2021 11th International Conference on Information Science and Technology (ICIST) | 978-1-6654-1266-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICIST52614.2021.9440591

An improved iterated greedy algorithm for the distributed flow shop scheduling problem with sequence-dependent setup times

1st Xue Han
School of Computer Science
Liaocheng University
Liaocheng, China
1979124154@qq.com

2nd* Yuyan Han
School of Computer Science
Liaocheng University
Liaocheng, China
hanyuyan@lcu-cs.com

3rd Yi-ping Liu
The College of Computer Science and Electronic Engineering
Hunan University
Changsha, China
yiping0liu@gmail.com

4th Quan-ke Pan
School of Mechatronic Engineering and Automation
Shanghai University
Shanghai, China
panquanke@mail.neu.edu.cn

5th Hao-xiang Qin
School of Computer Science
Liaocheng University
Liaocheng, China
987352978@qq.com

6th Junqing Li
School of Computer Science,
Shandong Normal University
Jinan, China
lijunqing@lcu-cs.com

***Abstract*—In various flow shop scheduling problems, it is very common that a large-scale production is done. Under this situation, more factories are of more practical interest than a factory. Thus, the distributed permutation flow shop scheduling problems (DPFSPs) have been attracted attentions by researchers. However, the DPFSP is more complicated than the traditional flow shop scheduling problems. It considers not only the processing order of the jobs, but also how to distribute the jobs to multiple factories for parallel processing. In addition, the sequence-dependent setup time (SDST) constraint of machines is taken into account to well study the above DPFSP with SDST. This paper presents a simple and effective iterated greedy algorithm. It is proposed to replace the traditional insertion-based local search with exchange-based local search, which greatly improves the search efficiency. The proposed new iterated greedy (NIG) algorithm is applied to test instances, and compares with the state-of-the-art algorithms. Our empirical results demonstrate that the proposed algorithm outperforms the compared algorithms and can obtain the best solution of DPFSP.**

Keywords—distributed permutation flow shop, iterated greedy algorithm, local search, swap

I. INTRODUCTION

With the continuous development of global economy, a company often needs several production centers, it is necessary to set up a distributed production mode [1]. In recent years, the distributed permutation flow shop problem (DPFSP) has gradually attracted much attentions by researchers. DPFSP can be described as follows: a total of J jobs need to be processed in F factories, each factory contains M serial machines, each factory can independently complete the processing of the job. DPFSP was first proposed by Naderi and Ruiz [1], who used iterated greedy algorithms to solve DPFSP with makespan criteria. They proposed six different mixed integer linear programming (MILP) models, and two simple factory assignment rules together with 14 heuristics based on the dispatching rules, the effective constructive heuristics and variable neighborhood descent methods. Bargaoui et al. designed a CRO algorithm with good single point crossover and effective greedy strategy embedding. The algorithm uses an effective NEH heuristic method to generate initial molecular groups [3]. Fernandez-Viagas and Perez-Gonzalez proposed 18 constructive heuristic methods and an improved iterated algorithm to obtain high-quality solutions [4]. Meng et al. developed three meta-heuristic methods to solve the DPFSP with customer order constraints [5]. In [6], Jing et al. proposed an iterated greedy (IG) algorithm with free time insertion evaluation to solve DPFSP with time window.

In practical production, different jobs will face operations such as tool replacement and equipment inspection when they are processed on the same machine. The time spent by these operations is often not only related to the job to be processed, but also related to the job previously processed on the submachine. Therefore, a sequence-dependent setup times (SDST) will be generated. About SDST / DPFSP, Huang gave a complete description in [2] and proposed an improved IG algorithm to solve this problem. Later, for the same problem, Huang integrated three constructive heuristic methods into discrete artificial bee colony (DABC) algorithm to generate good scheduling solutions [7].

In this paper, the iterated greedy (IG) algorithm with excellent performance in DPFSP is selected. Compared with other algorithms based on metaphor or inspired by nature, the IG algorithm is simple in structure and easy to understand [15]. IG algorithm is used in the study of multi-objective flow shop scheduling problem after continuous expansion and improvement [9]. In References, IG algorithm was applied to the no-idle flow shop scheduling problem [10]. In addition，the IG with taboo reconstruction strategy is used to solve the FSP with

no-wait [11]. Ruiz improves the performance of IG algorithm by improving initialization, destruction, construction and local search [12]. Karabulut proposed an IG algorithm based on the temperature calculation formula as the acceptance criterion, and hybridized it with a random search algorithm [13]. Li et al. Proposed a new simulated annealing algorithm with four domain structures [14]. Recently, Huang proposed a restart scheme with six different operators in [2].

Our contribution is that an improve IG algorithm is proposed, in which the key factory operation exchange strategy and LS-N method based on the exchange strategy are adopted to replace the insertion strategy used in the traditional IG algorithm. Through comparison experiments with other algorithms, it is proved that the performance of the proposed algorithm is significantly improved.

II. THE SDST/DPFSP PROBLEM

SDST/DPFSP described as follows: There are n jobs, $J_j = (J_1, J_2, ..., J_n)$, which need to be processed in f identical factories, $F_l = (F_1, F_2, ..., F_f)$, and each factory has m machines, $M = (M_1, M_2, ..., M_m)$. Each job $J_j (J_j \in J)$ can be processed in any factory f. The job is processed on the machine in the order from the first machine to the last machine, and the factory cannot be replaced during processing.

The processing time of Job J_j on machine M_i is $p_{i,j}$. When Job J_j is on machine M_i, when $J_{j'}$ is the previous Job processed on the machine, SDST is $s_{i,j',j}$. Each machine can only process one Job at any time, and one Job can only be processed in the same factory. Once a factory is selected, the factory cannot be replaced during processing. All operations are independent, and all factories start processing from 0 when processing the Job.

The purpose of SDST / DPFSP in this paper is to assign jobs reasonably to the factory and find a job sequence to minimize makespan ($C_{\max}$). The mathematical model of the problem is described as follows:

Notations:

f : Number of factories.

m : Number of machines in each factory.

n : Number of jobs needed to be processed.

$J = (J_1, J_2, ..., J_n)$: The set of n jobs to be processed.

$M = (M_1, M_2, ..., M_m)$: The set of m machines, where M_i is the i^{th} machine used to complete the i^{th} process of jobs, $M_i \in M$.

$F = \{F_1, F_2, ..., F_f\}$: The set of F parallel factories, where F_l is the l^{th} factory from set F, $F_l \in F$.

$o_{i,j}$: The operation of job J_j on machine M_i.

$p_{i,j}$: The processing time of job J_j on machine M_i.

$s_{i,j',j}$: The setup time of job J_j on machine M_i, When the job is the first job processed on machine M_i, then $j' = j$.

$ST_{i,j}$: The start time of $o_{i,j}$.

$CT_{i,j}$: The complete time of job J_j on machine M_i.

$MST_{l,i,q}$: The start time of the q^{th} job of factory F_l on machine M_i.

$MCT_{l,i,q}$: The complete time of the q^{th} job of factory F_l on machine M_i.

G : A fairly large positive integer.

$C_{\max}$: The completion time of all the jobs.

Decision variables:

$x_{j,i,l,q}$: when job J_j is the q^{th} job processed in factory F_l on machine M_i, the value of the decision variable is 1, otherwise it is 0.

$y_{i,l}$: when job J_j is processed in factory F_l, the value of the decision variable is 1, otherwise it is 0.

Objective:

$$\text{Min } C_{\max} \tag{1}$$

Subject to:

$$y_{j,l} = \sum_{q=1}^{n} x_{j,i,l,q}, \forall J_j \in J, \forall M_i \in M, \forall F_l \in F \tag{2}$$

$$\sum_{j=1}^{n} x_{j,i,l,\mathrm{q}} \le 1, \forall F_l \in F, \forall M_i \in M, \forall q \in \{1,2,...,n\} \tag{3}$$

$$\sum_{j=1}^{n} x_{j,i,l,q} \ge \sum_{j'=1}^{n} x_{j',i,l,q+1}, \forall F_l \in F, \forall M_i \in M, \forall q \in \{1,2,...,n-1\} \tag{4}$$

$$ST_{i+1,\mathrm{j}} \ge CT_{i,\mathrm{j}}, \forall \mathrm{J_j} \in \mathrm{J}, \forall M_i \in \{1,2,...,m-1\} \tag{5}$$

$$MCT_{l,i,\mathrm{q}} = MST_{l,i,q} + \sum_{j=1}^{n} p_{i,j} x_{j,i,l,q}, \forall M_{\mathrm{i}} \in \mathrm{M}, \forall \mathrm{F_l} \in \mathrm{F}, \forall \mathrm{q} \in \{1,2,...,\mathrm{n}\} \tag{6}$$

$$MST_{l,i,\mathrm{q}+1} \ge MCT_{l,i,q}, \forall M_{\mathrm{i}} \in \mathrm{M}, \forall \mathrm{F_l} \in \mathrm{F}, \forall \mathrm{q} \in \{1,2,...,\mathrm{n}-1\} \tag{7}$$

$$MST_{l,s,q+1} + G(1 - \mathrm{x_{j,i,l,q}}) \ge \mathrm{MCT_{l,i,q}} + \sum_{j'=1}^{n} s_{i,j',j} x_{j',i,l,q}, \forall M_i \in M, \forall F_l \in F, \forall q \in \{1,2,...,n-1\} \tag{8}$$

$$MST_{l,\mathrm{i},1} + G(1 - \mathrm{x_{j,i,l,1}}) \ge s_{i,j,j}, \forall \mathrm{Jj} \in \mathrm{J}, \forall M_i \in M, \forall F_l \in F \tag{9}$$

$$CT_{i,\mathrm{j}} = ST_{i,\mathrm{j}} + p_{i,j}, \forall \mathrm{J_j} \in \mathrm{J}, \forall M_i \in M \tag{10}$$

$$MST_{l,i,q} \ge 0, \forall M_i \in M, \forall F_l \in F, \forall q \in \{1,2,...,n\} \tag{11}$$

$$ST_{i,\mathrm{j}} \ge 0, \forall \mathrm{J_j} \in \mathrm{J}, \forall M_i \in M \tag{12}$$

Constraint (1) indicates that the indicator of the problem studied in this paper is to minimize makespan. Constraint (2) means that each job can only be processed on one machine in a factory at a time, and constraint (3) means that each machine can only process one job at a time. Constraint (4) states that the processing of operations on the machine can only be performed sequentially, and the processing time cannot be overlapped. Constraint (5) stipulates that the processing sequence of the job cannot be changed. Constraint (6) describes the start time and completion time of a workpiece processing. Constraint (7)

represents the start time must be equal or greater than the completion time of two adjacent jobs on a certain machine. Constraint (8) describes the constraints between the start time and completion time of the job including preparation time. Constraint (9) refers to the situation when the job is first processed on the machine. In constraint (10), the completion time of a job is the sum of the start time and processing time of the job. Constraints (11) and (12) indicate that the start time of each machine and each job is not less than 0, respectively.

Next, we use an example to illustrate how to calculate makespan. The example includes two factories ($f = 2$), two machines ($m = 2$) and five jobs ($n = 5$). Table 1 and Table 2 show the processing time and setup time. It is assumed that a solution is that jobs 1, 2 and 4 are processed in factory 1, and jobs 3 and 5 are processed in factory 2. The value of decision variables can be obtained as follows: $x_{1,1,1,1}=1$, $x_{1,2,1,1}=1$, $x_{2,1,1,2}=1$, $x_{2,2,1,2}=1$, $x_{4,1,1,3}=1$, $x_{4,2,1,3}=1$, $x_{3,1,2,1}=1$, $x_{3,2,2,1}=1$, $x_{5,1,2,2}=1$, $x_{5,2,2,2}=1$, $y_{1,1}=1$, $y_{2,1}=1$, $y_{4,1}=1$, $y_{3,2}=1$, $y_{5,2}=1$, The remaining decision variables are 0. The objective value is $C_{\max}$ = 333. The scheduling Gantt is shown in Fig. 1.

TABLE I. PROCESSING TIMES OF JOBS ON MACHINES M_1 AND M_2

	J_1	J_2	J_3	J_4	J_5
M_1	3	2	5	1	2
M_2	2	4	1	3	5

TABLE II. SEQUENCE-DEPENDENT SETUP TIMES OF JOBS ON MACHINES M_1 AND M_2

$J_{j'}/J_j$	M_1					$J_{j'}/J_j$	M_2				
	J_1	J_2	J_3	J_4	J_5		J_1	J_2	J_3	J_4	J_5
J_1	6	7	9	3	5	J_1	4	1	5	2	3
J_2	4	2	6	3	7	J_2	6	7	2	1	6
J_3	8	2	4	4	6	J_3	4	3	5	1	4
J_4	1	3	9	4	5	J_4	3	9	4	7	2
J_5	7	6	3	1	2	J_5	6	1	2	3	7

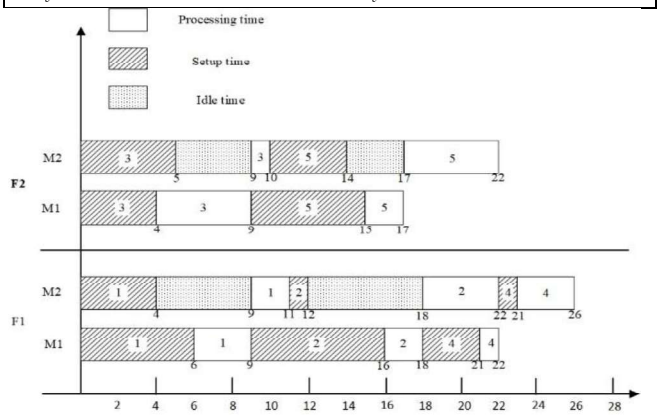


Fig. 1. Gantt Chart for the example problem

III. PROPOSED IG ALGORITHM FOR THE SDST/DPFSP

In this paper, we propose an improved iterated greedy algorithm with simple structure. IG algorithm has shown excellent performance in solving the flow shop scheduling problem [15]. For example, Ding and Song used tabu-based reconstruction strategy to enhance the search ability of the algorithm in [16]. Ruiz and Pan improved the local search, Destruction and Construction of IG algorithm, which greatly improved the performance of the algorithm [10]. The traditional IG algorithm uses a heuristic method, generally NEH, to create the initial solution. The iterative content includes four parts: destruction, reconstruction, local search and acceptance criteria. When the termination condition is satisfied, the iteration stops.

The traditional IG algorithm framework is as follows:

Algorithm 1 The traditional IG algorithm

01: **Begin:**
02: Set the parameters: T
03: $\pi^0 = InitialSolution\ M_2$
04: $\pi^0 = LocalSearch(\pi^0)$
05: **while** *termination criterion is not satisfied* **do**
06: $\pi^D, \pi^R = Destruction(\pi^0)$
07: $\pi = Construction(\pi^D, \pi^R)$
08: $\pi = LocalSearch(\pi)$
09: $\pi^0 = Accep\tan ceCriterion(\pi, \pi^0, T)$
10: **Endwhile**
11: **End**

At the same time, we use the flowchart to show the running process of the IG algorithm, as shown in Fig. 2:

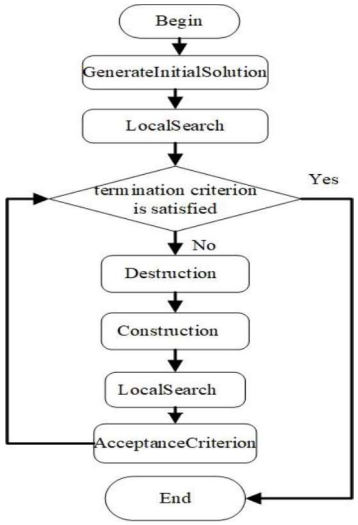


Fig. 2. IG algorithm flow chart

A. Initial Solution

The generation of initial solution is the first step of the program, and its quality is crucial. We retained NEH2_en in [12] to generate the initial solution that meets the requirements. Firstly, all jobs are sorted in descending order according to the sum of processing time on all machines to generate sequence

seeds, $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Then, the first job in the sequence is assigned to the first factory, and the second to the second factory. After all the factories are assigned, the remaining jobs are taken out one by one, and then the best position is tried and found in all sequences for insertion. After the insertion operation, the former or latter job of the insertion position is taken out, and tested at all positions of the same factory, and the insertion operation is performed at the optimal position. In addition, referring to the previous DPFSP literature, all the insertion processes adopt the well-known Taillard acceleration of the insertion neighborhood to improve the insertion speed.

The pseudocode of NEH2_en is as follows:

Algorithm 2 NEH2_en

01: **Begin:**
02: Compute $P_j = \sum_{i=1}^{m} p_{i,j}, J_j \in J$ % P_j *is the total processing time of job* J_j.
03: $\tau = \{\tau_1, \tau_2, ..., \tau_n\} \leftarrow$ sort jobs in descending order of P_j.
04: **for** $j = 1$ **to** f **do**
05: Assign job τ_j in τ to plant F_j
06: **endfor**
07: **for** $j = f + 1$ **to** n **do**
08: **for** $l = 1$ **to** f **do**
09: Test τ_j in all possible positions in π_l *% Taillard acceleration is applied*
10: Get the lowest makespan C_l' in factory F_j
11: Get the position of C_l' is pos_l
12: **endfor**
11: $l^* = \arg(\min_{l=1}^{f} C_l')$
12: Insert τ_j in the sequence π_{l^*} at position pos_{l^*}
13: Extract at random job h from position pos_{l^*}-1 or pos_{l^*}+1 from π_{l^*}
14: Insert job h in π_{l^*} at the position resulting in the lowest makespan
15: **endfor**
16: **Output** π

B. Local search

Local search in IG algorithm is an important factor to determine the quality of solution. Insertion operation is generally used in the local search of traditional IG algorithm. For example, for the case of n jobs, insertion attempts are generally required $n+1$ locations, and the time complexity is $o(n(n+1))$. If the selected location is p, the number of jobs that need to be moved is $n-p+1$, and the time complexity becomes $o(n(n+1)(n-p+1))$. If the result of the execution does not make the target value improved, the above operation will be repeated until the target value is improved, which makes the number of effective executions in a certain period of time reduced. Therefore, in this paper, we abandon the previous local operation dominated by the insertion strategy, and choose the swapping strategy based on the previous LS_3 [12]. We call it LS_N, and its time complexity is only $o(n^2)$. The operation steps of LS_N are as follows.

Firstly, find out the key factories in all factories, that is, the factory with the largest makespan, and then find out the factory with the largest makespan except the key factory. Then, two jobs are randomly selected in two factories to test the makespan $C_{\max}^*$ after the exchange. If the makespan $C_{\max}^*$ is less than the original $C_{\max}$, the exchange is retained. Otherwise, the solution remains unchanged. Then continue to select the job until all the jobs in the key factory are traversed. LS_N algorithm framework is as follows:

Algorithm 3 LS_N

01: **Begin**
02: Find the critical factory F_{c1} with the $C_{\max}$
03: Find the critical factory F_{c2} with the $C'_{\max}$ $\%\, C'_{\max}$ *is a value only less than* $C_{\max}$
04: $Cnt = 0$
05: **While** $Cnt < n$ **do** **% n is the number of jobs in factory** F_{c1}
06 $\pi^{Initial} = \pi$
06: τ' = randomly selected job in F_{c1}
07: τ'' = randomly selected job in F_{c2}
08: Test job τ' and τ'' for swap
09: $C_{\max}^*$ is the makespan of after swapping in all factories
10: **if** $C_{\max}^* > C_{\max}$
11: $\pi = \pi^{Initial}$
12: $Cnt = Cnt + 1$
13: **elseif**
14: Swap job τ' and τ''
15: $C_{\max} = \max_{f=1}^{F}\{C_{\max}(\pi_1), C_{\max}(\pi_2), ..., C_{\max}(\pi_F)\}$
16: Find the critical factory F_{c1} with the $C_{\max}$
17: Find the critical factory F_{c2} with the $C'_{\max}$
18: $Cnt = 0$
19: **endif**
20: **endwhile**
21: **Output** π

C. Destruction, construction and acceptance criteria

In the loop phase of IG algorithm, we repeat the four steps of Destruction, Construction, local search and acceptance criteria until the quality of the solution is improved. In the Destruction phase, first select a factory containing $n(n > d)$ jobs, then randomly select a job from it, add it to the sequence π^D (stored deleted jobs), and delete the job from the original sequence. Repeat the last operation d times to get two sequences, sequence π^D with d jobs and sequence π^R (store the remaining sequence of jobs without changing the sequence) with $n-d$ jobs.

The Destruction pseudo-code is given in Algorithm 4.

Algorithm 4 Destruction

01: **Begin**
02: **Procedure** Destruction(π , d)
03: $\pi^R = \pi$
04: $Cnt = 0, \pi^D = \varnothing$
05: **while** $Cnt < d$ **do**
06: $l = rand()\%f + 1$
07: **if** $n > 1$ **then**
08: $j = rand()\%\text{n}+1$
09: $\pi^D = \pi^D \cup \{\pi^R_{l,j}\}$
10: Delete job $\pi^R_{l,j}$ from π^R
11: $Cnt++$
12: **endif**
13: **endwhile**
14: Output π^D, π^R

The reconstruction phase is, firstly, the first job is taken out from π^D, and it is tried in all positions until the best insertion position is found, that is, the position with the smallest makespan. Then, the second job is taken out, and the above operation is performed… until all the jobs in π^D are taken out and inserted.

The Construction pseudocode is given in algorithm 5.

Algorithm 5 Construction

01: **Begin**
02: **Procedure** Construction (π^R , π^D)
03: **for** $j = 1$ **to** d
04: $Job =$ the j^{th} job of π^D
05: **for** $l = 1$ **to** f **do**
06: Test Job is inserted in all positions of π^R_l
07: Get the lowest makespan C^*_l in factory F_l
08: Get the position of C^*_l is pos_l
09: **endfor**
10: $l^* = \arg(\min^f_{l=1} C_l')$
11: Insert Job in the sequence $\pi^R_{l^*}$ at position pos_{l^*}
12: **endfor**
13: $\pi = \pi^R$
14: **Output** π

Since the code referenced in this paper is the code in Ruiz ' s [12], the code in the ' Receiving Criteria ' section still chooses Ruiz and Stützle (2007), a constant temperature acceptance criterion based on parameter T , as follows:

$$Temperature = T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{i,j}}{n \cdot m \cdot 10} \qquad (14)$$

T needs calibration, but has shown robustness (most values are not zero and not too high).

IV. Experiment and Analysis

The performance of the NIG algorithm is evaluated using 270 examples. In the example, n , m and f is the number of jobs, machines and factories, respectively. Their values are as follows: $n \in \{100, 200, 300, 400, 500\}$, $m \in \{5, 8, 10\}$, $f \in \{2, 3, 4, 5, 6, 7\}$. We randomly combine their values to generate instances of different sizes. In the example, the data are generated in a random way. The distribution range of processing time is [1, 99), and the value range of preparation time is $(1 + rand()\%99) \times factor / 100$. In this experiment, we repeat each instance five times, and each instance takes the minimum value in five runs. The index of this experiment is to minimize makespan ($C_{\max}$).

All test algorithms are compiled and coded by Visual Studio 2019 ,C++, running on Microsoft Windows 10 operating system, 16GB DDR4 memory and 1.00 GHZ Intel Core i5 - 1035G1 processor.

A. Computational evaluation

The DPFSP is a NP-hard problem, thus we cannot obtain the optimal solution. Therefore, in this paper, we will use the performance evaluation method of the single objective optimization algorithm commonly used in the literature to evaluate the performance of the algorithm, namely the percentage relative deviation RPI. The calculation method of RPI is as follows:

$$RPI = \frac{M_i - M_{best}}{M_{best}} \times 100\% \qquad (15).$$

Among them, M_i is the minimum makespan of the algorithm running five times in an instance, and M_{best} is the minimum makespan of all the algorithms running five times in this instance. It can be seen from the formula that in the comparison algorithm, the better the performance is, the closer the RPI is to 0, and the algorithm with RPI equals to 0 has the best performance in the current comparison algorithm.

In this paper, five algorithms are selected to compare with NIG, namely discrete artificial bee colony algorithm (DABC) [17], artificial chemical reaction optimization (CRO) [3], discrete differential evolution algorithm (DDE) [18], and improved iterated greedy algorithm IGA [12] and IGR [2]. For the 270 examples mentioned above, all algorithms are executed five times in comparison. In order to obtain fair results, we set the termination condition as:

$$TimeLimit = CPU \times n \times m \qquad (16)$$

n and m are the number of jobs and the number of machines in the current instance respectively.

B. RPI comparison

Table 3 shows the RPI under different instances, and the average RPI of each algorithm under different factory numbers. According to the above formula for calculating RPI, the performance of NIG algorithm is significantly better than that of the other five algorithms. With the increase of examples, the performance of DABC, DDE and IGA algorithms gradually deteriorates, while the performance of IGR gradually improves. Overall, the performance of DDE and IGR is good, but the performance of NIG is the best.

TABLE III. RPI OF COMPARISON ALGORITHM WHEN CPU = 10

Factory	J*M	DABC	CRO	DDE	IGA	IGM	NIG
f=2	100*5	0.054	0.049	0.015	0.039	0.018	0
	100*8	0.053	0.077	0.015	0.079	0.046	0
	100*10	0.082	0.064	0.021	0.050	0.031	0
	200*5	0.062	0.028	0.022	0.053	0.030	0
	200*8	0.067	0.035	0.010	0.052	0.033	0
	200*10	0.067	0.053	0.009	0.069	0.039	0
	300*5	0.051	0.037	0.010	0.046	0.037	0
	300*8	0.051	0.038	0.011	0.064	0.033	0
	300*10	0.055	0.028	0.008	0.037	0.034	0
	400*5	0.055	0.040	0.018	0.081	0.041	0
	400*8	0.043	0.028	0.010	0.060	0.033	0
	400*10	0.037	0.026	0.011	0.040	0.029	0
	500*5	0.038	0.030	0.015	0.044	0.024	0
	500*8	0.043	0.024	0.012	0.047	0.031	0
	500*10	0.040	0.031	0.005	0.038	0.028	0
	mean	**0.053**	**0.039**	***0.013***	**0.053**	**0.032**	**0**
f=3	100*5	0.052	0.057	0.003	0.043	0.031	0
	100*8	0.105	0.097	0.026	0.070	0.060	0
	100*10	0.073	0.082	0.021	0.055	0.034	0
	200*5	0.046	0.042	0.018	0.046	0.024	0
	200*8	0.076	0.061	0.005	0.066	0.038	0
	200*10	0.066	0.055	0.009	0.073	0.041	0
	300*5	0.060	0.036	0.006	0.032	0.022	0
	300*8	0.043	0.057	0.007	0.062	0.043	0
	300*10	0.054	0.031	0.004	0.051	0.033	0
	400*5	0.036	0.025	0.008	0.040	0.025	0
	400*8	0.057	0.042	0.004	0.055	0.035	0
	400*10	0.055	0.047	0.001	0.044	0.029	0
	500*5	0.026	0.029	0.006	0.035	0.019	0
	500*8	0.038	0.043	0.000	0.061	0.036	0.001
	500*10	0.043	0.021	0.005	0.047	0.029	0
	mean	**0.055**	**0.048**	***0.008***	**0.052**	**0.033**	**0.00009**
f=4	100*5	0.058	0.049	0.010	0.040	0.019	0
	100*8	0.067	0.086	0.019	0.057	0.024	0
	100*10	0.082	0.085	0.026	0.047	0.039	0
	200*5	0.054	0.047	0.008	0.034	0.017	0
	200*8	0.051	0.065	0.0003	0.061	0.039	0
	200*10	0.083	0.071	0.008	0.049	0.040	0
	300*5	0.068	0.055	0.013	0.055	0.026	0
	300*8	0.065	0.047	0.011	0.078	0.036	0
	300*10	0.066	0.036	0.006	0.047	0.033	0
	400*5	0.039	0.044	0.009	0.037	0.016	0
	400*8	0.073	0.042	0.003	0.048	0.030	0
	400*10	0.060	0.046	0.006	0.050	0.035	0
	500*5	0.053	0.038	0.016	0.054	0.030	0
	500*8	0.066	0.037	0.018	0.052	0.028	0
	500*10	0.056	0.047	0.028	0.066	0.040	0
	mean	**0.063**	**0.053**	***0.012***	**0.052**	**0.030**	**0.000**
f=5	100*5	0.054	0.027	0.009	0.018	0.007	0
	100*8	0.114	0.080	0.026	0.053	0.029	0
	100*10	0.116	0.115	0.060	0.061	0.051	0
	200*5	0.074	0.083	0.040	0.081	0.041	0
	200*8	0.095	0.069	0.000	0.056	0.045	0.005

	200*10	0.102	0.072	0.020	0.073	0.046	0
	300*5	0.057	0.039	0.007	0.048	0.015	0
	300*8	0.083	0.062	0.021	0.063	0.035	0
	300*10	0.107	0.069	0.006	0.073	0.044	0
	400*5	0.064	0.055	0.004	0.060	0.027	0
	400*8	0.077	0.054	0.017	0.055	0.029	0
	400*10	0.078	0.053	0.016	0.061	0.039	0
	500*5	0.062	0.046	0.014	0.064	0.030	0
	500*8	0.070	0.043	0.007	0.063	0.032	0
	500*10	0.080	0.041	0.022	0.066	0.042	0
	mean	**0.082**	**0.060**	***0.018***	**0.060**	**0.034**	**0.000348**
f=6	100*5	0.101	0.100	0.032	0.052	0.041	0
	100*8	0.116	0.072	0.024	0.059	0.042	0
	100*10	0.100	0.103	0.030	0.056	0.046	0
	200*5	0.087	0.074	0.025	0.055	0.030	0
	200*8	0.099	0.065	0.014	0.058	0.035	0
	200*10	0.093	0.083	0.058	0.048	0.042	0
	300*5	0.071	0.046	0.040	0.056	0.024	0
	300*8	0.090	0.059	0.034	0.070	0.033	0
	300*10	0.099	0.065	0.058	0.064	0.050	0
	400*5	0.083	0.054	0.061	0.063	0.029	0
	400*8	0.084	0.049	0.045	0.049	0.031	0
	400*10	0.095	0.047	0.028	0.067	0.034	0
	500*5	0.069	0.045	0.025	0.055	0.025	0
	500*8	0.074	0.050	0.040	0.050	0.030	0
	500*10	0.078	0.056	0.032	0.070	0.039	0
	mean	**0.089**	**0.065**	**0.036**	**0.058**	***0.035***	**0.000**
f=7	100*5	0.112	0.031	0.016	0.056	0.027	0
	100*8	0.082	0.079	0.049	0.060	0.014	0
	100*10	0.123	0.086	0.043	0.044	0.036	0
	200*5	0.114	0.063	0.024	0.059	0.029	0
	200*8	0.120	0.095	0.024	0.073	0.037	0
	200*10	0.102	0.082	0.048	0.050	0.039	0
	300*5	0.090	0.059	0.043	0.128	0.029	0
	300*8	0.127	0.071	0.048	0.076	0.043	0
	300*10	0.102	0.066	0.041	0.066	0.041	0
	400*5	0.073	0.047	0.039	0.039	0.018	0
	400*8	0.101	0.053	0.052	0.064	0.030	0
	400*10	0.104	0.043	0.018	0.070	0.034	0
	500*5	0.067	0.044	0.052	0.052	0.018	0
	500*8	0.074	0.053	0.036	0.061	0.027	0
	500*10	0.090	0.062	0.046	0.073	0.039	0
	mean	**0.099**	**0.062**	**0.039**	**0.065**	***0.031***	**0.000**

To identify the acquired data more clearly, we also use confidence intervals to analyze the data, as shown in Fig. 3. It can be seen from the graph that the performance of NIG algorithm is significantly better than other algorithms. In addition, DDE algorithm and IGR algorithm are also better, while DABC and IGA are relatively worse.

Next, we further analyze the convergence of the algorithm. We have selected four algorithms, DDE, IGA, IGR, and NIG, which perform well in the problem, as shown in Fig. 4. In order to make the expression clearer, we have selected three question examples of 300*5*3,400*5*6, 500*5*6. Since the termination condition of the programming operation is $TimeLimit = CPU \times n \times m$, we can change the running time of the algorithm by changing the value of the CPU. Therefore, the X axis in Fig. 3 is the value of the CPU, the optimal solution (makespan) obtained by running the algorithm five times is the Y axis. It can be seen from the three figures a, b, and c that DDE is easy to fall into the local optimum. The performance of the IGR algorithm gradually increases with the increase of the instance size. The NIG algorithm performs best in these four algorithms.

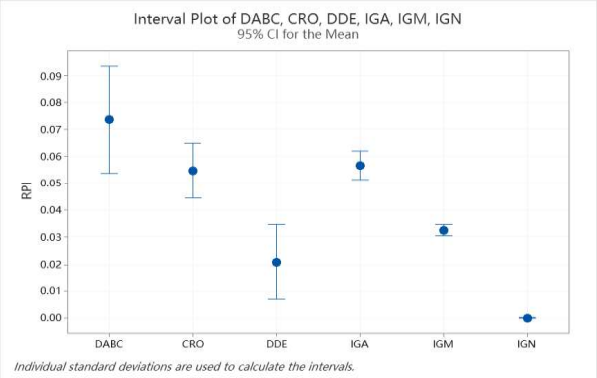
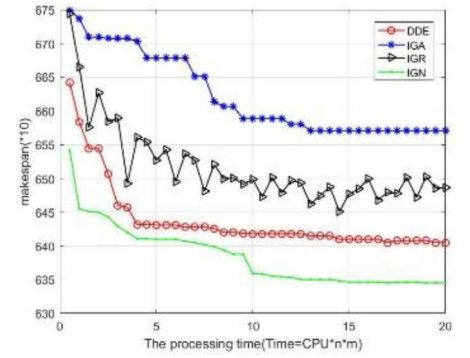
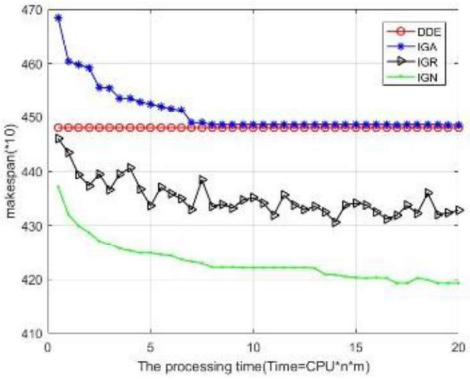


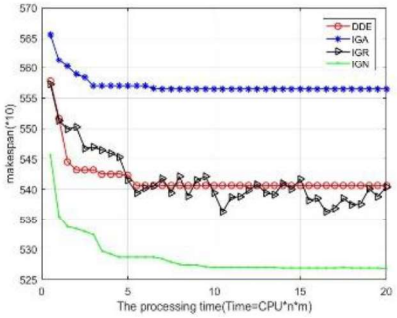
Fig. 3. Confidence Interval Diagram of the Algorithm with CPU = 10



(a) 300*5*3



(b) 400*5*6



(c) 500*5*6

Fig. 4. Convergence curve of DDE, IGA, IGR, NIG

V. CONCLUSIONS

In this paper, the DPFSP with sequence-dependent preparation time is studied and solved by the improved IG algorithm. Firstly, the mathematical model of SDST / DPFSP is described. Then, this paper proposes a local search based on exchange strategy, which improves the search efficiency. Through compared with other existing algorithms, the performance has been significantly improved.

In the future, we will do more research on problem-oriented strategies, focus more on algorithm thinking and consider different optimization objectives. For example, the green energy saving index is added to the problem of distributed displacement flow shop, and more attention is paid to the thinking of problems related to real life.

ACKNOWLEDGMENT

This work was jointly supported by National Natural Science Foundation of China with grant No. 61803192, 61973203, 61966012, 61773192, 61603169, 61773246, and 71533001. Thanks for the support of Shandong province colleges and universities youth innovation talent introduction and education program.

REFERENCES

[1] B. Naderi, and R. Ruiz . "The distributed permutation flowshop scheduling problem," Computers& Operations Research 37.4(2010):754-768.

[2] J. P. Huang , Q. K. Pan , and L. Gao . "An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times," in Swarm and Evolutionary Computation (2020):100742.

[3] Bargaoui, Hafewa , O. Belkahla Driss , and Ghédira, Khaléd. "A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion," in Computers & Industrial Engineering 111.sep.(2017):239-250.

[4] Fernandez-Viagas, Victor , P. Perez-Gonzalez , and J. M. Framinan . "The distributed permutation flow shop to minimise the total flowtime," Computers & Industrial Engineering 118.APR.(2018):464-477.

[5] T. Meng, Q. K. Pan, and L. Wang. "A distributed permutation flowshop scheduling problem with the customer order constraint," Knowledge-Based Systems. 184. 104894. 10.1016/j.knosys.2019.104894.

[6] X. L. Jing, Q. K. Pan, L. Gao, and Y. L. Wang. "An effective iterated greedy algorithm for the distributed permutation flowshop scheduling with due windows," Applied Soft Computing, 96, 106629.

[7] J. P. Huang, Q. K. Pan, Z. H. Miao, and L. Gao. "Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times," Engineering Applications of Artificial Intelligence, 97, 104016.

[8] Kenneth, and Sörensen. "Metaheuristics — the metaphor exposed," International Transactions in Operational Research (2015).

[9] R. Ruiz, and T. Stützle. "An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," European Journal of Operational Research 187.3(2008):1143-1159.

[10] R. Ruiz, Q. K. Pan. "An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem," Omega: The international journal of management science, 44(Apr.2014), 41-50.

[11] D. Yüksel, M. F. Tagetiren, L. Kandiller , and L. Gao. "An energy-efficient bi-objective no-wait permutation flowshop scheduling problem to minimize total tardiness and total energy consumption," Computers & Industrial Engineering, 145, 106431, 2020.

[12] R. Ruiz, Q. K. Pan , and B. Naderi . “Iterated Greedy methods for the distributed permutation flowshop scheduling problem,” Omega 83.MAR.(2019):213-222.

[13] K. Korhan . “A Hybrid Iterated Greedy Algorithm for Total Tardiness Minimization in Permutation Flowshops,” Computers & Industrial Engineering 98.aug.(2016):300-307.

[14] W. Li, J. Li, K. Gao, Y. Han, and Q.Sun. “Solving robotic distributed flowshop problem using an improved iterated greedy algorithm,” International Journal of Advanced Robotic Systems, (2019) 16(5), 172988141987981-.

[15] R. Ruiz, and T. Stützle. “A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem,” European Journal of Operational Research 177. 3(2007):2033-2049.

[16] J. Y. Ding, C. Raymond, R. Zhang. “An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem,” Applied Soft Computing 30(2015):604-613.

[17] J. Pan, W. Zou, J. Duan, “A discrete artificial bee colony for distributed permutation flowshop scheduling problem with total flow time minimization,” 2018 37th Chinese Control Conference (CCC), Wuhan, 2018, 8379-8383.

[18] G. H. Zhang , K. Xing , and F. Cao . “Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion,” Engineering Applications of Artificial Intelligence 76.NOV.(2018):96-107.