APEM journal
Advances in Production Engineering & Management
Volume 12 | Number 1 | March 2017 | pp 5–16
https://doi.org/10.14743/apem2017.1.235
ISSN 1854-6250
Journal home: apem-journal.org
Original scientific paper

Solving dual flexible job-shop scheduling problem using a Bat Algorithm

Xu, H.[a,*], **Bao, Z.R.**[a], **Zhang, T.**[a]

[a]School of Internet of Things Engineering, Jiangnan University, Wuxi, China

ABSTRACT

For the flexible job-shop scheduling problem with machine selection flexibility and process sequence flexibility in process design, types and characteristic of machine selection and process sequence flexibility are analyzed. The mathematical model of dual flexible job-shop scheduling problem is established, and an improved bat algorithm is proposed. For purpose of expressing the relationship effectively between the process and the bat population, a new method of encoding strategy based on dual flexibility degree is proposed. The crossover and mutation operation are designed to strengthen the searching ability of the algorithm. For purpose of overcoming the shortcomings of the fixed parameters in bat algorithm, the value of the inertia weight was adjusted, and a linear decreasing inertia weight strategy was proposed. We carried out experiments on actual examples, it can be seen from the experimental results that the robustness and optimization ability of the algorithm we proposed are better than Genetic Algorithm (GA) and Discrete Particle Swarm Optimization algorithm (DPSO). This shows that the proposed algorithm is more excellent in solving the flexible job-shop scheduling problem, and it is an efficient scheduling algorithm.

ARTICLE INFO

Keywords:
Flexible job-shop scheduling
Optimization
Process sequence flexibility
Machine selection flexibility
Bat algorithm
Genetic algorithm
Particle swarm optimization

**Corresponding author:*
joanxh2003@163.com
(Xu, H.)

Article history:
Received 16 November 2016
Revised 13 February 2017
Accepted 15 February 2017

1. Introduction

Dual Flexible Job-shop Scheduling Problem (DFJSP) based on machine selection flexibility and process sequence flexibility is an extension of the classical Job-shop Scheduling Problem (JSP) and the Flexible Job-shop Scheduling Problem (FJSP). It overcomes the problems of the fixed step sequence and uniqueness of machine selection, and increases the flexibility of scheduling problem, making it closer to an actual production environment. DFJSP needs to consider not only the processing sequence of the work, but also the problems of assigning operations to machines, which are a more complicated NP-hard problem [1]. Therefore, research on DFJSP has theoretical significance and application value.

At present, researches on FJSP are mainly focused on two aspects. The first involves assigning operations to machines. Yuan et al. [2] proposed a hybrid harmony search algorithm for solving FJSP, they converted the continuous vector to the discrete vector and introduced heuristic and random strategies for a resentful initialization scheme. Zhao et al. [3-4] proposed a hybrid genetic algorithm to solve FJSP. Luan et al. [5] proposed a new genetic algorithm to solve FJSP, in which they optimize the performance by minimizing the data sizes of the constrained model and reduced the time and space complexity of GA. The second involves processing sequence flexibility. A model is defined by Saygin[6] , which combines flexible process design and production

scheduling. Huang et al. [7] proposed an improved genetic algorithm for solving JSP based on process sequence flexibility. Ba et al. [8] proposed a new model of multi-resource flexible job shop scheduling problem, and designed a new genetic algorithm for solving this problem. Moreover, Modrák et al. [9] proposed an algorithm that can convert a multi-machines problem to a two-machines problem, and experiments show that the algorithm is effective in solving n-jobs and m-machine problems.

Bat Algorithm (BA) is a materialistic optimization algorithm developed by Yang [10] in 2010. Since its proposed the algorithm has caught the attention of scholars in different fields. It has been used in the study of power system stability by Ali [11]. Shi et al. [12] has applied BA to WSN position. Chen et al. [13-14] proposed a mind evolutionary bat algorithm which is applied to feature selection of mixed gases infrared spectrum. Besides its relevance to practical application BA is applied to the function optimization problem [15], pattern recognition [16] and optimization of engineering problems [17]. In area of workshop scheduling, there are a lot of articles concerning the influencing factors since the BA was presented. Marichelvam el al. [18-19] has applied the BA to the hybrid flow shop scheduling problem (FSP) in 2013, and it shows BA was more effective than genetic algorithm and particle swarm optimization in solving this problem through the simulation results. Luo et al. [20] proposed a discrete BA solution for permutation flow shop scheduling problem (PFSP) in 2014 and it also shows that BA was effective in solving PFSP through the simulation results. Zhang et al. [21] proposed an improved BA for solving PFSP, the simulation results show that the improved BA is feasible and effective. La et al. [22] proposed a hybrid BA to solve the two stages hybrid FSP and the result shows that the hybrid BA is better than the classical BA.

Be seen from the above, the studies of FJSP are primarily focused on the single FJSP. DFJSP is less researched. Although BA is applied to various fields, few investigators utilize it to solve DFJSP. Therefore, this paper establishes a model for DFJSP and proposes an improved bat algorithm for solving it.

2. Definition and formalization problem

Assumptions of notations for DFJSP are as follows:

- Let $J = J_{\mathrm{i}}$, $1 \leq i \leq n$, indexed i, be a set of n jobs.
- Let $M = M_{\mathrm{j}}$, $1 \leq j \leq m$, indexed j, be a set of m machines.
- Each job J_i consists of several operations. Certain jobs consist of unfixed process sequence. The operations of each job are not fixed numbers. Each operation can be processed for a given set of machines.

Some symbols used throughout the paper are as follows:

n	Total number of jobs
m	Total number of machines
λ	Total number of operations
λ_i	Number of operations of job J_i
O_{ij}	j-th operation of job J_i
S_{ij}	Total number of alternative machines of operation O_{ij}
W_{ijk}	j-th operation of job J_i on machine M_k
M_{ij}	Set of alternative machines of operation O_{ij}
T_{ijk}	Processing time of O_{ij} on machine M_k

In the actual production process there are many kinds of flexibility, such as machine selection flexibility and process sequence flexibility and so on. Descriptions of two of these flexibilities are described as below in details.

Process sequence flexibility

Process sequence flexibility is a kind of flexibility where there is not a fixed process sequence among operations for a job. Assume job J_i need λ_i operations to be finished. If order of operations from k_1-th to k_2-th $(k_1 < k_2)$ is unfixed, then it is regarded as flexible operations, and$< O_{ik_1}, O_{ik_2} >$ is denoted as the flexible process sequence span of job J_i. Flexible operation is classified into two types as follows: (1) Partial flexible operation: An operation which belongs to a flexible process sequence span can be inserted into any arbitrary position of the span. (2) Total flexible operation: An operation can be inserted into any arbitrary position of the process sequence.

Machine selection flexibility

The machine selection flexible scheduling problems can be classified into two main groups according to the relation between set M_{ij} and set M. (1) Total machine selection flexible scheduling problem: Each operation can be processed on any machine among the set M_{ij}, where $M_{ij} = M$. (2) Partial machine selection flexible scheduling problem: Each operation can be processed on any machine among the set M_{ij}, where $M_{ij} \subset M$. The partial machine selection flexible scheduling problem suits better than the total machine selection flexible scheduling problem in the practical manufacturing environments. But compared to the total machine selection flexible scheduling problem, the partial machine selection flexible scheduling problem has the disadvantages of great search space, great computation and difficulties solution.

Table 1 An example of processing time table of partial machine selection flexible scheduling problem

Job	Operation	M_1	M_2	M_3	M_4
J_1	O_{11}	2	7	—	6
	O_{12}	6	—	4	5
	O_{13}	—	3	2	4
J_2	O_{21}	3	1	6	3
	O_{22}	1	3	—	3

Table 2 An example of processing time table of total machine selection flexible scheduling problem

Job	Operation	M_1	M_2	M_3	M_4
J_1	O_{11}	2	7	2	6
	O_{12}	6	3	4	5
	O_{13}	4	3	2	4
J_2	O_{21}	3	1	6	3
	O_{22}	1	3	4	3

Examples of partial machine selection flexible scheduling problem and a total machine selection flexible scheduling problem are shown in Table 1 and Table 2 respectively. Processing time of each operation on the corresponding machine is indicated in the table. In Table 1, the tag "—" means that a machine cannot execute the corresponding operation.

Assumptions used throughout the DFJSP are as follows:

- In flexible process sequence span, each operation has the same priority.
- Each operation, once started, cannot be interrupted.
- Each machine can process only one job at the same time.
- Arrival time of a job is included in the processing time.
- When machines are available for re-scheduling when the corresponding operations are completed, machines can choose to stop or no-load running.

The task of DFJSP is to seek an appropriate schedule which cost minimum time to complete all operations. The makespan T_{max} can be calculated by the formula:

$$T_{max} = \max\{T_{max}^{j} \big| j = 1,2, \dots, m\} \tag{1}$$

where T_{max}^{j} is time of completion of last job on machine j.

3. The improved bat algorithm

To effectively avoid the bat algorithm from prematurity and to improve the global search capability and search precision, in this paper the following improvements of BA are made: (1) In order to express the relationship between the process flexibility, process sequence and bat population, a new coding strategy based on dual flexibility is proposed. (2) To enhance the ability of neighborhood search, the corresponding operations like crossover and mutation are designed. (3) A linear decreasing inertia weight strategy is adopted to effectively control the local and global search capability of BA during the optimization procedure.

3.1 Coding strategies

Coding is to abstract and specify the research problem, and then to develop mathematical model. Finally, it realizes the mapping between the solution space of feasible solution and the exploration space of BA. This is the key step in BA solving, and also the primary problem to be solved.

Classical JSP only needs to code based on the process sequence. But DFJSP need not only to code based on process sequence, but also need to select the corresponding machine for each operation. Therefore, the coding strategy needs to be done in the following three points: (1) The coding strategy should reflect machine selection flexibility and process sequence flexibility. (2) The coding strategy should show the sequence of operations for each job. (3) It has to show the corresponding processing machine that each job need for each operation.

Thus, we design three coding strategy based on above requirements:

A coding strategy base on the sequence

In the bat algorithm, all processes are required to be added to the code, present with vector $X_1 = (x_{11}, x_{12}, \dots, x_{1\lambda})$, the length is λ, among those vector elements, x_{11} indicates the first process of the first job, x_{12} indicates the second process of the first job. By that analogy, the total number of t is the operation of the i-th job characters of j-th job. In vector X_1, the order of the i-th corresponds to the order of processing sequence of each process. The advantages of this coding strategy are the high flexibility and always generate the feasible schedule after an exchange of the processing sequence.

A coding strategy base on the process sequence flexibility

In the DFJSP, after the coding based on sequence, it is needed to mark and determine the priority of each job with flexible process sequence span, here we use vector $X_1 = (x_{21}, x_{22}, \dots, x_{2\lambda})$ to present, the length is λ. Among them, the default initialization for x_{2i} is 0 or 1, 0 indicates that the process sequence is not flexible, to 1, indicates the opposite.

A coding strategy base on the machine selection flexibility

In the bat algorithm, we use vector $X_3 = (x_{31}, x_{32}, \dots, x_{3\lambda})$ to present machine selection. The length is λ. Among them, each variable is a positive integer which represents the position in the selected machine set. The advantage of this coding strategy is to ensure that generate the feasible schedule after the operation, and it can be applied to either totally FJSP or partially FJSP.

3.2 Bat algorithm and its improvement

Assume search space is a d-dimensional space, $v_i(t-1)$ and $x_i(t-1)$ denote the velocity and location of bat i at the time step t-1. The location $x_i(t)$ and velocity $v_i(t)$ at the time step t are as follows:

$$F_i = F_{min} + (F_{max} - F_{min})\beta \tag{2}$$

$$v_i(t) = v_i(t-1) + (x_i(t-1) - x^*)F_i \tag{3}$$

$$x_i(t) = x_i(t-1) + v_i(t) \tag{4}$$

$$x_{new} = x_{old} + \varepsilon A^{t-1} \tag{5}$$

$$A^t(i) = \alpha A^{t-1}(i) \tag{6}$$

$$R^t(i) = R^0(i)[1 - \exp(-\gamma(t-1))] \tag{7}$$

where F_i is the update frequency of bat i, F_{min} and F_{max} represent the minimum and maximum value of the update frequency. $\beta \in [0,1]$ and $\varepsilon \in [-1,1]$ are random numbers. Here x^* is the cur-

rent global optimal solution, x_{old} is an optimal solution from a random selection of the optimal solution set, $A^{t-1}(i)$ is the average loudness at the time step $t-1$. For any $0 < \alpha < 1$ ($\alpha \in R$) or $\gamma > 0$ ($\gamma \in R$), we have,

$$A^t(i) \to 0, R^t(i) \to R^0(i), \text{as } t \to \infty \tag{8}$$

The standard bat algorithm flow is presented in Fig. 1.

Begin
Step 1. Initialize parameters such as bat population x_i, velocity v_i, loudness $A(i)$, pulse rates $R(i)$ and so on
Step 2. Define pulse frequency F_i
Step 3. Calculated fitness value fit_{min} and the current global optimal solution x^*
Step 4. While (t < Max number iterations)
Step 5. For ($i = 1$ to pop size)
Step 6. Generate new solutions by adjusting frequency, and updating velocities and locations via Eq. 2 to Eq. 4
Step 7. If (rand>$R(i)$)
Step 8. Select a solution among the optimal solution set
Step 9. Generate a new solution via Eq. 5
Step 10. end if
Step 11. Calculated fitness value fit_{new}
Step 12. If (fit_{new}<$fit(i)$&rand< $A(i)$)
Step 13. Accept the new solutions
Step 14. Update $A(i)$ and $R(i)$
Step 15. end if
Step 16. end for
Step 17. Find the current optimal fitness value fit_{new} and the current optimal x^*
Step 18. end while
End

Fig. 1 The standard bat algorithm flow

BA was proposed for solving the constrained optimization problem of a continuous domain. Then, a binary bat algorithm (BBA) was proposed by Mirjalili et al. [23] in 2014 for solving the optimization problem of a discrete domain. Although DFJSP is a discrete combinatorial optimization problem, but BBA cannot be applied to DFJSP directly. Therefore, according to the characteristics of DFJSP and optimization mechanism of BBA, we designed the related operators for solving DFJSP. To use bat algorithm to solve DFJSP, doing the following:

In order to explain the jobs, process sequence, machines, processing time, processing order and the status of processing and other information, the following definitions are given based on the rule of encoding and decoding. Example is taken from table 1 for understanding of the following definitions.

Definition 1: The correlation matrix of job and operation, JO

The correlation matrix of job and operation (JO) represents the relationship between jobs and operations. According to the data in Table 1, a matrix can be obtained as follows:

$$JO = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 3 & 3 \\ 2 & 4 & 1 \\ 2 & 5 & 2 \end{bmatrix}$$

JO is a λ-by-3matrix. The first column indicates job number; the second column indicates operation number of all operations; the third column indicates the order of the operation for the corresponding job. For example, the first line (1,1,1) indicates that the operation 1 is the first operation of the job 1; the second line (1,2,2) indicates that the operation 2 is the second operation of the job 1. By that analogy, the last line (2,5,2) indicates that the operation 5 is the second operation of the job 2.

Definition 2: The matrix of processing of job, *PJ*
The matrix of processing of job (*PJ*) represents the working state of each job under process. According to the data in Table 1, a matrix can be obtained as below:

$$PJ = \begin{bmatrix} 1 & 3 & 1 & 0 \\ 2 & 2 & 2 & 1 \end{bmatrix}$$

PJ is a n-by-4matrix. The first column indicates the job number; the second column indicates the number of operations corresponding to the jobs; the third column indicates the processing progress, the default initialization is 0; the fourth column indicates processing state of the job with four status 0, 1, 2, 3, status 0 indicate job is before process, status 1 indicate job is under process, status 2 indicate job is waiting for process, and status 3 indicate job is after process. For example, the first line (1,3,1,0) indicates that machine 1 has 3 processes and first operation of machine 1 is before process; the second line (2,2,2,1) indicates that machine 2 has 2 process and second operation of machine 1 is under process.

Definition 3: The matrix of temporary resource pool, *TRP*
The matrix of temporary resource pool (*TRP*) represents the process state of each job under process. According to the data from Table 1, a matrix can be obtained as follows:

$$TRP = \begin{bmatrix} 1 & 1 & 2 & 1 & 1 & 2 \\ 3 & 1 & 1 & 2 & 1 & 6 \end{bmatrix}$$

TRP is a n-by-6matrix. The first column indicates machine number; the second column indicates operation number for corresponding job; the third column indicates the operation number for corresponding job; the fourth column indicates job number; the fifth column indicates the order of the operation for corresponding job; the sixth column indicates the processing time on the machine for corresponding job. For example, the first line (1,1,2,1,1,2) indicates that two jobs need to be processed on the machine 1 and the processing time is 2 for the first operation of the job 1 on machine 1; the second line (3,1,1,2,1,6) indicates that one job need to be processed on the machine 3 and the processing time is 6 for the first operation of the job 2 on machine 3.

Definition 4: The matrix of resource state, *RS*
The matrix of resource state (RS) represents the status of each machine. According to the data from Table 1, a matrix can be obtained as below:

$$RS = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 2 & 1 & 2 & 1 \end{bmatrix}$$

RS is a m-by-4matrix. The first column indicates the machine number; the second column indicates the availability of the machine which has two processing status represented by 0 and 1 respectively, status 0 indicates that the machine is available to be used, status 1 indicates the machine is not usable because of occupation or out of order; the third column indicates job number; the fourth column indicates the operation number for corresponding job. For example, the first line (1,0,1,1) indicates that the machine 1 is available and the first operation of job 1 can be processed on machine 1; the second line (2,1,2,1) indicates that the machine 2 is not usable and the first operation of job 2 cannot be processed on machine 1.

To make the current optimal individual of bat population get a better solution, four neighborhood search operators such as inserting, reversing, crossover and mutation are adopted. Detailed definitions of operators are provided below.

Definition 5: Neighborhood search operator
Exchanging operator: We use exchanging operator (i, j) to exchange the two elements i, j of the sequence. Take data from Table 1 as an example, assuming that the coding sequence is (4,1,2,3,2) based on operation sequencing, after the exchanging operator (2,4), the coding sequence becomes (4,3,2,1,2).

Inserting operator: The goal of inserting operator (i, j) is to insert element i next to element j. Take data of Table 1 as an example, assuming that the coding sequence is (4,1,2,3,2) based on operation sequencing, after the inserting operator (2,4), the code sequence becomes (4,2,3,1,2).

Reversing operator: Reversing operator (*i*, *j*) reverse the subsequence between element *i* and element *j*. Take data from Table 1 as an example, assuming that the coding sequence (4,1,2,3,2) is based on operation sequencing, after the reversing operator (2,5), the code sequence becomes (4,2,3,1,2).

Crossover operator: Assume a current sequence X, crossover operator choose randomly one sequence Y of bat population, and generate a new sequence Z by replacing the elements *i*, *j* of current sequence X with the elements *i*, *j* of sequence Y. Take data of Table 1 as an example, assuming that the current sequence (4,1,2,3,2) and randomly selected sequence (1,4,2,1,4) are both based on process sequence, after the crossover operator (2,4), the code sequence becomes (4,4,2,1,2).

Mutation operator: After each iteration, if the current optimal solution is no better than the population optimal solution, then the current optimal solution should be mutated. The number of variations cannot be more than the 1/4 of the total number of population. The mutation operator <i,j> means to mutate the selection of O_{ij} by value from M_{ij}.

Updating process of position and velocity in bat algorithm is similar to the updating process of PSO algorithm [24]. The inertia weight*w* is introduced to balance the local searching and the global searching via equation (9) [25]. In order to enhance the algorithm's ability of global searching at prophase of evolution, the weight exponential decline strategy is applied as shown in equation (10).

$$v_i(t) = wv_i(t-1) + (x_i(t-1) - x^*)F_i \tag{9}$$

$$w = \frac{T-t}{T}(w_{max} - w_{min}) + w_{min} \tag{10}$$

Where, t is the current number of iterations and T is maximum number of iterations; w_{max} and w_{min} are the maximum and minimum values of the inertia weight.

To summarize, the basic steps of the improved bat algorithm for solving DFJSP can be described as follows in Fig 2.

Step 1.	Initialize parameters, include the size of bat population $PopSize$, loudness $A(i)$, pulse rates $R(i)$, maximum inertia weight w_{max}, minimum inertia weight w_{min} and so on
Step 2.	The coding sequence is generated based on these three coding strategies
Step 3.	Calculate fitness value
Step 3.1	Load the information of all jobs into *PJ* and *TRP*
Step 3.2	Every job is processed in the beginning from the first operation by *PJ.* Firstly, judge by *RS* whether the machine which the first operation needing is processed is used by another operation. If not, then the job is into the processing status. Meanwhile, record the corresponding numbers of the job and the operation. Otherwise, the job is flagged as waiting for processing in *PJ*.
Step 3.3	The first job starts to be processed in *PJ* and *TRP*, which judge by *TRP* is being processed. If it is and the rest time of processing is greater than zero, the processing of the job is going on. If not, judge whether all operations of the job are finished. If the job is accomplished, make the machines free by *RS*. Otherwise, add the information of the next operation into *PJ* and *TRP.*
Step 3.4	If all jobs are already processed by *PJ*, go to *Step 3.5*; otherwise, go to *Step 3.2*
Step 3.5	Output fitness value
Step 4.	If the termination criterion is reached, go to S*tep 10*; otherwise, go to *Step 5*
Step 5.	Generate new solutions by adjusting frequency, and updating velocities and locations via Eqs. 2, 9 and 10
Step 6.	If rand>$R(i)$, select a solution from optimal solution set and generate a local solution x_{new} around the selected optimal solution via Eq. 5
Step 7.	Calculate fitness value fit_{new} of corresponding x_{new}
Step 8.	If $fit_{new} < fit(i)$& $rand < A(i)$, accept the new solutions, update $A(i)$ and $r(i)$ via Eq. 6 to Eq. 7; otherwise, do neighborhood search operator
Step 9.	Go to step 4
Step 10.	Post process results and visualization

Fig. 2 The improved bat algorithm flow

4. Case simulation and analysis

4.1 Experimental parameter settings

The improved bat algorithm is coded in Matlab platform and run on an Intel core i7-5500U, 3.0 GHz and 12.0 GB RAM PC. The parameters of three algorithms are as follows:

Table 3 Parameter settings of three algorithms

Parameter	Value		
	Improved bat algorithm	GA	DPSO
$PopSize$	100	100	100
T	500	500	500
F_{max}	1	—	—
F_{max}	0	—	—
α	0.9	—	—
γ	0.9	—	—
A	0.25	—	—
w_{max}	0.96	—	—
w_{min}	0.36	—	—
Pc	—	0.7	—
Pm	—	0.08	—
w	—	—	0.9
c_1	—	—	2
c_2	—	—	2

The *PopSize* represent the size of the bat population and T is the maximum number of iteration.

4.2 Experimental dates

To verify the efficiency and feasibility of the improved bat algorithm for solving DFJSP, this paper used the actual data of manufacturing enterprise which is the problem 6×8. Problem 6×8 is a DFJSP instance that consists of 6 jobs of 27 operations that can be implemented on 8 machines. The data of the machine selection flexibility, process sequence flexibility and processing time are as follows:

Table 4 The data of machine selection flexibility

Job	Job number	Flexible process span
J_1	6	<2,3>, <5,6>
J_2	3	—
J_3	5	<3,5>
J_4	3	<2,3>
J_5	6	<3,4>
J_6	4	—

The tag "—" means that the job does not have a flexible process span.

Table 5 Data of process sequence flexibility and processing time (min)

Job	Operation	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
J_1	O_{11}	—	12	—	9	14	—	20	—
	O_{12}	18	—	—	19	—	15	11	—
	O_{13}	—	12	14	9	—	17	—	—
	O_{14}	—	11	—	—	9	—	—	12
	O_{15}	15	—	—	8	—	—	—	18
	O_{16}	12	—	9	—	—	11	—	—
J_2	O_{21}	—	—	12	19	14	—	—	—
	O_{22}	8	—	—	9	11	—	15	—
	O_{23}	16	7	—	—	—	9	—	—
J_3	O_{31}	—	—	—	11	10	—	—	13
	O_{32}	—	—	12	18	—	—	14	—
	O_{33}	9	—	—	15	7	—	12	—
	O_{34}	—	12	—	15	—	7	—	9
	O_{35}	3	—	—	4	—	8	—	—
J_4	O_{41}	—	19	—	—	7	—	13	—
	O_{42}	—	—	8	—	11	—	—	16
	O_{43}	9	11	—	8	—	—	18	—
J_5	O_{51}	6	—	—	12	—	—	14	9
	O_{52}	—	—	—	22	—	12	17	—
	O_{53}	—	18	—	—	11	—	—	9
	O_{54}	9	—	12	—	—	—	—	7
	O_{55}	—	11	—	—	—	9	14	—
	O_{56}	8	—	—	12	6	—	—	9
J_6	O_{61}	—	—	11	—	17	—	—	18
	O_{62}	5	—	12	—	—	—	7	9
	O_{63}	—	11	—	—	—	8	13	7
	O_{64}	19	—	13	7	—	15	—	—

4.3 Experimental results and comparisons

This paper used the improved bat algorithm, discrete particle swarm optimization [26] (DPSO) and genetic algorithm [27] (GA) to solve the DFJSP. And the experimental results are compared and analyzed.

To obtain meaningful results, we run each algorithm fifty times on the above instance. The experimental results of these three kinds of algorithms are shown in Table 6 and Table 7.

The optimum solution, average solution and standard deviation of three algorithms in 50 experiments are shown in Table 6. The optimum solution obtained by this paper is 55, which is obviously better than 65 obtained by GA and 62 obtained by DPSO. Furthermore, the average value and standard deviation are also better than both GA and DPSO. In a conclusion, the algorithm in this paper is obviously better than GA and DPSO in searching performance.

Table 6 Makespan comparison between three algorithms

Algorithm	Optimum solution	Average solution	Standard deviation
GA	65	70.72	4.25
DPSO	62	65.56	3.15
Improved bat algorithm	55	57.32	2.24

Table 7 Distribution of 50 calculation results

Interval / Algorithm	Interval				
	[55,60)	[60,65)	[65,70)	[70,75)	[75,80)
GA	0	0	23	14	13
DPSO	0	23	19	8	0
Improved bat algorithm	40	10	0	0	0

The distribution of experiments results in each interval for running 50 times is presented in Table 7. It is clear from the table, the results of the improved bat algorithm are comparatively concentrated, which are mainly in the range of [55, 60], whereas the results of GA and DPSO algorithm are relatively distributed. Thus, the improved bat algorithm in terms of solving DFJSP

is more robust than GA and DPSO algorithm. Figs. 3, 4, and 5 shows the Gantt charts of optimal scheduling concerning three kinds of algorithms.

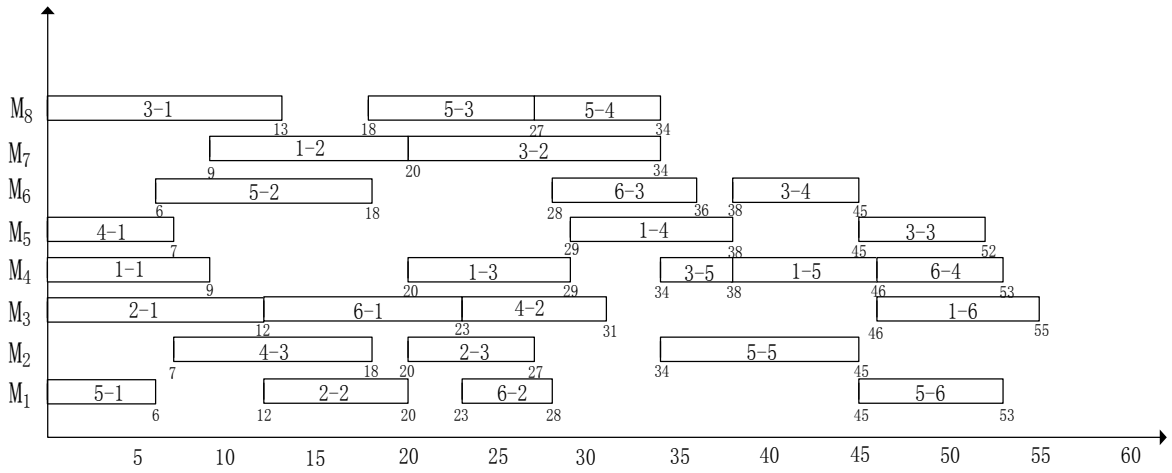

Fig. 3 The Gantt chart of optimal scheduling of the improved bat algorithm

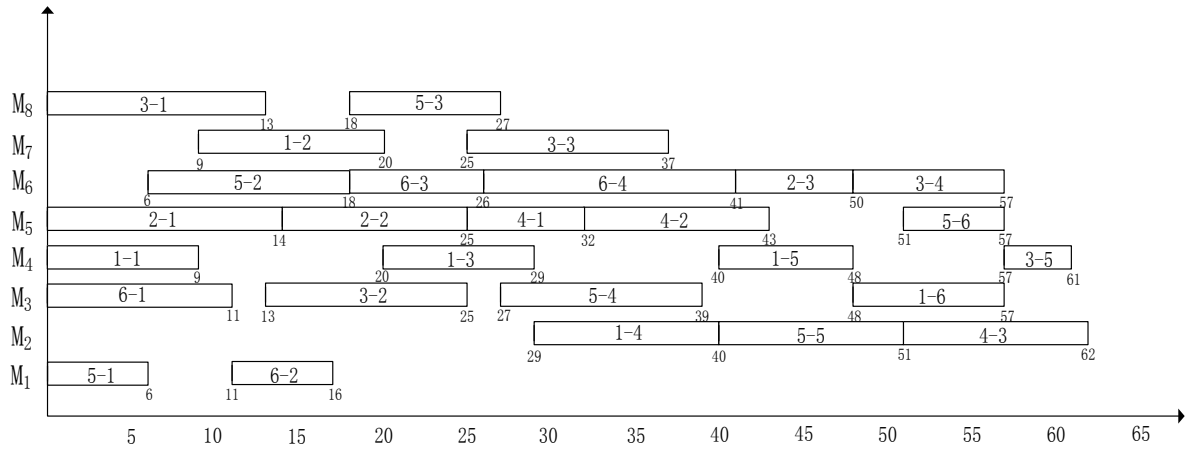

Fig. 4 The Gantt chart of optimal scheduling of DPSO algorithm

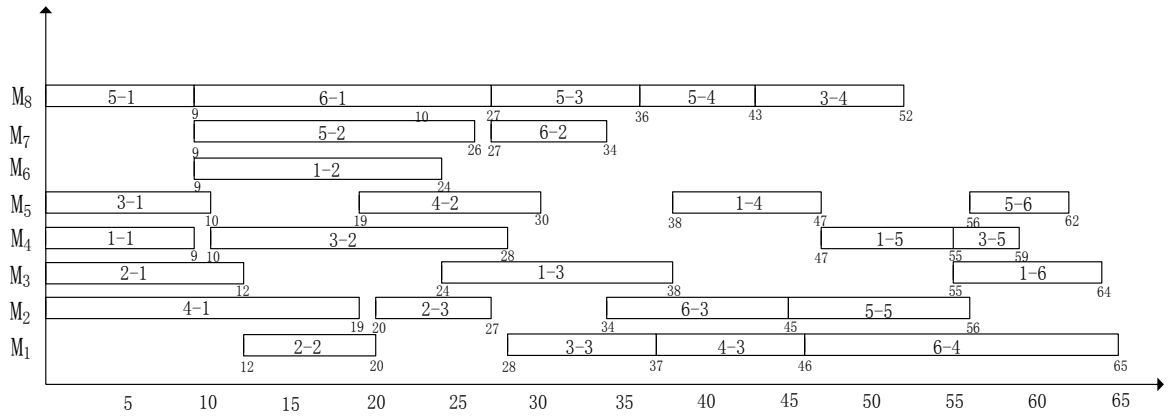

Fig. 5 The Gantt chart of optimal scheduling of GA

5. Conclusion

According to dual flexible job shop scheduling problems, we established a mathematical model based on process sequence flexibility and machine selection flexibility, and an improved bat algorithm is presented to solve it. The premise of BA used in DFJSP is to realize the mapping between operations and bat populations. Therefore, we proposed a dual flexible encoding strategy. In order to intensify neighborhood searching ability of the algorithm, several operations are designed such as crossover and mutation. Furthermore, a linear decreasing inertia weight strategy is put forward to effectively avoid the algorithm of premature convergence and to intensify the global searching ability and the precision of the algorithm. Aiming at solving actual job-shop scheduling problems, we set up the scheduling model and designed the detailed algorithm. Experimental results indicate that the improved bat algorithm for solving DFJSP is feasible and effective, which provides a new way to solve such problems. It is a further research direction to use the improved algorithm to solve multi-objective DFJSP and construct new dispatching rules.

Acknowledgement

The research has been supported by Natural Science Foundation of Jiangsu Province (Grant No. BK20140165) and China Scholarship Council Sponsored (Grant No. 201308320030).

References

[1] Röck, H. (1984). The three-machine no-wait flow shop problem is NP-complete, *Journal of the ACM*, Vol. 31, No. 2, 336-345, doi: 10.1145/62.65.
[2] Yuan, Y., Xu, H., Yang, J. (2013). A hybrid harmony search algorithm for the flexible job shop scheduling problem, *Applied Soft Computing*, Vol. 13, No. 7, 3259-3272, doi: 10/1016/j.asoc.2013.02.013.
[3] Zhao, S.-K., Fang, S.-L., Gu, X.-J. (2013). Genetic algorithm with new initialization mechanism for flexible job shop scheduling, *Journal of Zhejiang University (Engineering Science)*, Vol. 47, No. 6, 1022-1030, doi: 10.3785/j.issn.1008-973X.2013.06.013.
[4] Zhao, S.-K., Fang, S.-L., Gu, X.-J. (2014). Machine selection and FJSP solution based on limit scheduling completion time minimization, *Computer Integrated Manufacturing Systems*, Vol. 20, No. 4, 854-865, doi: 10.13196/j.cims.2014.04.zhaoshikui.0854.12.20140416.
[5] Luan, F., Wang, W., Fu, W.P., Bao, Y.T., Ren, G.C., Wang, J., Deng, M.M. (2014). FJSP solving by improved GA based on PST hierarchy structure, *Computer Integrated Manufacturing Systems*, Vol. 20, No. 10, 2494-2501.
[6] Saygin, C., Kilic, S.E. (1999). Integrating flexible process plans with scheduling in flexible manufacturing systems, *The International Journal of Advanced Manufacturing Technology,* Vol. 15, No. 4, 268-280, doi: 10.1007/s001700050066.
[7] Huang, X.W., Zhao, X.Y., Ma, X.L. (2014). An improved genetic algorithm for job-shop scheduling problem with process sequence flexibility, *International Journal of Simulation Modelling*, Vol. 13, No. 4, 510-522, doi: 10.2507/ijsimm13(4)co20.
[8] Ba, L., Li, Y., Yang, M.S., Gao, X.Q., Liu, Y. (2016). Modelling and simulation of a multi-resource flexible job-shop scheduling, *International Journal of Simulation Modelling*, Vol. 15, No. 1, 157-169, doi: 10.2507/IJSIMM15(1)CO3.
[9] Modrák, V. Pandian, R.S. (2010). Flow shop scheduling algorithm to minimize completion time for n-jobs m-machines problem, *Tehnički vjesnik –Technical Gazette*, Vol. 17, No. 3, 273-278.
[10] Yang, X.-S. (2010). A new materialistic bat-inspired algorithm, *Nature Inspired Cooperative Strategies for Optimization*, Vol. 284, 65-74, doi: 10.1007/978-3-642-12538-6_6.
[11] Ali, E.S. (2014). Optimization of power system stabilizers using bat search algorithm, *International Journal of Electrical Power & Energy Systems*, Vol. 61, 683-690, doi: 10.1016/j.ijepes.2014.04.007.
[12] Shi, H., Wang, W., Li, Y., Lu, L. (2015). Bat algorithm based on Lévy flight feature and its localization application in WSN, *Chinese Journal of Sensors and Actuators*, Vol. 6, 888-894, doi: 10.3969/j.issn.1004-1699.2015.06.019.
[13] Chen, Y., Wang, Z., Wang, Z. (2014). Feature selection of infrared spectrum based on improved bat algorithm, *Infrared and Laser Engineering*, Vol. 43, No. 8, 2715-2721, doi: 10.3969/j.issn.1007-2276.2014.08.054.
[14] Chen, Y., Wang, Z., Wang, Z. (2015). Mind evolutionary bat algorithm and its application to feature selection of mixed gases infrared spectrum, *Infrared and Laser Engineering*, Vol. 3, 845-851, doi: 10.3969/j.issn.1007-2276.2015.03.010.
[15] Yang, X.-S. (2011). Bat algorithm for multi-objective optimization, *International Journal of Bio-Inspired Computation*, Vol. 3, No. 5, 267-274, doi: 10.1504/IJBIC.2011.042259.
[16] Komarasamy, G., Wahi, A. (2012). An optimized k-means clustering technique using bat algorithm, *European Journal of Scientific Research*, Vol. 84, No. 2, 263-273.
[17] Yang, X.-S., Gandomi, A.H. (2012). Bat algorithm: A novel approach for global engineering optimization, *Engineering Compotations*, Vol. 29, No. 5, 464-483, doi: 10.1108/02644401211235834.

[18] Marichelvam, M.K., Prabaharan, T. (2012). A bat algorithm for realistic hybrid flowshop scheduling problems to minimize makespan and mean flow time, *ICTACT Journal on Soft Computing*, Vol. 3, No. 1, 428-433, doi: 10.21917 /ijsc.2012.0066.
[19] Marichelvam, M.K., Prabaharan, T., Yang, X.-S., Geetha, M. (2013). Solving hybrid flow shop scheduling problems using bat algorithm, *International Journal of Logistics Economics and Globalization*, Vol. 5, No. 1, 15-29, doi: 10.1504 /IJLEG.2013.054428.
[20] Luo, Q., Zhou, Y., Xie, J., Ma, M., Li, L. (2014). Discrete bat algorithm for optimal problem of permutation flow shop scheduling, *The Scientific World Journal*, Vol. 2014, 1-15, doi: 10.1155/2014/630280.
[21] Zhang, J.J., Li, Y.G. (2014). An improved bat algorithm and its application in permutation flow shop scheduling problem, *Advanced Materials Research*, Vol. 1049-1050, 1359-1362, doi: 10.4028/www.scientific.net/amr.1049-1050.1359.
[22] Dekhici, L., Belkadi, K. (2015). A bat algorithm with generalized walk for the two-stage hybrid flow shop problem, *International Journal of Decision Support System Technology,* Vol. 7, No. 3, 1-16, doi: 10.4018/ijdsst.2015070101.
[23] Mirjalili, S., Mirjalili, S.M., Yang, X.-S. (2014). Binary bat algorithm, *Neural Computing and Applications*, Vol. 25, No. 3, 663-681, doi: 10.1007/s00521-013-1525-5.
[24] Bai, J., Gong, Y.-G., Wang, N.-S., Tang, D.-B. (2010). Multi-objective flexible job shop scheduling with lot-splitting, *Computer Integrated Manufacturing Systems*, Vol. 16, No. 2, 396-403.
[25] Zhao, S. (2015). Bilevel neighborhood search hybrid algorithm for the flexible job shop scheduling problem, *Journal of Mechanical Engineering*, Vol. 51, No. 14, 175-184.
[26] Xu, H., Zhang, T. (2015). Improved discrete particle swarm algorithm for solving flexible flow shop scheduling problem, *Journal of Computer Application*, Vol. 35, No. 5, 1342-1347, doi: 10.11772/j.issn.1001-9081.2015.05. 1342.
[27] Cui, J.S., Li, T.K., Zhang, W.X. (2005). Hybrid flow shop scheduling model and its genetic algorithm, *Journal of University of Science and Technology Beijing*, Vol. 27, No. 5, 623-626, doi: 10.3321/j.issn:1001-053X.2005.05.027.