COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 04/06 pp 156–173
Volume 2, Number 2, June 2022
DOI: 10.23919/CSMS.2022.0010

Distributed Flexible Job-Shop Scheduling Problem Based on Hybrid Chemical Reaction Optimization Algorithm

Jialei Li, Xingsheng Gu*, Yaya Zhang, and Xin Zhou

Abstract: Economic globalization has transformed many manufacturing enterprises from a single-plant production mode to a multi-plant cooperative production mode. The distributed flexible job-shop scheduling problem (DFJSP) has become a research hot topic in the field of scheduling because its production is closer to reality. The research of DFJSP is of great significance to the organization and management of actual production process. To solve the heterogeneous DFJSP with minimal completion time, a hybrid chemical reaction optimization (HCRO) algorithm is proposed in this paper. Firstly, a novel encoding-decoding method for flexible manufacturing unit (FMU) is designed. Secondly, half of initial populations are generated by scheduling rule. Combined with the new solution acceptance method of simulated annealing (SA) algorithm, an improved method of critical-FMU is designed to improve the global and local search ability of the algorithm. Finally, the elitist selection strategy and the orthogonal experimental method are introduced to the algorithm to improve the convergence speed and optimize the algorithm parameters. In the experimental part, the effectiveness of the simulated annealing algorithm and the critical-FMU refinement methods is firstly verified. Secondly, in the comparison with other existing algorithms, the proposed optimal scheduling algorithm is not only effective in homogeneous FMUs examples, but also superior to existing algorithms in heterogeneous FMUs arithmetic cases.

Key words: scheduling problem; distributed flexible job-shop; chemical reaction optimization algorithm; heterogeneous factory; simulated annealing algorithm

1 Introduction

For a long time, flexible job-shop scheduling problem (FJSP) is not only a hot topic in academic research, but also plays an important role in modern manufacturing industries. The FJSP is an extension of job-shop scheduling problem (JSP), which is NP-hard problem and it has been widely studied in recent decades[1–7]. However, many manufacturing industries are changing from a traditional centralized manufacturing model to a distributed manufacturing model because of the emergence of globalization. The manufacturing of products is completed by multiple companies or factories located in different regions instead of only one company or factory because cooperative production between different factories can reduce response speed and production cost[8–12]. Therefore, the distributed flexible job-shop scheduling problem (DFJSP) has been gradually attracted the attention of scholars and applied in many fields, such as petroleum, chemical, metallurgy, steel, textile, and pharmaceutical industries[13].

The DFJSP solves the problem of production scheduling in distributed manufacturing environments, where tasks are handled cooperatively by several flexible manufacturing units (FMUs). And each

• Jialei Li, Xingsheng Gu, Yaya Zhang, and Xin Zhou are with the Laboratory of Smart Manufacturing in Energy Chemical Process, Ministry of Education, East China University of Science and Technology, Shanghai 200237, China. E-mail: xsgu@ecust.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2022-03-29; revised: 2022-06-08; accepted: 2022-06-20

operation can be handled by one or more available machines. DFJSP can be divided into homogeneous DFJSP and heterogeneous DFJSP. Under the homogeneous FMU environment, the number and performance of machines in each FMU are the same. While in heterogeneous FMU environment, the number and performance of machines in each FMU are different. In DFJSP, the three sub-problems of FMU selection, operation sequence scheduling, and machine selection must be solved, which is more difficult than FJSP.

To date, some literatures have been published on DFJSP. Among them, the study with makespan as the objective function is the most. Chan et al.[14] designed an encoding method with dominant genes (DG) which called genetic algorithm with dominant genes (GADG) to deal with DFJSP. Giovanni and Pezzella[15] extended some classic FJSP instances to DFJSP and proposed an improved genetic algorithm (IGA). Ziaee[16] developed a fast heuristic algorithm, which assigns different weights to factors such as machine limit time, average job time, and process processing time, so that the algorithm can quickly locate the region with better feasible solutions. To balance the load of the factory and the machine in the DFJSP, Lu et al.[17] proposed a concise encoding method and corresponding 3-D decoding method, which named GA-JS. The experiment was compared with IGA and some better optimal solutions were obtained. Chang and Liu[18] proposed a hybrid genetic algorithm (HGA) to solve the DFJSP. The authors used three methods for the crossover phase and divided the mutation operation into two parts. In comparison with GADG and IGA, HGA obtained better results in the mean and standard deviation. Compared with the above-mentioned existing genetic encoding methods, Wu et al.[19] developed a new encoding method for genetic algorithm, which called GA-OP, and designed corresponding decoding methods for factory allocation and machine allocation. Marzouki et al.[20] used chemical reaction optimization (CRO) to solve DFJSP. However, due to the use of basic CRO, the experimental results are not competitive with other algorithms. To solve the DFJSPs subject to preventive maintenance (PM), Chan et al.[21] developed a genetic algorithm with dominant genes (GADG) to identify chromosomes with good genes. Based on GADG, Chung et al.[22] added a local search mechanism and proposed an enhanced genetic algorithm (GA). Lin et al.[23] designed two incomplete chromosome representations and four GAs to solve DFJSP. Furthermore, an effective method to generate new chromosomes from high quality solutions was developed to improve the performance of the proposed algorithms. Considering the transportation time between factories, Chan et al.[24] proposed a hybrid algorithm based on tabu search and sample sort simulated annealing to solve DFJSP. Ziaee[25] integrated production scheduling and work-in-process planning decisions, established a corresponding mixed integer linear programming model, and developed a fast heuristic algorithm. To solve multi-objective DFJSPs, Li et al.[13] constructed a hybrid Pareto-based tabu search algorithm to deal with multi-objective DFJSP in steelmaking systems, and developed five types of neighbourhood structures to improve the algorithm ability. Luo et al.[26] proposed a model of DFJSP with transfers time and designed an efficient memetic algorithm to solve multi-objective DFJSP. In this algorithm, multiple crossover mutation operators and three kinds of neighbourhood structures are designed to expand the search space. Xu et al.[27] developed a hybrid genetic algorithm and tabu search with three-layer encoding to address multi-objective low carbon DFJSP of large-scale and complex manufacturing enterprises. Du et al.[28] combined a distribution estimation algorithm and variable neighbourhood search to solve multi-objective DFJSP with crane transportations.

In summary, the existing researches on DFJSP mainly focus on the homogeneous FMUs where the number and the processing capacity of machines in each FMU are assumed to be the same, and rarely pay attention to the DFJSP on the heterogeneous FMUs. Moreover, the experiments are mostly verified in small examples. The largest one of these examples contains only 20 jobs. But in the actual production, most of the FMUs are heterogeneous and the scale of DFJSP is large. Therefore, a new algorithm with strong applicability is proposed in this paper to be able to solve homogeneous and heterogeneous DFJSP at the same time, and its effectiveness is verified on large-scale examples.

In recent years, various intelligent optimization algorithms have been proposed to solve different production scheduling problem. Zhao et al.[29] proposed an ensemble discrete differential evolution algorithm to solve the blocking flow-shop scheduling problem with the minimization of the makespan in the distributed manufacturing environment. To solve the multi-objective energy-efficient no-wait flow shop scheduling problem and the multi-objective energy-efficient distributed no-idle flow shop scheduling problem, Zhao et al. designed a two-stage cooperative evolutionary algorithm with problem-specific knowledge[30] and a self-learning discrete Jaya algorithm[31], respectively. Wang et al.[32] used the multi-objective whale algorithm to study the energy-efficient distributed permutation flow shop scheduling problem with sequence dependent setup times. Lei et al. proposed an imperialist competition algorithm with memory[33] and an artificial bee colony algorithm with partitioning properties[34] to solve the distributed unrelated parallel machines scheduling problem, respectively. Şahman[35] employed discrete spotted hyena optimization algorithm to solve distributed job shop scheduling problem.

The above-mentioned researchers try to solve the scheduling problem with different intelligent optimization algorithms to expect close to the optimal solution. The experimental results also show that some new algorithms can obtain better solutions than other algorithms. It is very necessary to develop different intelligent optimization algorithms to solve different scheduling problems. Therefore, a hybrid chemical reaction optimization algorithm is proposed in this paper, which is suitable for solving DFJSP with the goal of minimizing the maximum makepan in both homogeneous and heterogeneous environments. The main reasons for using the chemical reaction optimization (CRO) algorithm to solve DFJSP in this paper are as follows. First, the CRO has been widely used to solve different combinatorial optimization problems, such as the resource constrained project scheduling problem[36], directed acyclic graph scheduling[37], the 0–1 knapsack problem[38], the FJSP[39, 40] , and distributed flow shop scheduling problem[41, 42] , but CRO is less used in DFJSP. Furthermore, evolutionary algorithms can explore a huge search space, but their ability to converge to the optimal solution is poor, while CRO can balance diversified search and intensive search. Therefore, CRO has good convergence and can obtain feasible solutions in a short time. Finally, CRO can adapt to different optimization problems by utilizing the defined molecular representation and the basic four reaction operations. This variety of operators helps us tailor algorithm to suit different problems. Based on the above advantages, it is considered that CRO can satisfactorily improve DFJSP based on the makespan criterion.

The main contributions of this work are as follows. (1) An operation-FMU encoding-decoding method is designed. (2) Corresponding operations are designed for four kinds of collisions of the algorithm. (3) The CRO algorithm is improved by combining with simulated annealing (SA) and critical-FMU refinement methods. In addition, orthogonal experiments are used for exploring the influence of parameters and the comparison with existing algorithms in homogeneous FMU cases and self-generated heterogeneous FMU cases are completed in the experimental part which verify the effectiveness and superiority of the proposed algorithm.

The remaining of this paper is organized as follows. The description of DFJSP is stated in Section 2. In Section 3, the proposed algorithm is described in detail, including basic chemical reaction optimization algorithm, encoding and decoding methods, four collision operators, SA new solution acceptance method, and critical-FMU refinement methods. The related parameters are designed in the anterior part of Section 4, and the experimental results are analysed in the posterior part of Section 4. Finally, the conclusions and future work are provided in Section 5.

2 Problem Description

For convenience, the notations used in this section are listed as follows.

Index

i index for job, $i = 1, 2, \ldots, N$;

j index for operation, $j = 1, 2, \ldots, p_i$;

f index for FMU, $f = 1, 2, \ldots, Q$;

k index for machine, $k = 1, 2, \ldots, mf$;

r index for processing sequence, $r = 1, 2, \ldots, q_{f,k}$.

Parameter

N total number of jobs;

p_i total number of operations for job i;

Q total number of FMUs;

mf total number of machines in FMU_f;

$t_{i,j}^{f,k}$ processing time of operation O_{ij} on machine $M_{f,k}$.

Variable

J_i the i-th job;

O_{ij} the j-th operation of job J_i;

FMU_f the f-th FMU;

$M_{f,k}$ the k-th machine in FMU_f;

$S_{i,j}$ starting time of operation O_{ij};

$D_{i,j}$ ending time of operation O_{ij};

F_i completion time of job i;

$C_{\max}$ maximum completion time;

$E_{f,k,r}$ starting time of the r-th processing sequence on machine k in FMU_f;

$q_{f,k}$ total number of operations processed on machine k in FMU_f.

Binary variable

$x_{i,j}^{f,k}$ binary variable that takes value 1 if O_{ij} is processed on machine k in FMU_f, and 0 otherwise.

$z_{i,j}^{f,k,r}$ binary variable that takes value 1 if the processing sequence of O_{ij} on machine k in FMU_f is r, and 0 otherwise.

The DFJSP can be stated as follows. A set of jobs $J = \{J_1, J_2, \ldots, J_N\}$ is given, which must be processed in a set of FMUs $FMU = \{FMU_1, FMU_2, \ldots, FMU_Q\}$. FMU_f is equipped with a set of machines $M = \{M_1, M_2, \ldots, M_{mf}\}$. It is worth noting that the number of machines in each FMU is not the same and each machine k exhibits a different level of performance. Each job J_i has an ordered set of operations $O_{ij} = \{O_{i1}, O_{i2}, \ldots, O_{ipi}\}$, each operation can be assigned only to one flexible manufacturing unit FMU_f for processing. The system schematic diagram of DFJSP is shown in Fig. 1.

The paper considers the following assumptions:

(1) All flexible manufacturing units, jobs, and machines are available at time zero.

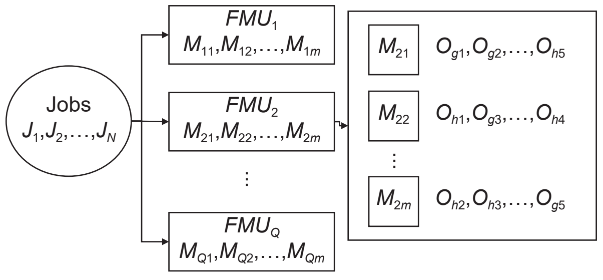
(2) The machining time of all operations is known in advance.

(3) Each operation that is processed by different machines may have different time.

(4) No consideration is given to transportation time, preparation time, release time, etc.

Based on the above assumptions, the model with the optimization objective of minimizing makespan is established as follows by referring to Ref. [43]:

$$\min\ C_{\max} = \max_{i=1}^{N}\{F_i\} \tag{1}$$



Jobs
$J_1, J_2, \ldots, J_N$
FMU_1
$M_{11}, M_{12}, \ldots, M_{1m}$
FMU_2
$M_{21}, M_{22}, \ldots, M_{2m}$
FMU_Q
$M_{Q1}, M_{Q2}, \ldots, M_{Qm}$
M_{21}
$O_{g1}, O_{g2}, \ldots, O_{h5}$
M_{22}
$O_{h1}, O_{g3}, \ldots, O_{h4}$
M_{2m}
$O_{h2}, O_{h3}, \ldots, O_{g5}$

Fig. 1 DFJSP system diagram.

$$F_i = D_{i,p_i} \tag{2}$$

$$D_{i,j} = S_{i,j} + \sum_{f=1}^{Q}\sum_{k=1}^{mf} x_{i,j}^{f,k} \times t_{i,j}^{f,k}, j \in \{1, 2, \ldots, p_i\} \tag{3}$$

$$S_{i,j+1} \geqslant D_{i,j}, j \in \{1, 2, \ldots, p_i - 1\} \tag{4}$$

$$E_{f,k,r+1} \geqslant E_{f,k,r} + \sum_{i=1}^{N}\sum_{j=1}^{p_i} t_{i,j}^{f,k} \times z_{i,j}^{f,k,r},$$
$$f \in \{1, 2, \ldots, Q\}, k \in \{1, 2, \ldots, mf\}, r \in \{1, 2, \ldots, q_{f,k} - 1\} \tag{5}$$

$$\sum_{j=1}^{p_i}\sum_{k=1}^{mf} x_{i,j}^{f,k} \in \{0, p_i\} \tag{6}$$

$$\sum_{f=1}^{Q}\sum_{k=1}^{mf} x_{i,j}^{f,k} = 1, j \in \{1, 2, \ldots, p_i\} \tag{7}$$

$$\sum_{r=1}^{q_{f,k}} z_{i,j}^{f,k,r} = x_{i,j}^{f,k}, j \in \{1, 2, \ldots, p_i\} \tag{8}$$

Equation (1) is the objective function, which means that the makespan is equal to the maximum completion time of all jobs. Equation (2) indicates that the completion time of the job is equal to the completion time of the last operation of this job. Equation (3) states that the completion time of the operation is equal to the starting processing time plus the actual processing time. Formula (4) is the operation constraint, meaning that the operations of the same job must be processed in order. Formula (5) ensures that the same machine can only process an operation at a time. Formula (6) represents that all operations of a job can only be processed in the same FMU. Equation (7) assures that an operation can only be processed by one machine in one FMU. Equation (8) defines that an operation can only be processed once by the selected processing machine.

3 Hybrid Chemical Reaction Optimization Algorithm for DFJSP

In this part, the specific content of the proposed algorithm is introduced with details. For convenience, the parameters involved are shown in Table 1.

The algorithm of HCRO is shown in Algorithm 1. Basic chemical reaction optimization algorithm is introduced in Section 3.1; Molecular encoding and decoding method is presented in Section 3.2; The population initialization is stated in Section 3.3; Four collisions are presented in Sections 3.4–3.7; Elitist

Table 1 Parameters of hybrid chemical reaction optimization (HCRO) algorithm.

Parameter	Description
ω	Molecular structure and solution of the problem
PE_ω	Potential energy of ω
KE_ω	Kinetic energy of ω
buffer	Central energy buffer, and initial buffer is 0.
InitialKE	Initial kinetic energy
MoleColl	A random number between 0 and 1 to control the occurrence of a uni-molecular or inter-molecular collision
a	Control parameter of decomposition reaction
b	The lower kinetic energy limit
p	Control parameter of critical-FMU refinement methods
T_0	Initial temperature
T_f	Threshold temperature
Beta	Value of attenuation factor
Popsize	Molecular population size
Maxgen	Maximum number of iterations

Algorithm 1 Hybrid chemical reaction optimization algorithm

```
1.Set parameters: Popsize, Maxgen, InitialKE, MoleColl, b, a, p, $T_0$, $T_f$, and Beta
2.Population initialization;
3.While the number of iterations does not reach Maxgen or the running time is less than 5000 s do
4.  Select molecules for optimal operations according to the elitist selection strategy;
5.  while go through the rest of the molecules do
6.    if rand(0,1) > MoleColl then
7.      if the optimal value of an iteration is not updated then
8.        Trigger decomposition reaction and accept the new molecule by simulated annealing algorithm;
9.      else
10.       Trigger on-wall ineffective collision and accept the new molecule by simulated annealing algorithm;
11.     end
12.   else
13.     if the kinetic energies of two molecules are all less than b then
14.       Trigger inter-molecular ineffective collision and accept the new molecule by simulated annealing algorithm;
15.     else
16.       Trigger synthesis reaction and accept the new molecule by simulated annealing algorithm;
17.     end
18.   end
19. end
20. Apply critical-FMU refinement methods to molecules with the top 20% fitness value;
21. end
```

selection strategy is introduced in Section 3.8; A new solution acceptance method based on simulated annealing algorithm and critical-FMU refinement methods are introduced in Sections 3.9 and 3.10.

3.1 Basic chemical reaction optimization algorithm

CRO is an emerging population evolution algorithm that appeared in 2012, which simulates the process of constantly changing molecules in a chemical reaction system trying to obtain the lowest potential energy[36]. The algorithm uses the concept of molecular structure ω to describe the solution of the optimization problem and describes the optimization process as a chemical reaction process. The molecular structure ω represents a feasible solution, and each molecule has two types of energies: potential energy PE_ω and kinetic energy KE_ω. PE_ω represents the objective function value of the corresponding solution ω while KE_ω represents the tolerance of the system to accept a worse solution. In this paper, KE of the new molecule is defined as the KE and PE of the old molecule minus the PE of the new molecules[44]. The specific formula expression will be listed in the following four collisions. CRO defines four basic reaction operators to search for feasible solutions: on-wall ineffective collision, decomposition reaction, inter-molecular ineffective collision, and synthesis reaction. These four reactions can be classified into uni-molecular collisions and bi-molecular collisions. The on-wall ineffective collision and decomposition reaction are single molecular collisions, while inter-molecular ineffective collision and synthesis reaction are of the second category.

Like other evolutionary algorithms, CRO contains three basic steps: initialization, iteration, and termination conditions. During the first step, *Popsize*, *MoleColl*, *InitialKE*, *buffer*, a, and b are initialized. In the iteration phase, CRO explores the solution space through four different collision reactions. The molecular changes caused by these collisions can be subtle or dramatic. Slight collision focuses on intensive search, which refers to searching the neighborhood of the current solution to improve the quality of the algorithm. In CRO, it is realized by on-wall ineffective collision and inter-molecular ineffective collision. While violent collisions focus on intensive search, they tend to search for different regions of the solution space through decomposition and synthesis reactions. The CRO algorithm will go through these four basic reactions until the termination conditions are met and the optimal solution is output.

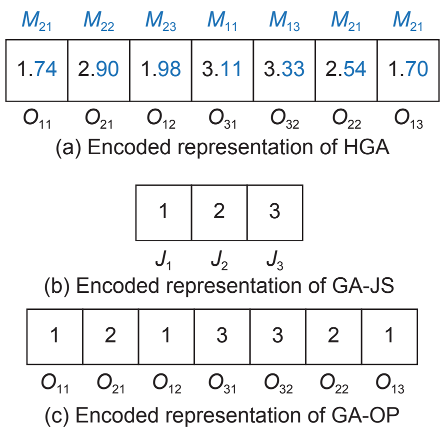
3.2 Molecular encoding and decoding

This section takes the data in Table 2 as an example and lists three relatively new encoding methods of DFJSP with $C_{\max}$ as the optimization goal, which are selected from HGA[18], GA-JS[17], and GA-OP[19], to explain the advantages of the encoding method proposed in this paper.

HGA explicitly expresses the three sub-problems of DFJSP using the encoding method shown in Fig. 2a. The integer part represents the sequence order of the operations, and the process sequence shown in Fig. 2a is 1213321; the decimal part uses the roulette wheel method to determine the machine selection, and the corresponding FMU selection is determined through the machine selection. This method does not need to use rules to decode and can search the whole solution space well. However, the FMU selection is determined by machine selection; DFJSP considers the constraint that operations of the same job are processed in the same FMU, so it is easy to generate illegal solutions, which will increase the running time of the algorithm to a certain extent.

Table 2 A sample DFJSP instance.

Job	Operation	FMU_1			FMU_2		
		M_{11}	M_{12}	M_{13}	M_{21}	M_{22}	M_{23}
J_1	O_{11}	2	1	3	3	–	2
	O_{12}	–	5	1	3	3	3
	O_{13}	3	5	1	2	1	–
J_2	O_{21}	4	6	2	5	4	–
	O_{22}	3	2	7	5	4	3
J_3	O_{31}	3	2	4	3	5	4
	O_{32}	–	3	4	5	3	4



M21 M22 M23 M11 M13 M21 M21
1.74 2.90 1.98 3.11 3.33 2.54 1.70
O11 O21 O12 O31 O32 O22 O13
(a) Encoded representation of HGA
1 2 3
J1 J2 J3
(b) Encoded representation of GA-JS
1 2 1 3 3 2 1
O11 O21 O12 O31 O32 O22 O13
(c) Encoded representation of GA-OP

Fig. 2 Three encoding methods of DFJSP.

The encoding of GA-JS only expresses the job processing sequence and decodes three sub-problems of DFJSP through three heuristic rules. As shown in Fig. 2b, this encoding method is simple, but too many heuristic rules can easily make the algorithm fall into a local optimum.

As shown in Fig. 2c, the encoding method of GA-OP explicitly expresses the sequence order of the operations, and the remaining machine selection and FMU selection are obtained by two heuristic rules to balance the load. The FMU selection of this method is greatly affected by the number of machines in the FMUs and is not suitable for DFJSP in a heterogeneous environment.

To overcome the above-mentioned shortcomings of encoding, we propose a novel operation-FMU encoding method to explicitly express the two sub-problems of DFJSP.

Each molecule ω has two parts of information. The first part of the molecule is the operation information, which adopts from the JSP solution encoding method proposed by Bierwirth[45]. The second part is the FMU information, whose length is N, indicating the FMU number of the corresponding job. Figure 3 shows a molecule ω based on Table 2. The right side of the dotted line is the FMU information. "2" in the first position means that job J_1 is allocated to FMU_2, "2" in the second position means that job J_2 is allocated to FMU_2, "1" in the third position means that job J_3 is allocated to FMU_1, and so on. This encoding method does not generate illegal solutions, which can save the time to check illegal solutions. The FMU selection is reflected in the encoding, so that the algorithm has strong applicability and can solve DFJSP in homogeneous and heterogeneous FMU environments at the same time.

DFJSP needs to solve three sub-problems, among which the processing order and FMU allocation can be obtained by the corresponding decoding of the above ω. Take the problem in Table 2 and the molecule representation in Fig. 3 as an example: J_1 and J_2 are assigned to FMU_2. Therefore, the operation sequence in FMU_2 (called C_{op2}) is $O_{11}\rightarrow O_{21}\rightarrow O_{12}\rightarrow O_{22}\rightarrow O_{13}$;

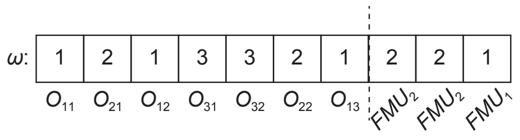


Fig. 3 Molecule representation.

and the operation sequence in FMU_1 (called C_{op1}) is O_{31}→O_{32} because J_3 is assigned to FMU_1. For the rest of the machine allocation, a heuristic rule is used for decoding to balance the FMU load to reduce makespan. Take the FMU_2 as an example, the specific operation of the rule is as follows: All C_{op2} operations are assigned to the FMU_2 machines in turn. Each operation is allocated to the machines with the lowest workload. O_{11} can be processed on M_{21} and M_{23}. M_{23} is selected since it has a lower workload, and the workload of each machine is then updated. O_{21} is allocated to M_{22} because of its lower workload, and the workload of each machine is updated again. If the same workload is obtained by several machines, then one of them will be selected randomly. Follow the above process to complete the process of machine selection, the final scheduling result can be obtained.

3.3 Population initialization

The initial population is crucial for an algorithm since it has influence on the following iterations[46]. To balance the load between FMUs and shorten $C_{\max}$ as much as possible, heuristic rules are developed to generate half of the initial population in this paper, while the remaining half of the population is randomly generated to maintain diversity.

Heuristic rule 1

This rule applies to the operation information in the first part of the molecule. Priority is given to the jobs with the most remaining operations. Once there exist some jobs with the same number of remaining operations, one of them is selected randomly.

Take the data in Table 2 as an example again, there are 3 operations left for J_1, 2 operations left for J_2, and 3 operations left for J_3. J_1 has the most remaining operations, so the operation of J_1 is processed first. After that, J_1, J_2, and J_3 are all remaining 2 operations, and one of them is randomly selected for processing and so on. One such operation sequence O_{11}→O_{21}→O_{12}→O_{31}→O_{32}→O_{22}→O_{13} can be generated as the first part of the molecule.

Heuristic rule 2

This rule applies to the FMU information in the second part of the molecule, which calculates the average processing time of each job in each FMU firstly. And then, the FMU with the lowest average processing time is preferred. If there are some FMUs with the same average processing time, one of them is selected randomly.

Take the data in Table 2 as an example. Firstly, calculate the average processing time of each job in each FMU. The calculation formula is as follows:

$$avg_i^f = \sum_j \overline{avg_{i,j}^f},$$

where $\overline{avg_{i,j}^f} = \sum_k t_{i,j}^{f,k}/mf$ denotes the average processing time of operation O_{ij} in FMU_f. Each element in Table 3 can be accordingly obtained.

Secondly, the FMU allocations are completed. The average processing time of J_1 in FMU_2 is the least, so the first place of the second part of molecule is “2”; J_2 has the lowest average processing time in FMU_1, so the second place is “1”; J_3 also has less average processing time in FMU_1, so the third place is “1”.

So, the resulting molecule from the heuristic rules is 1213321211.

3.4 On-wall ineffective collision

On-wall ineffective collision is a process in which one molecule ω produces another molecule ω'. Since this collision does not change the molecular structure greatly, this paper adopts the method of exchanging two coded positions randomly to realize the on-wall ineffective collision. Specific operations are as follows.

Step 1: If the on-wall ineffective collision conditions are met, molecule ω is randomly selected. Random numbers r_1, $r_2 \in [1, l_1]$ and r_3, $r_4 \in [l_1+1, l_1+l_2]$ are generated for the operation part and FMU part of the molecular, and $r_1 \neq r_2$, $r_3 \neq r_4$.

Step 2: Swap the element in position r_1 and r_2, r_3 and r_4. If r_1=3, r_2=5; r_3=8, r_4=10, the process of on-wall ineffective collision is shown in Fig. 4.

Step 3: If the new molecule is accepted, the old molecule is replaced by new molecule in order. During this collision, the lost energy is stored in the *buffer*.

Table 3 Average processing time of each job in each FMU.

Job	Average processing time	
	FMU_1	FMU_2
J_1	8.0	**6.0**
J_2	**8.0**	8.5
J_3	**6.5**	8.0

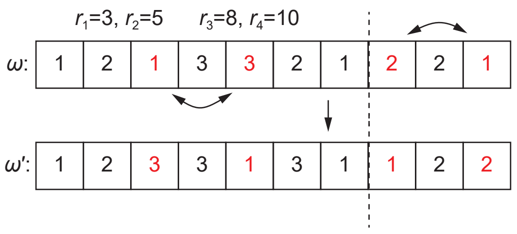


Fig. 4 On-wall ineffective collision diagram.

And the *KE* of the ω' and *buffer* are updated by Eqs. (9) and (10)[44].

$$KE_{\omega'} = KE_{\omega} + PE_{\omega} - PE_{\omega'} \tag{9}$$

$$buffer = buffer + KE_{\omega} + PE_{\omega} - KE_{\omega'} - PE_{\omega'} \tag{10}$$

3.5 Decomposition reaction

The decomposition reaction is the decomposition of one molecule ω into ω_1' and ω_2' molecules. If the algorithm cannot update the current optimal solution after iterating "a" times, it indicates that the algorithm may be trapped in a local optimum. In this paper, cyclic movement operation[36] is used to realize the decomposition reaction. The specific steps are as follows.

Step 1: If the decomposition conditions are met, molecule ω is randomly selected. Random numbers $r_1 \in [-l_1, l_1]$ and $r_2 \in [-l_2, l_2]$ are generated for the operation part and the FMU part of the molecule, respectively, where l_1 and l_2 are the encoding lengths of these two parts. Take Fig. 3 for example, l_1=7 and l_2=3.

Step 2: If $r_1 < 0$, the former r_1 elements in the encoding are moved to the end; if $r_1 \geqslant 0$, the last r_1 elements are moved to the front of the encoding. The judgment and operation for r_2 are the same as for r_1. If r_1=r_2=−1 and r_1=r_2=2, the process of decomposition reaction is shown in Fig. 5.

Step 3: If the new molecule is accepted, one of the new molecules is selected at random to replace the old molecules in order. Since decomposition produces two molecules from one molecule, the *KE* and *PE* of the old molecule may not be enough to produce two molecules. So, the energy in the *buffer* is used for facilitating the reaction. And the kinetic energy of the ω_1' and ω_2' are updated by Eqs. (11) and (12)[44].

$$KE_{\omega_1'} = buffer + KE_{\omega} + PE_{\omega} - PE_{\omega_1'} - PE_{\omega_2'} \tag{11}$$

$$KE_{\omega_2'} = buffer + KE_{\omega} + PE_{\omega} - PE_{\omega_1'} - PE_{\omega_2'} \tag{12}$$

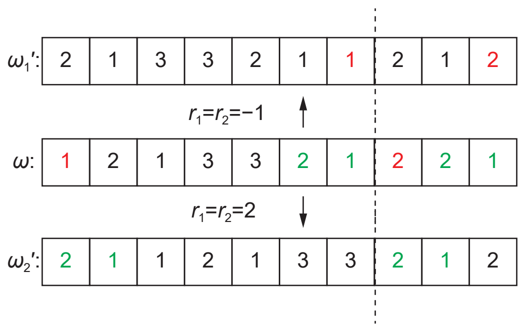


Fig. 5 Decomposition reaction diagram.

3.6 Inter-molecular ineffective collision

The inter-molecular ineffective collision is a process in which two molecules ω_1 and ω_2 generate ω_1' and ω_2'. The specific steps are listed as follows.

Step 1: If the inter-molecular ineffective collision is met, two different molecules are selected randomly. $J = \{J_1, J_2, \ldots, J_N\}$ is divided into two sets Job_1 and Job_2 randomly; $FMU = \{FMU_1, FMU_2, \ldots, FMU_Q\}$ is divided into two sets Fmu_1 and Fmu_2 randomly.

Step 2: The elements in ω_1 that belong to the sets Job_1 and Fmu_1 are retained in ω_1' directly and remain in their original positions. Similarly, elements in ω_2 that belong to the sets Job_1 and Fmu_1 are retained in ω_2' and remain in their original positions. Take Fig. 3 as an example, $Job_1 = \{1\}$ and $Fmu_1 = \{1\}$, so element "1" in two old molecules is retained directly and remain in original positions on the new molecules.

Step 3: The elements belonging to the sets Job_2 and Fmu_2 in ω_2 are filled into the gaps in ω_1' in order. As shown in Fig. 6, $Job_2 = \{2, 3\}$ and $Fmu_2 = \{2\}$, so the elements belonging to the sets Job_2 and Fmu_2 in ω_2 in order is 223322. And then these elements are filled into the gaps in ω_1' in order. Similarly, the elements in ω_1 that belong to the sets Job_2 and Fmu_1 are filled into the ω_2 space in turn.

Step 4: If the new molecules are accepted, the old molecules are replaced by new molecules in order. And the kinetic energy of the ω_1' and ω_2' are updated by Eqs. (13) and (14)[44].

$$KE_{\omega_1'} = KE_{\omega_1} + KE_{\omega_2} + PE_{\omega_1} + PE_{\omega_2} - PE_{\omega_1'} - PE_{\omega_2'} \tag{13}$$

$$KE_{\omega_2'} = KE_{\omega_1} + KE_{\omega_2} + PE_{\omega_1} + PE_{\omega_2} - PE_{\omega_1'} - PE_{\omega_2'} \tag{14}$$

3.7 Synthesis reaction

The synthesis reaction is a process in which two

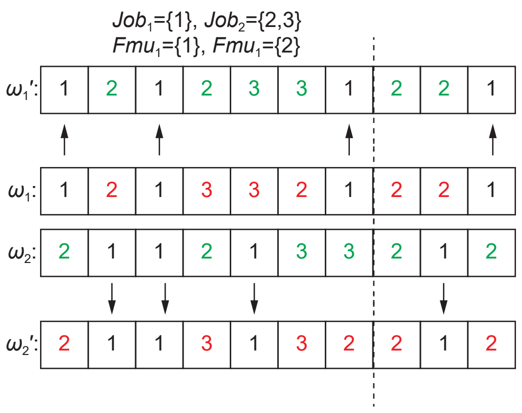


Fig. 6 Inter-molecular ineffective collision diagram.

molecules ω_1 and ω_2 are synthesized into the molecule ω'. If $KE_{\omega_1} \leqslant b$ and $KE_{\omega_2} \leqslant b$, which means that both ω_1 and ω_2 have a little kinetic energy, and the probability of collisions is very small. So synthetic reaction takes place to diversify the solutions. In this paper, a distance keeping crossover operation[36] is used to realize the synthesis reaction. The specific steps are as follows.

Step 1: If the synthesis conditions are met, two different molecules are selected randomly. And the elements in two molecules are compared.

Step 2: Copy the same elements at the same location in the old molecules to the new molecule. In a reasonable range, generate elements at other locations randomly to obtain the new molecules. The process of the synthesis reaction is shown in Fig. 7.

Step 3: If the new molecule is accepted, the old molecule is replaced by the new molecule in order. And the kinetic energy of ω' is updated by the Eq. (14)[44].

$$KE_{\omega'} = KE_{\omega_1} + KE_{\omega_2} + PE_{\omega_1} + PE_{\omega_2} - PE_{\omega'} \qquad (15)$$

3.8 Elitist selection

In the CRO algorithm, molecules with high fitness can generate new molecules with low fitness[36]. This phenomenon ensures the diversity of molecule population but also reduces the convergence speed of the algorithm. Therefore, elite retention scheme is introduced to the improved algorithm. This scheme reserves 20% of the optimal old molecules directly to the next iteration without reaction operations.

3.9 A new solution acceptance method based on simulated annealing algorithm

Since the new solutions generated by the chemical reaction optimization algorithm will change the number of population. To ensure that the overall population size remains unchanged, certain methods need to be adopted to accept new solutions. The calculation process of simulated annealing algorithm[47] is simple and general, and it can help the algorithm to jump out of the local optimum to a certain extent. SA has also shown its superiority in scheduling problem[48–50]. Therefore, we choose to integrate SA into HCRO to accept the new solutions after the four reactions of HCRO. Algorithm 2 displays the detailed pseudo-code of the SA algorithm.

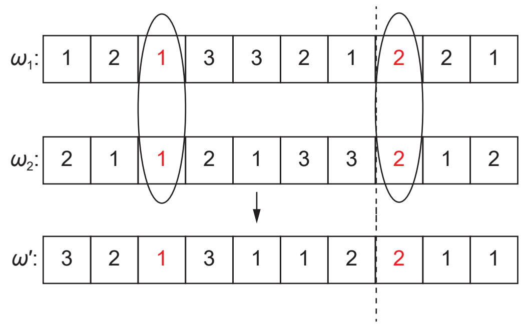


Fig. 7 Synthesis reaction diagram.

3.10 Critical-FMU refinement methods

For DFJSP, the maximum makespan in the whole production process depends on the FMU with the latest completion time, which is called critical FMU. Clearly, scheduling results can only be improved if the makespan of the critical FMU is reduced. Therefore,

Algorithm 2 A new solution acceptance method based on simulated annealing algorithm

```
1. if $T_0$>$T_f$ then
2.   if trigger on-wall ineffective collison then
3.     if $\Delta T$=$PE_{\omega'}$−$PE_\omega$ < 0 then
4.       $\omega$=$\omega'$;
5.     else if rand(0,1) < exp(−$\Delta T/T_0$) then
6.       $\omega$=$\omega'$;
7.     end if
8.   end if
9.   if trigger decomposition then
10.    if $\Delta T$=max($PE_{\omega_1'}$, $PE_{\omega_2'}$)−$PE_\omega$ < 0 then
11.      $\omega$=$\omega_1'$ or $\omega$=$\omega_2'$;
12.    else if rand(0,1) < exp (−$\Delta T/T_0$) then
13.      $\omega$=$\omega_1'$ or $\omega$=$\omega_2'$;
14.    end if
15.  end if
16. if trigger inter molecular ineffective collision then
17.    if $\Delta T$=max($PE_{\omega_1'}$, $PE_{\omega_2'}$)−max($PE_{\omega_1'}$, $PE_{\omega_2'}$)$PE_\omega$ < 0 then
18.      $\omega_1$=$\omega_1'$ and $\omega_2$=$\omega_2'$;
19.    else if rand(0,1) < exp(−$\Delta T/T_0$) then
20.      $\omega_1$=$\omega_1'$ and $\omega_2$=$\omega_2'$;
21.    end if
22.  end if
23. if trigger synthesis then
24.    if $\Delta T$= $PE_{\omega'}$ −max($PE_{\omega_1'}$, $PE_{\omega_2'}$)$PE_\omega$ then
25.      $\omega_1$=$\omega'$ or $\omega_2$=$\omega'$;
26.    else if rand(0,1) < exp(−$\Delta T/T_0$) then
27.      $\omega_1$=$\omega'$ or $\omega_2$=$\omega'$;
28.    end if
29.  end if
30. end if
```

the critical-FMU refinement methods are designed to adjust the job allocation and the processing order of the jobs. The specific ways are as follows and algorithm shows in Algorithm 3.

Method 1: In the FMU part of the molecule, a job is selected randomly from the critical FMU to the FMU with the shortest completion time. Take Fig. 3 as an example again, the critical FMU is FMU_2 and the FMU with the shortest completion time is FMU_1. While J_1 and J_2 are allocated to FMU_2. So, J_1 or J_2 is selected to allocate to FMU_1. And then, in the operation part of the molecule, operations of jobs are exchanged and inserted randomly.

Method 2: Rearrangement of operations in the critical FMU. Take Fig. 3 as an example, J_1 and J_2 are allocated to FMU_2. Rearrange the operations of J_1 and J_2 while keeping the same processing order of each operation in J_3.

4 Numerical Experiment

4.1 Introduction of the DFJSP instance

Experiment 1 involves 3 DFJSP instances, which is referenced by Chang and Liu[18]. Of these 3 DFJSP instances, one is a small-scale heterogeneous example, and the others are homogeneous FMUs examples. Five algorithms are compared in Experiment 1. Set parameter *Popsize* =100 and *Maxgen*=1000 in HGA, GA-JS, GA-OP, and HCRO, other parameter settings are the same as those in their articles.

Experiment 2 involves 23 DFJSP instances, which are proposed by Giovanni and Pezzella[15]. In Chang and Liu's research[18], experimental results of the 23 DFJSP instances are not available. Therefore, the lower bound, IGA[15], GA-JS[17], and GA-OP[19] are compared in this experiment. Set parameter *Popsize*=50 in IGA, GA-JS, GA-OP, and HCRO, and the other parameter settings are the same as those in their articles.

Algorithm 3 Critical-FMU refinement algorithm

```
1. if $rand(0,1)>p$ then
2.    a new molecule $\omega'$ is obtained by performing the operation of Method 1;
3.    if $PE_{\omega'} < PE_{\omega}$ then
4.       $\omega=\omega'$;
5.    end
6. else
7.    a new molecule $\omega'$ is obtained by performing the operation of Method 2;
8.    if $PE_{\omega'} < PE_{\omega}$ then
9.       $\omega=\omega'$;
10.   end
11. end
```

However, there is no standard heterogeneous FMUs example that can be used to test DFJSP, and the current experimental examples are all small-scale examples within 20 jobs. To study the different scales DFJSP on heterogeneous FMUs, examples of 10 to 120 jobs considering the case of 3, 4, 5 FMUs are generated in Experiment 3. The generated examples are named "SL number of jobs-number of FMUs". For example, when 10 jobs are assigned to 3 FMUs, the new example is named "SL10-3". The parameters used to generate the examples are shown in Table 4.

Take SL10-3 as an example, the number of jobs is 10, the number of operations in each job is a random integer from 5 to 7. The jobs need to be assigned to 3 FMUs. The number of machines in each FMU is a random integer from 5 to 7, and the processing time is a random integer from 1 to 7. Each FMU can process all operations. Among them, the size is measured by the number of jobs. 10 to 20 jobs are defined as small-scale examples; 30 to 60 jobs are defined as medium-scale examples; 80 to 120 jobs are defined as large-scale examples. For time comparison, the experiment reproduces HGA and GA-OP for comparison. To verify the effectiveness of rule initialization, SA new solution acceptance method, and critical-FMU refinement methods, the proposed algorithm without rule initialization is named HCRO1, and the proposed algorithm without SA new solution acceptance method is called HCRO2. The proposed algorithm without critical-FMU refinement methods is addressed as HCRO3. The better and newer HGA and GA-OP of Experiments 1 and 2 are selected as the comparison algorithms for Experiment 3.

Table 4 Parameters used to generate the examples.

Instance	Number of jobs	Number of operations	Number of FMUs	Number of machines in each FMU	Process time
SL10	10	[5,7]	3,4,5	[5,7]	[1, 7]
SL20	20	[3,10]	3,4,5	[6,10]	[5, 20]
SL30	30	[5,10]	3,4,5	[6,12]	[10, 30]
SL50	50	[5,10]	3,4,5	[6,12]	[10, 30]
SL60	60	[5,10]	3,4,5	[8,12]	[10, 30]
SL80	80	[8,12]	3,4,5	[8,17]	[10, 30]
SL100	100	[8,12]	3,4,5	[12,17]	[15, 30]
SL120	120	[10,15]	3,4,5	[12,17]	[15, 30]

4.2 HCRO parameter design

In the HCRO algorithm, the following parameters affect the performance: unimolecular collisions reaction probability *MoleColl*, initial kinetic energy *InitialKE*, the maximum number of iteration a in which the optimal molecular potential energy has not been improved, refinement operation probability p, and lower kinetic energy limit b. In order to investigate the influence of parameters on the algorithm performance, the orthogonal test method is used in this paper. SL20-3 is used for the test, and four different horizontal values are chosen for each test parameter, as shown in Table 5.

In the algorithm, the molecular population size *Popsize*=150, maximum number of iterations *Maxgen*=500, initial *buffer* =0, the initial temperature T_0=1000, the threshold temperature T_f $=10^{-8}$, and attenuation factor *Beta*=0.97. To avoid the randomness of test results, each group of experiments is run for 30 times. The average values of the results of 30 times are listed in Table 6. Lines from 1 to 4 in Table 7 show the average makespan at each level of different factors. The last line lists the standard deviations (S.D) of these means, indicating the importance of each factor. The larger the S.D of factor is, the greater the impact on the HCRO. Different parameter settings affect the performance of HCRO algorithm. In the selection of algorithm parameters, reasonable values of these five parameters should be obtained according to the problem. According to the experimental results, values of these five parameters are determined as follows: *MoleColl*=0.2, *InitialKE*=500, a=40, p=0.7, and b=20.

4.3 Experimental results and evaluation

The operating environment of the algorithm is: Intel Core i7-8550U CPU 1.8 GHz, RAM 16 GB, Windows 10, 64-bit operating systems, and the programming language is C.

Table 5 Horizontal values of the parameters in HCRO algorithm.

Set No.	*MoleColl*	*InitialKE*	*a*	*p*	*b*
1	0.2	100	20	0.4	20
2	0.4	500	40	0.5	40
3	0.6	1000	60	0.6	60
4	0.8	1500	80	0.7	80

Table 6 Average values for different combinations of parameters.

Test No.	*MoleColl*	*InitialKE*	*a*	*p*	*b*	*Av.*
1	1	1	1	1	1	61.90
2	1	2	2	2	2	62.10
3	1	3	3	3	3	62.10
4	1	4	4	4	4	62.17
5	2	1	2	3	4	62.23
6	2	2	1	4	3	61.90
7	2	3	4	1	2	62.20
8	2	4	3	2	1	62.70
9	3	1	3	4	2	62.53
10	3	2	4	3	1	61.60
11	3	3	1	2	4	62.23
12	3	4	2	1	3	62.23
13	4	1	4	2	3	63.43
14	4	2	3	1	4	62.47
15	4	3	2	4	1	62.00
16	4	4	1	3	2	63.07

Table 7 Average values for different parameters.

Set No.	*MoleColl*	*InitialKE*	*a*	*p*	*b*
1	62.07	62.53	62.28	62.20	62.05
2	62.26	62.02	62.14	62.62	62.48
3	62.15	62.13	62.45	62.25	62.42
4	62.74	62.54	62.35	62.15	62.28
S.D	0.30	0.23	0.11	0.19	0.17

The results of the three experiments are listed in Tables 8−12, respectively. *MK* represents the optimal solution in an instance, *Av.* denotes the average solution of an instance, T represents the average computation time to run the instance once, *Dev*=(*Deviation*)÷*Av.*, *LB* reports a lower bound proposed by Giovanni and Pezzella[15].

Table 8 displays the experiment results after 50 runs of the 3 DFJSP instances in Experiment 1. Five algorithms (IGA[15], HGA[18], GA-JS[17], GA-OP[19], and HCRO) are compared. In DFJSP1, HCRO

Table 8 Comparison of the results of the algorithms in Experiment 1.

Instance	IGA[15]			HGA[18]			GA-JS[17]			GA-OP[19]			HCRO		
	MK	*Av.*	*Dev* (%)	*MK*	*Av.*	*Dev* (%)	*MK*	*Av.*	*Dev* (%)	*MK*	*Av.*	*Dev* (%)	*MK*	*Av.*	*Dev* (%)
DFJSP1	11	N/A	N/A	**9**	N/A	N/A	**9**	9.0	0.0	**9**	9.0	0.0	**9**	9.0	0.0
DFJSP2	**37**	38.6	1.9	**37**	37.6	0.4	38	38.0	0.0	**37**	37.0	0.0	**37**	37.1	0.0
DFJSP3	**37**	38.3	1.9	**37**	37.5	0.6	38	38.2	0.2	**37**	37.0	0.0	**37**	37.1	0.0

Table 9 Performance comparison of 4-FMU DFJSP algorithms in Experiment 2.

Instance	*LB*	IGA[15]			GA-JS[17]			GA-OP[19]			HCRO		
		MK	*Av.*	*Dev* (%)	*MK*	*Av.*	*Dev* (%)	*MK*	*Av.*	*Dev* (%)	*MK*	*Av.*	*Dev* (%)
la01	413	**413**	413.0	0.0	**413**	413.0	0.0	**413**	413.0	0.0	**413**	413.0	0.0
la02	394	**394**	394.0	0.0	**394**	394.0	0.0	**394**	394.0	0.0	**394**	394.0	0.0
la03	349	**349**	349.0	0.0	**349**	349.0	0.0	**349**	349.0	0.0	**349**	349.0	0.0
la04	369	**369**	369.0	0.0	**369**	369.0	0.0	**369**	369.0	0.0	**369**	369.0	0.0
la05	380	**380**	380.0	0.0	**380**	380.0	0.0	**380**	380.0	0.0	**380**	380.0	0.0
la06	413	**413**	413.0	0.0	**413**	413.0	0.0	**413**	413.0	0.0	**413**	413.0	0.0
la07	376	**376**	376.0	0.0	**376**	376.0	0.0	**376**	376.0	0.0	**376**	376.0	0.0
la08	369	**369**	369.0	0.0	**369**	369.0	0.0	**369**	369.0	0.0	**369**	369.0	0.0
la09	382	**382**	382.0	0.0	**382**	382.0	0.0	**382**	382.0	0.0	**382**	382.0	0.0
la10	443	**443**	443.0	0.0	**443**	443.0	0.0	**443**	443.0	0.0	**443**	443.0	0.0
la11	413	**413**	413.0	0.0	**413**	413.0	0.0	**413**	413.0	0.0	**413**	413.0	0.0
la12	408	**408**	408.0	0.0	**408**	408.0	0.0	**408**	408.0	0.0	**408**	408.0	0.0
la13	382	**382**	386.0	9.9	**382**	382.0	0.0	**382**	382.0	0.0	**382**	388.9	0.0
la14	443	**443**	443.0	0.0	**443**	443.0	0.0	**443**	443.0	0.0	**443**	443.0	0.0
la15	378	397	402.0	2.3	**378**	381.9	4.7	**378**	385.8	7.8	**378**	395.5	5.2
la16	717	**717**	717.0	0.0	**717**	717.0	0.0	**717**	717.0	0.0	**717**	717.0	0.0
la17	646	**646**	646.0	0.0	**646**	646.0	0.0	**646**	646.0	0.0	**646**	646.0	0.0
la18	663	**663**	663.0	0.0	**663**	663.0	0.0	**663**	663.0	0.0	**663**	663.0	0.0
la19	617	**617**	617.0	0.0	**617**	617.0	0.0	**617**	617.0	0.0	**617**	617.0	0.0
la20	756	**756**	756.0	0.0	**756**	756.0	0.0	**756**	756.0	0.0	**756**	756.0	0.0
mt06	47	**47**	47.0	0.0	**47**	47.0	0.0	**47**	47.0	0.0	**47**	47.0	0.0
mt10	664	**664**	665.0	0.0	**664**	665.0	0.0	**664**	665.0	0.0	**664**	665.0	0.0
mt20	387	**387**	388.4	2.0	**387**	387.0	0.0	**387**	387.0	0.0	**387**	392.1	0.0

outperforms IGA in *MK* and gets the same *MK* as HGA, GA-JS, and GA-OP. HCRO can get the same *Av.* and *Dev* as GA-JS and GA-OP. Because there is no *Av.* and *Dev* data in the original articles of IGA and HGA, the mean and standard deviation of the experiment are not compared with that of HGA and IGA. Therefore, HCRO is competitive with GA-JS and GA-OP in small-scale example on heterogeneous FMUs. In DFJSP2 and DFJSP3, HCRO gets smaller *MK* than GA-JS and has the same *MK* as IGA, HGA, and GA-OP. HCRO outperforms IGA, HGA, and GA-JS in terms of *Av.* and *Dev*. But HCRO has slightly bigger *Av.* than GA-OP. This means that HCRO is competitive with the other four improved genetic algorithms in terms of the optimal solution. And the stability of HCRO is slightly inferior to that of GA-OP on homogeneous FMUs.

Table 9 displays the results of 23 DFJSP examples after 50 runs from Experiment 2 on four homogeneous FMUs. Four algorithms (IGA[15], GA-JS[17], GA-OP[19], and HCRO) are compared. GA-JS, GA-OP, and HCRO can obtain the lower bound of 23 examples while IGA can only get the lower bound of 22 examples. Compared with GA-JS and GA-OP, HCRO can get the same *MK* in all 23 examples, which means that HCRO has the same advantage in obtaining the optimal solution. In la13 and mt20, HCRO is slightly bigger than IGA, GA-JS, and GA-OP in *Av.*. In la15, HCRO is slightly bigger than GA-JS and GA-OP but smaller than IGA in *Av.* . Therefore, HCRO is competitive with IGA, GA-JS, and GA-OP in optimal solutions and inferior to GA-JS and GA-OP in the stability of solutions on homogeneous FMUs.

Tables 10 and 11 display the results of 24 DFJSP examples after 50 runs from Experiment 3 on heterogeneous FMUs. Six algorithms (HGA[18], GA-OP[19], HCRO1, HCRO2, HCRO3, and HCRO) are compared in Experiment 3. To ensure fairness, HGA and GA-OP adopt the same population number and iteration stop conditions as HCRO, and other parameter settings are the same as those set in their articles.

In Table 10, it can be seen from the comparison between HCRO1 and HCRO that HCRO can improve the *Av.* and *Dev* in small-scale examples. HCRO can

Table 10 Performance comparison of DFJSP in heterogeneous FMUs in Experiment 3.

Instance	HCRO1				HCRO2				HCRO3				HCRO			
	MK	*Av.*	*T*(s)	*Dev* (%)	*MK*	*Av.*	*T*(s)	*Dev* (%)	*MK*	*Av.*	*T*(s)	*Dev* (%)	*MK*	*Av.*	*T*(s)	*Dev* (%)
SL10-3	**11**	11.5	3.4	0.1	**11**	11.4	3.6	0.0	**11**	11.0	2.0	0.0	**11**	11.0	7.2	0.0
SL10-4	**10**	10.4	3.8	0.1	**10**	10.4	5.4	0.1	**10**	10.3	2.1	0.0	**10**	10.1	6.4	0.0
SL10-5	**9**	9.2	3.4	0.1	**9**	9.2	5.6	0.0	**9**	9.1	2.0	0.0	**9**	9.0	9.8	0.0
SL20-3	**60**	61.0	8.1	0.0	**60**	61.2	8.4	0.0	**60**	60.6	4.3	0.1	**60**	60.6	13.3	0.0
SL20-4	**59**	60.1	8.1	0.0	**59**	59.1	12.1	0.0	**59**	59.2	4.7	0.0	**59**	59.1	12.7	0.0
SL20-5	**59**	59.4	8.2	0.0	**59**	59.1	11.8	0.0	**59**	59.6	5.0	0.0	**59**	59.0	12.0	0.0
SL30-3	133	140.2	16.7	0.0	**130**	136.2	21.5	0.0	131	136.2	8.1	0.0	**130**	135.0	21.5	0.0
SL30-4	**109**	110.3	15.4	0.0	**109**	109.7	18.9	0.0	**109**	109.7	8.8	0.0	**109**	109.4	19.6	0.0
SL30-5	113	116.3	16.6	0.0	110	113.4	20.9	0.0	111	113.0	9.0	0.0	**109**	110.9	22.0	0.0
SL50-3	177	185.8	23.2	0.0	171	178.5	30.9	0.0	170	177.5	13.4	0.0	**169**	176.8	31.0	0.0
SL50-4	163	169.2	21.0	0.0	155	162.0	30.0	0.0	155	159.9	12.9	0.0	**153**	159.8	34.6	0.0
SL50-5	153	159.0	24.6	0.0	144	148.3	30.5	0.0	143	147.2	13.7	0.0	**142**	145.7	31.8	0.0
SL60-3	232	238.6	25.8	0.0	225	231.7	33.0	0.0	224	230.1	13.9	0.0	**223**	229.0	35.0	0.0
SL60-4	199	207.9	27.4	0.0	**190**	195.7	34.8	0.0	191	196.0	15.4	0.0	**190**	195.4	37.1	0.0
SL60-5	147	152.4	28.0	0.0	133	138.7	33.7	0.0	137	140.4	16.1	0.0	**132**	136.1	35.9	0.0
SL80-3	275	281.5	55.7	0.0	252	259.3	75.8	0.0	252	259.2	34.4	0.0	**251**	257.0	76.2	0.0
SL80-4	225	232.4	57.3	0.0	202	208.6	75.3	0.0	203	207.4	34.5	0.0	**201**	206.3	76.6	0.0
SL80-5	190	196.9	58.4	0.0	**169**	173.7	71.7	0.0	**169**	173.0	36.7	0.0	**169**	172.7	78.1	0.0
SL100-3	410	418.5	76.0	0.0	385	390.7	111.6	0.0	380	388.0	46.7	0.0	**379**	386.0	118.4	0.0
SL100-4	370	378.5	74.9	0.0	336	344.6	110.1	0.0	336	344.4	43.6	0.0	**334**	342.6	116.3	0.0
SL100-5	297	305.2	73.9	0.0	266	268.8	107.1	0.0	264	268.1	46.1	0.0	**263**	267.1	109.8	0.0
SL120-3	739	748.5	102.7	0.0	701	709.6	139.7	0.0	698	706.3	59.6	0.0	**693**	703.7	146.7	0.0
SL120-4	520	534.3	112.3	0.0	479	487.8	155.7	0.0	479	485.5	66.3	0.0	**476**	483.3	175.4	0.0
SL120-5	428	436.5	115.8	0.0	**377**	383.3	179.1	0.0	**377**	384.6	77.1	0.0	**377**	381.4	209.7	0.0
Average	−	−	40.4	0.0	−	−	−	0.0	−	−	24.0	0.0	−	−	59.9	0.0

get better *MK*, *Av.*, and *Dev* than HCRO1 in medium-scale and large-scale examples. Therefore, the effectiveness of the rule initialization is verified. From the comparison between HCRO2 and HCRO, HCRO can get the same *MK* as HCRO2 and better *Av.* than HCRO2 in small-scale examples. In the medium-scale examples, HCRO2 and HCRO obtain the same *MK* in SL30-3, SL30-4, and SL60-4, while HCRO obtains better *MK* than HCRO2 in other medium-scale examples. And HCRO gets better *Av.* in all medium-scale examples. In the large-scale examples, HCRO can obtain better *MK* than HCRO2 in all large-scale examples except SL80-5 and SL120-5. And HCRO also shows its advantage in *Av.*. The results reveal the effectiveness of SA. It can be seen from the comparison between HCRO3 and HCRO that HCRO can improve *Av.* and *Dev* in small-scale examples.

HCRO can optimize the *MK* and *Av.* of all medium-scale examples. In the large-scale examples, HCRO can obtain better *MK* than HCRO2 except SL80-5 and SL120-5 and improve *Av.* in all large-scale examples.

From the comparison between HCRO and HGA in Table 11, HCRO gets smaller *MK* in SL20-5 and the same *MK* as HGA in other small-scale examples. HCRO obtains smaller *MK* than HGA in medium-scale and large-scale examples. Wilcoxon signed rank test is used to statistically justify the performance difference between HCRO and HGA in terms of *Av.* HCRO outperforms HGA with *p-value*=1.8×10^{-5} <0.05. The average *T* of HCRO is 59.9 s and the average *T* of HGA is 2289.0 s. So HCRO shows superiority than HGA in terms of *T*. From the comparison between HCRO and GA-OP, HCRO gets the same *MK* as GA-OP in small-scale examples. This means that HCRO is about as competitive as GA-OP in small-scale examples. In medium-scale examples, GA-OP can get smaller *MK* than HCRO in SL50-3, SL50-4, SL50-5, and SL60-4. HCRO obtains the same *MK* as GA-OP in SL30-3, SL30-4, and SL60-5. HCRO has smaller *MK* than GA-OP in SL30-5 and SL60-3. HCRO can get better *MK* than GA-OP in all large-scale examples. In terms of *Av.* , HCRO outperforms GA-OP with *p*-

Table 11 Performance comparison between different algorithms in heterogeneous FMUs in Experiment 3.

Instance	HGA				GA-OP				HCRO			
	MK	*Av.*	*T*(s)	*Dev* (%)	*MK*	*Av.*	*T*(s)	*Dev* (%)	*MK*	*Av.*	*T*(s)	*Dev* (%)
SL10-3	**11**	12.5	33.4	0.1	**11**	11.0	10.9	0.0	**11**	11.0	7.2	0.0
SL10-4	**10**	11.2	36.6	0.1	**10**	10.0	10.9	0.0	**10**	10.1	6.4	0.0
SL10-5	**9**	9.9	38.7	0.1	**9**	9.0	12.1	0.0	**9**	9.0	9.8	0.0
SL20-3	**60**	63.2	96.6	0.0	**60**	60.4	25.3	0.0	**60**	60.6	13.3	0.0
SL20-4	**59**	62.1	94.1	0.0	**59**	59.1	30.3	0.0	**59**	59.1	12.7	0.0
SL20-5	60	63.5	96.6	0.0	**59**	59.2	32.1	0.0	**59**	59.0	12.0	0.0
SL30-3	140	151.4	391.6	0.0	**130**	135.5	52.9	0.0	**130**	135.0	21.5	0.0
SL30-4	116	123.2	475.9	0.0	**109**	109.9	56.0	0.0	**109**	109.4	19.6	0.0
SL30-5	119	121.7	447.3	0.0	110	114.4	63.7	0.0	**109**	110.9	22.0	0.0
SL50-3	180	193.0	1090.2	0.0	**166**	173.1	114.9	0.0	169	176.8	31.0	0.0
SL50-4	161	167.9	1058.4	0.0	**144**	150.9	112.5	0.0	153	159.8	34.6	0.0
SL50-5	147	159.2	1079.9	0.0	**134**	137.9	118.3	0.0	142	145.7	31.8	0.0
SL60-3	243	250.4	1790.5	0.0	225	230.7	123.1	0.0	**223**	229.0	35.0	0.0
SL60-4	218	226.3	1783.7	0.0	**187**	191.1	131.8	0.0	190	195.4	37.1	0.0
SL60-5	158	167.6	1908.4	3.0	**132**	137.5	136.1	0.0	**132**	136.1	35.9	0.0
SL80-3	389	307.7	4800.9	0.0	253	260.8	315.3	0.0	**251**	257.0	76.2	0.0
SL80-4	234	243.8	4787.9	0.0	206	211.7	414.6	0.0	**201**	206.3	76.6	0.0
SL80-5	205	209.8	4926.1	0.0	171	176.3	385.2	0.0	**169**	172.7	78.1	0.0
SL100-3	437	448.2	5000	0.0	394	402.9	498.1	0.0	**379**	386.0	118.4	0.0
SL100-4	393	406.1	5000	0.0	354	368.4	470.6	0.0	**334**	342.6	116.3	0.0
SL100-5	324	335.1	5000	0.0	264	267.7	464.1	0.0	**263**	267.1	109.8	0.0
SL120-3	764	787.6	5000	0.0	712	727.0	673.4	0.0	**693**	703.7	146.7	0.0
SL120-4	552	569.5	5000	0.0	527	537.5	769.7	0.0	**476**	483.3	175.4	0.0
SL120-5	467	480.7	5000	0.0	384	394.0	875.6	0.0	**377**	381.4	209.7	0.0
Average	–	–	2289.0	0.0	–	–	249.8	0.0	–	–	59.9	0.0

Table 12 Influence of the number of machines in the FMUs on the algorithms.

Instance	Number of machines (FMU_1, FMU_2, and FMU_3)	Standard deviation	HGA		GA-OP		HCRO	
			MK	*Av.*	*MK*	*Av.*	*MK*	*Av.*
SL30-3	9, 8,12	1.7	140	151.4	**130**	135.5	**130**	135.0
SL30-3*	10,6,12	2.5	144	152.0	135	143.7	**130**	134.7
SL50-3	12,9,11	1.2	180	193.0	**166**	173.1	169	176.8
SL50-3*	11,11, 6	2.4	215	222.0	224	244.7	**189**	202.1
SL80-3	16,13,15	1.2	389	307.7	253	260.8	**251**	257.0
SL80-3*	17,9,15	3.4	310	322.0	323	339.5	**252**	274.2

value=0.0475<0.05, which means HCRO shows better stability than GA-OP. In addition, the average *T* of GA-OP is 249.8 s, slower than that of HCRO (59.9 s). To sum up, HCRO gets the same optimal solutions as HGA in small-scale examples and shows its advantages in medium-scale and large-scale examples. Meanwhile, HCRO outperforms HGA in terms of the stability of solutions and computation time. HCRO is competitive with GA-OP in small and medium-scale examples, but HCRO is better than GA-OP in large-scale examples. And HCRO is also better than GA-OP in the stability of solutions and computation time.

To explore the influence of the number of machines in each FMU on the algorithms, SL30-3, SL50-3, and SL80-3 are selected for further experiments. The second column of Table 12 indicates the number of machines in the three FMUs. The third column of Table 12 shows the standard deviation of machines in three FMUs which indicates the difference in the number of machines between three FMUs. When the number of machines in the three FMUs is not much different, GA-OP has absolute advantages in MK and $Av.$ than HGA. In SL80-3, GA-OP is worse than HCRO both in MK and $Av.$. GA-OP can get the same MK as HCRO in SL30-3 and even get smaller MK than HCRO in SL50-3. When the number of machines in the three FMUs differs greatly, in SL30-3*, GA-OP gets the better MK than HGA and gets worse MK than HCRO. HCRO has 3.7% improvement compared to GA-OP in MK in SL30-3*. In SL50-3*, HGA can get better MK and $Av.$ than GA-OP and HCRO has 15.6% improvement compared to GA-OP in MK. In SL80-3*, HGA can also get better MK and $Av.$ than GA-OP and HCRO has 22.0% improvement compared to GA-OP in MK. Therefore, the GA-OP is inferior to HCRO and HGA when the number of machines in these FMUs differs greatly. And as the size of the job increases, the disadvantage becomes more obvious. This is because the encoding method of GA-OP only involves the arrangement of operations, and the choices of FMU and machine are both subject to rules. Therefore, when the number of machines differs greatly between FMUs, the rule of FMU allocation makes the algorithm fall into a local optimum. The operation-FMU encoding-decoding method proposed in this paper can explicitly express two of the three sub-problems of DFJSP. The choice of the FMU is not affected by the rule, and the solution space can be searched to a greater extent. Therefore, GA-OP is affected by the number of machines in each FMU while HCRO and HGA are slightly affected by the number of machines in each FMU.

In conclusion, the operation-FMU encoding-decoding method can search the solution space to a large degree without the affection of the number of machines in each FMU, while the machine selection rule can balance the workload of the machines at the same time to get a better solution quickly. This encoding and decoding method makes HCRO have strong applicability to solve DFJSP in homogeneous and heterogeneous environments at the same time.

The rule is used for initializing partial solutions, which enables HCRO to search in a better area while maintaining diversity in the initial stage of the algorithm, which can improve the quality of the initial solution and speed up the convergence rate. The new solution acceptance method based on SA can improve the optimization ability and stability of the algorithm on the basis of ensuring the constant number of molecular populations. The critical-FMU refinement methods focus on critical FMU, which can optimize the target to a certain extent and enhance the local search ability of the algorithm. In this case, the HCRO algorithm can search for better solutions compared with the comparative algorithms, which is especially outstanding in large-scale calculation examples. In addition, HCRO is superior to the comparison algorithms in terms of time due to the superior convergence of CRO algorithm and its encoding method.

5 Concluding Remark

In this paper, an HCRO algorithm is proposed to solve DFJSP on heterogeneous FMUs. Aiming at the characteristics of the problem, an operation-FMU based encoding-decoding method is designed. The method of random generation and rule generation is used for generating initial population. Four collision operations of CRO algorithm are designed and simulated annealing algorithm is combined with CRO algorithm to accept new solutions from the four collision reactions to improve the search capability. In addition, a refinement method is designed for critical-FMU. Finally, three groups of comparative experiments which include 50 DFJSP instances in total are carried out. Compared with the existing algorithms with better performance, HCRO has slightly less stability but can get the same optimal solution in small-scale examples on homogeneous FMUs. Compared with two existing algorithms, the superiority of HCRO in solving the large, medium, and small-scale examples on heterogeneous FMUs is verified, and the effectiveness of initialization rules, the critical-FMU refinement methods, and new solution acceptance method based on simulated annealing algorithm are also verified. In future research, we try to apply chemical reaction optimization algorithm to the energy-efficient DFJSP because the energy-efficient DFJSP has attracted significant attention from the viewpoint of sustainable development and green manufacturing, which includes developing a new chromosome representation, establishing corresponding mathematical model, and designing the corresponding scheduling rules.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 61973120, 62076095, 61673175, and 61573144).

References

[1] H. J. Ding and X. S. Gu, Improved particle swarm optimization algorithm based novel encoding and decoding schemes for flexible job shop scheduling problem, *Computers & Operations Research*, vol. 121, p. 104951, 2020.

[2] K. Z. Gao, Z. G. Cao, L. Zhang, Z. H. Chen, Y. Y. Han, and Q. K. Pan, A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems, *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 904–916, 2019.

[3] K. Z. Gao, F. J. Yang, M. C. Zhou, Q. K. 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.

[4] R. L. Burdett, P. Corry, P. K. D. V. Yarlagadda, C. Eustace, and S. Smith, A flexible job shop scheduling approach with operators for coal export terminals, *Computers & Operations Research*, vol. 104, pp. 15–36, 2019.

[5] D. Deliktas, O. Torkul, and O. Ustun, A flexible job shop cell scheduling with sequence-dependent family setup times and intercellular transportation times using conic scalarization method, *International Transactions in Operational Research*, vol. 26, no. 6, pp. 2410–2431, 2019.

[6] Q. W. Deng, G. L. Gong, X. R. Gong, L. K. Zhang, W. Liu, and Q. H. Ren, A bee evolutionary guiding nondominated sorting genetic algorithm II for multiobjective flexible job-shop scheduling, *Computational Intelligence & Neuroscience* , vol. 2017, p. 5232518, 2017.

[7] G. L. Gong, Q. W. Deng, R. Chiong, X. Gong, and H. Z. Y. Huang, An effective memetic algorithm for multi-objective job-shop scheduling, *Knowledge-Based Systems*, vol. 182, p. 104840, 2019.

[8] I. Chaouch, O. B. Driss, and K. Ghedira, A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm, *Applied Intelligence*, vol. 49, no. 5, pp. 1903–1924, 2019.

[9] Q. K. Pan, L. Gao, L. Wang, J. Liang, and X. Y. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, *Expert Systems with Applications*, vol. 124, pp. 309–324, 2019.

[10] J. Q. Li, M. X. Song, L. Wang, P. Y. Duan, Y. Y. Han, H. Y. Sang, and Q. K. Pan, Hybrid artificial bee colony algorithm for a parallel batching distributed flow-shop problem with deteriorating jobs, *IEEE Transactions on Cybernetics*, vol. 50, no. 6, pp. 2425–2439, 2020.

[11] S. C. Zhang, X. Li, B. W. Zhang, and S. Y. 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.

[12] J. Zheng, L. Wang, and J. J. Wang, A cooperative coevolution algorithm for multi-objective fuzzy distributed hybrid flow shop, *Knowledge-Based Systems*, vol. 194, p. 105536, 2020.

[13] J. Q. Li, P. Y. Duan, J. D. Cao, X. P. Lin, and Y. Y. Han, A hybrid Pareto-based tabu search for the distributed flexible job shop scheduling problem with E/T criteria, *IEEE Access*, vol. 6, no. 99, pp. 58883–58897, 2018.

[14] F. T. S. Chan, S. H. Chung, and P. L. Y. Chan, Application of genetic algorithms with dominant genes in a distributed scheduling problem in flexible manufacturing systems, *International Journal of Production Research*, vol. 44, no. 3, pp. 523–543, 2006.

[15] L. D. Giovanni and F. Pezzella, An improved genetic algorithm for the distributed and flexible job-shop scheduling problem, *European Journal of Operational Research*, vol. 200, no. 2, pp. 395–408, 2010.

[16] M. Ziaee, A heuristic algorithm for the distributed and flexible job-shop scheduling problem, *Journal of Supercomputing*, vol. 67, no. 1, pp. 69–83, 2014.

[17] P. H. Lu, M. C. Wu, H. Tan, Y. H. Peng, and C. F. Chen, A genetic algorithm embedded with a concise chromosome representation for distributed and flexible job-shop scheduling problems, *Journal of Intelligent Manufacturing*, vol. 29, no. 1, pp. 19–34, 2018.

[18] H. C. Chang and T. K. Liu, Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms, *Journal of Intelligent Manufacturing*, vol. 28, no. 8, pp. 1973–1986, 2017.

[19] M. C. Wu, C. S. Lin, C. H. Lin, and C. F. Chen, Effects of different chromosome representations in developing genetic algorithms to solve DFJS scheduling problems, *Computers & Operations Research*, vol. 80, pp. 101–112, 2017.

[20] B. Marzouki, O. B. Driss, and K. Ghédira, Solving distributed and flexible job shop scheduling problem using a chemical reaction optimization metaheuristic, *Procedia Computer Science*, vol. 126, no. 1, pp. 1424–1433, 2018.

[21] F. T. S. Chan, S. H. Chung, L. Y. Chan, G. Finke, and M. K. Tiwari, Solving distributed FMS scheduling problems subject to maintenance: Genetic algorithms approach, *Robotics and Computer-Integrated Manufacturing*, vol. 22, nos. 5&6, pp. 493–504, 2006.

[22] S. H. Chung, F. T. S. Chan, and H. K. Chan, A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling, *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 1005–1014, 2009.

[23] C. S. Lin, I. L. Lee, and M. C. Wu, Merits of using chromosome representations and shadow chromosomes in genetic algorithms for solving scheduling problems, *Robotics and Computer-Integrated Manufacturing*, vol. 58, no. 1, pp. 196–207, 2019.

[24] F. T. S. Chan, A. Prakash, H. L. Ma, and C. S. Wong, A hybrid tabu sample-sort simulated annealing approach for solving distributed scheduling problem, *International Journal of Production Research*, vol. 51, no. 9, pp. 2602–2619, 2013.

[25] M. Ziaee, Modeling and solving the distributed and flexible job shop scheduling problem with WIPs supply planning and bounded processing times, *International Journal of Supply and Operations Management*, vol. 4, no. 1, pp. 78–89, 2017.

[26] Q. Luo, Q. W. Deng, G. L. Gong, L. K. Zhang, W. W. Han, and K. X. Li, An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers, *Expert Systems with Applications*, vol. 160, p. 113721, 2020.

[27] W. X. Xu, Y. W. Hu, W. Luo, L. Wang, and R. Wu, A multi-objective scheduling method for distributed and flexible job shop based on hybrid genetic algorithm and tabu search considering operation outsourcing and carbon emission, *Computers & Industrial Engineering*, vol. 157, p. 107318, 2021.

[28] Y. Du, J. Q. Li, C. Luo, and L. L. Meng, A hybrid estimation of distribution algorithm for distributed flexible job shop scheduling with crane transportations, *Swarm and Evolutionary Computation*, vol. 62, p. 100861, 2021.

[29] F. Q. Zhao, L. X. Zhao, L. Wang, and H. B. Song, An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion, *Expert Systems with Applications*, vol. 160, p. 113678, 2020.

[30] F. Q. Zhao, X. He, and L. Wang, A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem, *IEEE Transactions on Cybernetics*, vol. 51, no. 11, pp. 5291–5303, 2020.

[31] F. Q. Zhao, R. Ma, and L. Wang, A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system, *IEEE Transactions on Cybernetics*, doi: 10.1109/TCYB.2021.3086181.

[32] G. C. Wang, X. Y. Li, L. Gao, and P. G. Li, A multi-objective whale swarm algorithm for energy-efficient distributed permutation flow shop scheduling problem with sequence dependent setup times, *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 235–240, 2019.

[33] D. M. Lei, Y. Yuan, J. C. Cai, and D. Y. Bai, An imperialist competitive algorithm with memory for distributed unrelated parallel machines scheduling, *International Journal of Production Research*, vol. 58, no. 2, pp. 597–614, 2020.

[34] D. M. Lei, Y. Yuan, and J. C. Cai, An improved artificial bee colony for multi-objective distributed unrelated parallel machine scheduling, *International Journal of Production Research*, vol. 59, no. 17, pp. 5259–5271, 2021.

[35] M. A. Şahman, A discrete spotted hyena optimizer for solving distributed job shop scheduling problems, *Applied Soft Computing*, vol. 106, p. 107349, 2021.

[36] A. Y. S. Lam and V. O. K. Li, Chemical-reaction-inspired metaheuristic for optimization, *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 3, pp. 381–399, 2010.

[37] Y. M. Xu, K. L. Li, L. G. He, and T. K. Truong, A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization, *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1306–1322, 2013.

[38] T. K. Truong, K. L. Li, and Y. M. Xu, Chemical reaction optimization with greedy strategy for the 0–1 knapsack problem, *Applied Soft Computing*, vol. 13, no. 4, pp. 1774–1780, 2013.

[39] J. Q. Li and Q. K. Pan, Chemical-reaction optimization for solving fuzzy job-shop scheduling problem with flexible maintenance activities, *International Journal of Production Economics*, vol. 145, no. 1, pp. 4–17, 2013.

[40] B. Marzouki, O. B. Driss, and K. Ghédira, Multi agent model based on chemical reaction optimization with greedy algorithm for flexible job shop scheduling problem, *Procedia Computer Science*, vol. 112, no. C, pp. 81–90, 2017.

[41] H. Bargaoui, O. B. Driss, and K. Ghédira, A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion, *Computers & Industrial Engineering*, vol. 111, no. C, pp. 239–250, 2017.

[42] H. Bargaoui, O. B. Driss, and K. Ghédira, Towards a distributed implementation of chemical reaction optimization for the multi-factory permutation flowshop scheduling problem, *Procedia Computer Science*, vol. 112, pp. 1531–1541, 2017.

[43] R. Wu, S. S. Guo, Y. B. Li, L. Wang, and W. X. Xu, Improved artificial bee colony algorithm for distributed and flexible job-shop scheduling problem, (in Chinese), *Control and Decision*, vol. 34, no. 12, pp. 2527–2536, 2019.

[44] H. J. Xiao, Z. L. Chai, C. Y. Zhang, L. L. Meng, Y. P. Ren, and H. W. Mei, Hybrid chemical-reaction optimization and tabu search for flexible job shop scheduling problem, (in Chinese), *Computer Integrated Manufacturing Systems*, vol. 24, no. 9, pp. 2234–2245, 2018.

[45] C. Bierwirth, A generalized permutation approach to job shop scheduling with genetic algorithms, *OR Spectrum*, vol. 17, pp. 87–92, 1995.

[46] X. Han, Y. Y. Han, Q. D. Chen, J. Q. Li, H. Y. Sang, Y. P. Liu, Q. K. Pan, and Y. S. Nojima, Distributed flow shop scheduling with sequence-dependent setup times using an improved iterated greedy algorithm, *Complex System Modeling and Simulation*, vol. 1, no. 3, pp. 198–217, 2021.

[47] S. P. Brooks and B. J. T. Morgan, Optimization using simulated annealing, *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 44, no. 2, pp. 241–257, 1995.

[48] L. Hernández-Ramírez, J. Frausto-Solis, G. Castilla-Valdez, J. J. Gonzalez-Barbosa, D. Teran-Villanueva, and M. L. Morales-Rodriguez, A hybrid simulated annealing for job shop scheduling problem, *International Journal of Combinatorial Optimization Problems & Informatics*, vol. 10, no. 1, pp. 6–15, 2019.

[49] C. Ramesh, R. Kamalakannan, R. Karthik, C. Pavin, and S. Dhivaharan, A lot streaming based flow shop scheduling problem using simulated annealing algorithm, *Materials Today: Proceedings* , vol. 37, pp. 241–244, 2021.

[50] M. Wang, L. M. Liu, and K. Y. Song, Analysis of flexible shop scheduling problem based on a genetic simulated annealing algorithm, *AIP Conference Proceedings*, vol. 2258, no. 1, p. 020016, 2020.

Jialei Li received the BSc degree from Nanjing Tech University in 2019, and the MSc degree from East China University of Science and Technology in 2022. Her research interests include production scheduling and intelligent optimization algorithms.

Xingsheng Gu received the BS degree from Nanjing Institute of Chemical Technology in 1982, the MS and PhD degrees from East China University of Chemical Technology in 1988 and 1993, respectively. He is currently a professor at East China University of Science and Technology. His research interests include planning and scheduling for process industry, modeling, control, and optimization for industry processes, intelligent optimization, faults detection and diagnosis, etc.

Yaya Zhang received the BSc degree from East China University of Science and Technology in 2017. She is pursuing the PhD degree in East China University of Science and Technology. Her research interests include batch production scheduling problems and distributed scheduling problems in industry process.

Xin Zhou received the BEng degree from Xi'an Jiaotong-Liverpool University and University of Liverpool, UK in 2015, and the MEng degree from Australian National University, Australia in 2017. She received the PhD degree from East China University of Science and Technology in 2022. Her research interests include multi-/many-objective optimization and welding robot path planning.