COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 03/06 pp 198−217
Volume 1, Number 3, September 2021
DOI: 10.23919/CSMS.2021.0018

Distributed Flow Shop Scheduling with Sequence-Dependent Setup Times Using an Improved Iterated Greedy Algorithm

Xue Han, Yuyan Han*, Qingda Chen, Junqing Li, Hongyan Sang,
Yiping Liu, Quanke Pan, and Yusuke Nojima

Abstract: To meet the multi-cooperation production demand of enterprises, the distributed permutation flow shop scheduling problem (DPFSP) has become the frontier research in the field of manufacturing systems. In this paper, we investigate the DPFSP by minimizing a makespan criterion under the constraint of sequence-dependent setup times. To solve DPFSPs, significant developments of some metaheuristic algorithms are necessary. In this context, a simple and effective improved iterated greedy (NIG) algorithm is proposed to minimize makespan in DPFSPs. According to the features of DPFSPs, a two-stage local search based on single job swapping and job block swapping within the key factory is designed in the proposed algorithm. We compare the proposed algorithm with state-of-the-art algorithms, including the iterative greedy algorithm (2019), iterative greedy proposed by Ruiz and Pan (2019), discrete differential evolution algorithm (2018), discrete artificial bee colony (2018), and artificial chemical reaction optimization (2017). Simulation results show that NIG outperforms the compared algorithms.

Key words: distributed permutation flow shop; iterated greedy; local search; swapping strategy

1 Introduction

Under the influence of globalization, distributed manufacturing and scheduling have become a trend in the production industry because many enterprises are gradually turning to multiregional cooperation. In this context, enterprises need several production centers and must establish a distributed production model[1, 2]. Thus, the distributed production process flow has gradually attracted researchers' attention and become a hotspot for research[2]. The objective of the distributed permutation flow shop scheduling problem (DPFSP) is to assign some jobs to a factory and to balance the efficiency of all the factories. Thus, the DPFSP consists of two subproblems: the first is the distribution of jobs among factories, and the second is the scheduling sequence of jobs to be processed on machines, which demonstrates that DPFSP is more complicated than the traditional performance flow shop scheduling problem.

In actual factory production, operations such as machine maintenance or blade replacement are often required after a job is processed on the machine, thereby creating extra time. These times become sequence-dependent setup times (SDST) when their length is related to the job being processed and to the

• Xue Han, Yuyan Han, and Hongyan Sang are with the School of Computer Science, Liaocheng University, Liaocheng 252000, China. E-mail: 1979124154@qq.com; hanyuyan@lcu-cs.com; sanghongyan@lcu-cs.com.
• Qingda Chen is with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China. E-mail: cqd0309@126.com.
• Junqing Li is with the School of Information Science and Engineering, Shandong Normal University, Jinan 252000, China. E-mail: lijunqing@lcu-cs.com.
• Yiping Liu is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China. E-mail: yiping0liu@gmail.com.
• Quanke Pan is with the School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, China. E-mail: panquanke@mail.neu.edu.cn.
• Yusuke Nojima is with the Department of Computer Science and Intelligent Systems, Osaka Prefecture University, Osaka 599-8531, Japan. E-mail: nojima@cs.osakafu-u.ac.jp.
* To whom correspondence should be addressed.
Manuscript received: 2021-05-31; revised: 2021-07-13; accepted: 2021-08-15

previous job[3]. Therefore, this paper considers the SDSTs and addresses the DPFSP-SDST to minimize makespan.

We note that intelligent optimization algorithms based on metaphors or inspired by nature have been developed to solve the DPFSP, such as the estimation of distribution algorithm (EDA)[4], the chemical reaction optimization (CRO) algorithm[5], the discrete artificial bee colony (DABC) algorithm[6], and the iterative greedy (IG) algorithm[7–9]. These swarm intelligence algorithms can provide multiple solutions that are helpful in improving the diversity of solutions. However, in the exploration of a single solution neighborhood, these swarm intelligence algorithms are slightly less effective than the IG algorithm. Especially for flow shop scheduling problems, few local optimums exist. Thus, we can select an algorithm with good local exploitation ability to optimize the above problems. Compared with traditional swarm intelligence algorithms, the IG algorithm is a simple and effective optimization algorithm that has shown excellent local exploitation ability when solving scheduling problems[6–9].

The IG algorithm is characterized by ease of implementation, simple structure, few parameters, and few mathematical requirements. It has two key stages: destruction and construction, and local search, thereby making it a parallel search framework[1]. Thus, many heuristics, metaheuristics, and problem-dependent local search methods, as well as operators, can be embedded into the above search framework to further enhance exploration and exploitation. Since the proposal of the IG algorithm by Ruben and Thomas, it has been continuously expanded and improved for solving flow shop scheduling problems[1]. From Ref. [10], the simulation experimental results verify that the IG algorithm is appropriate and competitive for solving discrete optimization problems. To the best of our knowledge, the IG algorithm has not yet been well studied up to the present for the DPFSP-SDST. With the above motivations, we propose an improved IG algorithm to solve the DPFSP-SDST.

We know that the DPFSP has two key issues to be solved: how to allocate jobs to appropriate factories and how to generate the scheduling sequence of operations on machines with minimal makespan. To tackle these issues effectively, our contribution to the algorithm is twofold.

The process of allocating jobs to appropriate factories is often finished after the initialization solution is generated. In this paper, an allocation strategy based on the idle time of a factory and an independent insertion operator is adopted according to the initialization scheduling sequence.

Based on the distributed feature of DPFSP-SDST, a two-stage local search strategy based on a single job exchange and a job block swapping is proposed to disturb the current solution within the key factory. This two-stage local search has low computational complexity, and it has more iterations and opportunities to improve the quality of the solution than the algorithms based on insertion operators.

This paper is organized as follows: Section 2 reviews some related references. Section 3 introduces the mathematical model of DPFSP-SDST. Section 4 introduces the IG algorithm and its improvement. Section 5 provides experimental results and analysis. The last section summarizes the strengths of our algorithms and gives some perspectives for future works.

2 Literature Review

The DPFSP-SDST is a multifactory production model; thus, it has significant applications in real-world problems. Therefore, this problem has attracted the attention of many researchers. In this section, we have summarized some single- and multi-objective DPFSPs and the IG algorithm in recent years.

The change in production from a single factory to multiple factories is to reduce processing times while increasing processing efficiency or reducing energy consumption. Thus, some significant research has been conducted on the DPFSPs with the makespan minimization, and many improvements of the algorithms have been made, such as an efficient EDA[4], a scattered search (SS) algorithm containing restart and local search strategies[11], and a CRO algorithm[5]. Recently, for the same criterion, Zhao et al.[12] integrated two heuristics and a stochastic policy to generate an initial solution and proposed the ensemble discrete differential evolution algorithm. Meng et al.[13] studied the three metaheuristics, namely, variable neighborhood descent, artificial bee colony, and IG. Li et al.[14] employed an improved DABC algorithm, and the experimental results show that the performance of the DABC is better than that of the genetic algorithm and IG algorithm.

Except for the makespan, the objective of the total

flow time is also important. Thus, many studies have been developed for DPFSPs with the minimization of the total flow time. Fernandez-Viagas et al.[15] proposed some constructive heuristics and an iterative improvement algorithm to minimize the total flow time. Next, Pan et al.[16] designed the three constructive heuristics and four metaheuristics algorithms based on a high-performance framework of a discrete artificial swarm, SS, iterative local search, and IG. Recently, for the same objective, Zhang et al.[17] proposed an innovative 3D matrix-based distribution-based estimation algorithm, and Song and Lin[18] proposed a genetic program-based hyper-heuristic algorithm.

In various real-world applications, the DPFSPs optimize not only one objective but also several objectives. Deng and Wang[19] proposed a competitive modal algorithm to optimize the two objectives of the makespan and total tardiness criteria. Wang et al.[20] employed a multiobjective whale swarm algorithm (MOWSA), in which a problem-specific coding scheme, crossover, and variational operations, as well as efficient local search, are proposed to solve multiobjective DPFSP-SDST. Furthermore, Wang et al.[21] adopted the above-improved MOWSA to solve the energy-efficient DPFSP. Chen et al.[22] proposed a collaborative optimization algorithm using attributes and some synergistic mechanisms for reducing makespan and total energy consumption.

In production, after a machine has finished machining a job, it often takes a certain amount of time for operations, such as tool changes and machine maintenance. When these operations are associated with two jobs before and after machining on the same machine, this time is referred to as SDST. SDST allows for a more precise consideration of setup time and is more coincident with the actual production activities in most factories. Therefore, the study of SDST is more relevant than the common fixed lead time. Most of the early studies on SDST were conducted on specific real-world problems. For example, Parthasarathy and Rajendran[3] studied the problem of a flow shop of the production of drill bits. Later, Mirabi[23] proposed an improved ant colony optimization algorithm to solve the permutation flow shop scheduling problem with SDST/PFSP. For the same problem, an improved neighborhood-based heuristic[24], an enhanced migrating birds optimization algorithm[25], and an effective DABC algorithm[6] are proposed to optimize the makespan of DPFSP-SDST. In addition, for SDST-PFSP with the total process time, Nagano et al.[26] applied a new construction heuristic called QUARTS to solve the above problem.

In the existing literature, some researchers proposed many excellent heuristic and metaheuristic algorithms for distributed flow shops with different constraints[27]. Li et al.[28] proposed a DABC algorithm to solve the distributed heterogeneous no-wait flow shop scheduling problem. Next, considering the distributed heterogeneous hybrid flow shop scheduling problem with unrelated parallel machines and the SDST, Li et al.[29] studied a machine position-based mathematical model and designed an improved artificial bee colony algorithm. For the distributed assembly flow shop scheduling problem, Zhao et al.[30] proposed a cooperative water wave optimization algorithm to minimize the maximum assembly completion time. Next, based on the features of the same problem mentioned above, Shao et al.[31] considered a constructive heuristic based on a new assignment rule of jobs and a product-based insertion procedure.

Among the above algorithms, the IG algorithm has shown good performance in solving PFSP[10]. With its simplicity, ease of operation, and superiority over many other metaphor-based algorithms[10, 32, 33], IG has attracted great attention from researchers for use in solving various PFSP problems. IG was first proposed to solve DPFSP by Naderi and Ruiz[1], and the experimental results show its great performance. The use of insertion operations in the local search phase in their work leads to an improvement in the quality of the solution. Subsequently, scholars and producers have spent many efforts to modify the IG algorithms and achieve significant improvements in their performance[34−37]. More recently, Fernandez-Viagas and Framinan[38] compared the existing IG algorithms and their variants to derive a new best-in-class algorithm. Mao et al.[39] improved the initial phase of the IG algorithm and the damage reconstruction phase. Instead of applying a simple simulated annealing criterion to the IG algorithm[40], Lin et al.[34] used an acceptance criterion with a settling temperature value and included the number of elements to be removed in the destruction step as a variable. Ruben et al.[7] employed an improved IG algorithm to optimize the makespan of DPFSP. Jing et al.[41] adopted an improved IG algorithm to solve the DPFSP with windows. Huang et al.[8] combined the proposed six different operators with the IG algorithm to greatly

improve the performance of the IG algorithm and applied two different local searches based on insertion operations to improve the quality of the solution.

In summary, research on the above DPFSP-SDSTs is relatively few. In addition, although IG has shown good performance in solving DPFSP, for most existing IG algorithms, the local search based on insertion operator is often adopted. We know that the insertion operations require more running time and will lose opportunities to generate promising solutions by several iterations. Thus, efforts are needed to reduce the time complexity of the IG algorithm. In this paper, a two-stage local search strategy is designed and integrated into the IG algorithm.

3 SDST-DPFSP Problem

For example, the parallelization of cutoff pair interactions is mature on CPUs and typically employs a voxel-based method.

DPFSP-SDST has been described as follows: There are n jobs, which need to be processed in f identical factories, and each factory has m machines. This problem has the following constraints: (1) Each job can be processed in any factory. (2) The job is processed in the order from the first machine to the last machine, and the factory cannot be changed during processing. (3) Each machine can process only one job at any time. (4) Only one job can be processed in the same factory. (5) All operations are independent, and all factories start processing from 0 moment when processing the job. The purpose of DPFSP-SDST 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 and its notations are described as follows:

Notations:

f: The number of factories.

m: The number of machines in each factory.

n: The number of jobs that need 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 is M_i machine used to complete the q-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 completion 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 completion time of the q-th job of factory F_l on machine M_i.

$C_{max}(\pi_{F_f})$: The completion time of the jobs processed in factory F_f.

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 on machine M_i in factory F_l, 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:

$$\text{Min}C_{max} = max_{i=1}^{f}\left\{C_{max}(\pi_{F_1}), C_{max}(\pi_{F_2}), ..., C_{max}(\pi_{F_f})\right\} \tag{1}$$

Subject to

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

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

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

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

$$MCT_{l,i,q} = MST_{l,i,q} + \sum_{j=1}^{n} p_{i,j} x_{j,i,l,q}, \forall M_i \in M, \forall F_1 \in F, \forall q \in \{1,2,...,n\} \tag{6}$$

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

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

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

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

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

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

Equation (1) is the objective function to be minimized. Constraint (2) is that each job can be processed on only one machine in a factory at a time, and Constraint (3) means that each machine can process only one job at a time. Constraint (4) states that the processing of operations on the machine can be performed only sequentially, and the processing time cannot be overlapped. Constraint (5) shows that the processing sequence of the job cannot be changed. Constraint (6) describes the start time and completion time of a job processing. Constraint (7) represents that 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 the 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.

The following example illustrates a scheduling case considered in DPFSP-SDST. Suppose that there is a scheduling sequence with six jobs, two factories, and two machines per factory. The processing time of the 6 jobs on two machines is (4, 3, 1, 3, 6, 8) and (3, 7, 2, 1, 9, 4), respectively. The setup times of the six jobs on two machines with different sequences are shown in Table 1. The jobs assigned to the first factory are 2, 1, 5, and the jobs assigned to the second factory are 4, 3, 6. For decision variables, $x_{2,1,1,1} = 1$, $x_{1,2,1,1} = 1$, $x_{5,3,1,1} = 1$, $x_{2,1,1,2} = 1$, $x_{1,2,1,2} = 1$, $x_{5,3,1,2} = 1$, $x_{4,1,2,1} = 1$, $x_{3,2,2,1} = 1$, $x_{6,3,2,1} = 1$, $x_{4,1,2,2} = 1$, $x_{3,2,2,2} = 1$, $x_{6,3,2,2} = 1$, $y_{2,1} = 1$, $x_{2,1,1,1} = 1$, $y_{5,1} = 1$, $y_{4,2} = 1$, $y_{3,2} = 1$, and $y_{6,1} = 1$. The remaining decision variables are 0. Table 1 presents the setup time for jobs on different machines.

Figure 1 gives the Gantt chart of the scheduling sequence (2, 4, 1, 3, 6, 5) on the two machines and factories. In this study, we consider minimizing the maximum makespan of scheduling. From this Gantt chart, we see that the value of makespan is 38 units of the time given by the maximum completion time of the last job on the second machine in the two factories. It is easy to understand that if the number of factories is equal to one, then the makespan must be larger than 38.

Table 1 Sequence-dependent setup times $s(i,j',j)$ of jobs on machines M_1 and M_2

J'_j	M_1						M_2					
	J_1	J_2	J_3	J_4	J_5	J_6	J_1	J_2	J_3	J_4	J_5	J_6
J'_1	4	3	6	1	2	4	1	4	6	7	3	2
J'_2	7	8	2	7	5	1	5	2	8	1	7	8
J'_3	4	5	8	1	3	7	2	3	2	4	6	3
J'_4	1	2	6	9	4	9	9	8	4	2	1	3
J'_5	3	7	5	4	8	6	6	7	1	5	2	3
J'_6	3	6	1	2	6	4	4	5	3	1	2	6

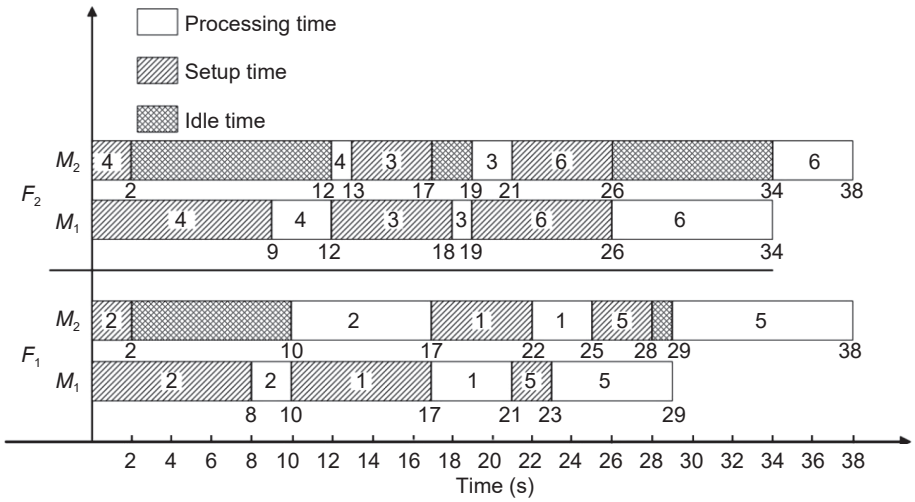


Fig. 1 Gantt chart for a solution to the example problem.

From the Gantt chart, we can see that distributed flow shop scheduling can reduce the production cycle or completion time, accelerate the manufacturing process, and enhance production efficiency. Thus, it has applications that are more important in manufacturing than the traditional PFSPs.

4 Iterative Greedy Algorithm

As mentioned above, the IG algorithm is a simple local search method. In the framework of the IG algorithm, two local searches are performed. However, most local search methods are based on insertion operations, which will consume running time. In addition, for the distributed characteristic, DPFSP-SDST is lack of operators assigning jobs to factories. The above difficulties may be encountered when solving the DPFSP-SDST. To address the above drawbacks, this section presents an improved IG algorithm for solving the DPFSP- SDST, including the initialization, the two local searches, destruction and construction, and acceptance criteria. In the proposed IG algorithm, two local search strategies based on job swapping and job block swapping within the key factory are proposed. Notably, a discrete job permutation-based coding scheme is utilized to directly solve the discrete problem considered in this paper. Algorithm 1 shows the main structure of the proposed IG algorithm. In the next section, we describe each of its components.

4.1 Initialization method

The construction heuristic aims to obtain a feasible solution in a reasonable amount of time. The quality of the solution obtained using a constructive heuristic is better than that obtained by using a random method to generate the initial solution. A good initial solution can enhance the convergence of an algorithm. The relevant references on the constructive heuristic algorithms prove their excellent performance in solving flow shop scheduling problems[6, 31, 42–45]. Thus, a construction heuristic is adopted in initializing the solution.

The IG algorithm usually uses a heuristic, i.e., NEH (Nawaz, Enscore, and Ham), to generate the initial solution. To solve the DPFSP, Naderi and Ruiz[1] used a modified NEH called NEH2 to generate the initial solution. In 2019, Ruiz et al.[7] extended NEH2, called NEH2_en. The experimental results demonstrated that NEH2_en performs better when optimizing the makespan of the DPFSP than other heuristics[7]. Thus, in this paper, we adopted NEH2_en to generate the initial solution. The main steps are as follows: (1) The sum of the processing times for each job on all machines is calculated; (2) A nonincreasing sequence based on the above sum values is obtained; (3) The first jobs in the sequence are sequentially assigned to each factory, and the remaining jobs are taken out of the sequence in turn; (4) All locations in all factories are evaluated until the location pos_{l^*}, and when the smallest makespan is found, it is inserted; (5) After the insert operation, the job before or after the position pos_{l^*} is randomly extracted and tested in all positions in the same factory. (6) Steps 3 and 4 are repeated until all jobs have been inserted. Algorithm 2 lists the initialization procedure.

Algorithm 1 Two-stage IG algorithm

1: Defining constants d, T

2: $\pi = GenerateInitialSolution$

3: $\pi^0 = LocalSearch(\pi) \rightarrow LS\_N$// The new local search based on single job swapping is applied

4: While (Satisfying the cyclic condition) do

5: $\pi^D = \pi^R = Destruction(\pi^0, d)$

6: $\pi' = Construction(\pi^D, \pi^R)$

7: $\pi'' = LocalSearch(\pi') \rightarrow LS\_N2$// The local search based job block swapping is proposed.

8: $\pi^0 = AcceptaneCriterion(\pi'', \pi^0, T)$

9: end while

Algorithm 2 Initialization based on NEH2_en

1: Calculate $TotalP_j = \sum_{i=1}^{m} p_i, Jj \in J$ ($TotalP_j$ is the total processing time for job J_j)

2: $\tau = \tau_1, \tau_2, ..., \tau_n$ (sort jobs according to the decreasing $TotalP_j$)

3: for $j = 1$ to f

4: Take job τ_i from τ and assign it to factory F_j

5: endfor

6: for $j = f + 1$ to n do

7: for $l = 1$ to f

8: Test τ_j in all possible positions in π_l// Taillard acceleration is applied

9: C'_l is the lowest makespan of factory F_j obtained

10: pos_l is the position where the C'_l is generated

11: endfor

12: $l\times = arg(min_{l=1}^{f} C'_l)$

13: Insert τ_j in the sequence π_{l^*} at position pos_{l^*}

14: Extract at random job h from position $pos_{l^*} - 1$ or $pos_{l^*} + 1$ from π_{l^*}

15: Test job h in all possible positions of π_{l^*}

16: Insert job h in π_{l^*} at the position resulting in the lowest makespan

17: end for

4.2 New local search based on single swapping

For example, the parallelization of cutoff pair interactions is mature on CPUs and typically employs a voxel-based method.

In most IG algorithms, the local search based on insertion operator is adopted. However, the insertion operations consume running time and will lose opportunities to generate promising solutions by several iterations. Thus, in this paper, we propose a new local search based on single job swapping within the key factory named LS_N. Algorithm 3 lists the proposed local search procedure.

In Line 1 of Algorithm 3, $F_{c1} = argmax_{c1=(1,2,...,f)} C_{max}(\pi_{F_{c1}})$ and $F_{c2} = argmax_{c1=(1,2,...,f)/c1} C_{max}(\pi_{F_{c2}})$ aim to find the critical factory, F_{c1}, with the maximal makespan, as well as the secondary critical factory, F_{c2}, with the secondary maximal makespan among the factories. In Lines 5 to 7, we randomly select two jobs from F_{c1} and F_{c2}, respectively, to perform the swap operator, and reevaluate the two new solutions obtained by implementing the swap operator. Next, Line 8 finds the maximal makespan, denoted as C^*_{max}. Lines 9 to 15 employ the acceptance criterion. If C^*_{max} is smaller than C_{max}, then the swap is kept. Otherwise, $Cnt = Cnt + 1$.

4.3 New local search based on job block swapping

After the destruction and reconstruction operators are used, the second local search strategy is employed, which is called LS_N2. Similarly, to reduce the computational complexity, a job block-based swapping strategy is considered in the second local search stage. First, the critical factory, F_{c1}, with the maximal makespan and subcritical factory, F_{c2}, with the secondary maximal makespan among the factories are obtained (see Line 1 of Algorithm 4). Furthermore, some adjacent jobs are selected randomly from the two factories, respectively, block_1 and block_2. As with the job-based exchange strategy, block_1 and block_2 are swapped, and two new solutions of F_{c1} and F_{c2} are obtained (see Lines 5 to 8 of Algorithm 4). Next, $C^*_{max} = max^f_{i=1}\{C_{max}(\pi_{F_1}), C_{max}(\pi_{F_2}), ..., C_{max}(\pi_{F_f})\}$ is computed, and the maximal makespan of all the factories is recorded. Finally, the acceptance criterion is executed. If C^*_{max} is smaller than C_{max}, then the swap is kept. Otherwise, $Cnt = Cnt + 1$.

To further clearly describe the above job block selection process, a simple example is given. Suppose that jobs are available in the scheduling sequence, and the size of the job block is $l(l = 3)$. First, the job (i ranges from 1 to n) is selected randomly. When $i \leqslant n - l + 1$, the jobs at interval $[i, i + 2]$ are selected. When $i > n - l + 1$, the jobs at interval $[n - 2, n]$ are selected. The above selection strategy ensures the

Algorithm 3 Local search based on single job swapping

Input: π

Output: π, C_{max}

1 : $F_{c1} = argmax_{c1=(1,2,...,f)} C_{max}(\pi_{F_{c1}})$,
$F_{c2} = argmax_{c1=(1,2,...,f)/c1} C_{max}(\pi_{F_{c2}})$

2: $Cnt = 0$

3: While $Cnt < n$ do // n is the number of jobs in factory F_{C1}

4: $\pi^{Initial} = \pi$

5: τ' = randomly selected job in F_{c1}

6: τ'' = randomly selected job in F_{c2}

7: Swap job τ' and τ'', and revaluate $\pi_{F_{c1}}$ and $\pi_{F_{c2}}$

8: $C^*_{max} = max^f_{i=1}\{C_{max}(\pi_{F_1}), C_{max}(\pi_{F_2}), ..., C_{max}(\pi_{F_f})\}$

9: if $C^*_{max} < C_{max}$

10: $C_{max} < C^*_{max}$

11: $F_{c1} = argmax_{c1=(1,2,...,f)} C_{max}(\pi_{F_{c1}})$

12: $F_{c2} = argmax_{c1=(1,2,...,f)/c2} C_{max}(\pi_{F_{c1}})$

13: $Cnt = 0$

14: else

15: $\pi = \pi^{Initial}$ and $Cnt = Cnt + 1$

16: end if

17: end while

Algorithm 4 Local search based on job block swapping

Input: π

Output: π, C_{max}

1 : $F_{c1} = argmax_{c1=(1,2,...,f)} C_{max}(\pi_{F_{c1}})$,
$F_{c2} = argmax_{c1=(1,2,...,f)/c1} C_{max}(\pi_{F_{c2}})$

2: $Cnt = 0$

3: While $Cnt < n$ do // n is the number of jobs in factory F_{C1}

4: $\pi^{Initial} = \pi$

5: τ' = randomly selected job in F_{c1}

6: τ'' = randomly selected job in F_{c2}

7: Determine the job block block_1 and block_2.

8: Swap block_1 and block_2, and revaluate $\pi_{F_{c1}}$ and $\pi_{F_{c2}}$

9: $C^*_{max} = max^f_{i=1}\{C_{max}(\pi_{F_1}), C_{max}(\pi_{F_2}), ..., C_{max}(\pi_{F_f})\}$

10: if $C^*_{max} < C_{max}$

11: $C_{max} < C^*_{max}$

12: $F_{c1} = argmax_{c1=(1,2,...,f)} C_{max}(\pi_{F_{c1}})$

13: $F_{c2} = argmax_{c1=(1,2,...,f)/c2} C_{max}(\pi_{F_{c1}})$

14: $Cnt = 0$

15: else

16: $\pi = \pi^{Initial}$ and $Cnt = Cnt + 1$

17: end if

18: end while

legitimacy of the positions of the selected job.

Through the above example, we describe the process of job block selection. In this paper, we also proposed an algorithm based on job block swapping called LS_N2. The proposed local search procedure is listed in Algorithm 4.

4.4 Destruction, reconstruction, and acceptance criteria

In the IG algorithm, destruction and reconstruction are constantly performed within the IG loop to keep the algorithm from falling into a local optimum. The destruction operator is applied to the original π. In this paper, we first randomly select djobs and place them into π^D in turn. Then a subsequence π^R is obtained by deleting d jobs from π ($\pi^R = \pi - \pi^D$). Based on the distributed characteristic of the DPFSP, we delete $d/2$ jobs from the critical factory, F_{C1}, and delete the $d - d/2$ jobs from the noncritical factories. Next, a reconstruction operation is employed to generate a completed sequence.

The reconstruction operation aims to reinsert the deleted jobs into π^R. The process is given as follows:

(1) The job is taken from π^D in turn and inserted at all the possible positions, respectively.

(2) Second, the position with the smallest makespan is selected, and the job is inserted into the selected position.

(3) Steps (1) and (2) are repeated until all jobs in have been removed. After the destruction and reconstruction operators, a simple thermostatic acceptance criterion proposed by Ruiz and Stutzle is applied.

$$Temperature = T \times \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{i,j}}{n \times m \times 10} \tag{13}$$

where T is a constant temperature value, $p_{i,j}$ is the processing time of job J_j on machine M_i, and n and m refer to the number of jobs and machines of the example, respectively. For the T value, the experimental results verify that some acceptance criteria without parameters, which were proposed by Hatamit et al.[45], did not yield significant improvements in initial testing[27]. T needs calibration but has shown robustness (most values are not zero and not too high).

5 Experiment and Experimental Result

The experimental data in this paper are the same as in Ref. [8], with a total of 150 test cases, where the number of factories is $f \in \{2,3,4,5,6,7\}$, the number of jobs is $n \in \{100,200,300,400,500\}$, the number of machines is $m \in \{5,8,10\}$, and the impact factor is $factor \in \{25,50,100\}$. We use $(1 + rand()\%99) \times factor/100$ to generate serially relevant preparation times and processing times through the impact factors. Thus, the values of processing times and SDSTs are in the range [1, 99). Each instance is independently executed with five replications, and the minimum makespan was taken as the final result of that algorithm.

In these experiments, all the algorithms are written in Visual C++ 2019, and the same library functions are employed to make fair comparisons. All the algorithms are implemented on a PC with Microsoft Windows 10 operating system, 16 GB DDR4 memory, and a 1.00 GHz Intel Core i5-1035G1 processor. For the termination criterion of these algorithms, the same maximal elapsed CPU time of $TimeLimit = CPU \times n \times m$ millisecond is employed.

For the evaluation indicator, we adopt the relative percentage increase (RPI) to test the efficiency of the proposed algorithm. The RPI is calculated as follows:

$$\text{RPI} = \frac{M_i - M_{best}}{M_{best}} \times 100 \tag{14}$$

where M_{best} is the minimum makespan obtained by all the compared algorithms for each test instance. M_i is the best makespan of the i-th algorithm for each test instance. A small RPI corresponds to improved results obtained by the algorithm.

In this paper, to demonstrate the performance of the proposed algorithm, we select the existing five compared algorithms used to solve the DPFSP. The compared algorithms are artificial CRO[5], DABC[46], DDE[47], improved iterative greedy algorithm (IGA)[7], and iterative greedy algorithm with a restart scheme (IGR)[8].

During solving the above instances, if we obtain a better solution of an instance by using the proposed algorithm than that of the comparative algorithms, then we update its upper bound. In the experiments, we report the computational results related to the following aspects:

- Comparison results of the local search based on single job swapping and job block swapping.
- Comparison results between the proposed algorithm and the five compared ones.
- Update of upper bounds of some benchmark problems.

5.1 Comparison results of a local search based on single job and job block swapping

In this paper, we propose a two-stage local search; that is, the local search is divided into a local search for the initial solution and a local search within the IG loop. NIG_*X* is the proposed improved Iterated greedy algorithm in which the local search based on single job swapping is performed in the first stage, and a job block swapping is performed in the second local search. NIG2_*X* is the algorithm in which job block swapping is employed in the two local search stages. "*X*" represents the number of jobs in the job block and is taken to be 1 to 5 in this experiment.

Figure 2 shows the interval plot of NIG_*X* and NIG2_*X*, where "*X*" equals 2, 3, 4, and 5, respectively. From the results of Fig. 2, the strategies of NIG_*X* and NIG2_*X* show good performance, suggesting that the proposed two-stage local search can enhance the performance of the proposed algorithm. Among these proposed strategies, only when the job block length is equal to 2 ($X = 2$) NIG_2 performs slightly better than NIG2_*X*. The performances of NIG_*X* and NIG2_*X* become increasingly worse as the number of jobs in the job block increases. In addition, for NIG_*X* and NIG2_*X*, the RPI of the former is slightly better than that of the latter. The reason may be that the job block that includes more than two jobs destroys the sequence within the critical factory to a large degree, resulting in reduced local exploitation ability of the proposed algorithm.

To further verify the performance of the proposed NIG and NIG_*X* (X = 2, 3, 4, 5), Table 2 and Figs. 3 and 4 list the experimental results of the comparison of NIG, NIG_*X*, IGA, and IGR at CPU = 10. In Table 2, when X = 2, the makespan values obtained by the proposed NIG, NIG_2, NIG_3, NIG_4, and NIG_5 are smaller than those of IGA and IGR for all the test instances, suggesting that the two-stage local search strategy can generate better solutions than all the compared algorithms, and more upper bounds obtained by the proposed algorithms are updated. As the number of factories increase, the superiority of the NIG, NIG_2, NIG_3, and NIG_4 over the IGA and IGR algorithms is demonstrated for the mean values of all test sets in Table 2. NIG is better than the compared algorithms because the two-stage local search based on single job swapping can slightly disrupt the current solution within the critical factory and improve the exploitation ability of NIG.

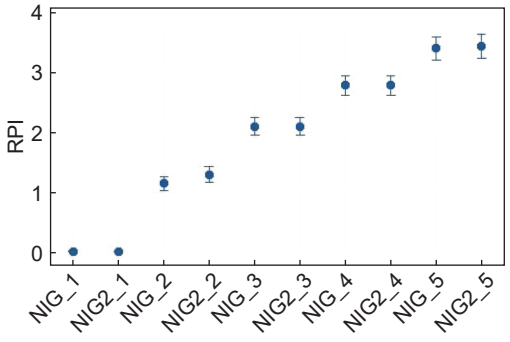


Fig. 2 Confidence intervals for two-stage IG and job block-based exchange IG.

To effectively demonstrate the convergence of the proposed algorithm, we also plotted the evolutionary curves of the seven algorithms above. According to requirements, all the compared algorithms are run using the same CPU time with the step size of 0.5 s on the aforementioned PC.

We randomly chose a small-scale instance and a large-scale instance of 100×5×2 and 500×10×5, respectively. Compared with the convergence curves of IGA, IGR, NIG, and NIG_*X*, the convergence curve of NIG is the highest as the runtime increases, followed by NIG_2, NIG_3, NIG_4, IGR, IGA, and NIG_5, suggesting that the proposed algorithms have the capacity to guide the solution to the optimal solution. The superiority of NIG is due to the two-stage local search.

In summary, NIG is significantly superior in all the test instances with respect to makespan, convergence curve, and interval plot compared with NIG_*X*, IGA, and IGAR. For this case, the reason for this superior performance may be that the NIG adopts a two-stage local search based on a single job swapping operator rather than an insertion operator, thereby decreasing the computational complexity of the local search and having more iterations to improve the quality of the solution than the compared algorithms.

5.2 Performance of all the compared algorithms

The experimental results in Subsection 5.1 show that the proposed NIG algorithm outperforms NIG_*X* and NIG2_*X*. Thus, in this section, we will further validate the effect of the proposed NIG algorithm. We compare NIG with DABC[46], CRO[5], DDE[47], IGA[7], and IGR[8] in terms of makespan and PRI on 150 test instances. All the compared algorithms have the same computational time and experimental environment. Tables 3 and 4 highlight the best mean result of the comparative methods.

Table 2 Makespan values of all the compared algorithms when CPU = 10.

factory	$J \times M$	IGA	IGR	NIG_2	NIG_3	NIG_4	NIG_5	NIG
f=2	100×5	3423	3405	3335	3343	3371	3415	3329
	100×8	3832	3703	3579	3610	3658	3652	3537
	100×10	3864	3804	3675	3755	3731	3776	3652
	200×5	6630	6506	6328	6415	6451	6435	6300
	200×8	6937	6812	6670	6680	6774	6740	6588
	200×10	7313	7117	6913	6959	6986	7045	6818
	300×5	9853	9682	9436	9498	9549	9587	9345
	300×8	10305	10027	9760	9870	9867	9908	9667
	300×10	10257	10260	9997	10058	10141	10047	9881
	400×5	13517	12984	12645	12802	12841	12860	12477
	400×8	13402	13053	12753	12787	12879	12894	12672
	400×10	13578	13399	13116	13261	13319	13365	13069
	500×5	15795	15468	15287	15357	15426	15383	15080
	500×8	16603	16385	15979	16064	16114	16223	15877
	500×10	16838	16752	16274	16385	16416	16454	16215
	Mean	10143.1	9957.13	9716.47	9789.6	9834.87	9852.27	**9633.8**
f=3	100×5	2345	2331	2305	2334	2336	2359	2249
	100×8	2632	2566	2469	2523	2509	2543	2446
	100×10	2840	2805	2752	2776	2793	2833	2728
	200×5	4467	4367	4338	4357	4390	4403	4298
	200×8	4835	4689	4597	4648	4689	4678	4508
	200×10	5083	4894	4816	4827	4897	4934	4745
	300×5	6678	6533	6457	6469	6537	6578	6410
	300×8	7015	6795	6630	6685	6733	6778	6545
	300×10	7152	7094	6880	6948	6951	6994	6823
	400×5	8792	8696	8528	8627	8694	8687	8475
	400×8	9331	9107	8913	8999	9083	9057	8811
	400×10	9247	9165	8969	9056	9081	9164	8881
	500×5	10839	10570	10473	10595	10608	10645	10419
	500×8	11248	10964	10658	10760	10827	10862	10620
	500×10	11473	11344	11033	11144	11221	11301	10956
	Mean	6931.8	6794.67	6654.53	6716.53	6756.6	6787.73	**6594.27**
f=4	100×5	1861	1816	1784	1798	1814	1846	1793
	100×8	2136	2064	2045	2047	2091	2082	2015
	100×10	2250	2237	2177	2221	2217	2254	2143
	200×5	3431	3371	3344	3385	3408	3424	3316
	200×8	3821	3677	3573	3641	3628	3721	3578
	200×10	3973	3965	3846	3893	3906	3961	3814
	300×5	5091	4963	4854	4911	4930	4973	4820
	300×8	5503	5229	5132	5155	5225	5240	5078
	300×10	5580	5514	5408	5455	5500	5535	5348
	400×5	8792	8696	8528	8627	8694	8687	8475
	400×8	9331	9107	8913	8999	9083	9057	8811
	400×10	9247	9165	8969	9056	9081	9164	8881
	500×5	8297	8037	7947	8017	8023	8105	7833
	500×8	8728	8556	8430	8472	8508	8587	8309
	500×10	8964	8720	8546	8549	8621	8688	8414
	Mean	5800.33	5674.47	5566.4	5615.07	5648.6	5688.27	**5508.53**

(To be continued)

Table 2 Makespan values of all the compared algorithms when CPU = 10. (Continued)

factory	$J \times M$	IGA	IGR	NIG_2	NIG_3	NIG_4	NIG_5	NIG
f=5	100×5	1515	1482	1483	1504	1506	1532	1474
	100×8	1826	1796	1751	1793	1777	1818	1723
	100×10	1954	1912	1858	1889	1909	1944	1832
	200×5	2858	2728	2701	2724	2742	2782	2629
	200×8	3037	2988	2950	2986	2998	3004	2908
	200×10	3303	3236	3151	3195	3244	3226	3105
	300×5	4096	3997	3933	3974	4002	4012	3920
	300×8	4422	4322	4217	4271	4295	4310	4160
	300×10	4693	4554	4453	4528	4535	4541	4386
	400×5	5479	5264	5210	5230	5237	5273	5171
	400×8	5758	5599	5523	5554	5633	5590	5477
	400×10	5951	5878	5668	5735	5773	5789	5632
	500×5	6807	6604	6456	6514	6558	6597	6401
	500×8	7130	6877	6751	6803	6846	6953	6688
	500×10	7337	7125	7014	7033	7088	7155	6877
	Mean	4411.067	4290.8	4207.93	4248.87	4276.2	4301.73	**4158.87**
f=6	100×5	1345	1307	1287	1305	1306	1335	1291
	100×8	1548	1488	1455	1482	1512	1529	1456
	100×10	1736	1689	1673	1658	1721	1703	1630
	200×5	2403	2357	2285	2340	2344	2366	2268
	200×8	2685	2579	2517	2552	2607	2618	2514
	200×10	2839	2803	2740	2750	2800	2798	2694
	300×5	3451	3326	3284	3334	3363	3380	3269
	300×8	3851	3707	3620	3613	3666	3679	3553
	300×10	3988	3934	3810	3845	3872	3919	3740
	400×5	4525	4355	4329	4361	4363	4375	4248
	400×8	4909	4828	4717	4742	4809	4771	4677
	400×10	5210	5042	4941	4953	4990	5012	4842
	500×5	5570	5397	5379	5417	5458	5503	5287
	500×8	6011	5866	5759	5896	5910	5978	5724
	500×10	6305	6124	5951	6036	6055	6111	5925
	Mean	3758.4	3653.47	3583.13	3618.93	3651.73	3671.8	**3541.2**
f=7	100×5	1185	1156	1142	1152	1174	1160	1133
	100×8	1390	1362	1332	1344	1353	1377	1320
	100×10	1546	1548	1493	1499	1518	1526	1487
	200×5	2131	2093	2055	2078	2103	2122	2032
	200×8	2413	2334	2302	2320	2329	2357	2252
	200×10	2608	2535	2482	2526	2540	2550	2445
	300×5	3422	2909	2861	2897	2922	2942	2838
	300×8	3325	3206	3135	3148	3175	3202	3071
	300×10	3539	3456	3366	3433	3455	3455	3320
	400×5	3930	3843	3802	3839	3880	3890	3752
	400×8	4334	4202	4137	4151	4201	4237	4086
	400×10	4602	4460	4363	4405	4399	4461	4308
	500×5	4840	4679	4604	4687	4702	4747	4584
	500×8	5213	5037	4955	5012	5041	5077	4902
	500×10	5525	5370	5235	5283	5351	5394	5176
	Mean	3333.53	3212.67	3150.93	3184.93	3209.53	3233.13	**3113.733**

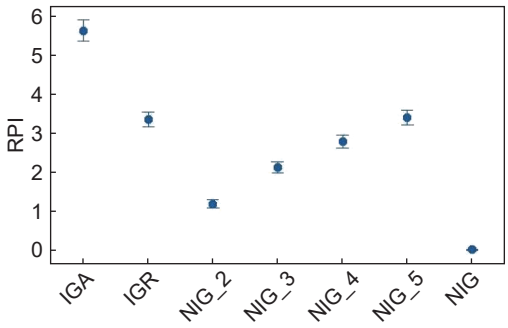


Fig. 3 Confidence intervals for some improved IG algorithms.

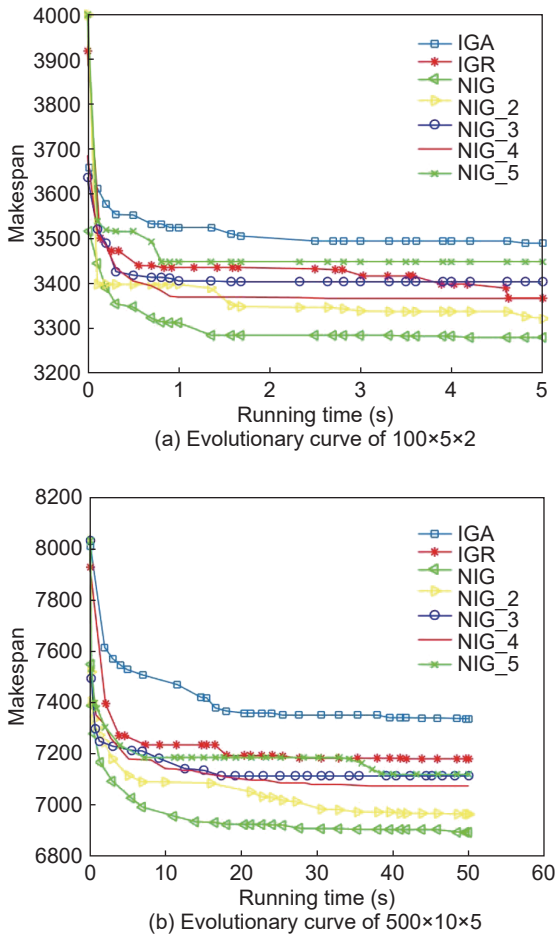


Fig. 4 Evolutionary curves for some improved IG algorithms.

Tables 3 and 4 list the new upper bounds of makespan and RPI produced by all the six compared algorithms when CPU = 5 and CPU = 10, respectively. From Tables 3 and 4, the makespan and RPI values produced by the proposed NIG algorithm are smaller than the ones obtained by the compared algorithms in all the test sets. In addition, when the number of factories is equal to 2, 3, 4, 5, 6, and 7, the proposed algorithm still achieves good performance in all the test sets. Except for NIG, the IGR and DDE algorithms are superior to the DABC, CRO, and IGA algorithms. Furthermore, the performance of IGR improves progressively as the size increases.

To further evaluate the performance of the proposed NIG algorithm, we investigated the convergence of different algorithms in this section. We randomly select an instance with 100 jobs, 5 machines, and 2 factories, and an instance with 500 jobs, 10 machines, and 5 factories. Figure 5 gives the best makespan values obtained by DABC, CRO, DDE, IGA, IGR, and NIG algorithms as the computation time increases. Figure 6 indicates that the convergence curve of the proposed NIG reaches the lowest levels among the compared algorithms for two given instances as the computation time increases, followed by the convergence curve of DDE, IGR, CRO, IGA, and DABC.

The above results indicate that the superiority of NIG is mainly attributed to the two-stage local search strategy proposed in Subsections 4.2 and 4.3 because they enhance the exploitation abilities of the algorithm. NIG adopts a two-stage local search based on a single job swapping operator rather than an insertion operator, which is why the computational complexity of NIG is lower than that of the compared algorithms. In addition, all algorithms adopt the same maximal elapsed CPU time with the unit of a millisecond as the termination criterion. Thus, the NIG has more iterations and opportunities to improve the quality of the solution than the compared algorithms. In summary, the proposed algorithm is effective and can generate the solution with good convergence.

5.3 Gantt charts of specific instances

To show the optimal scheduling sequence, Figure 6 shows the Gantt chart of a job sequence with 100 jobs, 3 factories, and 5 machines. In the Gantt chart, the horizontal axis represents the makespan value. The yellow rectangle represents the processing time of a job on a machine, and the blue rectangle is the preparation time. The serial numbers of the jobs are marked in the yellow rectangle. Figures 6a − 6c provide the optimal scheduling plan for managers in 3 factories, respectively. In the first factory, the optimal scheduling plan is 18-8-47-98-45-12-80-26-75-56-65-55-3-84-92-21-1-36-78-27-54-70-35-60-10-97-91-15-62-0-66-57 and the makespan value is equal to 2248. In the second factory, the optimal scheduling plan is 16-7-76-42-71-74-61-95-64-29-37-81-11-49-51-2-40-72-33-69-23-68-82-46-17-89-41-30-31-58-43-99-50-32-77 and the makespan value is equal to 2248. In the third factory,

Table 3 Makespan and RPI values of all the six algorithms when CPU = 5.

factory	*J*×*M*	DABC		CRO		DDE		IGA		IGR		NIG	
		Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI
f=2	100×5	3473	4.13	3405	2.1	3366	0.93	3441	3.18	3351	0.48	3335	0
	100×8	3823	7.9	3794	7.08	3584	1.16	3832	8.16	3694	4.26	3543	0
	100×10	3917	7.25	3931	7.64	3712	1.64	3864	5.81	3799	4.03	3652	0
	200×5	6553	3.67	6539	3.45	6426	1.66	6637	5	6478	2.48	6321	0
	200×8	6943	5.29	6894	4.55	6678	1.27	6937	5.2	6841	3.75	6594	0
	200×10	7245	6.13	7137	4.56	6854	0.41	7313	7.13	7093	3.91	6826	0
	300×5	9802	4.73	9674	3.37	9478	1.27	9853	5.28	9632	2.92	9359	0
	300×8	10094	4.41	10025	3.69	9778	1.14	10333	6.88	10046	3.91	9668	0
	300×10	10364	4.81	10247	3.63	9975	0.88	10282	3.98	10257	3.73	9888	0
	400×5	13202	5.68	13224	5.85	12714	1.77	13517	8.2	12992	3.99	12493	0
	400×8	13222	4.34	13167	3.91	12759	0.69	13402	5.76	13045	2.94	12672	0
	400×10	13742	5.11	13468	3.01	13206	1.01	13578	3.85	13526	3.46	13074	0
	500×5	15535	3	15702	4.1	15346	1.74	15795	4.72	15554	3.12	15083	0
	500×8	16509	3.98	16375	3.14	16101	1.41	16603	4.57	16367	3.09	15877	0
	500×10	16753	3.31	16828	3.77	16336	0.73	16838	3.83	16684	2.88	16217	0
	Mean	10078	4.92	10027	4.26	9754	1.18	10148	5.44	9957	3.26	**9640**	**0**
f=3	100×5	2434	8.23	2449	8.89	2257	0.36	2345	4.27	2338	3.96	2249	0
	100×8	2589	5.76	2638	7.76	2484	1.47	2638	7.76	2573	5.11	2448	0
	100×10	2888	5.9	2876	5.46	2728	0.04	2850	4.51	2803	2.79	2727	0
	200×5	4406	2.13	4438	2.87	4337	0.53	4467	3.55	4385	1.65	4314	0
	200×8	4755	5.36	4858	7.64	4542	0.64	4848	7.42	4726	4.72	4513	0
	200×10	5027	5.97	5040	6.24	4767	0.48	5083	7.15	4948	4.3	4744	0
	300×5	6696	4.45	6543	2.06	6432	0.33	6678	4.16	6525	1.78	6411	0
	300×8	6905	5.34	6817	4	6599	0.67	7022	7.12	6841	4.36	6555	0
	300×10	7151	4.81	7036	3.12	6833	0.15	7161	4.95	7027	2.99	6823	0
	400×5	8868	4.56	8740	3.05	8519	0.45	8811	3.89	8655	2.05	8481	0
	400×8	9306	5.51	9211	4.43	8865	0.51	9375	6.29	9149	3.73	8820	0
	400×10	9376	5.38	9150	2.84	8902	0.06	9247	3.93	9124	2.55	8897	0
	500×5	10808	3.68	10711	2.75	10482	0.56	10839	3.98	10683	2.48	10424	0
	500×8	11095	4.6	11013	3.83	10607	0	11251	6.07	10976	3.48	10620	0.12
	500×10	11516	5.02	11290	2.95	11030	0.58	11473	4.62	11359	3.58	10966	0
	Mean	6921	5.11	6854	4.53	6626	0.45	6939	5.31	6807	3.3	**6599**	**0.01**
f=4	100×5	1876	4.86	1838	2.74	1789	0	1865	4.25	1829	2.24	1793	0.22
	100×8	2156	6.89	2132	5.7	2042	1.24	2136	5.9	2080	3.12	2017	0
	100×10	2293	7	2306	7.61	2194	2.38	2261	5.51	2239	4.48	2143	0
	200×5	3542	6.82	3472	4.7	3342	0.78	3436	3.62	3382	1.99	3316	0
	200×8	3816	6.62	3777	5.53	3597	0.5	3823	6.82	3669	2.51	3579	0
	200×10	4034	5.74	4012	5.16	3832	0.45	3983	4.4	3931	3.04	3815	0
	300×5	5071	5.1	4966	2.92	4862	0.77	5091	5.51	4927	2.11	4825	0
	300×8	5364	5.63	5335	5.06	5126	0.95	5504	8.39	5274	3.86	5078	0
	300×10	5670	6.02	5533	3.46	5380	0.6	5580	4.34	5531	3.42	5348	0
	400×5	6704	5.38	6601	3.76	6422	0.94	6637	4.32	6519	2.47	6362	0
	400×8	7226	7.04	7070	4.73	6754	0.04	7032	4.16	6942	2.83	6751	0
	400×10	7419	6.69	7264	4.46	6984	0.43	7309	5.1	7187	3.35	6954	0
	500×5	8306	5.96	8082	3.1	7940	1.29	8313	6.05	8064	2.87	7839	0
	500×8	8772	5.26	8596	3.14	8438	1.25	8728	4.73	8589	3.06	8334	0
	500×10	9025	7.2	8740	3.81	8626	2.46	8965	6.49	8729	3.68	8419	0
	Mean	5418	6.15	5315	4.39	5155	0.94	5378	5.31	5259	3	**5105**	**0.01**

(To be continued)

Table 3 Makespan and RPI values of all the six algorithms when CPU = 5. (Continued)

factory	*J*×*M*	DABC		CRO		DDE		IGA		IGR		NIG	
		Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI
f=5	100×5	1563	6.04	1547	4.95	1487	0.88	1519	3.05	1483	0.61	1474	0
	100×8	1890	9.69	1836	6.56	1758	2.03	1836	6.56	1786	3.66	1723	0
	100×10	1998	8.65	1957	6.42	1917	4.24	1954	6.25	1902	3.43	1839	0
	200×5	2887	9.69	2879	9.38	2728	3.65	2858	8.59	2743	4.22	2632	0
	200×8	3115	8.27	3053	6.12	2877	0	3037	5.56	3007	4.52	2908	1.08
	200×10	3394	9.24	3297	6.12	3141	1.09	3303	6.31	3248	4.54	3107	0
	300×5	4144	5.71	4058	3.52	3944	0.61	4101	4.62	3992	1.84	3920	0
	300×8	4477	7.62	4436	6.63	4214	1.3	4422	6.3	4267	2.57	4160	0
	300×10	4709	7.59	4667	6.63	4377	0	4693	7.22	4554	4.04	4386	0.21
	400×5	5479	5.87	5409	4.52	5229	1.04	5486	6.01	5279	2.01	5175	0
	400×8	5933	8.13	5642	2.82	5564	1.4	5758	4.94	5587	1.82	5487	0
	400×10	6067	7.65	5903	4.74	5694	1.03	5951	5.59	5818	3.23	5636	0
	500×5	6810	6.34	6772	5.75	6489	1.33	6816	6.43	6600	3.06	6404	0
	500×8	7158	6.98	6836	2.17	6720	0.43	7139	6.7	6908	3.24	6691	0
	500×10	7420	7.82	7226	5	7012	1.89	7343	6.7	7135	3.68	6882	0
	Mean	4470	7.69	4368	5.42	4210	1.4	4414	6.05	4287	3.1	**4162**	**0.09**
f=6	100×5	1387	7.44	1357	5.11	1294	0.23	1345	4.18	1304	1.01	1291	0
	100×8	1549	6.02	1579	8.08	1479	1.23	1548	5.95	1513	3.56	1461	0
	100×10	1787	9.63	1752	7.48	1673	2.64	1736	6.5	1697	4.11	1630	0
	200×5	2476	9.12	2394	5.51	2321	2.29	2403	5.91	2341	3.17	2269	0
	200×8	2727	8.47	2602	3.5	2526	0.48	2689	6.96	2608	3.74	2514	0
	200×10	2980	10.5	2919	8.27	2834	5.12	2839	5.3	2802	3.93	2696	0
	300×5	3506	7.25	3452	5.6	3366	2.97	3451	5.57	3323	1.65	3269	0
	300×8	3881	9.17	3809	7.14	3672	3.29	3851	8.33	3715	4.5	3555	0
	300×10	4007	7.14	4001	6.98	3944	5.45	3999	6.93	3924	4.92	3740	0
	400×5	4622	8.8	4417	3.98	4480	5.46	4529	6.61	4346	2.31	4248	0
	400×8	5095	8.94	4900	4.77	4878	4.3	4935	5.52	4828	3.23	4677	0
	400×10	5283	9.09	5150	6.34	4963	2.48	5219	7.76	5042	4.11	4843	0
	500×5	5688	7.58	5518	4.37	5425	2.61	5570	5.35	5428	2.67	5287	0
	500×8	6199	8.15	6019	5.01	5929	3.44	6011	4.87	5894	2.83	5732	0
	500×10	6291	6.14	6147	3.71	6070	2.41	6305	6.38	6102	2.95	5927	0
	Mean	3832	8.23	3734	5.72	3657	2.96	3762	6.14	3658	3.25	**3543**	**0**
f=7	100×5	1229	8.57	1190	5.12	1132	0	1185	4.68	1142	0.88	1134	0.18
	100×8	1436	8.79	1415	7.2	1375	4.17	1390	5.3	1357	2.8	1320	0
	100×10	1666	12	1582	6.39	1545	3.9	1546	3.97	1542	3.7	1487	0
	200×5	2234	9.94	2144	5.51	2061	1.43	2131	4.87	2087	2.71	2032	0
	200×8	2549	12.8	2400	6.29	2294	1.59	2422	7.26	2359	4.47	2258	0
	200×10	2723	11.2	2585	5.6	2561	4.62	2608	6.54	2558	4.49	2448	0
	300×5	3097	9.13	3050	7.47	2944	3.74	3422	20.5	2917	2.78	2838	0
	300×8	3387	10	3257	5.78	3194	3.73	3325	7.99	3181	3.31	3079	0
	300×10	3731	11.7	3509	5.12	3443	3.15	3539	6.02	3463	3.74	3338	0
	400×5	4097	9.14	3947	5.14	3896	3.78	3931	4.71	3824	1.86	3754	0
	400×8	4508	10.3	4295	5.12	4278	4.7	4344	6.31	4203	2.86	4086	0
	400×10	4741	10	4502	4.45	4420	2.55	4602	6.77	4465	3.6	4310	0
	500×5	4996	8.68	4760	3.55	4807	4.57	4842	5.33	4688	1.98	4597	0
	500×8	5319	8.51	5083	3.69	5067	3.37	5213	6.34	5063	3.28	4902	0
	500×10	5679	9.72	5445	5.2	5390	4.13	5538	6.99	5379	3.92	5176	0
	Mean	3426	10.05	3278	5.44	3227	3.29	3336	6.91	3215	3.09	**3117**	**0.01**

Table 4 Makespan and RPI of the compared algorithms when CPU = 10.

factory	*J*×*M*	DABC		CRO		DDE		IGA		IGR		NIG	
		Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI
f=2	100×5	3965	8.04	3981	8.47	3769	2.7	3913	6.62	3840	4.63	3670	0
	100×8	4453	7.77	4462	7.99	4132	0	4381	6.03	4271	3.36	4141	0.22
	100×10	4578	7.26	4490	5.2	4287	0.45	4489	5.18	4419	3.54	4268	0
	200×5	7521	6.06	7329	3.36	7218	1.79	7387	4.17	7255	2.31	7091	0
	200×8	7976	5.21	7918	4.45	7759	2.35	8079	6.57	7944	4.79	7581	0
	200×10	8338	6.54	8056	2.94	7893	0.86	8242	5.32	8154	4.19	7826	0
	300×5	11067	5.24	10892	3.58	10821	2.9	11204	6.54	11003	4.63	10516	0
	300×8	11880	5.52	11826	5.05	11377	1.06	11927	5.94	11724	4.14	11258	0
	300×10	12185	5.18	12073	4.21	11714	1.11	12229	5.56	12118	4.6	11585	0
	400×5	14995	6.67	14666	4.32	14458	2.85	14884	5.88	14667	4.33	14058	0
	400×8	15425	5.59	14945	2.3	14752	0.98	15503	6.12	15179	3.9	14609	0
	400×10	15820	4.5	15697	3.69	15352	1.41	16068	6.14	15768	4.15	15139	0
	500×5	18400	6.28	18242	5.37	17843	3.06	18526	7.01	18246	5.39	17313	0
	500×8	19031	4.77	18844	3.74	18529	2.01	19183	5.61	19006	4.64	18164	0
	500×10	19787	4.25	19794	4.29	19292	1.64	20211	6.49	19883	4.76	18980	0
	Mean	11694	5.93	11547	4.6	11279	1.68	11748	5.94	11565	4.22	**11079**	**0.01**
f=3	100×5	2757	6.57	2783	7.58	2618	1.2	2725	5.33	2693	4.1	2587	0
	100×8	3081	8.72	3055	7.8	2857	0.81	3037	7.16	2947	3.99	2834	0
	100×10	3201	6.99	3209	7.25	3030	1.27	3190	6.62	3131	4.65	2992	0
	200×5	5002	6.95	4937	5.56	4787	2.35	4925	5.3	4894	4.64	4677	0
	200×8	5491	5.76	5529	6.49	5252	1.16	5454	5.05	5411	4.22	5192	0
	200×10	5860	6.2	5807	5.24	5589	1.29	5835	5.74	5736	3.95	5518	0
	300×5	7562	6.1	7543	5.84	7253	1.77	7514	5.43	7438	4.36	7127	0
	300×8	8243	5.8	8240	5.76	7831	0.51	8209	5.37	8122	4.25	7791	0
	300×10	8288	5.24	8282	5.17	7969	1.19	8333	5.82	8211	4.27	7875	0
	400×5	10164	6.17	10028	4.75	9807	2.44	10144	5.96	9945	3.89	9573	0
	400×8	10462	5.14	10445	4.96	10048	0.97	10415	4.66	10342	3.93	9951	0
	400×10	10863	5.5	10568	2.63	10347	0.49	10895	5.81	10655	3.48	10297	0
	500×5	12615	5.94	12476	4.77	12097	1.59	12633	6.09	12398	4.11	11908	0
	500×8	13178	6.46	12901	4.23	12674	2.39	13022	5.2	12914	4.33	12378	0
	500×10	13468	5.5	13262	3.89	12874	0.85	13509	5.82	13261	3.88	12766	0
	Mean	8015	6.2	7937	5.46	7668	1.35	7989	5.69	7873	4.14	**7564**	**0**
f=4	100×5	2160	7.78	2129	6.24	2033	1.45	2103	4.94	2084	3.99	2004	0
	100×8	2507	9.24	2494	8.67	2346	2.22	2473	7.76	2388	4.05	2295	0
	100×10	2723	9.58	2720	9.46	2541	2.25	2626	5.67	2605	4.83	2485	0
	200×5	3873	7.11	3865	6.89	3616	0	3875	7.16	3763	4.07	3624	0.22
	200×8	4404	6.02	4423	6.48	4205	1.23	4437	6.81	4292	3.32	4154	0
	200×10	4620	7.44	4644	8	4403	2.4	4620	7.44	4511	4.91	4300	0
	300×5	5978	8.51	5951	8.02	5608	1.8	5892	6.95	5846	6.12	5509	0
	300×8	6128	4.31	6229	6.03	5917	0.71	6242	6.25	6180	5.19	5875	0
	300×10	6634	6.79	6541	5.3	6356	2.32	6639	6.87	6471	4.17	6212	0
	400×5	7673	5.24	7619	4.5	7401	1.51	7704	5.66	7610	4.38	7291	0
	400×8	8168	6.74	8031	4.95	7751	1.29	8083	5.63	7957	3.99	7652	0
	400×10	8496	7.5	8415	6.48	8051	1.87	8449	6.91	8250	4.39	7903	0
	500×5	9467	7.74	9196	4.65	9052	3.02	9307	5.92	9144	4.06	8787	0
	500×8	9901	6.07	9740	4.35	9489	1.66	9904	6.11	9702	3.94	9334	0
	500×10	10340	6.46	10129	4.28	9820	1.1	10301	6.05	10125	4.24	9713	0
	Mean	6204	7.1	6141	6.29	5905	1.66	6177	6.41	6061	4.38	**5809**	**0.01**

(To be continued)

Table 4 Makespan and RPI of the compared algorithms when CPU = 10. (Continued)

factory	*J*×*M*	DABC		CRO		DDE		IGA		IGR		NIG	
		Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI	Makespan	RPI
f=5	100×5	1845	10.88	1803	8.35	1691	1.62	1763	5.95	1740	4.57	1664	0
	100×8	2108	9.68	2050	6.66	1964	2.19	2017	4.94	1982	3.12	1922	0
	100×10	2332	12.93	2248	8.86	2079	0.68	2209	6.97	2137	3.49	2065	0
	200×5	3278	8.83	3233	7.34	3106	3.12	3241	7.6	3142	4.32	3012	0
	200×8	3666	9.43	3627	8.27	3360	0.3	3549	5.94	3484	4	3350	0
	200×10	3905	8.62	3844	6.93	3630	0.97	3836	6.7	3761	4.62	3595	0
	300×5	4605	6.87	4602	6.8	4486	4.11	4653	7.98	4524	4.99	4309	0
	300×8	5377	10.59	5142	5.76	4987	2.57	5092	4.73	5020	3.25	4862	0
	300×10	5450	7.64	5399	6.64	5141	1.54	5378	6.22	5302	4.72	5063	0
	400×5	6288	8.47	6107	5.35	5886	1.54	6155	6.18	5991	3.35	5797	0
	400×8	6764	9.17	6576	6.13	6280	1.36	6640	7.17	6468	4.39	6196	0
	400×10	7035	8.36	6789	4.57	6600	1.66	6990	7.67	6778	4.41	6492	0
	500×5	7803	8.75	7548	5.2	7354	2.49	7667	6.86	7452	3.86	7175	0
	500×8	8329	7.82	8189	6.01	8127	5.2	8216	6.36	8076	4.54	7725	0
	500×10	8666	8.07	8430	5.13	8265	3.07	8599	7.23	8376	4.45	8019	0
	Mean	5163	9.07	5039	6.53	4863	2.16	5067	6.57	4948	4.14	**4749**	**0**
f=6	100×5	1573	14.73	1536	12.04	1419	3.5	1470	7.22	1449	5.69	1371	0
	100×8	1804	10.81	1806	10.93	1643	0.92	1721	5.71	1708	4.91	1628	0
	100×10	2011	10.62	2002	10.12	1878	3.3	1938	6.6	1893	4.13	1818	0
	200×5	2814	9.24	2803	8.81	2618	1.63	2712	5.28	2661	3.3	2576	0
	200×8	3151	10.02	3100	8.24	2943	2.76	3066	7.05	3023	5.55	2864	0
	200×10	3436	12.07	3280	6.98	3096	0.98	3260	6.33	3203	4.47	3066	0
	300×5	4005	8.33	3928	6.25	3829	3.57	3922	6.09	3813	3.14	3697	0
	300×8	4558	10.12	4485	8.36	4282	3.45	4438	7.22	4317	4.3	4139	0
	300×10	4668	9.71	4543	6.77	4382	2.98	4576	7.54	4487	5.45	4255	0
	400×5	5414	9.75	5250	6.43	5093	3.24	5274	6.91	5111	3.61	4933	0
	400×8	5719	8.27	5587	5.77	5554	5.15	5618	6.36	5480	3.75	5282	0
	400×10	6159	10.14	5928	6.01	5751	2.84	6014	7.55	5828	4.22	5592	0
	500×5	6530	6.8	6412	4.87	6368	4.15	6534	6.87	6349	3.84	6114	0
	500×8	7142	9.61	6905	5.97	6904	5.95	6934	6.41	6789	4.19	6516	0
	500×10	7368	8.42	7174	5.56	7060	3.88	7288	7.24	7024	3.35	6796	0
	Mean	4423	9.91	4316	7.54	4188	3.22	4317	6.69	4209	4.26	**4043**	**0**
f=7	100×5	1378	10.42	1354	8.49	1315	5.37	1312	5.13	1276	2.24	1248	0
	100×8	1637	13.52	1608	11.51	1498	3.88	1547	7.28	1517	5.2	1442	0
	100×10	1704	7.17	1741	9.5	1642	3.27	1687	6.1	1648	3.65	1590	0
	200×5	2465	10.39	2403	7.61	2333	4.48	2396	7.3	2305	3.22	2233	0
	200×8	2770	9.79	2722	7.89	2583	2.38	2690	6.62	2622	3.92	2523	0
	200×10	3065	11.7	2982	8.67	2829	3.1	2960	7.87	2869	4.56	2744	0
	300×5	3590	10.6	3522	8.5	3388	4.37	3438	5.91	3375	3.97	3246	0
	300×8	3880	9.95	3816	8.13	3652	3.49	3767	6.74	3672	4.05	3529	0
	300×10	4230	11.17	4060	6.7	3920	3.02	4056	6.6	3955	3.94	3805	0
	400×5	4640	8.54	4578	7.09	4342	1.57	4538	6.15	4429	3.6	4275	0
	400×8	5189	11.38	4952	6.29	4707	1.03	4954	6.33	4841	3.91	4659	0
	400×10	5395	9.72	5227	6.3	5028	2.26	5279	7.36	5108	3.88	4917	0
	500×5	5810	9.58	5593	5.49	5537	4.43	5648	6.53	5518	4.07	5302	0
	500×8	6127	8.35	5992	5.96	5959	5.38	6105	7.96	5866	3.73	5655	0
	500×10	6554	9.76	6266	4.94	6135	2.75	6384	6.92	6184	3.57	5971	0
	Mean	3895	10.14	3787	7.54	3657	3.38	3784	6.72	3679	3.84	**3542**	**0**

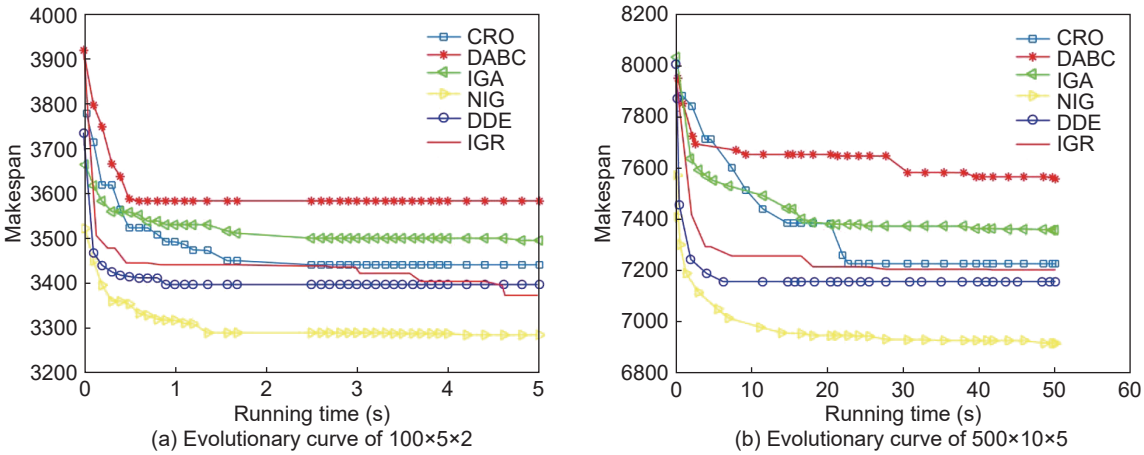


Fig. 5 Evolutionary curves for CRO, DABC, IGA, NIG, DDE, and IGR.

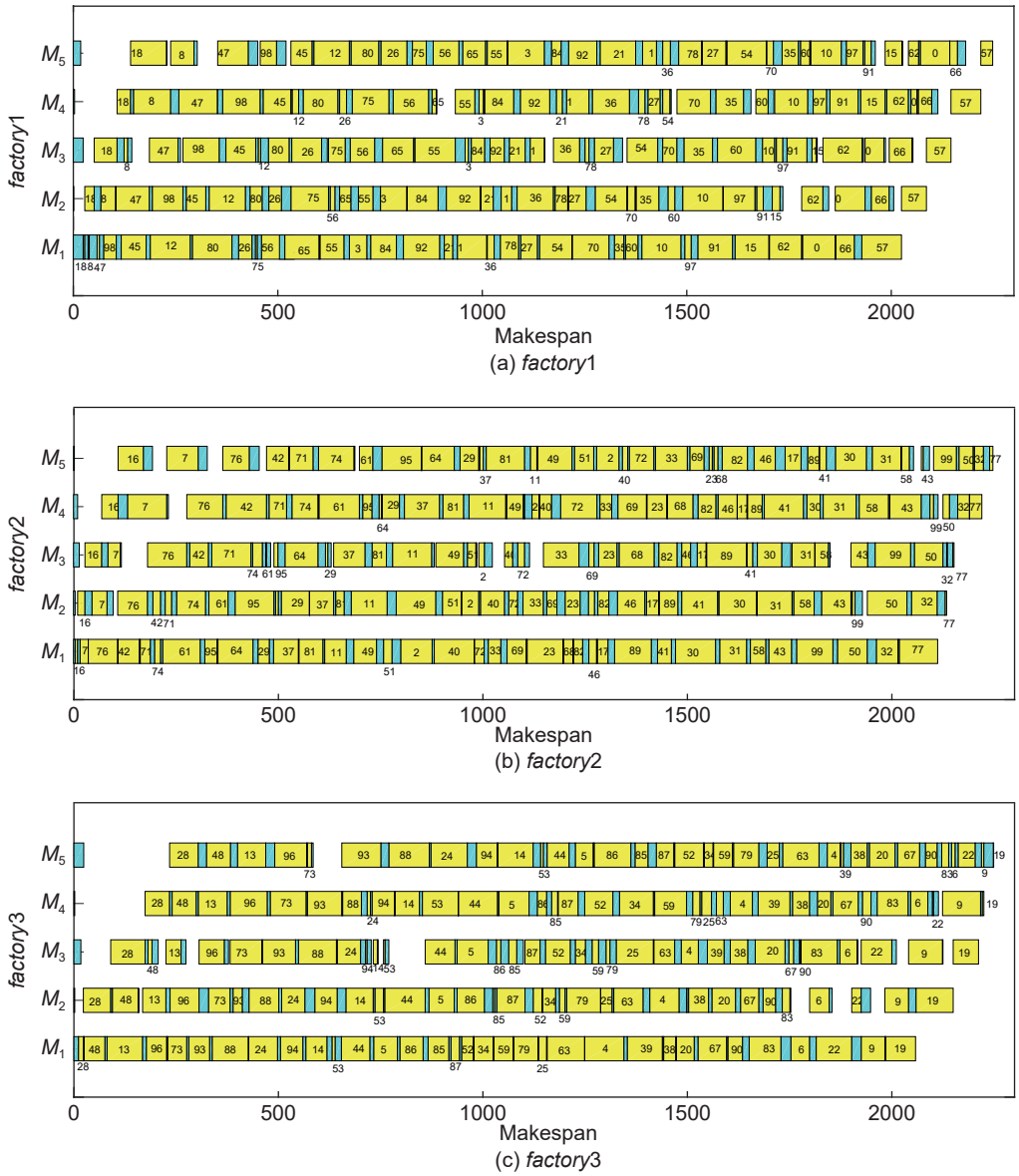


Fig. 6 Gantt chart of the optimal scheduling sequence with 100 jobs and 5 machines in 3 factories.

the optimal scheduling plan is 28-48-13-96-73-93-88-24-94-14-53-44-5-86-85-87-52-34-59-79-25-63-4-39-38-20-67-90-83-6-22-9-19 and the makespan value is equal to 2249 . Thus, the final makespan for this instance is 2249.

6 Conclusion and Future Prospect

In this paper, we proposed an NIG algorithm based on a two-stage local search strategy for solving DPFSP-SDST. Based on the distributed feature of DPFSP-SDST, a single job swapping operator is proposed to disrupt the current solution within the critical factory in the first local search stage. In the second local search stage, job block swapping is designed to further enhance the exploitation ability of the proposed algorithm. This two-stage local search has low computational complexity and more iterations and provides more opportunities to improve the quality of the solution than the algorithms based on insertion operators. Computational experiments are given and compared with the results obtained by the IGA, IGR, DDE, DABC, and CRO algorithms.

Several problems and opportunities on DPFSP-SDST need to be addressed in the future. For example, DPFSP with multiobjective, blocking constraints, and energy consumption or green objective problems can be focused on in future research. In addition, we can further consider DPFSP-SDST with uncertainties, such as machine breakdowns, nondeterministic processing time, operator illness, and the change of due date. We believe that an increasing number of excellent findings will be obtained as a result.

Acknowledgment

This work was jointly supported by the National Natural Science Foundation of China (Nos. 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, *Comput. Oper. Res.*, vol. 37, no. 4, pp. 754–768, 2010.

[2] Y. P. Fu, Y. S. Hou, Z. F. Wang, X. W. Wu, K. Z. Gao, and L. Wang, Distributed scheduling problems in intelligent manufacturing systems, *Tsinghua Sci. Technol.*, vol. 26, no. 5, pp. 625–645, 2021.

[3] S. Parthasarathy and C. Rajendran, An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs, *Int. J. Prod. Econom.*, vol. 49, no. 3, pp. 255–263, 1997.

[4] S. Y. Wang, L. Wang, M. Liu, and Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, *Int. J. Prod. Econ.*, vol. 145, no. 1, pp. 387–396, 2013.

[5] H. Bargaoui, O. B. Driss, and K. Ghédira, A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion, *Comput. Ind. Eng.*, vol. 111, pp. 239–250, 2017.

[6] 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, *Eng. Appl. Artif. Intell.*, vol. 97, p. 104016, 2021.

[7] R. Ruiz, Q. K. Pan, and B. Naderi, Iterated Greedy methods for the distributed permutation flowshop scheduling problem, *Omega*, vol. 83, pp. 213–222, 2019.

[8] 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, *Swarm Evol. Comput.*, vol. 59, p. 100742, 2020.

[9] H. X. Qin, Y. Y. Han, Q. D. Chen, J. Q. Li, and H. Y. Sang, A double level mutation iterated greedy algorithm for blocking hybrid flow shop scheduling, (in Chinese), *Control Decision*, doi: 10.13195/j.kzyjc.2021.0607.

[10] K. Sörensen, Metaheuristics—The metaphor exposed, *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 3–18, 2015.

[11] B. Naderi and R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, *Eur. J. Oper. Res.*, vol. 239, no. 2, pp. 323–334, 2014.

[12] 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 Syst. Appl.*, vol. 160, p. 113678, 2020.

[13] T. Meng, Q. K. Pan, and L. Wang, A distributed permutation flowshop scheduling problem with the customer order constraint, *Knowledge-Based Syst.*, vol. 184, p. 104894, 2019.

[14] Y. L. Li, F. Li, Q. K. Pan, L. Gao, and M. F. Tasgetiren, An artificial bee colony algorithm for the distributed hybrid flowshop scheduling problem, *Procedia Manuf.*, vol. 39, pp. 1158–1166, 2019.

[15] V. Fernandez-Viagas, P. Perez-Gonzalez, and J. M. Framinan, The distributed permutation flow shop to minimise the total flowtime, *Comput. Ind. Eng.*, vol. 118, pp. 464–477, 2018.

[16] 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 Syst. Appl.*, vol. 124, pp. 309–324, 2019.

[17] Z. Q. Zhang, B. Qian, R. Hu, H. P. Jin, and L. Wang, A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem, *Swarm Evol. Comput.*, vol. 60, p. 100785, 2021.

[18] H. B. Song and J. Lin, A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times, *Swarm Evol. Comput.*, vol. 60, p. 100807, 2021.

[19] J. Deng and L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, *Swarm Evol. Comput.*, vol. 32,

pp. 121–131, 2017.

[20] 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.

[21] G. C. Wang, L. Gao, X. Y. Li, P. G. Li, and M. F. Tasgetiren, Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm, *Swarm Evol. Comput.* , vol. 57, p. 100716, 2020.

[22] J. F. Chen, L. Wang, and Z. P. Peng, A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling, *Swarm Evol. Comput.*, vol. 50, p. 100557, 2019.

[23] M. Mirabi, Ant colony optimization technique for the sequence-dependent flowshop scheduling problem, *Int. J. Adv. Manuf. Technol.*, vol. 55, nos. 1–4, pp. 317–326, 2011.

[24] R. Vanchipura, R. Sridharan, and A. S. Babu, Improvement of constructive heuristics using variable neighbourhood descent for scheduling a flow shop with sequence dependent setup time, *J. Manuf. Syst.* , vol. 33, no. 1, pp. 65–75, 2014.

[25] A. Sioud and C. Gagné, Enhanced migrating birds optimization algorithm for the permutation flow shop problem with sequence dependent setup times, *Eur. J. Oper. Res.*, vol. 264, no. 1, pp. 66–73, 2018.

[26] M. S. Nagano, H. H. Miyata, and D. C. Araújo, A constructive heuristic for total flowtime minimization in a no-wait flowshop with sequence-dependent setup times, *J. Manuf. Syst.*, vol. 36, pp. 224–230, 2015.

[27] F. Q. Zhao, L. X. Zhang, Y. Zhang, W. M. Ma, C. Zhang, and H. B. Song, A hybrid discrete water wave optimization algorithm for the no-idle flowshop scheduling problem with total tardiness criterion, *Expert Syst. Appl.*, vol. 146, p. 113166, 2020.

[28] H. R. Li, X. Y. Li, and L. Gao, A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem, *Appl. Soft Comput.*, vol. 100, p. 106946, 2021.

[29] Y. L. Li, X. Y. Li, L. Gao, and L. L. Meng, An improved artificial bee colony algorithm for distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup times, *Comput. Ind. Eng.*, vol. 147, p. 106638, 2020.

[30] F. Q. Zhao, L. X. Zhang, J. Cao, and J. X. Tang, A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem, *Comput. Ind. Eng.*, vol. 153, p. 107082, 2021.

[31] Z. S. Shao, W. S. Shao, and D. C. Pi, Effective constructive heuristic and metaheuristic for the distributed assembly blocking flow-shop scheduling problem, *Appl. Intell.*, vol. 50, no. 12, pp. 4647–4669, 2020.

[32] R. Ruiz and T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *Eur. J. Oper. Res.* , vol. 177, no. 3, pp. 2033–2049, 2007.

[33] V. Fernandez-Viagas, R. Ruiz, and J. M. Framinan, A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation, *Eur. J. Oper. Res.* , vol. 257, no. 3, pp. 707–721, 2017.

[34] S. W. Lin, K. C. Ying, and C. Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, *Int. J. Prod. Res.*, vol. 51, no. 16, pp. 5029–5038, 2013.

[35] V. Fernandez-Viagas and J. M. Framinan, A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem, *Int. J. Prod. Res.*, vol. 53, no. 4, pp. 1111–1123, 2015.

[36] Q. K. Pan and R. Ruiz, An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem, *Omega*, vol. 44, pp. 41–50, 2014.

[37] J. Y. Ding, S. J. Song, J. N. D. Gupta, R. Zhang, R. Chiong, and C. Wu, An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem, *Appl. Soft Comput.*, vol. 30, pp. 604–613, 2015.

[38] V. Fernandez-Viagas and J. M. Framinan, A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective, *Comput. Oper. Res.*, vol. 112, p. 104767, 2019.

[39] J. Y. Mao, X. L. Hu, Q. K. Pan, Z. H. Miao, C. X. He, and M. F. Tasgetiren, An iterated greedy algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance to minimize total flowtime, in *2020 39th Chinese Control Conf., (CCC)*, Shenyang, China, 2020, pp. 1507−1512.

[40] K. C. Ying, S. W. Lin, and C. Y. Huang, Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic, *Expert Syst. Appl.*, vol. 36, no. 3, pp. 7087–7092, 2009.

[41] 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, *Appl. Soft Comput.*, vol. 96, p. 106629, 2020.

[42] W. S. Shao, D. C. Pi, and Z. S. Shao, Local search methods for a distributed assembly no-idle flow shop scheduling problem, *IEEE Syst. J.* , vol. 13, no. 2, pp. 1945–1956, 2019.

[43] D. P. Ronconi, A note on constructive heuristics for the flowshop problem with blocking, *Int. J. Prod. Econ.*, vol. 87, no. 1, pp. 39–48, 2004.

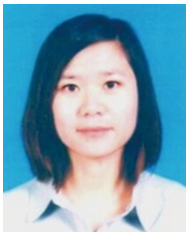
[44] Z. K. Zhang and Q. H. Tang, Integrating preventive maintenance to two-stage assembly flow shop scheduling: MILP model, constructive heuristics and meta-heuristics, *Flex. Serv. Manuf. J.*, doi: 10.1007/s10696-021-09403-0.

[45] S. Hatami, R. Ruiz, and C. Andrés-Romano, Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times, *Int. J. Prod. Econ.* , vol. 169, pp. 76–88, 2015.

[46] J. Q. Pan, W. Q. Zou, and J. H. Duan, A discrete artificial bee colony for distributed permutation flowshop scheduling problem with total flow time minimization, in *2018 37th Chinese Control Conf. (CCC)*, Wuhan, China, 2018, pp. 8379−8383.

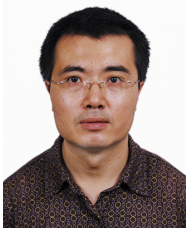
[47] G. H. Zhang, K. Y. Xing, and F. Cao, Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion, *Eng. Appl. Artif. Intell.*, vol. 76, pp. 96–107, 2018.

Xue Han received the BS degree from Liaocheng University, Liaocheng, China, in 2020. Currently she is studying as a master student in the School of Computer Science at Liaocheng University, Liaocheng, China. She is under the supervision of associate professor Yuyan Han. Her research interests include intelligent optimization methods and scheduling.



Yuyan Han received the MS degree from Liaocheng University, Liaocheng, China, in 2012, and the PhD degree in control theory and control engineering from China University of Mining and Technology, Xuzhou, China, in 2016. Since 2016, she has been an associate professor with the School of Computer Science, Liaocheng University and department head of the School of Computer Science. Her current research interests include evolutionary computation, multi-objective optimization, and flow shop scheduling. She has authored more than 30 refereed papers.

Yiping Liu received the BEng degree in electrical engineering and automation and the PhD degree in control theory and control engineering from China University of Mining and Technology, Xuzhou, China in 2012 and 2017, respectively. He is currently an associate professor in the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. During 2018−2020, he was a research assistant professor in the Department of Computer Science and Intelligent Systems, Osaka Prefecture University, Sakai, Japan. During 2016−2017, he was a visiting scholar in the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, USA. His research interests include evolutionary computation, multi-objective optimization, and machine learning.



Quanke Pan received the BS and PhD degrees from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1993 and 2003, respectively. From 2003 to 2011, he was with the School of Computer Science, Liaocheng University, China, where he became a full professor in 2006. From 2011 to 2014, he was with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, China. From 2014 to 2015, he was with the State Key Laboratory of Digital Manufacturing and Equipment Technology, Huazhong University of Science and Technology, China. He has been with the School of Mechatronic Engineering and Automation, Shanghai University, China since 2015. He has authored one academic book and more than 200 refereed papers. His current research interests include intelligent optimization and scheduling algorithms.

Qingda Chen received the PhD degree in control theory and engineering from Northeastern University, Shenyang, China, in 2020. Since 2021, he has been working in the State Key Laboratory of Synthetical Automation for Process Industries of Northeastern University. His current research interests include modeling, plant-wide control and optimization for the complex industrial systems, stochastic distribution control, and multiobjective evolutionary algorithms and its application. Around above research, he has published more than ten papers.

Junqing Li received the master degree in computer science and technology from Shandong Economic University, Shandong, China in 2004, and the PhD degree from Northeastern University, Shenyang, China in 2016. Since 2017, he has been with the School of Information Science and Engineering, Shandong Normal University. His current research interests are intelligent optimization and scheduling. He has authored more than 60 refereed papers.

Hongyan Sang received the MS degree from Liaocheng University, Liaocheng, China, in 2010, and the PhD degree in industrial engineering from Huazhong University of Science Technology, Wuhan, China, in 2013. Since 2003, she has been with the School of Computer Science, Liaocheng University, where she became a professor in 2021. Her current research interests include intelligent optimization and scheduling. She has authored more than 60 refereed papers.

Yusuke Nojima received the BS and MS degrees in mechanical engineering from Osaka Institute of Technology, Osaka, Japan, in 1999 and 2001, respectively, and the PhD degree in system function science from Kobe University, Hyogo, Japan, in 2004. Since 2004, he has been with Osaka Prefecture University, Osaka, Japan, where he is currently a professor in the Department of Computer Science and Intelligent Systems. His research interests include evolutionary fuzzy systems, evolutionary multiobjective optimization, and parallel distributed data mining. He was a guest editor for several special issues in international journals. He was a task force chair on Evolutionary Fuzzy Systems in Fuzzy Systems Technical Committee of IEEE Computational Intelligence Society. He was an associate editor of *IEEE Computational Intelligence Magazine* (2014−2019).