Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 2856056, 17 pages
https://doi.org/10.1155/2022/2856056

Research Article

An Improved Immune Genetic Algorithm for Solving the Flexible Job Shop Scheduling Problem with Batch Processing

Libo Song,[1,2,3,4] **Chang Liu**,[1,2,3] **Haibo Shi**,[1,2,3] **and Jun Zhu**[1,2,3]

[1]*Key Laboratory of Networked Control Systems, Chinese Academy of Sciences, Shenyang 110016, China*
[2]*Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China*
[3]*Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110016, China*
[4]*University of Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Libo Song; songlibo@sia.cn

Received 22 April 2022; Revised 30 May 2022; Accepted 2 June 2022; Published 26 June 2022

Academic Editor: Kuruva Lakshmanna

This paper presents a mathematical model for the flexible job shop scheduling problem (FJSP) with batch processing for manufacturing enterprises with both the flexible job shop scheduling problem and a batch process (BP) problem in actual production. An improved immune genetic algorithm (IGA) based on greedy thought combined with local scheduling rules is used to solve this scheduling problem. In the flexible job shop part, the greedy optimal solution is obtained through the greedy thought. The concept of cross-entropy is then introduced to improve the standard IGA. Calculating the cross-entropy of the individual and greedy optimal solutions for optimization considerably accelerates the optimization speed of the algorithm and enhances the ability of the algorithm to escape the local optimum. In the batching process, effective batching rules are designed to reduce blockage and improve batching efficiency; thus, the job can quickly and effectively pass the batching process and complete the entire production process. In the algorithm verification stage, standard FJSP datasets are used to simulate and verify the proposed algorithm. Considering the specific FJFP with BP problem, we perform simulation experiments with actual production data of a transformer manufacturer. The results show that the proposed method can effectively solve such problems.

1. Introduction

The flexible job shop scheduling problem (FJSP) is an extension of the traditional job shop scheduling problem (JSP) and was first proposed by Brucker and Schlie [1] in 1990. In FJSP, each job operation can be assigned to multiple machines, which may have different processing times; thus, FJSP is a more complicated nondeterministic polynomial-hard problem than JSP. Recently, more methods have been applied to solving the FJSP and its expansion problem. For example, Zhao [2] proposed an improved neighborhood structure hybrid algorithm and achieved good results. Lin et al. [3] proposed a hybrid multiverse optimization to address the fuzzy FJSP. An et al. [4] proposed an improved nondominated sorting biogeography-based optimization to solve the (hybrid) multiobjective FJSP. Li et al. [5] examined the FJSP with transportation resource constraints, which increased the problem complexity and practicability. Cao et al. [6] studied the FJSP with sequencing flexibility. Lei et al. [7] proposed a two-phase metaheuristic for multiobjective FJSP with a total energy consumption threshold. Gao et al. [8] focused on a flexible job shop rescheduling problem (FJRP) for new job insertion and solved the problem using a discrete Jaya algorithm. Gong et al. [9] developed energy- and labor-aware multiobjective flexible job shop scheduling under dynamic electricity pricing. Yang et al. [10] proposed mining dispatching rules from dispatching-related historical data with the characteristics of industrial big data to solve the FJSP. Other extended research issues of the FJSP include distributed FJSP [11], FJSP considering automated guided vehicle planning [12], and FJSP considering machine breakdowns [13–15]. Meanwhile, certain new methods have been applied to solving the FJSP, and examples are distributed particle swarm optimization algorithm [16], two-stage genetic algorithm [17], and state transition algorithm based on the normal cloud model [18–22].

Batch scheduling is an optimization problem with a strong application background. Its basic assumption is that the machine can process multiple jobs simultaneously. Jia et al. [23] proposed a two-objective collaborative optimization algorithm based on the ant colony to solve the parallel batch machine scheduling problem. Chi et al. [24] constructed a class of forward-looking batching algorithms for uncertain environments. Wang et al. [25] studied a new type of rolling batch scheduling problem arising in continuous casting and rolling processes. Li et al. [26] investigated the problem of scheduling jobs with unit processing time and nonidentical sizes on single or parallel identical batch machines. Tan et al. [27] addressed the green batch scheduling problem on nonidentical parallel machines with time-of-use electricity prices. Wang et al. [28] studied a new, mixed batch scheduling problem that arises in vacuum heat treatment. Shahvari and Logendran [29] discussed the batch scheduling problem in a hybrid flow shop with a learning effect.

In the research of the combination of the FJSP and batch process (BP), Huang et al. [30] studied the batch scheduling algorithm of the flexible flow shop for incompatible job families in a flexible flow shop for mold heat treatment comprising quenching and tempering processes. Zhou [31] studied the scheduling problem of two batch processing machines with different jobs. Zhu et al. [32] used a hybrid algorithm that combines the particle swarm algorithm and Nawaz–Enscore–Ham heuristic algorithm to study the batch scheduling problem of a differential job shop. Yin et al. [33] studied the scheduling problem of a large-scale flexible job shop, adopted the batch scheduling method based on job group to degrade the problem scale, and used the adaptive genetic algorithm to optimize the solution. Currently, only a few studies have investigated the scheduling problem comprising flexible job shop and BP scheduling, which are combined in different ways. Therefore, it is crucial to investigate flexible job shop scheduling and batch scheduling thoroughly.

The rest of this paper is organized as follows. Section 2 describes the problem and mathematical model of the FJSP with the BP. Section 3 introduces the method to solve the FJSP–BP problem: Section 3.1 describes a proposed improved immune genetic algorithm (IIGA) to solve the FJSP part of the problem, including designing an effective coding method, genetic operators, cross-entropy thought, and greedy optimal solution; Section 3.2 describes the design of effective batching methods and batching rules for the batching process. Section 4 describes the experimental analysis: using the standard FJSP benchmark example to verify the effectiveness of the algorithm and using the actual data of a transformer manufacturing enterprise that conforms to the FJSP–BP problem to verify the effectiveness and feasibility of the algorithm. Section 5 summarizes the study and highlights the scope of future work.

2. Problem Description

In this study, we developed the FJSP–BP, which is an extension of the FJSP. The FJSP–BP is widely used in manufacturing enterprises, such as transformer manufacturing, semiconductor production, engine parts manufacturing, and steel production line. In this type of problem, most job operations are flexibly distributed on multiple machines. At the same time, specific operations must be unified in batches through batch processing machines, such as drying ovens and dryers, which is a combination of batch scheduling and FJSP.

Figure 1 shows the production process of a transformer, which illustrates the FJSP–BP problem. The entire production process is divided into two parts. The first part is the FJSP, where n jobs are assigned to M machines as an FJSP. The operations consist of several sections, such as the core process, coil process, and lead process setting. The second part is batch processing. Here, the jobs pass through the drying oven in batches, which is equivalent to batch scheduling problems with different arrival times. Figure 2 illustrates a simplification of the production process 2.

The FJSP–BP can be described as follows: n jobs $(J_1, J_2, \cdots, J_n)$ must be processed on $M+1$ machines $(M_1, M_2, \cdots, M_m, M_B)$. When the jobs are processed on $(M_1, M_2, \cdots, M_m, M_B)$, it is an FJSP, which meets the following conditions: each job contains one or more operations; the sequence of operations is predetermined; each operation can be processed on multiple processing machines; the processing time of the operation varies with the processing machine. All the jobs must pass through the batch machine M_B in a batch mode; the batch machine M_B has volume or quality constraints; the batch processing times for different jobs are different; the time for the entire batch to pass through the batch machine is the maximum of all the single-batch processing times in the batch. To optimize a certain performance index of the entire system, scheduling was aimed at selecting the appropriate machine for each operation, determining the best processing sequence and start time of each operation on each machine, and determining the batching method when all jobs pass through the batch machine. Therefore, the FJSP–BP includes three subproblems: determining the processing machine of each job (machine selection sub-problem), determining the sequencing of the operations on each machine (operations sequencing sub-problem), and determining the way each job passes through the batch machine (batching problem).

In addition, the following constraints must be met during processing.

Flexible job shop part:

(1) A machine can only process one job per time

(2) An operation of the job can only be processed by one machine per time

(3) Once each operation of each job starts, the processing cannot be interrupted

(4) Different jobs have the same priority

(5) No sequential constraints exist between the operations of different jobs, and sequential constraints exist between the processes of the same job

(6) All jobs can be processed at zero time

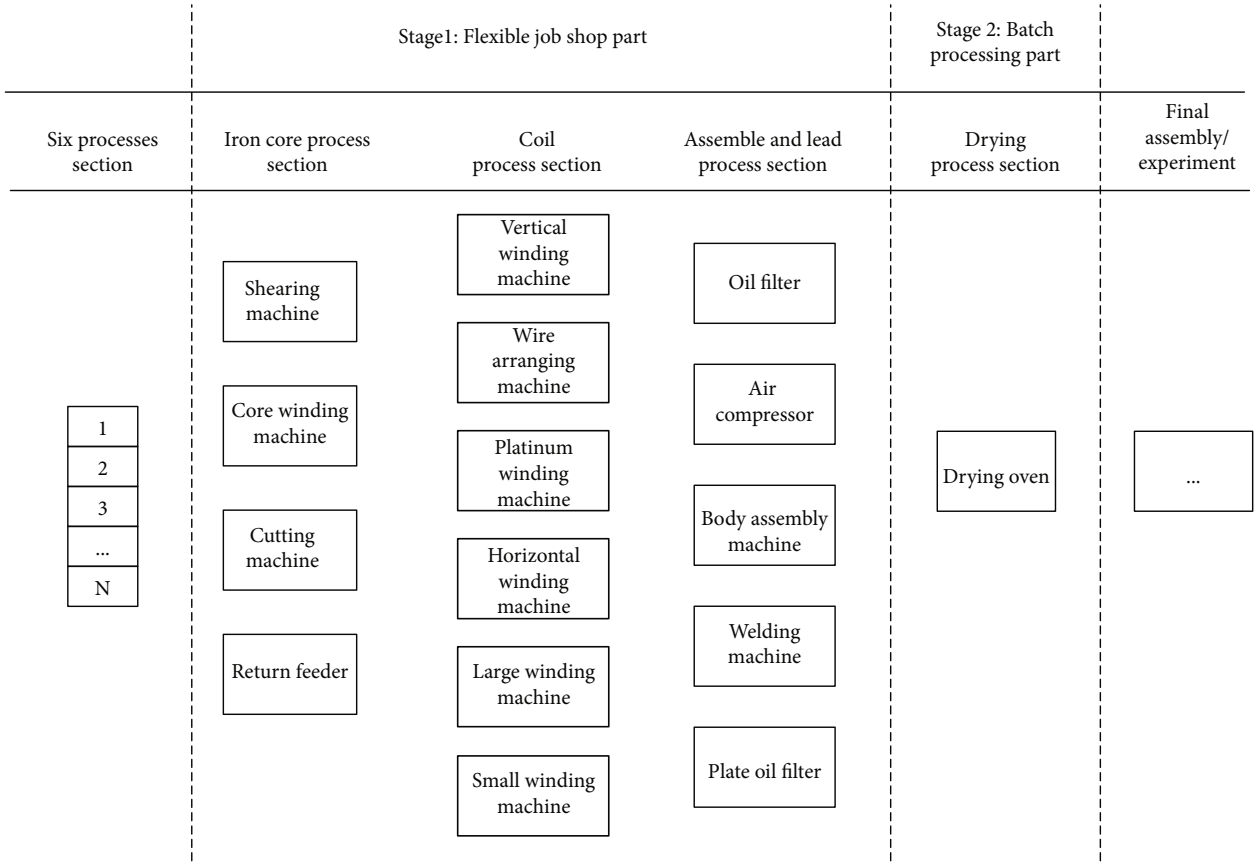

Figure 1: Transformer production flow chart.

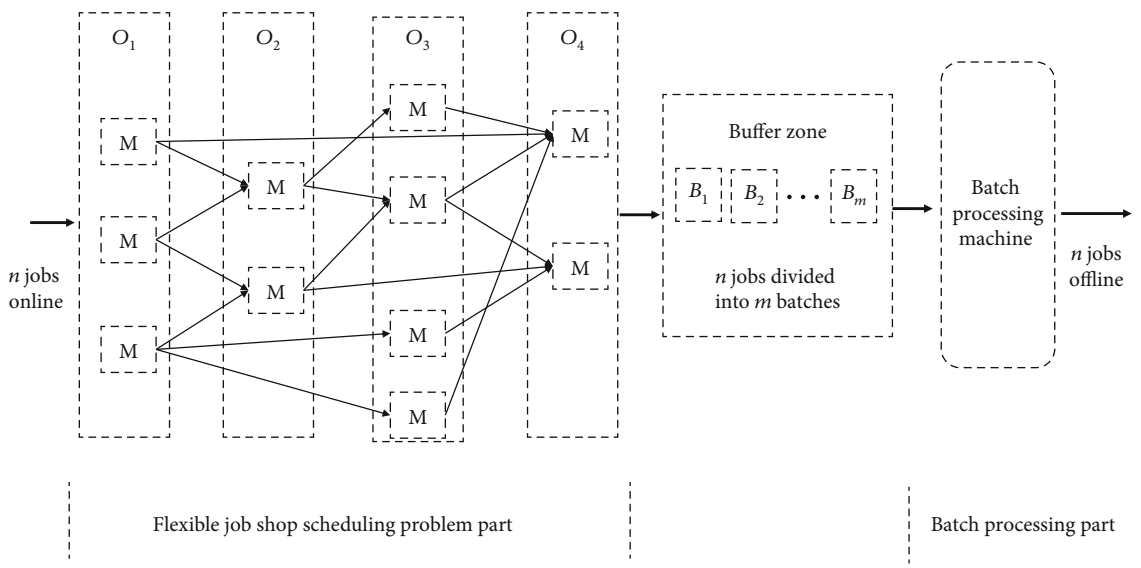

Figure 2: Schematic of flexible job shop scheduling problem with batch processing.

Table 1: Mathematical symbols and their definitions.

Mathematics symbol	Symbol definition and interpretation
n	Total number of jobs
m	Total number of machines
Ω	Total machine set
i, e	Machine serial number, $i, e = 1, 2, 3, \cdots, m$
j, k	Job serial number, $j, k = 1, 2, 3, \cdots, n$。
h_j	Total number of operations of the job j
l	Operation sequence, $l = 1, 2, 3, \cdots, h_j$
Ω_{jh}	Optional processing machine set for the operation h of the job j
m_{jh}	Number of optional processing machines for the operation h of the job j
O_{jh}	Operations h of the job j
M_{ijh}	Operations h of the job j is processed on machine i
p_{ijh}	Processing time of the operations h of the job j on machine i
s_{jh}	Processing start time of the operations h of the job j
c_{jh}	Completion time of the operations h of the job j
I	A positive number large enough
d_j	Due date of the job j
C_j	Completion time of each job
p_j	Batch processing time of the job j
s_j	Size of the job j
r_j	Arrival time of the job j to the buffer of the batch processing machine
C	Batch processing machine capacity
J_b	Set of jobs in batch b
B	Sets of all batches
RT^b	Available time of batch b, the maximum arrival time of all jobs in batch b
ST^b	Start time of batch b
PT^b	Processing time of batch b, the maximum processing time of all jobs in batch b
CT^b	Completion time of batch b

Batch processing part:

(1) The batch machine can only process one batch at a certain time, and the batch cannot be interrupted once it starts processing

(2) The sum of the volume or mass of all jobs in each batch is less than or equal to the volume or mass threshold of the batch machine

For the convenience of subsequent description, the following mathematical symbols are defined, as presented in Table 1.

The mathematical model of the FJSP–BP is expressed as

$$\min\ C_{\max} \tag{1}$$

$$\text{s.t.} s_{jh} + x_{ijh} \times p_{ijh} \le c_{jh}, \tag{2}$$

where $i = 1, 2, \cdots, m$, $j = 1, 2, \cdots, n$, and $h = 1, 2, \cdots, h_j$.

$$c_{jh} \le s_{j(h+1)}, \quad j = 1, 2, \cdots, n, h = 1, 2, \cdots, h_j - 1, \tag{3}$$

$$c_{jh_j} \le C_{\max}, \quad j = 1, 2, \cdots, n, \tag{4}$$

$$s_{jh} + p_{ijh} \le s_{kl} + I\left(1 - y_{ijhkl}\right), \quad j = 0, 1, \cdots, n, k = 1, 2, \cdots, n, h = 1, 2, \cdots, h_j, \quad l = 1, 2, \cdots, h_k, i = 1, 2, \cdots, m, \tag{5}$$

$$c_{jh} \le s_{j(h+1)} + I\left(1 - y_{iklj(h+1)}\right), \quad j = 1, 2, \cdots, n, k = 1, 2, \cdots, n, h = 1, 2, \cdots, h_j - 1, \quad l = 1, 2, \cdots, h_k, i = 1, 2, \cdots, m, \tag{6}$$

$$\sum_{i=1}^{m_{jh}} x_{ijh} = 1, \quad h = 1, 2, \cdots, h_j, j = 1, 2, \cdots, n, \tag{7}$$

$$\sum_{j=1}^{n}\sum_{h=1}^{h_j} y_{ijhkl} = x_{ikl}, \quad i = 1, 2, \cdots, m, k = 1, 2, \cdots, n, l = 1, 2, \cdots, h_k, \tag{8}$$

$$\sum_{k=1}^{n}\sum_{l=1}^{h_k} y_{ijhkl} = x_{ijh}, \quad i = 1, 2, \cdots, m, j = 1, 2, \cdots, n, h = 1, 2, \cdots, h_k, \tag{9}$$

$$s_{jh} \ge 0, c_{jh} \ge 0, \quad j = 1, 2, \cdots, n, h = 1, 2, \cdots, h_j,$$

$$x_{ijh} = \begin{cases} 1, \text{ if operation } O_{jh} \text{ select machine } i \\ 0, \text{ else,} \end{cases} \tag{10}$$

$$y_{ijhkl} = \begin{cases} 1, \text{ if operation } O_{ijh} \text{ precedes } O_{ikl} \\ 0, \text{ else,} \end{cases} \tag{11}$$

$$\sum_{b \in B} x_{jb} = 1, \quad j = 1, 2, \cdots, n, \tag{12}$$

$$\sum_{s=1}^{B} y_{bs} = 1, \tag{13}$$

$$\sum_{j=1}^{n} s_j x_{jb} \le C, \tag{14}$$

$$RT^b \ge r_j x_{jb}, \tag{15}$$

$$PT^b \ge p_j x_{jb}, \tag{16}$$

$$ST^b \ge RT^b, \tag{17}$$

$$ST^{b'} \ge CT^b\left(\forall b, b' \in B\right) \cap \left(\forall s \le B, \sum_{i=1}^{s} y_{bi} \ge \sum_{i=1}^{s} y_{b'i}\right), \tag{18}$$

$$CT^b = ST^b + PT^b, \tag{19}$$

$$x_{jb}, y_{bs} \in (0, 1). \tag{20}$$

Equation (1) is the optimization goal, which is to maximize completion time. Equations (2) and (3) express the sequence constraints of each job. Equation (4) expresses the constraint of the job completion time, where the completion time of each job cannot exceed the total completion time. Equations (5) and (6) indicate that a machine can only process one process per time. Equation (7) represents machine constraints, and an operation can only be processed by one machine per time. Equations (8) and (9) indicate that the operation of each machine can be cyclic. Equation (10) indicates that each parameter must be positive. Equation (12) ensures that each job j can only be arranged in one batch b. Equation (13) ensures that each batch is processed only once on the machine, where $s = 1, 2, \cdots, B$ represents the processing sequence of the batch on the machine. If the processing order of batch b on the machine is s, then $y_{bs} = 1$; otherwise, $y_{bs} = 0$. Equation (14) ensures that the sum of the sizes of the jobs in each batch does not exceed the machine capacity C. Equations (15) and (16) indicate that the available time and processing time of the batch are determined by the maximum arrival and processing times of the jobs in the batch, respectively. Equation (17) indicates that the start time of a batch cannot be earlier than its available time. Equation (18) indicates that batch processing cannot be interrupted, and processing can only begin when a previous batch has been processed. Equation (19) is the expression for calculating batch completion time. Equation (20) indicates that the decision variables x_{jb} and y_{bs} are all 0 or 1 variables [34].

Table 2: A 4×5 FJSP problem.

Job	Operation	Optional processing machine				
		M_1	M_2	M_3	M_4	M_5
J_1	O_{11}	2	6	5	3	5
	O_{12}	—	8	2	4	—
J_2	O_{21}	3	7	—	—	6
	O_{22}	4	6	5	10	—
	O_{23}	—	7	11	—	8
J_3	O_{31}	—	3	8	6	—
	O_{32}	7	8	2	9	—
J_4	O_{41}	—	8	6	4	4
	O_{42}	3	12	—	7	9
	O_{43}	—	5	6	10	—

Table 2 illustrates an example of a 4×5 FJSP problem, which is an example of an FJSP with four jobs, five machines, and 12 total operations. The numbers presented in the table are the processing times of the corresponding machine operation, and "—" indicates that the process cannot be processed on the corresponding machine.

3. FJSP–BP Solution

To obtain the solution to the FJSP–BP, we solve the flexible job shop part and the batch processing part separately. The IIGA is used to solve the flexible job shop part, and the concepts of greedy thought and cross-entropy thought are introduced. The introduced concepts accelerate the optimization speed and search ability of the algorithm. The batch processing part is equivalent to solving the batch scheduling problem of a single-batch processing machine with different arrival times; the rule-based method is used for batch processing and batch sorting.

3.1. Solving the FJSP Part Using an Improved Immune Genetic Algorithm. To obtain the solution to the FJSP part,

Table 3: FJSP chromosome coding.

(a)

Machine selection coding(MS)									
1	3	2	4	2	3	2	5	4	2
J_1		J_2			J_3			J_4	
O_{11}	O_{12}	O_{21}	O_{22}	O_{23}	O_{31}	O_{32}	O_{41}	O_{42}	O_{43}
M_1	M_3	M_2	M_4	M_2	M_3	M_2	M_5	M_4	M_2

(b)

Operation sequencing coding(OS)									
2	1	3	4	4	2	1	2	3	4
J									
O_{21}	O_{11}	O_{31}	O_{41}	O_{42}	O_{22}	O_{12}	O_{23}	O_{32}	O_{43}

we propose the IIGA. The performance of the algorithm is improved by the concepts of greedy thought and cross-entropy thought. We design effective encoding and decoding methods and optimize the selection, crossover, mutation, and other operations in the algorithm framework.

3.1.1. Chromosome Coding. The FJSP part includes two subproblems: machine selection and operation sequencing, which are independent of each other. Therefore, we code machine selection and operation sequencing separately. The chromosome is divided into two parts, A/B, representing the machine selection and operation sequencing subproblems, respectively. The length of both parts of the chromosome is equal to T_0, which represents the total number of operations.

Machine selection subproblem: the length of the machine selection part of the chromosome is T_0. Each gene locus is represented by an integer, and the integers are arranged sequentially according to the job and its order. Each integer represents the serial number of the processing machine selected in the current operation. The FJSP example presented in Table 2 illustrates the machine coding method. The encoding result is presented in Table 3.

Operation sequencing subproblem: the chromosome length of the process sequencing part is equal to the total number of processes T_0. Each gene is directly coded with a process number, and the order in which the part number appears indicates the process sequence among the job operations; thus, the chromosomes are compiled from left to right. For example, the h-th operation number indicates the h-th operation of job j, and the number of occurrences of the job number is equal to the total number of operations h_j of the job.

3.1.2. Chromosome Decoding. The decoding part also comprises the machine selection and operation sequencing decoding.

Machine selection decoding: the machine selection part is decoded from left to right and converted into a machine sequence matrix J_M and time sequence matrix T. $J_M(j,h)$ represents the machine number of operation O_h of job J_j; $J_M(j,\bullet)$ represents the arrangement of the machine numbers processed by all operations of the job J_j in order of priority; and $T(j,h)$ represents the processing time of operation O_h of job J_j. $J_M(j,h)$ and $T(j,h)$ have a corresponding relationship.

$$J_M = \begin{bmatrix} 1 & 3 & \\ 2 & 4 & 2 \\ 3 & 2 & \\ 5 & 4 & 2 \end{bmatrix}, \quad T = \begin{bmatrix} 2 & 2 & \\ 7 & 10 & 7 \\ 8 & 8 & \\ 4 & 7 & 5 \end{bmatrix}. \tag{21}$$

Operation sequencing decoding: the chromosomes of the operation sequencing part are read from left to right, and the processing machine and processing times corresponding to each job process are obtained according to the machine and time sequence matrices corresponding to the machine selection part. Finally, the operation is sorted to obtain the scheduling result.

3.1.3. Chromosome Crossover. Machine selection part: the machine selection part must ensure that the sequence of each gene remains the same, using the uniform crossover.

Step 1. Randomly generate an integer r in the interval $[1,T_0]$ to represent the number of gene positions that remain unchanged.

Step 2. Generate r unequal integers, which correspond to the specific positions of the r invariant gene positions.

Step 3. According to the r integers generated in Step 2, copy the genes at the corresponding positions in the parent chromosomes P_1 and P_2 to the offspring chromosomes C_1 and C_2, maintaining their position and order.

Step 4. Genetically copy the remaining genes of P_1 and P_2 to C_1 and C_2, maintaining their position and order.

As Figure 3 illustrates and taking Table 2 as an example, P_1 and P_2 are the two parent chromosomes. Assuming $r = 3$ and the three random numbers of Step 2 are 2, 5, and 7, then the offspring chromosomes C_1 and C_2 are obtained by the uniform crossover.

Operation sequencing part: the operation sequencing part adopts the precedence preserving order-based crossover method, and the operation of multiple jobs in each chromosome can better inherit the excellent characteristics of the parent individual.

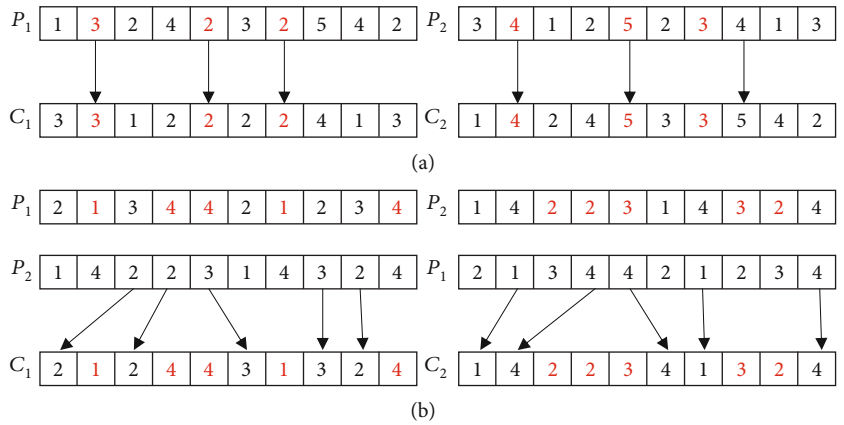

Figure 3: FJSP chromosome crossover. (a) Machine selection uniform crossing. (b) Operation sequencing precedence preserving order-based crossover crossing.

Step 1. Randomly divide the job set $\{J_1, J_2, \cdots, J_n\}$ into two job sets, Jobset1 and Jobset2.

Step 2. Copy the processes of the jobs contained in Jobset1/Jobset2 in the parent chromosomes P_1 and P_2 to C_1/C_2, maintaining their positions and sequences.

Step 3. Copy the processes of the jobs excluded from Jobset1/Jobset2 in P_1 and P_2 to C_2/C_1, and maintain only their order.

In Figure 3, Jobset1 = $\{1, 4\}$.

3.1.4. Chromosome Variation. Both the machine selection and operation sequencing parts use the random variation method, and the steps are outlined as follows.

Machine selection part:

Step 1. Randomly select r positions in the variant chromosome.

Step 2. Select each position in turn and randomly replace the machine in each position with one of the machines in the set of optional machines.

Operation sequencing part:

Randomly select r genes in the mutant chromosome, maintain the position and order of genes in other positions, and randomly order the selected r genes to replace their original positions and orders.

3.1.5. Greedy Optimal Solution Thought. We improve the IGA and introduce the greedy thought. Before each iteration of the algorithm, a "greedy optimal solution" is obtained through the greedy thought.

This greedy optimal solution is not necessarily the global optimal solution, but it is highly similar to the global optimal solution with a high probability. Therefore, the optimization can be accelerated by selecting highly similar individuals to the greedy optimal solution in the IIGA.

The greedy optimal solution selection thought is as follows:

(1) When selecting the machine for each job operation, choose the machine with the shortest process time in the optional machine collection

(2) During machine selection, distribute the process evenly to each machine as much as possible

(3) Considering the constraints of the operation sequence, start processing the earlier processes as soon as possible

Greedy optimal solution selection: the steps for selecting the greedy optimal solution for n jobs and m machines are as follows.

Step 1. Set up an integer array with a length equal to the total number of machines m, followed by the machine serial number $[M_1, M_2, \cdots, M_m]$; the array corresponds to the processing time, and each element in the array is initialized to zero.

Step 2. Among the initialized n jobs, randomly select a job J_{j^1} and set its first operation as $O_{j^1 1}$. Select the machine M_{m^1} with the shortest processing time in the collection of processing machines where the operation $O_{j^1 1}$ is located.

Assign operation $O_{j^1 1}$ to machine M_{m^1} and define the processing time of the corresponding machine. $M_{m^1} = T_{j^1 1 m^1}.T_{j^1 1 m^1}$ is the processing time of the job J_{j^1} on machine M_{m^1}.

Step 3. Randomly select J_{j^2} from the remaining $n-1$ jobs and set its first operation as $O_{j^2 1}$. Select the machine M_{m^2} with the shortest processing time in the collection of processing machines where the operation $O_{j^2 1}$ is located. (At this time, the processing time of the first operation $O_{j^2 1}$ of job J_{j^2} on machine M_{m^1} should be $T_{j^2 1 m^1} = T_{j^2 1 m^1} + M_{m^1}$.) Update $M_{m^1} = T_{j^2 1 m^1}$. If the time is unchanged, the machine with a small number of assigned operations is preferred.

Steps	Random job	Corresponding operation	Machine selection	Machine array					Machine coding
				M_1	M_2	M_3	M_4	M_5	
Stage1	J_2	O_{21}	M_1	3	0	0	0	0	1
	J_4	O_{41}	M_4	3	0	0	4	0	4
	J_3	O_{31}	M_2	3	3	0	4	0	2
	J_1	O_{11}	M_3	3	3	5	4	0	3
Stage2	J_3	O_{32}	M_3	3	3	7	4	0	3
	J_4	O_{42}	M_1	5	3	7	4	0	1
	J_1	O_{12}	M_4	5	3	7	8	0	4
	J_2	O_{22}	M_2	5	9	7	8	0	2
Stage3	J_2	O_{23}	M_5	5	9	7	8	8	5
	J_4	O_{43}	M_3	5	9	13	8	8	3

Illustrate by the FJSP in Table 2 . In stage 1, the order of randomly selected jobs is 2-4-3-1. According to the greedy optimal solution solving rule, and the processing time of their first operation, the machine selection result is 1-4-2-3. Then complete stage 2 and stage 3 according to the solving rules. Finally, the machine selection result is converted into MS code

J_1		J_2			J_3		J_4		
3	4	1	2	5	2	3	4	1	3

Figure 4: Flow chart description of solving the greedy optimal solution, converted to machine selection coding.

Step 4. By analogy, complete the machine allocation problem of the first operation of all jobs.

Step 5. Repeat Steps 2–4 to complete the machine allocation problem of all operations of all jobs.

Step 6. Get the greedy optimal solution X_{best} of the final machine allocation result.

The solution process is illustrated in Figure 4.

3.1.6. Information Entropy and Cross-Entropy. In the IGA, concepts such as antibody information entropy, antibody similarity, and antibody concentration are introduced. The immune optimization method calculates the affinity between the antibody and antigen as well as the similarity between antibodies. The adoption of a concentration-based selection mechanism not only encourages antibodies with high adaptability but also inhibits antibodies with high concentrations, which reflects the regulatory function of the immune system and can prevent falling into a local optimal solution.

We introduce the concept of cross-entropy into the IGA. Cross-entropy is an important concept in Shannon's information theory, mainly used to measure the difference in information between two probability distributions. If $p(x)$ is the true probability distribution of the data and $q(x)$ is the probability distribution calculated from the data, then the cross-entropy is defined as

$$H(p, q) = -\sum_{x} p(x) \ln q(x). \tag{22}$$

Cross-entropy is often used in machine learning. It is hoped that the data distribution $q(x)$ obtained by the algorithm is as close as possible to the true distribution $p(x)$ of the data. In machine learning, cross-entropy is generally defined as a loss function.

In the initial stage of the population, we obtain the greedy optimal solution X_{best} through greedy thought. X_{best} corresponds to the true distribution $p(x)$ according to the cross-entropy thought, and other ordinary individuals in the population correspond to $q(x)$. We expect individuals to be as similar to X_{best} as possible. By calculating the cross-entropy, similarity, and other indicators of each individual x_i and X_{best}, the better individual is selected in the selection process, which speeds up the optimization process of the entire algorithm. The calculation method of cross-entropy is illustrated in Table 4 and by Equations (23) and (24).

$$q_k(x) = \frac{\text{The number of } X_{\text{best\_}k} \text{in the k-th gene}}{\text{The number of the k-th gene}}, \tag{23}$$

$$p_k(x) = \frac{\text{The number of } X_{\text{best\_}k} \text{ in the k-th gene}}{\text{Number of optional machines for the k-th gene}}. \tag{24}$$

Table 4: Cross-entropy calculation explanation.

X_{best}	3	4	1	2	5	2	3	4	1	3
X_1	4	4	1	2	5	2	3	4	1	3
X_2	3	2	1	2	5	2	3	4	1	3

$$H(X_{\text{best}}, X_1) = -\sum_{k=1}^{10} p_k(x) \ln q_k(x) = -\left(\frac{1}{5} \bullet \ln \frac{1}{2} + 0 + \cdots \_0\right)$$

$$H(X_{\text{best}}, X_2) = -\sum_{k=1}^{10} p_k(x) \ln q_k(x) = -\left(0 + \frac{1}{3} \bullet \ln \frac{1}{2} + 0 + \cdots \_0\right)$$

Cross-entropy similarity definition:

$$A_{X_{\text{best}}, X_i} = \frac{1}{1 + H(X_{\text{best}}, X_i)}. \tag{25}$$

The greater the value of the cross-entropy similarity is, the higher the similarity between this individual and the greedy optimal solution.

3.1.7. Algorithm Steps of the IIGA to Solve the FJSP Part. The IIGA framework is outlined as follows.

The flow chart of the IIGA is shown in Figure 5.

3.2. Batch Process Part. For each solution in the flexible job shop part, all the jobs will arrive sequentially in the buffer in front of the batch machine with different arrival times. For the batch machine, this is equivalent to the batch processing of problems with different arrival times [35].

Batch rules: BF (best fit) rules:

(1) Given a sequence of jobs, a solution for the flexible job shop part is calculated by calculating the arrival time of all the jobs in the front buffer of the batch machine. Sort the jobs according to the arrival time from smallest to largest

(2) Select the first job J_1 in the sequence as the first job of batch b_1. Then, check whether the second job can be added to the batch b_1; if it can be added, add job J_2 to batch b_1; otherwise, create a batch b_2 with job J_2 as the first job

(3) Then check whether the job J_3 can be added to batches b_1 and b_2 in sequence. If it is possible, add the job to the batch; if not, create a new batch b_3

(4) When the capacity of a certain batch reaches the maximum value, no other job can be added. For job J_i, find the batch b_k that can accommodate the modified job and the earliest arrival time among all unfinished batches; then, add job J_i to the batch. Otherwise, create a new batch b_l and add the job to the new batch

(5) By analogy, repeat the above steps until all the jobs are finished in batches

When multiple formed batches wait in the buffer, the batch sorting problem is involved.

Batch sorting rules: ERT–LPT (earliest release time–longest processing time):

(1) Calculate the arrival and processing times of all batches. The arrival time of the batch is the arrival time of the latest job in the batch, and the processing time of the batch is the maximum processing time of the job in the batch

(2) Arrange the batches in the order of arrival time. If there are multiple batches with the same arrival time, they are further arranged in the order of the batch processing time

(3) Select the machine that is currently idle and arrange the first batch in the batch sequence for processing on this machine

(4) Repeat Step (3) until every batch scheduling is completed; finally, calculate the makespan

4. Experimental Results and Analysis

For the IIGA verification in this study, we used MATLAB 2014 run on an environment with an Intel Core I5 fourth-generation CPU (3.20 GHz main frequency), 8 GB memory, and Windows 7 operating system (64-bit, professional edition). The experiments were divided into two parts. In the first part, to verify the effectiveness of the IIGA, we used a standard FJSP example to compare the performance of different algorithms. In the second part, we focused on the FJSP–BP. Taking the actual data of a transformer manufacturing enterprise as an example, we solved the FJSP–BP by the IIGA and used batching rules to verify the feasibility and effectiveness of the algorithm.

4.1. Algorithm Parameter Setting. The IIGA used in this study had three parameters at the algorithmic level: crossover probability *P*c, mutation probability *P*m, and selection parameter τ in the selection process.

Determining and optimizing parameters is extremely complicated, thereby requiring continuous simulation experiments. For the crossover probability *P*c and mutation probability *P*m under the genetic algorithm framework, the crossover probability *P*c was set to (0.6, 0.9), while the mutation probability *P*m ranged between (0.005, 0.02). These values were based on previous experiments. In the following examples, the operating parameters of the crossover

1: Randomly generate initial populations P_0 and Q_0; the population size is N; set $t = 0$.
2: Combine the parent and offspring populations, namely, $R_t = P_t \cup Q_t$.
3: Obtain X_{best} through the greedy thought and then calculate the fitness value of the 2N individuals in R_t and the cross-entropy value and similarity with X_{best}, respectively.
4: Arrange F_{2N} and H_{2N} according to their values and take the largest of the first N phases as F_N and H_N.
5: Let $T_E = F_N \cap H_N$
6: Define $P_{t+1} = T_E + X_{best} + (F_N - T_E)_{\tau\bullet(N-1-E)} + (H_N - T_E)_{(1-\tau)\bullet(N-1-E)}$; update population P_{t+1}
7: Perform a crossover operation on the population P_{t+1}.
8: Perform a mutation operation on the population P_{t+1} to generate a new population Q_{t+1}.
9: If $t <$ maxgen, then $t = t + 1$; return to Step 2; otherwise, the algorithm terminates.

Algorithm 1: IIGA framework.

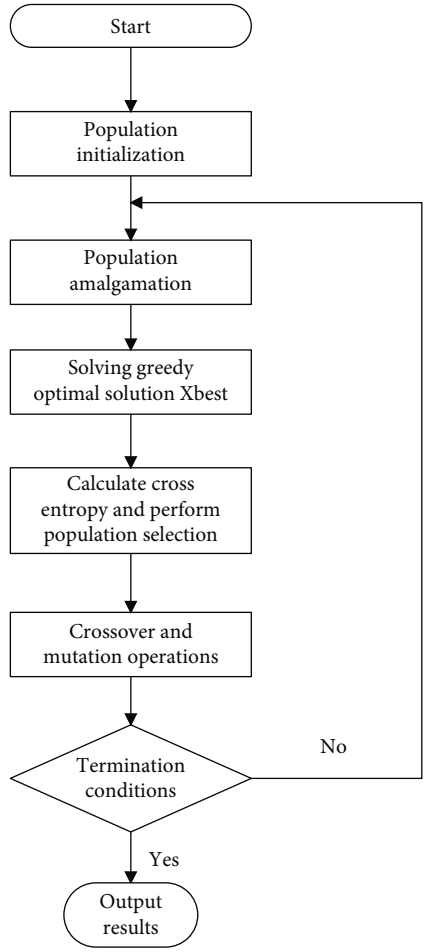

Figure 5: Flow chart of IIGA for solving the FJSP part.

probability Pc and mutation probability Pm were set to 0.75 and 0.01, respectively. For the selection parameter τ introduced by the improved algorithm in this study, we used the MK01 example to test the selection of τ.

The MK01 example is one of the ten classic FJSP test problems designed by Brandimarte. The problem contains ten jobs, six machines, and 55 operations. We considered different settings of τ to determine the influence of the parameter on the performance of the algorithm. The population size and the maximum number of iterations were both 100, and the algorithm ran 20 times continuously. The experimental results are illustrated in Figure 6.

Figure 6 shows that when τ is relatively small, the part selected by cross-entropy in the selection process is larger, the average number of iterations when the algorithm converges is small, and the algorithm converges quickly, but the number of optimal solutions and the average optimal solution are inferior. When τ is relatively large, the selection process based on fitness is larger, the average number of iterations when the algorithm converges is large, the algorithm

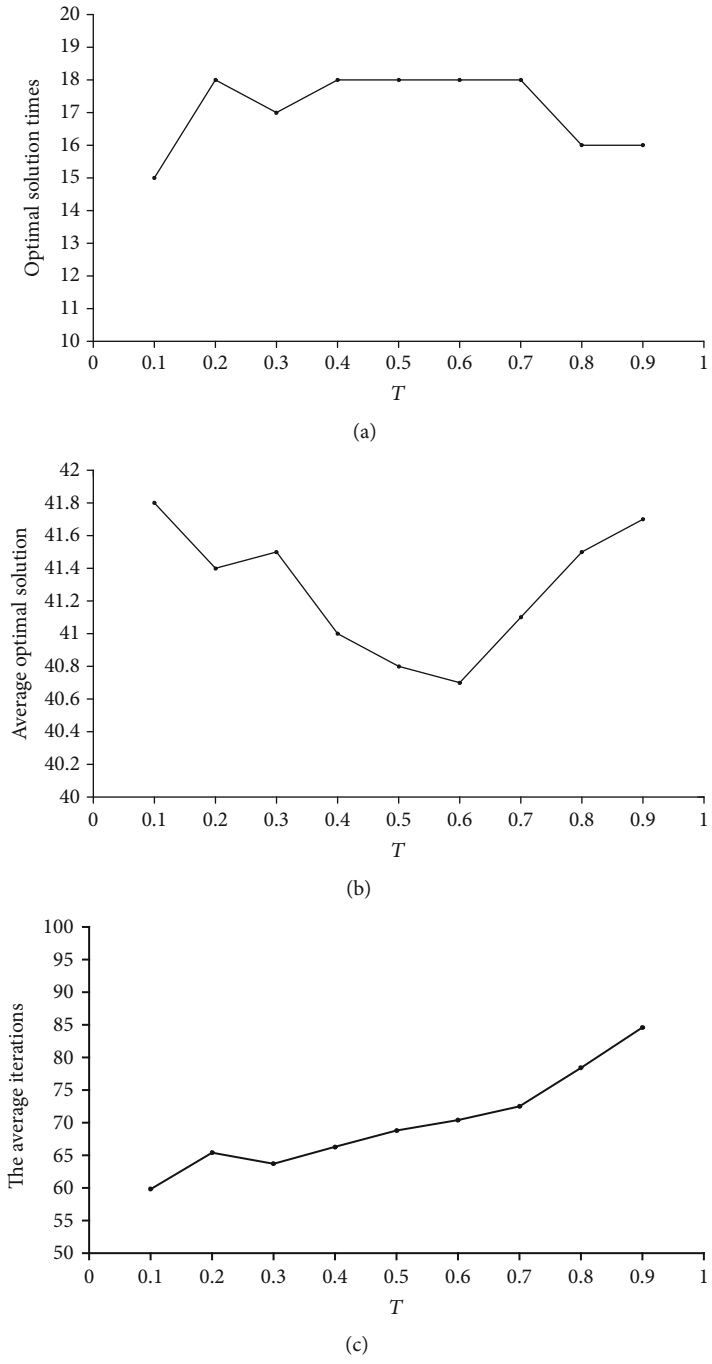

Figure 6: Optimization of τ. (a) Relationship between the optimal solution times and τ. (b) Relationship between the average optimal solution and τ. (c) Relationship between the average iterations when converging to the optimal solution and τ.

convergence speed is low, and the number of optimal solutions and the average optimal solution effect are moderate. When the value of τ is in the range of (0.4, 0.6), the algorithm considers both the convergence speed and optimal solution quality. Although the convergence speed is moderate, the number of optimal solutions and the average optimal solution effect are both good. Therefore, in the following experiment, the value of τ was set to 0.5.

4.2. Standard FJSP Example to Verify the Effectiveness of the IIGA. To verify the improvement effect of the improved algorithm, we used 27 sets of FJSP standard case data to test and analyze the IIGA. The genetic algorithm, IGA, and the emerging group intelligence algorithm JAYA were used for comparison. The 27 sets of data used include the following:

(1) Five groups of Kacem calculation examples. The Kacem calculation example is the earliest standard FJSP calculation example. The five questions are combined from the number of jobs from 4 to 15 and the number of machines from 5 to 15

Table 5: Kacem and BR problem test results.

Problem	$n \times m$	LB,UB	GA		IGA		JAYA		IIGA	
			C_M	AV_{C_M}	C_M	AV_{C_M}	C_M	AV_{C_M}	C_M	AV_{C_M}
Kacem1	4 × 5	11,*	11	11.5	11	11	11	11	11	11
Kacem2	8 × 8	14, *	23	24.6	15	16.2	14	14.6	14	14.3
Kacem3	10 × 7	11, *	19	21.6	14	14.8	11	12	11	11.6
Kacem4	10 × 10	7, *	13	16.4	8	8.3	7	7.8	7	7.5
Kacem5	15 × 15	11,*	27	31.5	17	18.4	14	14.7	11	12.1
Mk01	10 × 6	36,42	40	41.5	40	40	40	40	40	40
Mk02	10 × 6	24,32	29	29.1	26	26	26	26	26	26
Mk03	15 × 8	204,211	204	204	204	204	204	204	204	204
Mk04	15 × 8	48,81	67	47.34	60	60	60	60	60	60
Mk05	15 × 4	168,186	176	178.1	173	176.8	173	174.4	172	175.2
Mk06	10 × 15	33,86	67	68.82	58	60.5	58	60.5	57	58
Mk07	20 × 5	133,157	147	152.9	144	146.3	144	148.5	139	140.2
Mk08	20 × 10	523	523	523.34	523	523	523	523	523	523
Mk09	20 × 10	299,369	320	327.74	311	311	307	309	307	310.8
Mk10	20 × 15	165,296	229	235.72	201	203.6	197	200.2	196	198.6

(2) Ten sets of BRdata calculation examples. The BRdata calculation example is one of the classic standard FJSP calculation examples proposed by Brandimarte. The ten questions are combined from the number of jobs from 10 to 20 and the number of machines from 4 to 15. The process of each group of questions ranges from 5 to 15, and the average number of processing machines for each process in each group of questions ranges from 1.43 to 4.10

(3) Twelve groups of BCdata calculation examples. The BCdata example is a 21-example problem proposed by Barnes and Chambers. The 21 calculation examples are mainly constructed from the three most challenging problems in the classic JSP (mt10, la24, and la40) according to certain principles. The average number of processing machines in each process of the BCdata calculation example is relatively small, and its data type and flexibility are similar to those in the flexible job shop part of the transformer production line. Therefore, 12 of 21 groups of BCdata calculation examples were selected for algorithm verification

Each algorithm ran 30 times on each group of data, the initial population of the four algorithms was 100, and the maximum evolutionary number was 200 generations. $n \times m$ represents the problem scale; LB and UB represent the lower and upper bounds of the optimal solution, respectively; C_M is the minimum value of the maximum completion time; and AV_{C_M} represents the average optimal value obtained by running the algorithm 30 times continuously. The test results are presented in Tables 5 and 6.

Table 5 shows that for the Kacem and BR examples, IGA has a similar optimization effect to that of JAYA, but JAYA is slightly better. The average completion time of the two algorithms is relatively small, while their optimization effect is rather better. Compared with IGA and JAYA, both C_M and AV_{C_M} of the GA are larger, and the optimization effect is relatively poor. In terms of AV_{C_M}, IIGA is slightly worse than JAYA in the MK05 and MK09 instances but better than JAYA in the seven instances of kacem2-kacem5, MK06, MK07, and MK10; the AV_{C_M} of the two algorithms is the same in the other six instances. Therefore, the performance of the two algorithms is similar in terms of AV_{C_M}. IIGA is slightly better than JAYA but far better than the other two algorithms. In terms of C_M, IIGA has the best value on five instances of kacem5, MK05, MK06, MK07, and MK10; in the remaining ten instances, the best value of C_M is parallel to other three algorithms. Thus, IIGA is significantly better than the other three algorithms in terms of C_M.

For BCdata, in terms of the optimization effect, IIGA has a significant improvement compared with the other three algorithms. In terms of the average optimization effect, the optimization effect of IIGA is also significantly better than that of GA, IGA, and JAYA.

During testing, we found that when four algorithms used data for 30 simulation experiments, the C_M obtained each time had obvious volatility. Because the algorithm easily falls into local extreme values, its C_M may even have large abnormal values. To better compare and evaluate the optimization effects of the four algorithms, we selected the MK06 group of data as an example and drew the C_M obtained by running the four algorithms 30 times into a box plot, as shown in Figure 7.

The box plot is a statistical graph used to show the degree of dispersion in a set of data. The stability of the optimization effect can be reflected through the box plot. The interquartile range (IQR) is used in the box plot to measure the degree of dispersion in the data.

Table 6: Twelve BCdata problem test results.

Problem	$n \times m$	LB,UB	GA		IGA		JAYA		IIGA	
			C_M	AV_{C_M}	C_M	AV_{C_M}	C_M	AV_{C_M}	C_M	AV_{C_M}
mt10c1	10 × 11	655,927	928	928.2	927	927.2	927	927.4	927	927
mt10cc	10 × 12	655,914	910	912.4	910	911.7	908	908.8	908	908
mt10x	10 × 11	655,929	918	919.6	918	918	918	918	918	918
mt10xx	10 × 12	655,936	918	918.6	918	918	918	918	918	918
setb4c9	15 × 11	857,924	919	920.4	919	919.2	914	914.6	914	914.2
setb4cc	15 × 12	857,909	909	915.0	909	911.6	907	910.0	907	908.5
setb4x	15 × 11	846,937	925	934.3	925	926.8	925	925	925	925
setb4xx	15 × 12	847,930	925	933.7	925	925.4	925	925	925	925
seti5c12	15 × 16	1027,1185	1174	1184.7	1174	1176.0	1174	1174.2	1170	1171
seti5cc	15 × 17	955,1136	1136	1146.5	1136	1137.2	1136	1136.4	1135	1135.8
seti5x	15 × 16	955,1218	1209	1213.2	1200	1209.0	1201	1203.6	1198	1199.4
seti5xx	15 × 17	955,1204	1204	1205.9	1199	1200.6	1198	1202.4	1197	1198.3

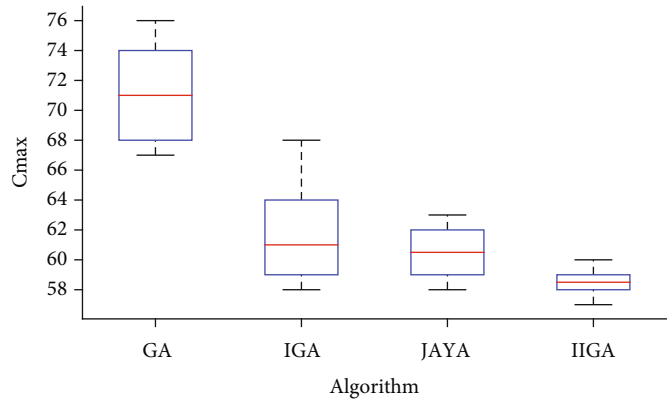

Figure 7: Box plot of the four algorithms.

Figure 7 shows that in the 30 runs of the algorithm, IGA had the widest solution range fluctuation, indicating that it is easier for the algorithm to fall into the local extreme value because of the immune mechanism. The position of the box plot generated by IIGA is the lowest, indicating that the overall quality of the solution generated by the algorithm is better than that of the other three algorithms. Moreover, the IQR of the box plot generated by IIGA is smaller than that of the other three algorithms, showing that the degree of dispersion of the solution produced by the algorithm is smaller than that of the other three algorithms. Its stability is also the best among the four algorithms.

We plotted the relationship between the maximum completion time and the number of iterations of the four algorithms under the MK06 data, as shown in Figure 8.

Figure 8 shows that IIGA has the fastest convergence speed and the smallest optimal solution, and it converges to the optimal solution in the shortest time: $C_{\max}$ of 57 after about 60 iterations. The final convergence value of GA is 67, and the performance is the worst. The optimal values of IGA and JAYA are both 58, but the optimization speed of IGA is faster. Analyzing the search mechanism of the two algorithms shows that JAYA uses the optimal individual X_{best} and worst individual X_{worst} in the search process for optimization according to the iterative formula, and it only selects individuals according to the fitness function value, making the algorithm easy to fall into a local optimum. Meanwhile, IGA calculates the similarity between antibodies and antibodies in addition to the affinity between antibodies and antigens in the optimization process, and the algorithm adopts a concentration-based selection mechanism, which encourages the adaptation of antibodies with a high degree of concentration to inhibit antibodies with a high concentration. The algorithm reflects the regulation function of the immune system, which can escape the local optimal solution and accelerate the convergence speed. The selection mechanism and results of the convergence speed of both algorithms are different.

Figure 9 shows the Gantt chart of IIGA under the MK06 example.

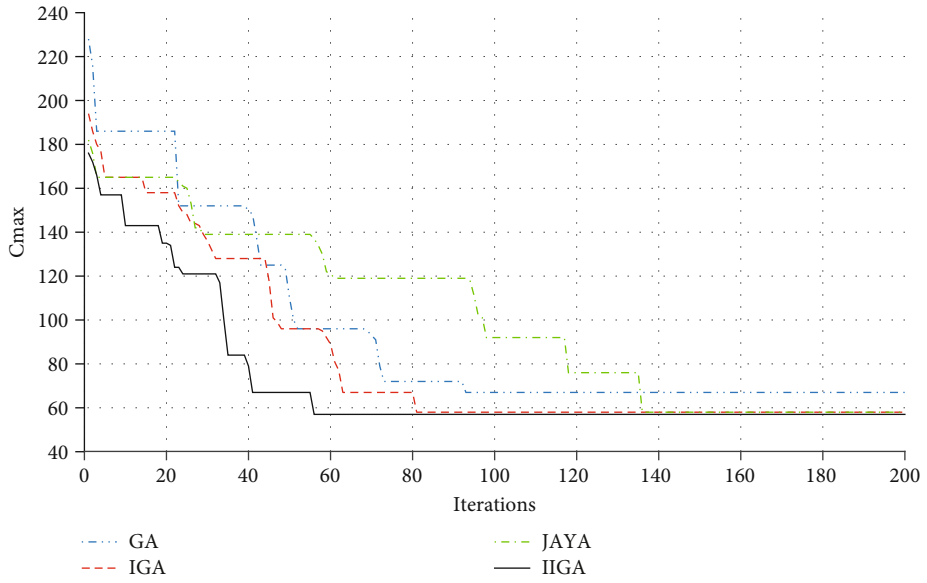

Figure 8: Iteration graph of the four algorithms.

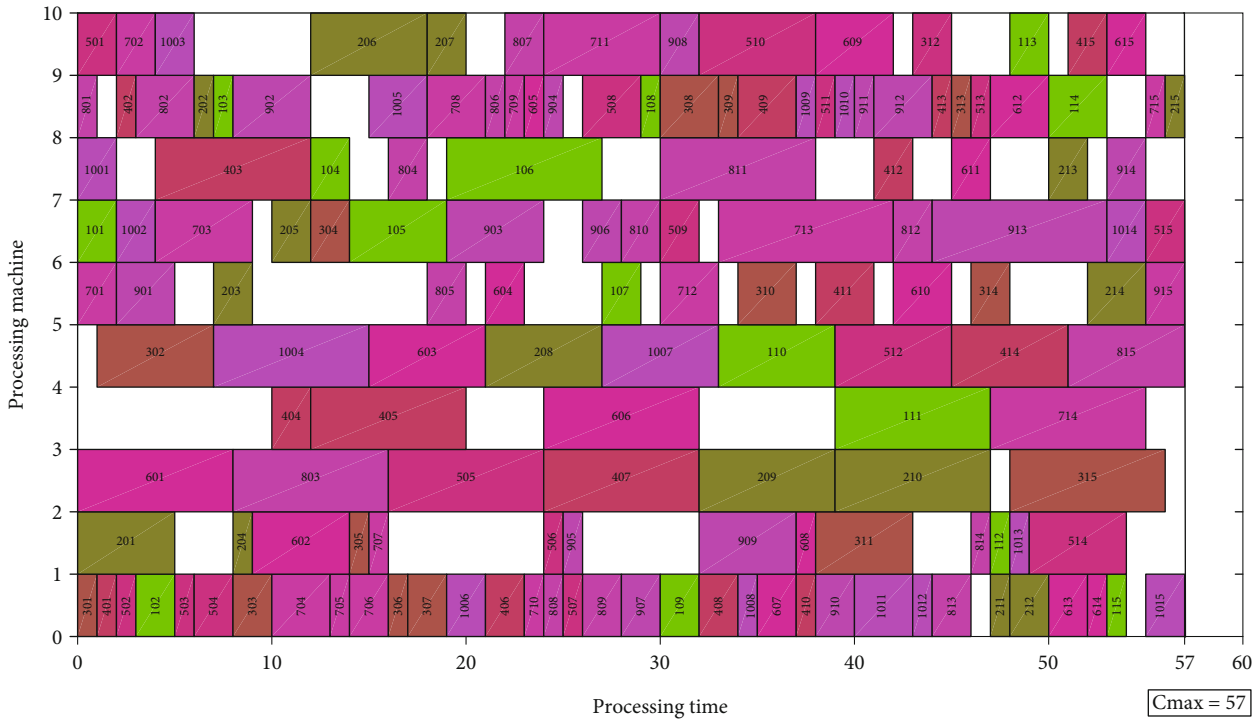

Figure 9: MK06 optimal solution Gantt chart.

4.3. Transformer Manufacturer Data. We considered the data from an actual transformer manufacturing enterprise as an example. The production workshop of the transformer manufacturer includes several sections, including the coil process, iron core, and lead process setting sections, as the FJSP part; the transformer body drying process section is the BP part.

The FJSP part contains multiple machines and equipment, and the drying oven is batched with job volume as a constraint. We simplified the FJSP part into a total of 20 machines. In the BP part, the selected jobs are divided into two categories. Similar jobs have the same volume, and the FJSP part has different processing times; the job volume differs by category, and the batch processing times differ. The batch processing part normalizes the volume, and the batch machine has a capacity of 12; the volumes of the two job types are three and four, respectively.

Among the two job types, the job of class A is a laminated core transformer, while the job of class B is an

Table 7: Basic information on two job types.

Job category	Number of operations	Number of machines	Job volume	Batch machine volume	Batch processing time
Category A	11+1	20+1	3	12	10
Category B	11+1		4		12

Table 8: Detailed processing information on two job types.

Category A				Category B			
Operation number	Operation name	Optional machine sets	Processing time	Operation number	Operation name	Optional machine sets	Processing time
1	Low voltage winding	1-3	5-7	1	Low voltage winding	1-3	6-8
2	High voltage winding	4-7	6-9	2	Pressing and drying	8-9	3-5
3	Pressing and drying	8-9	4-5	3	High voltage winding	4-7	7-10
4	Remove the mold	10-11	2-3	4	Pressing and drying	8-9	3-4
5	Turn measurement and brush glue	10-11	7-8	5	Remove the mold	10-11	2-3
6	Inspection	12-13	3-4	6	Turn measurement and brush glue	10-11	8-10
7	Coil assembly	14-20	8-10	7	Inspection	12-13	3-4
8	Insert silicon steel sheet	14-20	6-8	8	Coil assembly	14-20	7-9
9	Body assembly	14-20	4-6	9	Body assembly	14-20	4-6
10	Lead assembly	14-20	3-5	10	Lead assembly	14-20	2-5
11	Inspection	12-13	3-4	11	Inspection	12-13	3-4
12	Drying	21	10	12	Drying	21	12

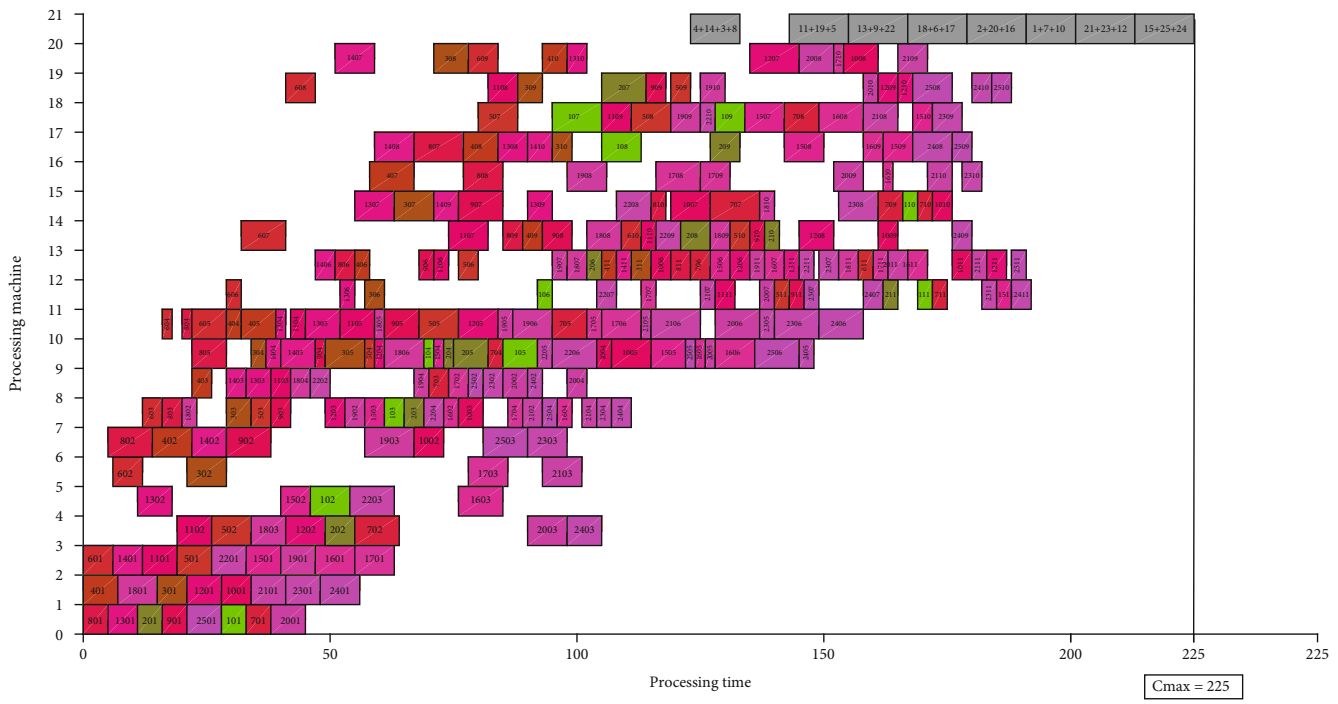

Figure 10: Gantt chart of the transformer production process ($J = 25$).

amorphous alloy transformer. Table 7 presents basic information about the two job types.

Both types of jobs in the FJSP part have 11 operations. Compared with class A jobs, class B jobs require two press-fitting and drying operations, and class B jobs have no insertion operation. In the actual production process, the iron core and coil operations of class A products are produced in parallel, and the iron core of class B products is purchased out. In this study, we investigated the coil process, body equipment, and drying process sections. The iron core part was set as a prefabricated part, and unified installation was performed in the coil assembly operation. The detailed processing information of the two job types is shown in Table 8.

According to the actual production situation of the transformer manufacturer, we use half a month's production data to verify the effectiveness of the algorithm. The total number of jobs is $J = 25$, where $A = 15$ and $B = 10$. The Gantt chart of scheduling results is shown in Figure 10.

Figure 10 shows that the jobs are evenly distributed to each machine, and the completion time is 225. In the actual production process, the completion time is 282. Thus, the completion time is increased by 25.3%, reflecting the effectiveness of the algorithm in this paper.

5. Conclusion

In this study, a combination of intelligent methods and scheduling rules is used to solve the FJSP–BP problem. An improved IGA based on the concept of greed and cross-entropy is proposed, and an effective batching method and batching rules are designed. The standard FJSP benchmark example and data conforming to the FJSP-PB problem from a transformer manufacturing enterprise were used to verify the effectiveness and practicability of the proposed algorithm. This work provides a significant reference for solving the FJSP–BP problem.

From a scheduling problem perspective, we only investigated the single-batch machine problem under small-scale data. In future research, we will thoroughly study the FJSP with parallel batch machines under medium- and large-scale data. As the problem is more complicated, the selection of batch machines must be considered based on MS and OS. Meanwhile, analysis and research will be conducted for different optimization objectives, including machine load, equipment energy consumption, and early/delay penalty. In addition, in terms of research methods, it is necessary to determine a global optimization algorithm that considers both the FJSP and BP parts as well as a multiobjective optimization algorithm that includes multiple optimization objectives.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no competing interests.

Authors' Contributions

Libo Song's contributions are conceptualization, original draft writing, methodology design, formal analysis, and MATLAB program development. Chang Liu's contributions are investigation, resource acquisition, and manuscript verification. Haibo Shi and Jun Zhu's contributions are funding acquisition, supervision, and guidance.

Acknowledgments

This work was supported in part by the Liaoning Revitalization Talents Program under Grant (XLYC1808009) and in part by the Key R & D Projects in Liaoning Province (2020JH2/10100039).

References

[1] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[2] S. K. Zhao, "Improved hybrid algorithm of neighborhood structure for flexible job shop scheduling," *Computer Integrated Manufacturing Systems*, vol. 24, no. 12, pp. 3060–3072, 2018.

[3] J. Lin, L. Zhu, and Z. J. Wang, "A hybrid multi-verse optimization for the fuzzy flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 127, pp. 1089–1100, 2019.

[4] Y. J. An, X. H. Chen, Y. H. Li, Y. Han, J. Zhang, and H. Shi, "An improved non-dominated sorting biogeography-based optimization algorithm for the (hybrid) multi-objective flexible job-shop scheduling problem," *Applied Soft Computing*, vol. 99, article 106869, 2021.

[5] J. Q. Li, Y. Du, J. Tian, P. Y. Duan, and Q. K. Pan, "Artificial bee colony algorithm for flexible job shop scheduling with transportation resource constraints," *Acta Electronica Sinica*, vol. 49, no. 2, pp. 324–330, 2021.

[6] Z. C. Cao, C. R. Lin, and M. C. Zhou, "A knowledge-based Cuckoo search algorithm to schedule a flexible job shop with sequencing flexibility," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 1–14, 2019.

[7] D. M. Lei, M. Li, and L. Wang, "A two-phase meta-heuristic for multi-objective flexible job shop scheduling problem with Total energy consumption threshold," *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 1097–1109, 2019.

[8] K. Z. Gao, F. J. Yang, M. C. Zhou, Q. Pan, and P. N. Suganthan, "Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm," *IEEE Transactions on Cybernetics*, vol. 49, no. 5, pp. 1944–1955, 2019.

[9] X. Gong, T. D. Pessemier, and L. Martens, "Energy- and labor-aware flexible job shop scheduling under dynamic electricity pricing: a many-objective optimization investigation," *Journal of Cleaner Production*, vol. 209, pp. 1078–1094, 2019.

[10] H. T. Yang, Y. H. Fei, and Q. F. Chen, "Dynamic scheduling of flexible job shop based on industrial big data," *Computer Integrated Manufacturing System*, vol. 26, no. 9, pp. 2497–2510, 2020.

[11] X. L. Wu and X. J. Liu, "Differential evolution algorithm for solving distributed flexible job shop scheduling problem," *Computer Integrated Manufacturing System*, vol. 25, no. 10, pp. 2539–2558, 2019.

[12] X. Deng, X. B. Hu, and D. Y. Jiang, "Flexible job shop machine and AGV planning based on hybrid genetic algorithm," *Journal of Sichuan University (Natural Science Edition)*, vol. 58, no. 2, pp. 73–82, 2021.

[13] Y. Yang, M. Huang, and Z. Y. Wang, "Robust scheduling based on extreme learning machine for bi-objective flexible job-shop problems with machine breakdowns," *Expert Systems with Applications*, vol. 158, pp. 113545–113556, 2020.

[14] J. Q. Li, J. W. Deng, C. Y. Li et al., "An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times," *Knowledge-Based Systems*, vol. 200, no. 3, article 106032, 2020.

[15] W. Zhao, K. Wang, and A. D. Xu, "The health assessment method of industrial robots for intelligent manufacturing," *Robot*, vol. 42, no. 4, pp. 460–468, 2020.

[16] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 29, no. 3, pp. 603–615, 2018.

[17] F. M. Defersha and D. Rooyani, "An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time," *Computers & Industrial Engineering*, vol. 147, article 106605, 2020.

[18] B. B. Wu, H. L. Zhang, and C. Wang, "State transition algorithm based on normal cloud model to solve multi-objective flexible job shop scheduling problem," *Control and Decision*, vol. 36, no. 5, pp. 1181–1190, 2021.

[19] L. L. Meng, C. Y. Zhang, X. Y. Shao, and Y. Ren, "MILP models for energy-aware flexible job shop scheduling problem," *Journal of Cleaner Production*, vol. 210, no. 10, pp. 710–723, 2019.

[20] X. B. Pei, R. Zhang, and X. Y. Yu, "Hybrid firefly algorithm for solving multi-objective replacement flow shop scheduling problem," *Information and Control*, vol. 49, no. 4, pp. 478–488, 2020.

[21] A. Baykasolu, F. S. Madenolu, and A. Hamzaday, "Greedy randomized adaptive search for dynamic flexible job-shop scheduling," *Journal of Manufacturing Systems*, vol. 56, pp. 425–451, 2020.

[22] S. C. Zhang, X. Li, B. W. Zhang, and S. Wang, "Multi-objective optimisation in flexible assembly job shop scheduling using a distributed ant colony system," *European Journal of Operational Research*, vol. 283, no. 2, pp. 441–460, 2020.

[23] Z. H. Jia, Y. Wang, and Y. W. Zhang, "A bi-objective synergy optimization algorithm of ant colony for scheduling on non-identical parallel batch machines," *Acta Automatica Sinica*, vol. 46, no. 6, pp. 1121–1135, 2020.

[24] Y. R. Chi, J. J. Liu, and Q. X. Chen, "Lookahead batching heuristic for batch scheduling problem of two-stage hybrid flow shop," *Computer Integrated Manufacturing System*, vol. 25, no. 10, pp. 2559–2570, 2019.

[25] G. S. Wang, J. Y. Liu, and L. X. Tang, "Branch-and-price algorithm for rolling batch scheduling problem in continuous-casting and rolling processes with hybrid production mode," *Acta Automatica Sinica*, vol. 43, no. 7, pp. 1178–1189, 2017.

[26] R. Q. Li, Z. Y. Tan, and Q. Y. Zhu, "Batch scheduling of non-identical job sizes with minsum criteria," *Journal of Combinatorial Optimization*, vol. 26, no. 2, pp. 165–179, 2019.

[27] M. Tan, H. L. Yang, and Y. X. Su, "Genetic algorithms with greedy strategy for green batch scheduling on nonidentical parallel machines," *Memetic Computing*, vol. 11, no. 4, pp. 439–452, 2019.

[28] J. Q. Wang, G. Q. Fan, and Z. Liu, "Mixed batch scheduling on identical machines," *Journal of Scheduling*, vol. 23, no. 4, pp. 487–496, 2020.

[29] O. Shahvari and R. Logendran, "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*, vol. 195, no. 2, pp. 227–248, 2018.

[30] J. T. Huang, J. J. Liu, Q. X. Chen, and N. Mao, "Batch scheduling algorithm for flexible flow workshop of incompatible job family," *Mechanical Design and Manufacturing*, vol. 6, pp. 75–77, 2016.

[31] S. C. Zhou, *Research on Several Problems of Machine Batch Scheduling for Differential Jobs*, University of Science and Technology of China, 2016.

[32] Q. Zhu, C. D. Chen, and H. P. Chen, "Solution of batch scheduling problem in flow shop with different jobs," *Computer Engineering and Applications*, vol. 49, no. 13, pp. 221–227, 2013.

[33] M. Yin, S. Wang, J. Zhang, and Y. S. Zou, "Research on batch scheduling and solution method of large-scale flexible job shop," *Mechanical Design and Manufacturing*, vol. 6, pp. 32–34, 2020.

[34] L. Gao, G. H. Zhang, and X. J. Wang, *Intelligent Algorithm of Flexible Job Shop Scheduling and Its Application*, Huazhong University of Science and Technology Press, Wuhan, China, 2012.

[35] Z. H. Han, C. Han, S. Lin, Dong, and Shi, "Flexible flow shop scheduling method with public buffer," *Processes*, vol. 7, no. 10, p. 681, 2019.