. Thus the batch size of the sub-batch j of job i is exactly equal to $b_{ij} + n_{ij}$. It can be expressed as

$$b_{ij} + n_{ij} = \frac{b_{ij} + n'_{ij}}{b_i + n'_{ij}} \cdot b_i \tag{14}$$

where $b_{ij}+n'_{ij}$ is the virtually increased batch size of the sub-batch j of a job i, $\frac{b_{ij}+n'_{ij}}{b_i+n'_{ij}}$ indicates the ratio, $\frac{b_{ij}+n'_{ij}}{b_i+n'_{ij}} \cdot b_i$ is the actual increased batch size. Hence, the virtually increased batch size n'_{ij} can be calculated with (15).

$$n'_{ij} = \frac{b_i \cdot n_{ij}}{b_i - n_{ij} - b_{ij}}, \quad \forall i, j \tag{15}$$

After n'_{ij} is obtained, the final batch size b'_{ij} of each sub-batch can be calculated with (16), where $\frac{b_i}{b_i+n'_{ij}}$ is the ratio, and b_{ij} is the original batch size of the sub-batch j of job i. If $y_{ij} = 1$, the increased batch size is $n'_{ij} \cdot y_{ij}$; otherwise $y_{ij} = 0$, the batch size is reduced according to the ratio $\frac{b_i}{b_i+n'_{ij}}$.

$$b'_{ij} = \frac{b_i}{b_i + n'_{ij}} \cdot (b_{ij} + n'_{ij} \cdot y_{ij}), \quad \forall i, j \tag{16}$$

BOA-IS makes full use of the idle time slot of the scheduling solution, obtained by the gap extrusion method (GEM) [25]. It adjusts the batch size of each sub-batch with the aims to reduce the idle time slot, improve the utilization of machines, and reduce the makespan. The detailed steps are as follows. The first step is to find all the idle time slots available on the machine for the current operation to be scheduled. The second step is to find the idle time slots whose starting time are greater than the completion time of the precedence operation and the slot length is greater than its processing time.

To make it easier to understand the BOA-IS, we give an example in Table 2, where M_1, M_2 and M_3 indicate machine 1, machine 2 and machine 3, respectively.

Table 2 An example for FJSP-VB

Job	Batch size	Operation	M_1	M_2	M_3
1	12	O_{11}	10	10	—
		O_{12}	2	2	—
		O_{13}	—	—	3
2	6	O_{21}	2	—	—
		O_{22}	—	5	—
		O_{23}	—	—	5

During the processing process, job 1 is divided into two sub-batches to process and job 2 is only one batch to process. The batch scheduling information in a chromosome is [O_{111}, O_{121}, O_{211},O_{112}, O_{113}, O_{212}, O_{213}, O_{122}, O_{123}]. The scheduling Gantt chart is shown in Fig. 2. Each rectangle represents an operation. The data in each rectangle is in the form of "a/b/c", where a represents the job index, b indicates the sub-batch index and c means the operation index. For example, "1/2/2" means the operation 2 in sub-batch 2 of job 1.

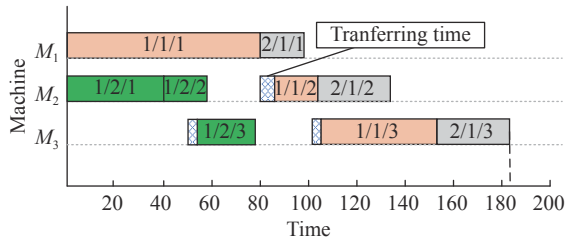

Fig. 2 Scheduling solution

It can be seen from Fig. 2, the length of the idle time slot on machine 2 is 32. The processing time of the first operation of job 1 on machine 2 is 10 and the second operation of job 1 on machine 2 is 2. If we add the batch size by 1 for the second sub-batch of job 1, the idle time slot length is reduced by 10+2. Hence, for this idle time slot, we adjust the sub-batch size by b. The following constraint should therefore be met.

$$b \cdot (10+2) \leqslant 32 \leqslant (b+1) \cdot (10+2) \tag{17}$$

According to (17), the batch size of the second sub-batch of job 1 can be accurately increased by 2. Since the total demand amount is a constant, the batch size of the first sub-batch of job 1 has to be reduced by 2 accordingly. Thus, the adjusted scheduling solution is shown in Fig. 3. Comparing Fig. 2 with Fig. 3, one can see that the

makespan is reduced from more than 180 time units to fewer than 180 time units. Hence, it can be concluded that the BOA-IS is effective in reducing the makespan of a scheduling solution.

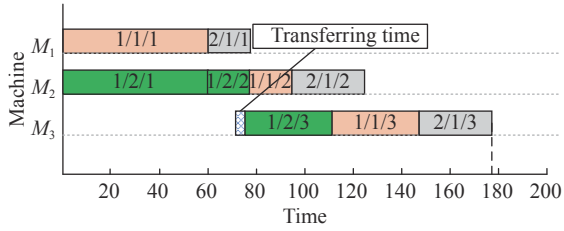

Fig. 3 Adjusted scheduling solution

To be more general, the pseudo of the BOA-IS is given belows. Each step of the BOA-IS is as follows.

1.$l = 0, j = 1$
2.Get scheduling information
3.Generate a scheduling solution with GEM
4.WHILE $l < m$
5. WHILE $j \leqslant$ len (M_l.)
6. IF $j = 1$
7. IF $S(M_{lj}) > 0$ AND $b_{M_{lj}} > 1$ AND $J_{M_{lj}} > 1$
8. Adjust the batch size of each sub-batch of the job corresponding to M_{lj}
9. END IF
10. ELSE
11. IF $S(M_{lj}) > S(M_{l(j-1)})$ AND $J_{M_{lj}} > 1$ AND $M_{lj!} = M_{l(j-1)}$
12. Adjust the batch size of each sub-batch of the job corresponding to $M_{l(j-1)}$
13. END IF
14. $j = j + 1$
15. END IF
16. END WHLIE
17. $l = l$+1
18. IF $l == m$
19. IF The solution before adjustment > The solution after adjustment
20. Update the solution before adjustment with the solution after adjustment
21. $l = 0$
22. ELSE
23. Output the solution before adjustment
24. END IF
25. END IF
26.END WHILE

Step 1 Input a chromosome and select the batch scheduling information of the chromosome, where M_{lj} represents the jth operation of machine l and the total number of M_{lj} is defined as len(M_l); $S(M_{lj})$ indicates the start time of the jth operation of machine l; $E(M_{lj})$ indicates the completion time of the jth operation of machine m; $b_{M_{lj}}$ is the batch size of M_{lj}.

Step 2 Call the gap extrusion method to generate a scheduling solution and record the result before adjustment.

Step 3 Traverse each of the machines.

(i) Traverse the operations on the current machine in turn.

(ii) Judge whether M_{lj} is the first operation processed on the current machine. If not, go to Step 6; otherwise judge whether the start time of M_{lj} is larger than zero, the batch size of M_{lj} is larger than 1, and the demand amount of the corresponding job is larger than 1. If yes, reduce the batch size of the current sub-batch and adjust the batch sizes of the other sub-batch of the job; otherwise, return to (i).

(iii) Check whether the start time of M_{lj} is greater than the completion time of its precedence operation and the batch number of the current job is larger than 1, and M_{lj} and $M_{l(j-1)}$ are not in the same sub-batch. If yes, increase the batch size of $M_{l(j-1)}$, and reduce the batch size of the other sub-batches of the corresponding job.

Step 4 Check whether each machine has been traversed. If not, return to Step 3; otherwise, check whether the adjusted solution dominates the original one. If yes, update the original solution with the adjusted solution and return to Step 3 to continue traversing machines; otherwise, output the solution before adjustment.

The detailed steps of batch adjustment of jobs in the pseudo of the BOA-IS corresponding to $M_{l(j-1)}$ are as follows.

Step 1 Calculate the idle time slot length T_1 of the current machine.

Step 2 Find all the operations that are located before M_{lj} and belong to the same sub-batch as M_{lj} on machine l and store them into record set O.

Step 3 Sum the processing time of the operation in set O and record it as T_2.

Step 4 Calculate the increment or decrement of batch size $b_{M_{lj}}$ corresponding to M_{lj} according to the constraint (16).

Step 5 If the batch size after adjustment is larger than the demand amount of the job or less than zero, readjust it; otherwise, accept this adjustment.

The pseudocode is shown below.

1. Find the idle time slot for the machine T_1
2. Store the operations belonging to the sub-batch before M_{lj} on machine l into the operation set O
3. Sum the processing time of the operations in set O to obtain T_2
4. IF $T_1 > T_2$
5. Calculate the adjusted sub-batch size b of M_{lj}
6. IF $b_{M_{lj}} + b > b_i$ OR $b_{M_{lj}} + b < 0$

7. Readjust the batch size of the sub-batch
8. ELSE
9. Adjust batch size of each sub-batch corresponding to M_{lj}
10. END IF
11. END IF

Since the chromosome encoding method does not include the machine assignment, it is necessary to decode the chromosome according to GEM proposed by Wu et al. [25]. The detailed steps of GEM are as follows.

Step 1 Input a chromosome.

Step 2 Divide the chromosome into three parts, i.e., the batch number of each job, the batch size of each sub-batch and the batch scheduling information. Then calculate the processing time of each sub-batch according to the batch size.

Step 3 In the batch scheduling part, denoted as $\boldsymbol{X}$, select an operation O_{ijk} in $\boldsymbol{X}$.

(i) Select all the available machines M for operation O_{ijk}.

(ii) Search each of the idle time slots on each machine from the completion time of operation $O_{ij(k-1)}$(i.e., the starting point) to the current completion time on the current machine (i.e, the end point). Compare the processing time of the operation and the idle time slot length. If the length is greater than the processing time of the operation, record the idle time slot and append it in the record set Q.

(iii) Check whether each of the idle time slots on all the available machines has been traversed. If yes, choose the idle time slot whose start time is the earliest in record set Q to insert the operation O_{ijk}. Otherwise, return to (ii).

(iv) Judge if all operations have been traversed. If yes, the GEM ends; otherwise, return to Step 3.

3.2.3 Clustering algorithm

By setting a threshold, the updating part of the neighbors can avoid the rapid homogenization of the neighborhood individuals, but the neighborhood individuals will gradually lose diversity with iterations. In order to improve the diversity of the population, IMOA introduces the clustering algorithm to dynamically adjust the diversity of neighborhood individuals. Fig. 4 is its flow chart.

The detailed steps are as follows:

Step 1 Generate the neighborhood of the chromosome;

Step 2 Select an individual x that is not clustered into a cluster.

Step 3 Calculate the similarity between individual x and the other individual in the neighborhood, and add individual x to the cluster that includes the individual with the highest similarity with individual x. Judge if all the individuals have been added to one of the clusters. If yes, go to Step 4. Otherwise, return to Step 2. The method to calculate the similarity between two chromosomes is shown in Fig. 5. If the alleles of two chromosomes are the same, the score will be 1; 0, otherwise. For the example in Fig. 5, the total score is 8. Hence the similarity of the two chromosomes is 8.

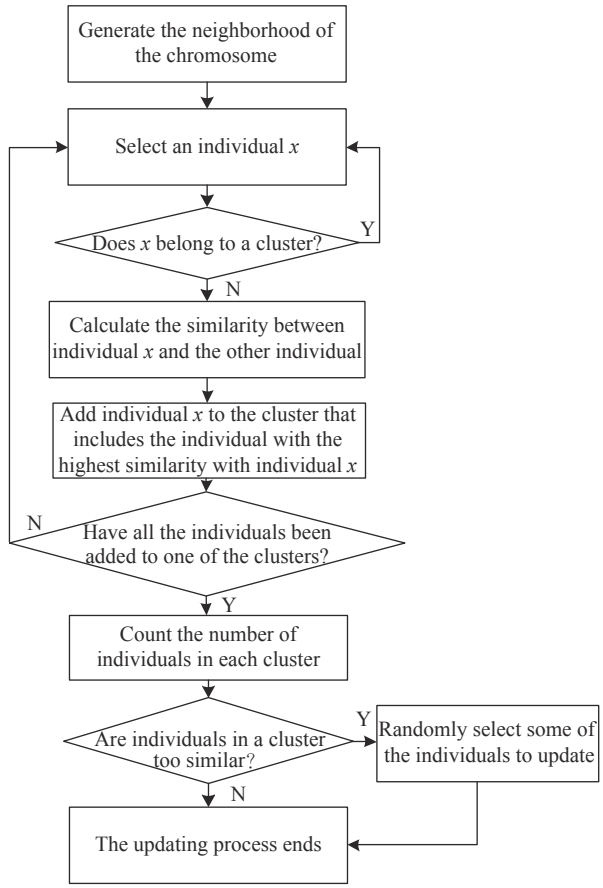

Fig. 4 Flowchart of the clustering algorithm

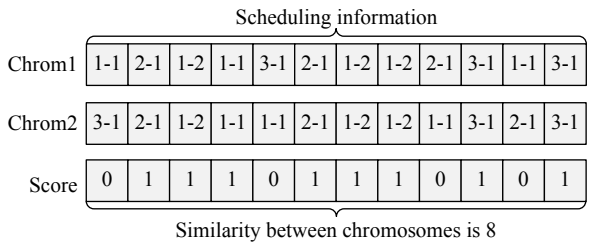

Fig. 5 An example to show how to calculate the similarity

Step 4 Count the number of individuals in each cluster. If the number of individuals in a certain cluster is larger than half of the number of individuals in the neighborhood, randomly select some of the individuals to update. Two updating strategies are randomly selected to

take into account the distribution expansion and the distribution uniformity of Pareto solutions. The two methods are as follows. If the number of individuals in all clusters is smaller than half of the number of individuals in the neighborhood, go to Step 5.

(i) Initialize the individual directly.

(ii) Select the individual of the external cluster which are inclined to the peak or the long tail. First cross it with the individual to be updated to obtain an offspring and then mutate the offspring.

Step 5 The algorithm ends and the updating is completed.

3.2.4 Crossover

In IMOA, in order to avoid the disadvantage of the single crossover method and explore the solution space better, we employ two crossover methods and they are randomly chosen to generate new individuals, including linear order crossover (LOX) and multi-point crossover (MPX).

(i) LOX operation (see Fig. 6). Randomly select two positions in the batch scheduling information part of the chromosome. Exchange the genes between two positions of the parents to generate the offspring.

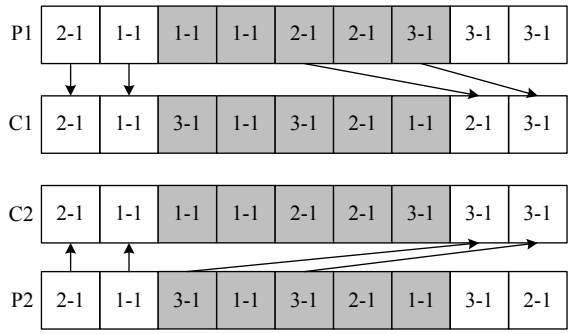

Fig. 6 An example to show how the LOX works

(ii) MPX operation (see Fig. 7). Randomly select two positions in the batch scheduling information part of the chromosome, and exchange the genes in the corresponding positions in the parents and inset the rest genes to the other positions in order.

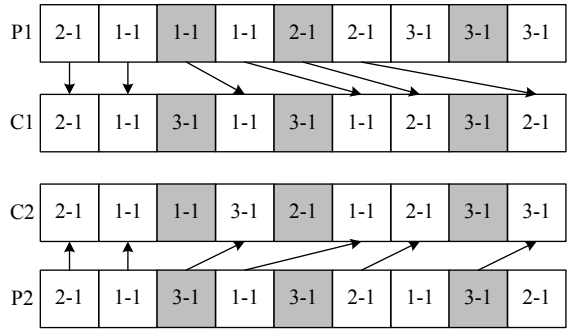

Fig. 7 An example to show how the MPX works

3.2.5 Mutation

Three mutation methods are randomly employed to further explore the solution space and improve the local search ability of the algorithm. They are the reverse based mutation, the exchange based mutation and the insertion based mutation.

(i) The reverse based mutation (see Fig. 8) is to randomly select two positions, reverse the genes within the two positions, and then fill the corresponding position of the chromosome to generate a new chromosome.

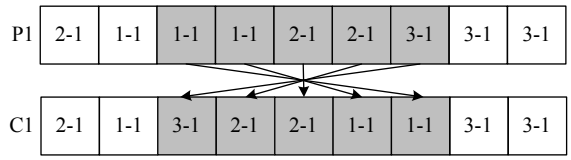

Fig. 8 An example to show how the reverse based mutation works

(ii) The exchange based mutation (see Fig. 9) is to select two positions randomly in the batch scheduling information part, and exchange the genes at these two positions to generate a new chromosome.

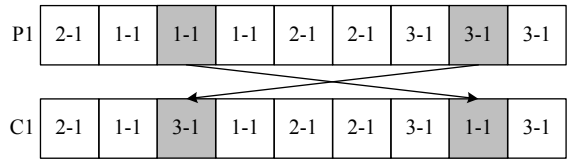

Fig. 9 An example to show how the exchange based mutation works

(iii) The insertion based mutation (see Fig. 10) is to first select two positions randomly in the batch scheduling information gene segment, and then insert the gene in one selected position into another selected position to generate a new chromosome.

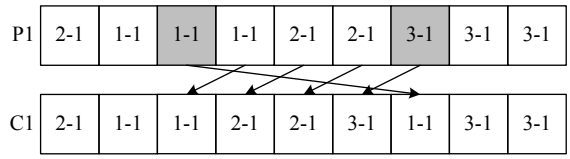

Fig. 10 An example to show how the insertion based mutation works

3.3 Computational complexity of the IMOA

To analyze the performance of the proposed IMOA, we need to compute its computational complexity. From the pseudocode of the IMOA, the "while" loop between line 3 and line 27 is the main process, in which the "for" loop between line 4 and line 11 needs the most computing effort. That means the computational complexity in the worst case is mainly determined by the "for" loop. Hence, we first compute the computational complexity in the worst case for each step in the "for" loop.

For line 5: $O(n\max|\boldsymbol{K}_i|)$

For line 6: $O(n\max|\boldsymbol{K}_i|)$
For line 7: $O(mn^2(\max|\boldsymbol{K}_i|)^2)$
For line 8: $O(1)$
For line 9: $O(\text{popsize})$
For line 10: $O(\text{popsize}\cdot n\cdot\max|\boldsymbol{K}_i|)$

Thus, the total computational complexity for the IMOA in the worst case can be calculated as follows.

$$O(\text{Max\_iterate}\times\text{popsize}(n\max|K_i|+n\max|K_i|+mn^2(\max|K_i|)^2+1+\text{popsize}+\text{popsize}\times n\times\max|K_i|)=O\left(\text{Max\_iterate}\times\text{popsize}\times mn^2(\max|K_i|)^2\right)$$

It can be concluded that the computational complexity for the IMOA in the worst case is proportional to the maximal iteration (i.e., Max_iterate), the population size (i.e., popsize), the machine amount (i.e., m), the square of the job amount (i.e., n^2) and the square of the maximal operation amount (i.e., $(\max|\boldsymbol{K}_i|)^2$).

4. Case study

4.1 Design of experiments

4.1.1 Environment configuration

The approach is compiled and run in Intel(R) Core i54210U 1.70 GHz CPU, 8.00 G RAM, Win 8.1 64 32-bit operating system, and the Python 3.7 programming language and Pycharm programming environment.

Data from Novas [10], Zhao et al. [26] and Defersha et al. [27] are used as test data. By adjusting the parameters, the processing time of Novas [10] is reduced by 20 times. The instances data is shown in Table 3.

Table 3 The instances

Number	Instance	$n\times m$	Batch size of jobs
Test01	Novas [10]	3×3	[32, 48, 31]
Test02	Zhao et al [26]	3×4	[54, 58, 49]
Test03	Novas [10]	3×5	[21, 39, 36]
Test04	Novas [10]	4×5	[9, 20, 25, 6]
Test05	Defersha et al. [27]	4×6	[32, 47, 20, 17]
Test06	Defersha et al. [27]	6×6	[17, 49, 43, 23, 46, 19]
Test07	Novas [10]	8×7	[17, 29, 33, 33, 46, 19, 9, 43]
Test08	Novas [10]	9×8	[22, 47, 35, 19, 36, 38, 28, 30, 38]
Test09	Novas [10]	11×8	[46, 9, 30, 47, 7, 7, 44, 9, 18, 46, 17]
Test10	Novas [10]	12×8	[46, 35, 39, 9, 28, 17, 43, 5, 13, 7, 50, 21]

4.1.2 Purpose of experiments

In order to verify the performance of the proposed algorithm in solving FJSP-VB, we carry out the following five experiments.

Experiment 1 Parameter setting experiment is carried out to determine the best parameter setting of the crossover probability pc, the mutation probability pm and the number of domain weight vectors R.

Experiment 2 The distribution performance experiment of IMOA for solving multi-objective FJSP-VB is carried out to analyze the distribution performance of the Pareto solutions with the instances proposed by Zhao et al. [26].

Experiment 3 The batching mode (i.e., the equal batch size and the variable batch size) comparison experiment is carried out to verify the advantages and disadvantages of the proposed IMOA with the two batching modes.

Experiment 4 BOA-IS performance experiment is carried out to verify the effectiveness of BOA-IS in adjusting the batch size of the scheduling solution.

Experiment 5 The algorithm performance experiment is carried out to verify the performance of IMOA in solving the multi-objective FJSP-VB. The algorithm is compared with the non-dominated sorting genetic algorithm-II (NSGA-II) [2] and the multi-objective differential evolution (MODE) algorithm [28]. NSGA-II and MODE algorithms have good performance in solving multi-objective problems. Therefore, by comparing with NSGA-II and MODE algorithms, the performance of the proposed IMOA can be verified in solving this problem.

4.2 Results of experiments

In Experiment 1, in order to determine the optimal parameters of the crossover probability pc, the mutation probability pm and the threshold R in the algorithm, the hypervolume index (HV), the independent dominating region of the hypervolume index (IHV) and the inverse generational distance (IGD) are taken as the objectives. Take the instance Test02 as the test data, $N=100$ and Max_iterate= 50. The results are shown in Table 4.

In order to analyze the above data clearly, take the mean value of the current objective under the condition that other objectives remain unchanged (see Table 5).

It can be concluded from Table 5 that when the cross probability *pc* is 0.9 and the mutation probability *pm* is 0.1, the three objectives are the best. When *R* is 0.1, HV is the best and when *R* is 0.15, IHV and IGD are the best. Therefore, *pc*, *pm* and *R* are set as 0.9, 0.1 and 0.15, respectively.

In Experiment 2, compare the MOEA/D with modified IMOA and verify the distribution performance of IMOA in terms of the number of Pareto solutions, HV and IHV. Still take Test02 as the test data and the results of running 10 times are shown in Table 6.

Table 4 Results

Indicator	*R*	*pc*=0.7			*pc*=0.8			*pc*=0.9		
		pm=0.1	*pm*=0.15	*pm*=0.2	*pm*=0.1	*pm*=0.15	*pm*=0.2	*pm*=0.1	*pm*=0.15	*pm*=0.2
HV	0.05	**606.61**	1 944.64	1 820.26	2 220.54	944.31	1 429.1	1 970.26	1 273.82	2 396.7
	0.1	1 585.36	1 896.8	**639.58**	2 204.65	2 059.63	1 506.86	2 115.18	134.93	2 911.07
	0.15	1 499.31	941.59	1 030.54	1 092.39	1 102.42	891.66	**564.98**	1 716.67	1 110.39
IHV	0.05	441.24	**98.63**	168.22	209.71	180.28	167.88	108.44	129.01	173.12
	0.1	95.32	91.55	266.91	132.28	258.94	214.67	130.66	**74.2**	192.6
	0.15	92.85	**60.99**	191.52	194.12	123.01	224.42	110.2	205.12	137.68
IGD	0.05	1 291.05	1 439.39	1 348.38	1 411.64	1 430.61	1 888.81	**1 248.72**	1 385.6	1 587.37
	0.1	1 547.89	1 396.05	1 945.29	1 334.35	1 954.2	1 976.07	1 414.73	**899.4**	1 517.29
	0.15	1 553.3	1 225.08	1 438.2	1 464.17	1 193.79	1 248.57	**1 044.2**	1 532.24	1 334.99

Table 5 Analyzed results

HV			IHV			IGD		
pc	*pm*	*R*	*pc*	*pm*	*R*	*pc*	*pm*	*R*
0.7(1 329.41)	**0.1(1 539.92)**	0.05(1 622.91)	0.7(167.47)	**0.1(140.11)**	0.05(186.28)	0.7(1 464.95)	**0.1(1 367.78)**	0.05(1 447.95)
0.8(1 494.61)	0.15(1 334.97)	**0.1(1 672.67)**	0.8(189.47)	0.15(241.41)	0.1(161.90)	0.8(1 544.69)	0.15(1 384.04)	0.1(1 553.91)
0.9(1 577.11)	0.2(1 526.24	0.15(1 105.55)	**0.9(140.11)**	0.2(211.61)	**0.15(148.87)**	**0.9(1 329.39)**	0.2(1 587.21)	**0.15(1 337.17)**

Table 6 Results of the distributed performance experiment

Number	Number of Pareto solutions		HV		IHV	
	IMOA	MOEA/D	IMOA	MOEA/D	IMOA	MOEA/D
1	**11**	7	**1 392.98**	877.17	**135.42**	143.21
2	9	**11**	**1 384.82**	1 175.43	**117.44**	139.39
3	**17**	13	**3 283.24**	1 738.94	**119.16**	267.15
4	**16**	11	**3 432.73**	1 357.31	**48.03**	323.70
5	**20**	12	**3 056.51**	1 722.89	175.76	**127.31**
6	**18**	11	**3 197.70**	1 807.41	**222.03**	358.76
7	**14**	12	**3 787.31**	2 140.67	1 054.95	**294.01**
8	**15**	6	**2 316.36**	201.77	313.59	**77.95**
9	**14**	**14**	1 408.93	**2 296.49**	**128.13**	194.86
10	**20**	12	**3 660.51**	2 461.24	**317.63**	364.25

It can be seen from Table 6 that the number of Pareto solutions obtained by IMOA is more than that by MOEA/D, i.e., IMOA outperforms for eight times. Hence the diversity of solutions obtained by IMOA is better than that by MOEA/D. The HV obtained by IMOA is better than that by MOEA/D, i.e., IMOA outperforms for nine times. Therefore, the expansion of solutions obtained by IMOA are better than those by MOEA/D, respectively. IHV obtained by IMOA is better than that by MOEA/D, i.e., IMOA outperforms for seven times. Hence, the uniformity of solutions obtained by IMOA is better than that by MOEA/D. From the view of statistical analysis, it can

be concluded that the IMOA outperforms MOEA/D in terms of the distribution of the solutions.

In Experiment 3, in order to compare quantitatively the results of equal batching and variable batching, the Pareto solutions obtained by two modes are normalized first, and then calculate the average value [29]. Equation (18) is for normalization, and (19) is for calculating the mean value.

$$f'_{ji} = \frac{f_{ji} - f_{j\min}}{f_{j\max} - f_{j\min}} \tag{18}$$

$$F = \frac{1}{n}\sum_{j=1}^{3}\sum_{i=1}^{n} f'_{ji} \tag{19}$$

Among them, $f_{j\min}$ and $f_{j\max}$ denote the minimum and maximum values of the objective j in the Pareto solution set, respectively. f_{ji} represents the jth objective of the ith Pareto solution; f'_{ji} indicates the normalized f_{ji}; F represents the average of the sum value of all objectives after normalization. The Wilcoxon rank sum test is used to analyze the results. The hypothesis is that there is no significant difference between the two batching methods.

The results are shown in Table 7. For the ten instances, the results of the variable batching are better than those of the equal batching. When the significance level is set as $\alpha = 0.05$, the z value of the standard normal distribution is 2.608, which is larger than the upper critical value of 1.645 under the significance level. Hence, the original hypothesis is rejected and the experimental results produced by the two batching methods are significantly different. Therefore, it can be concluded that the results obtained by the IMOA using the variable batching dominate the results obtained by the IMOA using equal batching.

Table 7 Comparison of experimental results

Instance	Variable batching	Rank	Equal batching	Rank
Test01	**0.90**	17	1.09	20
Test02	**0.51**	3	0.76	9
Test03	**0.59**	4	0.85	15
Test04	**0.72**	8	1.02	19
Test05	**0.43**	1	0.59	5
Test06	**0.67**	7	0.80	11
Test07	**0.65**	6	0.83	13
Test08	**0.50**	2	0.94	18
Test09	**0.78**	10	0.85	16
Test10	**0.83**	12	0.84	14
$z = 2.608 > 1.645$	$n_1 = 10$	$T_1 = 70$	$n_2 = 10$	$T_2 = 140$

In Experiment 4, ten experiments are run with the test case proposed by Zhao et al. [26]. Compare the IMOA integrating BOA-IS with the IMOA integrating GEM. The Pareto solutions are dotted in Fig. 11 for each run. The results obtained with BOA-IS are dotted by red star and the results obtained with GEM are dotted by blue balls.

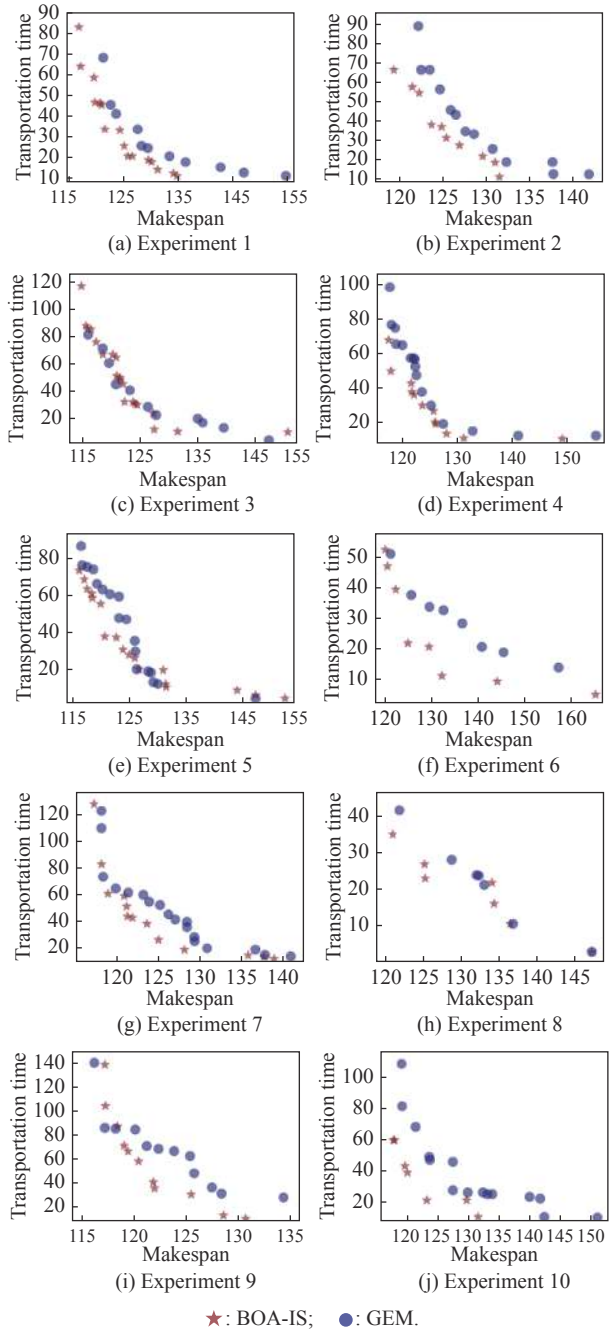

Fig. 11 Comparison of Pareto solutions

In order to further analyze the Pareto solutions obtained by BOA-IS and GEM, select the typical Pareto solution obtained with the two algorithms, respectively. In each running result, for the same batching and batch size, an optimal Pareto solution is selected from the Pareto solutions obtained by the GEM algorithm according to the

priority order of completion time and transportation time. The optimal Pareto solution is found in the Pareto solutions obtained by BOA-IS with the priority that the completion time and transportation time are better than the typical Pareto solutions. The typical Pareto solutions obtained by the two decoding algorithms are shown in Table 8.

Table 8 Comparison of BOA-IS and GEM

Times of test	Makespan		Transferring time	
	BOA-IS	GEM	BOA-IS	GEM
1	**119.70**	121.20	**46.72**	68.40
2	**119.40**	122.20	**66.29**	88.73
3	**115.60**	116.00	88.00	**81.59**
4	**117.50**	117.70	**68.12**	98.55
5	**116.20**	116.50	**73.42**	86.37
6	**120.50**	121.20	**46.91**	50.95
7	118.20	118.20	**83.16**	109.83
8	**120.90**	121.80	**34.99**	41.46
9	117.23	**116.20**	**104.40**	139.90
10	**117.70**	119.20	**59.940**	108.16

From the distribution of Pareto solutions in Fig. 11, it can be concluded that the BOA-IS dominates the GEM. Analyze the experimental results in Table 8, it can be found that only the typical Pareto solution obtained by the GEM in the 9th experiment is better than that by BOA-IS in terms of the completion time. In terms of the transferring time, only in the third experiment, the Pareto solution obtained by the GEM is better than that of BOA-IS. Only in the 3rd and the 9th experiments, BOA-IS and GEM dominate each other in the 10th experiment. Therefore, for the multi-objective FJSP-VB, BOA-IS outperforms GEM from a statistical point of view.

In Experiment 5, in order to compare IMOA, MODE and NSGA-II more intuitively, calculate the mean value of Pareto solutions generated by IMOA, MODE and NSGA-II and report them in Table 9. It can be seen from Table 9 that the results obtained by IMOA are better than those obtained by NSGA-II in most instances except for the 2nd and the 4th experiments, in which IMOA and NSGA-II dominate each other. Except for the 5th and the 7th tests, the results obtained by IMOA are better than those obtained by MODE.

Table 9 Comparison of three algorithms

Test	Makespan			Transportation time		
	IMOEA/ISD	MODE	NSGA-II	IMOEA/ISD	MODE	NSGA-II
Test01	**322.70**	328.52	326.21	**61.27**	69.77	65.19
Test02	195.29	188.87	**186.19**	**37.07**	41.94	59.92
Test03	**348.96**	355.56	367.99	**71.17**	78.72	77.80
Test04	391.89	414.45	**383.82**	**85.68**	86.76	94.31
Test05	**304.79**	356.05	327.18	95.01	**88.51**	98.35
Test06	**832.20**	842.26	834.61	**496.87**	498.05	503.50
Test07	1 244.15	**1 214.49**	1 248.01	**152.24**	223.63	177.51
Test08	**1 395.91**	1 435.80	1 441.40	**348.16**	376.39	376.33
Test09	**1 409.55**	1 565.45	1 501.74	**518.93**	519.10	573.55
Test10	**1 516.48**	1 556.72	1 579.18	**485.99**	506.87	487.03

5. Conclusions

This paper studies FJSP-VB, which improves the MOEA/D and proposes an IMOA. Firstly, two strategies are designed to control the updating of the individuals: one is to update neighbors with a threshold and the other is to introduce the clustering algorithm to dynamically increase the diversity of the population. Secondly, in order to further optimize the batch size of each batch of a job, this paper proposes a batch optimization algorithm on the basis of the inverse scheduling to optimize the batch size of each sub-batch. Finally, in order to verify the performance of the proposed algorithm, five groups of experiments are carried out. The results show that the proposed algorithm solves FJSP-VB effectively.

In the FJSP-VB, challenges include optimization of the batch number and the batch size of each job and reduction of the computing complexity of the algorithm are worthy of in-depth investigation. Besides, although there are many studies on the optimization algorithm for solving the flexible job shop batch scheduling, our future work aims to integrate heuristics and self-learning to improve the performance of the algorithm, and study the uncertainties [30] in the scheduling problem.

References

[1] ZHANG M, TAN Y, ZHU J H, et al. A competitive and cooperative migrating birds optimization algorithm for vary-

sized batch splitting scheduling problem of flexible job-shop with setup time. Simulation Modelling Practice and Theory, 2020, 100: 1–19.
[2] WU X L, PENG J J, XIAO X, et al. An effective approach for the dual-resource flexible job shop scheduling problem considering loading and unloading. Journal of Intelligent Manufacturing, 2021, 32: 707–728.
[3] LI Y J, LI S G. Scheduling jobs with sizes and delivery times on identical parallel batch machines. Theoretical Computer Science, 2020, 841(11): 1–9.
[4] WANG Y, JIA Z H, LI K. A multi-objective co-evolutionary algorithm of scheduling on parallel non-identical batch machines. Expert Systems with Applications, 2021, 167: 114145.
[5] SONG C. Improved greedy genetic algorithm for solving the hybrid flow-shop scheduling problem. Systems Engineering and Electronics, 2019, 41(5): 1079–1086. (in Chinese)
[6] OMID S, RASARATNAM L. A comparison of two stage-based hybrid algorithms for a batch scheduling problem in hybrid flow shop with learning effect. International Journal of Production Economics, 2018, 195: 227–248.
[7] MENG T, PAN Q K, LI J Q, et al. An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem. Swarm and Evolutionary Computation, 2018, 38: 64–78.
[8] LIU S W, PEI J, CHENG H, et al. Two-stage hybrid flow shop scheduling on parallel batching machines considering a job-dependent deteriorating effect and non-identical job sizes. Applied Soft Computing, 2019, 84: 105701.
[9] CHANG J H, CHIU H N. A comprehensive review of lot streaming. International Journal of Production Research, 2005, 43(8): 1515–1536.
[10] NOVAS M J. Production scheduling and lot streaming at flexible job-shops environments using constraint programming. Computers & Industrial Engineering, 2019, 136: 252–264.
[11] GENG Z C, YUAN J J, YUAN J L. Scheduling with or without precedence relations on a serial-batch machine to minimize makespan and maximum cost. Applied Mathematics and Computation, 2018, 332: 1–18.
[12] HUANG R H, YU T H. An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting. Applied Soft Computing, 2017, 57: 642–656.
[13] DEFERSHA F M. Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming. Computers & Industrial Engineering, 2018, 117: 319–335.
[14] HAM A M, CAKICI E. Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches. Computers & Industrial Engineering, 2016, 102: 160–165.
[15] ZHOU Y, YANG J J, ZHENG L Y. Multi-agent based hyper-heuristics for multi-objective flexible job shop scheduling: a case study in an aero-engine blade manufacturing plant. IEEE Access, 2019, 7: 21147–21176.
[16] BOZEK A, WERNER F. Flexible job shop scheduling with lot streaming and sublot size optimization. International Journal of Production Research, 2017, 56(19): 6391–6411.
[17] JIA Z H, WANG Y, WU C, et al. Multi-objective energy-aware batch scheduling using ant colony optimization algorithm. Computers & Industrial Engineering, 2019, 131: 41–56.
[18] REITER S. A system for managing job-shop production. Journal of Business, 1966, 39(3): 371–393.
[19] PENG J G, LIU M Z, ZHANG X, et al. Hybrid heuristic algorithm for multi-objective scheduling problem. Journal of Systems Engineering and Electronics, 2019, 30(2): 327–342.
[20] DONG Z M, WANG X P, TANG L X. MOEA/D with a self-adaptive weight vector adjustment strategy based on chain segmentation. Information Sciences, 2020, 521: 209–230.
[21] JIANG E D, WANG L. Multi-objective optimization based on decomposition for flexible job shop scheduling under time-of-use electricity prices. Knowledge-Based Systems, 2020, 204: 106–177.
[22] WANG L P, FENG M L, QIU F Y, et al. Decomposition multi-objective evolutionary algorithm for recursive replacement optimization strategy. Journal of Chinese Computer Systems, 2018, 39(6): 1135–1141. (in Chinese)
[23] QI Y T, MA X L, LIU F, et al. MOEA/D with adaptive weight adjustment. Evolutionary Computation, 2014, 22(2): 231–264.
[24] BRUCKER P, SHAKHLEVICH N V. Inverse scheduling: two-machine flow-shop problem. Journal of Scheduling, 2011, 14(3): 239–256.
[25] WU X L, SUN Y J. A green scheduling algorithm for flexible job shop with energy-saving measures. Journal of Cleaner Production, 2018, 172: 3249–3264.
[26] ZHAO Y W, WANG H Y, XU X X. A new hybrid parallel algorithm for consistent-sized batch splitting job shop scheduling on alternative machines with forbidden intervals. The International Journal of Advanced Manufacturing Technology, 2010, 48: 1091–1105.
[27] DEFERSHA F M, CHEN M Y. Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time. International Journal of Production Research, 2012, 50(8): 2331–2352.
[28] WU X L, YUAN Q, WANG L. Multi-objective differential evolution algorithm for solving robotic cell scheduling problem with batch-processing machines. IEEE Trans. on Automation Science and Engineering, 2000. DOI: 10.1109/TASE.2020.2969469.
[29] JAMALI A, MALLIPEDDI R, SALEHPOUR M, et al. Multi-objective differential evolution algorithm with fuzzy inference-based adaptive mutation factor for Pareto optimum design of suspension system. Swarm and Evolutionary Computation, 2020, 54: 100666.
[30] YUE F, SONG S J, JIA P, et al. Robust single machine scheduling problem with uncertain job due dates for industrial mass production. Journal of Systems Engineering and Electronics, 2020, 31(2): 350–358.

Biographies

WU Xiuli was born in 1977. She received her B.S., M.S. and Ph.D. degrees in mechanical and electronic engineering from Northwestern Polytechnical University, Xi'an, Shaanxi Province, China, in 2000, 2003 and 2006, respectively. Since 2006, she has been with the Department of Logistics, School of Mechanical Engineering, University of Science and Technology Beijing, where she became a professor in 2020. She has authored two books and over 50 refereed papers. Her current research interests include intelligent optimization, production scheduling, and logistic optimization.
E-mail: wuxiuli@ustb.edu.cn

PENG Junjian was born in 1992. He received his B.S. degree in industrial engineering from Henan Polytechnical University, Jiaozuo, Henan Province, China, in 2018. He is a master of University of Science and Technology Beijing. His research interests include intelligent optimization, and production scheduling.
E-mail: 18810458661@163.com

XIE Zirun was born in 1996. She received her B.S. degree in logistics engineering from University of Science and Technology Beijing, Beijing, China, in 2019. She is a master of University of Science and Technology Beijing. Her research interests include intelligent optimization and production scheduling.
E-mail: xiezirain@163.com

ZHAO Ning was born in 1978. He received his Ph.D. degree in mechanical manufacturing and automation from Beijing Institute of Technology, Beijing, China, in 2005. He is a Ph.D. and a professor in University of Science and Technology Beijing. His research interests include applied intelligent manufacturing system, scheduling and simulation decision.
E-mail: nickzhao@me.ustb.edu.cn

WU Shaomin was born in 1965. He received his Ph.D. degree in applied statistics from Southeast University, Nanjing, China, in 1995. He is a Ph.D. and a professor in Kent Business School, University of Kent, UK. His research interests include applied stochastic processes, business data analysis, statistical data analysis, data mining, and risk management.
E-mail: s.m.wu@kent.ac.uk