

Multi-objective Optimization of the Distributed Permutation Flow Shop Scheduling Problem with Transportation and Eligibility Constraints

Shuang Cai^{1,2} · Ke Yang^{1,2} · Ke Liu^{1,2,3,4}

Received: 4 September 2016 / Revised: 22 February 2017 / Accepted: 19 April 2017 /

Published online: 28 May 2017

© Operations Research Society of China, Periodicals Agency of Shanghai University, Science Press, and Springer-Verlag Berlin Heidelberg 2017

Abstract In this paper, we consider the distributed permutation flow shop scheduling problem (DPFSSP) with transportation and eligibility constraints. Three objectives are taken into account, i.e., makespan, maximum lateness and total costs (transportation costs and setup costs). To the best of our knowledge, there is no published work on multi-objective optimization of the DPFSSP with transportation and eligibility constraints. First, we present the mathematics model and constructive heuristics for single objective; then, we propose an improved The Nondominated Sorting Genetic Algorithm II (NSGA-II) for the multi-objective DPFSSP to find Pareto optimal solutions, in which a novel solution representation, a new population re-/initialization, effective crossover and mutation operators, as well as local search methods are developed. Based on extensive computational and statistical experiments, the proposed algorithm

This research was partially supported by the National Natural Science Foundation of China (Nos. 71390334 and 11271356).

✉ Shuang Cai
caishuang@amss.ac.cn

Ke Yang
keyang@amss.ac.cn

Ke Liu
kliu@amss.ac.cn

- ¹ Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
- ² University of Chinese Academy of Sciences, Beijing 100049, China
- ³ Key Laboratory of Management, Decision and Information Systems, Chinese Academy of Sciences, Beijing 100190, China
- ⁴ National Center for Mathematics and Interdisciplinary Sciences, Chinese Academy of Sciences, Beijing 100190, China

performs better than the well-known NSGA-II and the Strength Pareto Evolutionary Algorithm 2 (SPEA2).

Keywords Multi-objective optimization · Distributed scheduling · Permutation flow shop scheduling · Transportation · NSGA-II

Mathematics Subject Classification 68M14 · 68M20 · 97M40

1 Introduction

Scheduling is a decision-making process that plays important roles in both manufacturing and service industries [1]. The flow shop scheduling problem (FSSP) is one of the most common manufacture layouts in practice where all productions have the same processing sequence. The permutation FSSP (PFSSP) is a special case of the FSSP, which satisfies every job to be processed on all machines is in the same order. This results in a set of $n!$ different candidate solutions, where n is the number of jobs to be processed. The PFSSP is proved to be strongly NP-hard when there are more than three machines [2]. A few heuristics have been proposed with aim to obtain high-quality solutions, such as shortest processing time (SPT), largest processing time (LPT), Johnsons' rule [3], the index heuristics of Palmer [4], Campbell-Dudek-Smith (CDS) method of Campbell, Dudek and Smith [5] and so on. Among these algorithms, the Nawaz-Enscore-Ham (NEH) heuristics developed by Nawaz et al. [6] is believed to be the most effective one.

In the PFSSP, there is only one production place or factory, which means that all jobs must be assigned to the same factory. However, to meet the demand of making factories closer to customers, the managers usually locate their factories in different geographic locations. Scheduling in distributed systems is more difficult than the classical non-distributed scheduling, as we need to make two decisions: which factory to produce each job and in which order each job to be produced in its assigned factory. Compared with considerable literatures on single factory scheduling, few literatures addressed distributed scheduling. The distributed job shop scheduling problem is probably the first field to be studied in distributed scheduling by Jia et al. [7], and genetic algorithm was employed to solve the problem. Later, researchers proposed several GA-based algorithms to solve the distributed job shop scheduling problem [8–12]. In recent years, different types of the distributed flow shop scheduling problem with assembly lines have been studied by many researchers, such as Deng et al. [13] and Wang et al. [14]. Naderi and Ruiz [15] first studied the DPFSSP with objective makespan. In their study, six different mixed integer linear programming models for the DPFSSP and 12 heuristic algorithms were presented. The results demonstrated that the NEH method [6] with the second job to factory assignment rule (NEH2) has the best performance among the 12 heuristic algorithms. Liu and Gao [16] presented a complex electromagnetism meta-heuristic (EM) for the DPFSSP, and they extended Variable Neighborhood Descent (VND) local search which is proposed by [15] to a more powerful variable neighborhoods such as insertion within the critical factory, swap in the critical factory and general insertion and swap. Gao and Chen [17] proposed a GA

with local search, in which NEH2 and VND (a) were employed for initialization. Gao et al. presented an improved NEH2 heuristics in [18] and a knowledge-based GA in [19] that performs better than GA\LS. Recently, Gao et al. [20] proposed a tabu search method which outperforms the hybrid GA (HGA) of Gao and Chen [17]. Lin et al. [21] presented an iterated greedy (IG) algorithm. Wang et al. [22] proposed an estimation of distribution algorithm (EDA) which outperforms VND (a). Xu et al. [23] proposed a hybrid immune algorithm (HIA) for solving the DPFSSP, and the effectiveness of the HIA was demonstrated by comparing with some heuristics and the variable neighborhood descent methods. Naderi and Ruiz [24] present a scatter search (SS) method for the DPFSSP to optimize makespan. Compared with 10 existing algorithms, the results showed that the proposed SS has better performance than existing algorithms by a significant margin. Recently, a hybrid discrete cuckoo search algorithm was proposed by Wang et al. [25] for the DPFSSP and a fuzzy logic-based hybrid estimation of distribution algorithm was proposed by Wang et al. [26] for the DPFSSP under machine breakdown.

Practical scheduling is usually faces multi-objective decisions. The multi-objective optimization problems are originally conceived with finding Pareto optimal solutions, i.e., efficient solutions [27]. Such solutions are non-dominated, i.e., no other solutions are superior to them when all objectives are taken into account. A significant number of researches studied the flow shop scheduling problem with more than two objectives, e.g., [28–31]. The objectives include makespan, maximum lateness, maximum tardiness, total flow time, total costs and so on. Yagmahan et al. [32] presented an ant colony algorithm to solve the multi-objective flow shop scheduling problem with three objectives makespan, total flow time and total machine idle time. To minimize the sum of weighted flow time and weighted tardiness of jobs for the m -machine flow shop scheduling problem, Gelders and Sambandam [33] developed four simple heuristics to obtain approximate Pareto solutions. Ponnambalam et al. [34] proposed a GA based on the traveling salesman algorithm for the m -machine flow shop scheduling problem with three scheduling objectives, i.e., the weighted combination of makespan, mean flow time and machine idle time. This hybrid GA employed randomly generated weights to sum the objectives as a scalar fitness function. Although rich papers have been published for solving multi-objective of flow shop scheduling [35], however, seldom literatures are dedicated for solving the multi-objective DPFSSP. Till now, only Deng et al. [36] solved the bi-objective DPFSSP with respect to the minimization of makespan and total tardiness by competitive memetic algorithm (CMA). Compared with the random algorithm and the well-known NSGA-II, the CMA performed better.

As complexity of the multi-objective scheduling problem, researchers developed heuristic and meta-heuristic algorithms to solve this kind of problem. Among the meta-heuristics, the SPEA2 (Zitzler et al. [37]) and NSGA-II (Deb et al. [38]) are considered to be the most well known algorithm for providing better results. Bandyopadhyay et al. [39] presented a modified NSGA-II to solve the parallel machine scheduling problem with three objectives (total cost due tardiness, the deterioration cost and makespan) and compared the results with the SPEA2 and NSGA-II. Bolanos et al. [40] used the NSGA-II for solving the multiple traveling salesman problems. For task scheduling in grid computing, Sahu et al. [41] applied the NSGA-II to optimize the problem with two objectives: maximizing availability and minimizing makespan.

In order to solve practical scheduling with release times in steel plants, Long et al. [42] proposed a new algorithm based on the NSGA-II. Autuori et al. [43] considered the flexible job shop problem with two objectives: makespan and to producing jobs just in time. The authors compared the space exploration between the NSGA-II and SPEA2.

In most manufacturing and distribution systems, the unfinished jobs (raw materials) are transferred from supplies to manufacturing factories by transporters and the finished jobs are delivered to clients or warehouses by vehicles [44]. As transportation is one of the most important parts of logistics, it has great effects on the performance of competitiveness of modern enterprises [45]. Considerable research efforts have been put on minimizing transportation costs since 1960 ([46, 47]). Specifically, in flow shop scheduling, researches usually considered the transportation time from one machine to the next machine, such as [48–51].

The constraints of factory eligibility in the DPFSSP could be considered as the extension of conventional machine eligibility constraints in FSSP. Factory eligibility puts limitation on the assignment of jobs to factory, which means that a job cannot be assigned to a factory if this job cannot be processed on at least one machine of this factory. Machine eligibility has been concentrated on parallel machines or hybrid flow shop scheduling such as [52, 53], while it has seldom been considered in the DPFSSP.

Based on the above literature analysis, it could be concluded that there is no published work on multi-objective optimization of the DPFSSP with transportation and eligibility constraints. The objective of this investigation is explicitly set out to fulfill this role. In this paper, three stages are considered. In stage 1, each job needs to be transported from suppliers to one of its available factories. In stage 2, each job is produced in its assigned factory. After production, each job will be transported to the clients 1, 2, \dots , S in stage 3. Each factory transports the productions to fixed clients or warehouses (only related to factories) in the third or last stage. The production times of job j on machine i are related to its assigned factory. For each job, it must be assigned to one of its available factories. At any factory, each assigned job goes through processing over m machines in the flow shop mode. We focus on permutation flow shop such that all the jobs are processed in the same order on the m machines. We can denote the problem by the notation of $\alpha/\beta/\gamma$ in [1]: DPFSSP/Transportation, $M_j, d_j/C_{\max}, L_{\max}, TC$, where the factory eligibility M_j and due date d_j of job j are known; C_{\max}, L_{\max}, TC are three scheduling objectives, i.e., makespan, maximum lateness and total costs, respectively. The overall process is depicted in Fig. 1. However, even though the problem we considered has three stages, it can be transformed into two stages by combining the first stage and the last stage into one stage, as jobs assigned the same factory have the same transportation time to the final client or warehouse. Then the problem can be cast as: in stage 1, each job needs to be transported from suppliers to one of its available factories (the transportation time (cost) of each job is the sum of transportation times (costs) from suppliers to factories and from factories to the final client or warehouse); in stage 2, jobs are processed sequentially in the distributed factories. Once the assignment is determined for each factory, it can be seen as PFSSP with releasing time.

The remaining paper is arranged as follows. In Sect. 2, we give the mixed integer mathematic programming model of the DPFSSP with transportation and eligibility

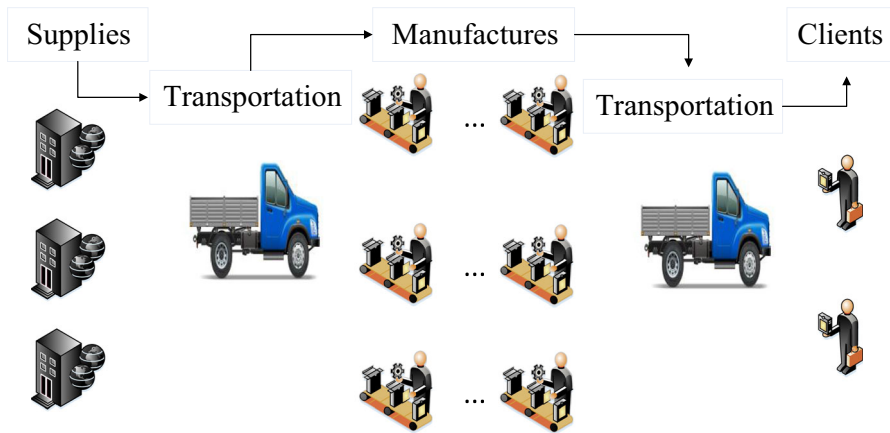


Fig. 1 Framework of procedure from the suppliers to the final clients

constraints. In Sect. 3, several simple constructive heuristic algorithms for minimizing makespan and maximum lateness and a greedy algorithm for minimizing total costs are described. In Sect. 4, an improved NSGA-II is introduced; and in Sect. 5, computational results and comparisons with the NSGA-II and SPEA2 are provided. Finally, we conclude the paper and present the future research in Sect. 6.

2 Mathematic Model

This section contains three parts: the assumptions, the notations and formulation of the problem.

2.1 Assumptions

Here are all the assumptions for the problem:

1. Each job can be processed in at least one factory;
2. Each job can only be assigned to one of its available factories;
3. Each factory can be seen as a permutation flow shop;
4. Each machine of any factory can process only one job at a time; each job can only be processed by one machine of a factory at a time;
5. There may be different transportation times (costs) when transporting a job to different factories;
6. The setup cost of each factory is a fixed value when we assign at least one job to it; otherwise, its setup cost is 0;
7. There is a due date for each job.

2.2 Notations

Variables are as follows:

- $X_{k,j,f}$: 1 if job J_j is processed in factory f , immediately after job J_k , 0 otherwise.
- $Y_{j,f}$: 1 if job J_j is processed in factory f , 0 otherwise.
- $I(x)$: 1 if x is positive and 0 otherwise.
- $C_{j,i}$: Continuous variable for the complete time of job J_j on machine i .
- C_{\max} : Continuous variable for the complete time of all the jobs.
- L_{\max} : Continuous variable for the maximum lateness of all the jobs.

Parameters and indices are as follows:

- F : Number of factories.
- n : Number of jobs.
- m : Number of machines in each factory.
- j, k : Index of job J_j, J_k .
- f : Index of factory, $f = 1, 2 \dots F$.
- $p_{j,i}^f$: Processing time of job j on machine i in factory f .
- $M_{j,f}$: 1 if job j can be processed in factory f ; 0 otherwise.
- D_j : The due date of job j .
- S : A sufficiently large positive number.
- $T_{j,f}$: Transportation time of job j when transporting it to factory f .
- $c_{j,f}$: Transportation cost of job j when transporting it to factory f .
- c_f^g : The fixed setup cost of factory f .

2.3 Formulation of the Proposed Problem

We introduce a dummy job 0 which precedes the first job in the sequence. Three objectives can be formulated as:

$$\min \quad C_{\max}, \tag{2.1}$$

$$L_{\max}, \tag{2.2}$$

$$TC = \sum_{f=1}^F \sum_{j=1}^n c_{j,f} Y_{j,f} + \sum_{f=1}^F I \left(\sum_{j=1}^n Y_{j,f} \right) c_f^g, \tag{2.3}$$

where makespan (C_{\max}) represents the maximum complete time of all jobs, maximum lateness (L_{\max}) means the maximum of lateness of all jobs, total costs (TC) include the transportation costs of all jobs and setup costs of all factories which process at least one job.

Subject to the constraints:

$$\sum_{k=0, k \neq j}^n \sum_{f=1}^F X_{k,j,f} = 1, \quad \forall 1 \leq j \leq n, \tag{2.4}$$

$$\sum_{f=1}^F Y_{j,f} = 1, \quad \forall 1 \leq j \leq n, \quad (2.5)$$

$$\sum_{k=1, k \neq j}^n (X_{k,j,f} + X_{j,k,f}) \leq 2Y_{j,f}, \quad \forall 1 \leq j \leq n, 1 \leq f \leq F, \quad (2.6)$$

$$\sum_{j=1, j \neq k}^n \sum_{f=1}^F X_{k,j,f} \leq 1, \quad \forall 0 \leq k \leq n, \quad (2.7)$$

$$\sum_{f=1}^F (X_{k,j,f} + X_{j,k,f}) \leq 1, \quad \forall 1 \leq k \leq n-1, j > k, \quad (2.8)$$

$$Y_{j,f} \leq M_{j,f}, \quad \forall 1 \leq j \leq n, 1 \leq f \leq F, \quad (2.9)$$

$$C_{j,i} \geq C_{j,i-1} + P_{j,i}^f - S(1 - Y_{j,f}), \quad \forall 1 \leq j \leq n, 2 \leq i \leq m, 1 \leq f \leq F, \quad (2.10)$$

$$C_{j,i} \geq C_{k,i} + P_{j,i}^f - S(1 - X_{k,j,f}), \quad \forall 1 \leq j, k \leq n, j \neq k, 1 \leq i \leq m, 1 \leq f \leq F, \quad (2.11)$$

$$C_{\max} \geq C_{j,m}, \quad \forall 1 \leq j \leq n, \quad (2.12)$$

$$L_{\max} \geq C_{j,m} - D_j, \quad \forall 1 \leq j \leq n, \quad (2.13)$$

$$C_{j,1} \geq T_{j,f} + P_{j,1}^f - S(1 - Y_{j,f}), \quad \forall 1 \leq j \leq n, 1 \leq f \leq F, \quad (2.14)$$

$$X_{j,k,f}, Y_{j,f} \in \{0, 1\}, \quad \forall 1 \leq j, k \leq n, j \neq k, 1 \leq f \leq F, \quad (2.15)$$

where constraint (2.4) controls every job has exactly at one position and only at one factory; constraint (2.5) ensures every job must be exactly assigned to one factory; constraint (2.6) represents every job can be at most one successor of a job and one predecessor of another job; constraint (2.7) assures every job has no more than one successor; constraint (2.8) avoids cross-precedence of any two jobs, which means that a job cannot be both a predecessor and a successor of another job; constraint (2.9) avoids assigning a job to an unavailable factory; constraint (2.10) represents every job is processed at most on one machine at a time while constraint (2.11) guarantees a machine can only process one job at a time; constraints (2.12–2.13) define the makespan and maximum lateness; constraint (2.14) makes sure every job cannot be processed until it arrives the assigned factory.

3 Heuristics for the DPFSSP with Single Objective

3.1 Heuristics for Makespan or Maximum Lateness

In this subsection, we propose two rules for assigning jobs to factories and eight heuristics for generating a permutation of jobs. The heuristics for generating a permutation are SPT, LPT, Johnson [3], Palmer [4], earliest releasing time (ERT), earliest due date (EDD), NEH [6] and a new heuristic NEH adaptive (NEHA) proposed in this

paper based on least flexible job (LFJ) and NEH (by Taillard method [54]), respectively.

Here the new algorithm NEHA is proposed to solve the PFSSP with releasing time based on Taillard method. The procedure is described as follows.

- Step 1 Sequence the n jobs (n is the total number of jobs) by LFJ method. If two jobs have the same number of available factories, the job with a smaller total processing time in these factories will have a forward position;
- Step 2 Take the first two jobs and schedule them in order to minimize the partial makespan (or maximum lateness);
- Step 3 For $k = 3$ to n do:
- Step 4 Insert job k at the place, among the k possible ones, which approximately minimize the partial makespan (or maximum lateness).

The time complexity of step 1 is $O(\log n)$ and that of step 2 is $O(m)$. For every $3 \leq k \leq n$ in step 4, it needs $O(km)$ operations to calculate one partial makespan. Here we give an approximate algorithm to calculate the k makespan (or maximum lateness) in $O(km)$ time:

Determining CM_j , the makespan after insertion of job k at the j th position and the maximum lateness time LM_j , r_s is the releasing time of job s , $t_{j,i}$ is the processing time of the j th job on machine i .

1. Compute the earliest completion time $e_{j,i}$ of the j th job on the i th machine; the starting time of the first job on the first machine is its releasing time;

$$\begin{aligned}
 e_{1,1} &= r_1 + t_{1,1}, \\
 e_{1,i} &= e_{1,i-1} + t_{1,i}, \\
 e_{j,1} &= \max(r_j, e_{j-1,1}) + t_{j,1}, \\
 e_{j,i} &= \max(e_{j,i-1}, e_{j-1,i}) + t_{j,i}, \\
 & \quad j = 2, \dots, k-1, \quad i = 2, \dots, m.
 \end{aligned}$$

2. Compute the tail $q_{j,i}$, i.e., the real duration between the starting time of the j th job on the i th machine and the end of the operations before inserting job k ;

$$\begin{aligned}
 q_{j,i} &= e_{k-1,m} - e_{j,i} + t_{j,i}, \\
 & \quad j = k-1, \dots, 1, \quad i = m, \dots, 1.
 \end{aligned}$$

3. Compute the earliest completion time $f_{j,i}$ on the i th machine of job k inserted at the j th position;

$$\begin{aligned}
 f_{1,1} &= r_k, \quad f_{1,i} = f_{1,i-1} + t_{k,1}, \\
 f_{j,1} &= \max(e_{j-1,1}, r_k) + t_{k,1}, \\
 f_{j,i} &= \max(f_{j,i-1}, e_{j-1,i}) + t_{j,i}, \\
 & \quad i = 2, \dots, m, \quad j = 2, \dots, k.
 \end{aligned}$$

4. The value of the partial makespan CM_j and maximum lateness LM_j when adding job k at the j th position is

$$CM_j = \max_{1 \leq i \leq m} (f_{j,i} + q_{j,i}),$$

$$LM_j = \max (f_{j,m} - D_k, \max_{j' \geq j} (e_{j',m} - D_{j'} + CM_j - e_{k-1,m})),$$

$$i = 1, \dots, m, j = 1, \dots, k.$$

All these computations (2.1–2.4) can be calculated in time $O(km)$. Considering step (4) of the NEHA algorithm which has $O(km)$ time complexity, we conclude the NEHA algorithm runs in time $O(n^2m)$.

The processing (releasing) time of each job is calculated by the average of all processing (transportation) times in its available factories. Then the 8 heuristic algorithms can be used as before.

According to Naderi and Ruiz [15], two rules for assignment are proposed (1) assign a given job j to its available factory with the lowest current makespan, not including job j ; and (2) assign job j to its available factory which completes it at the earliest time, i.e., the available factory resulting in the lowest makespan after assigning job j . We should make two decisions: (1) assign each job to one of its available factories (2) the permutation of each factory. Then we can obtain 16 heuristic algorithms for the combination of job assignment and job priority. Latter we will give the results of these 16 heuristic algorithms with 400 examples (150 small-sized instances and 250 large-sized instances) in Table 1 with objectives makespan and maximum lateness. The relative percentage deviation (RPD) is regarded as the results of algorithm:

$$RPD_{g,h} = \left| \frac{V_{g,h} - V_{g,\min}}{V_{g,\min}} \right|, \quad (3.1)$$

where $V_{g,h}$ is the objective value makespan or maximum lateness of the g th instance using algorithm h , $V_{g,\min}$ is the minimum objective value of the g th instance among all algorithms. It can be seen from Table 1 that the second rule of assignment is better compared with the first one and the new heuristic algorithm has better performance with objective makespan than the other algorithms except NEH1 and NEH2. When the objective is maximum lateness time, the permutation of EDD is better and the performance of EDD2 is almost the same as NEH2. This is partly because any two jobs may have different due date in our instances, as the due-date time of each job is different in our instances and the rule of EDD makes sure that the job with smaller due date can be completed earlier. It can be seen that EDD is an effective rule for scheduling jobs with different due dates for objective maximum lateness.

Table 1 Average RPD of the heuristic algorithm

Makespan	Algorithm							
	SPT1	LPT1	Johnson1	Palmer1	ERT1	EDD1	NEH1	NEHA1
Small	0.13	0.17	0.10	0.15	0.18	0.15	0.06	0.05
Large	0.19	0.21	0.19	0.18	0.19	0.20	0.03	0.13
	Algorithm							
	SPT2	LPT2	Johnson2	Palmer2	ERT2	EDD2	NEH2	NEHA2
Small	0.11	0.13	0.07	0.13	0.13	0.12	0.02	0.05
Large	0.15	0.18	0.17	0.16	0.17	0.17	0.00	0.11
MaxLateness	Algorithm							
	SPT1	LPT1	Johnson1	Palmer1	ERT1	EDD1	NEH1	NEHA1
Small	1.03	0.85	0.90	1.02	0.85	0.20	0.31	0.43
Large	2.61	2.69	2.40	2.51	2.22	0.24	0.24	0.37
	Algorithm							
	SPT2	LPT2	Johnson2	Palmer2	ERT2	EDD2	NEH2	NEHA2
Small	0.92	0.74	0.93	0.85	0.64	0.10	0.04	0.16
Large	2.12	2.20	2.04	2.11	1.83	0.03	0.02	0.25

400 instances

3.2 A Greedy Algorithm for Objective Total Costs

In Sect. 3.2, we proposed a greedy algorithm for objective total costs. As the costs are not relevant to the schedule of each factory, the greedy algorithm only deals with the assignment of jobs.

The greedy algorithm can be easily described as follows:

1. For each job j from 1 to n ,
2. Assign job j to factory f which has the smallest transportation cost $c_{j,f}$.

The complexity of this greed algorithm is $O(nF)$. Even though the number of all available assignments is $|M_1| |M_2| \cdots |M_n|$ (M_j is the set of available factories of job j), there has an exact algorithm which has the complexity of $O(nF2^F)$ based on the greedy algorithm.

The exact algorithm is based on greedy algorithm:

For each subset of $\{1, 2, \dots, F\}$,

If every job has at least one available factory, use the above greed algorithm.

There are 2^F subsets of $\{1, 2, \dots, F\}$ and the complexity of step 2 for each subset is $O(nF)$, so the complexity of this algorithm is $O(nF2^F)$. Because the minimum total costs must be one of the conditions, this algorithm can be easily proved to be an exact algorithm.

4 An Improved NSGA-II Algorithm

In this section, an improved NSGA-II (Non-dominated Sorting Genetic Algorithm) is proposed to solve multi-objective optimization of the DPFSSP with transportation

and eligibility constraints. NSGA-II could be characterized by population based on GA and non-domination sorting which assigns rank and crowding distance to each individual (chromosome) in the population. Besides, NSGA-II is the most widely applied multi-objective evolutionary algorithm (MOEA) as observed in the existing literature, especially in scheduling. Both crossover and mutation operators based on the type of problem have been embedded in the proposed modified NSGA-II. After crossover and mutation, local search is used to find the better neighborhoods of individuals. Figure 3 shows the procedure for the improved NSGA-II. The algorithm continues to execute till the maximum experiment time. The main components of the algorithm are summarized below.

4.1 The Representation of Individuals

For each individual (an available schedule), we have to make two decisions for every job: which factory it is assigned to and which position in its assigned factory. The first decision is called job assignment, and second decision is called job priority. We encode each individual in $2 \times n$ matrix if the total number of jobs is n . Here we give a theorem to explain the rationality of this encoding.

Theorem 4.1 *Each individual can be encoded in a $(2 \times n)$ matrix A , which represents job priority (a permutation from 1 to n) in the first row and job assignment in the second row, respectively, when the number of jobs is n ; each $2 \times n$ matrix A , which consists of a permutation from 1 to n in the first row and integers between 1 and F in the second row, respectively, when there are n jobs and F factories, is an available schedule when $A_{2,j} \in M_{A_{1,j}}$.*

Proof For any available schedule, each job must be assigned to a position of its available factory. If factory f ($1 \leq f \leq F$) has the production sequence (the number of jobs assigned to factory f is N_f) f_1, f_2, \dots, f_{N_f} , then the job priority can be written as $1_1, \dots, 1_{N_1}, \dots, F_{N_F}$; job assignment recodes the factories where jobs are processed and if the mapping between the jobs and the factories is $\varphi : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, F\}$. Here we have $\varphi(f_i) = f$. Just let A be

$$\begin{pmatrix} 1_1 & 1_2 & \dots & 1_{N_1} & \dots & F_1 & \dots & F_{N_F} \\ 1 & 1 & \dots & 1 & \dots & F & \dots & F \end{pmatrix}.$$

If a matrix A is given, we get the job priority in the first row and job assignment in the second rule. Every job will be processed in the fixed factory according to the job assignment and jobs assigned to the same factory are processed sequentially by job priority, i.e., a job with higher priority will be processed before a job with lower priority. If every job is assigned to its available factory, the matrix A represents an available schedule.

The standard encoding scheme satisfies the second row of A is non-reduced. The individuals in our proposed algorithm are encoded as a standard coding almost all the time.

4.2 Initialization

In order to obtain a population with better performance for the three objectives, the initial population are initialized by three parts with the three scheduling objectives, respectively. After obtaining P_{size} individuals as an initial population, we copy them to the archives.

- Part 1 The assignment of each job to factory is generated by using the greedy algorithm with objective TC if each job can be processed in one of the selected r (generated randomly) factories; else, continue adding factory until we get a set of factories which satisfies that each job can be processed in one of the selected factories. The sequence of each factory is generated by applying NEHA for objective makespan and EDD for objective maximum lateness.
- Part 2 First an individual is obtained by using EDD2 with objective maximum lateness. The job priority of each of the remaining individuals is randomly generated. Then, the remaining individuals are obtained by applying NEHA2 with objective maximum lateness.
- Part 3 Each individual is obtained by using the NEHA2 heuristic algorithm with objective makespan, and its job priority is randomly generated. Then we can get different solutions when using different job priorities.

4.3 Crossover

As there are factory eligibility constraints in the considered problem, a two-point crossover operator is applied in the algorithm based on the structure of individuals.

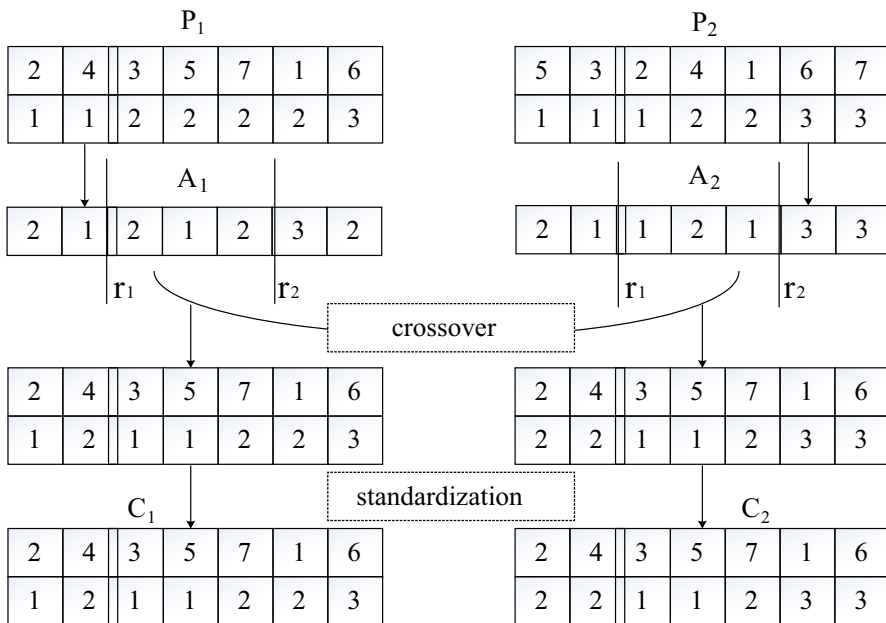


Fig. 2 Procedure of crossover operator

By the following method, two generated children will be available schedules after crossover. The crossover operator is as follows (the procedure of crossover is shown in Fig. 2):

- Step 1 Generate two points randomly: first get one integer point r_1 randomly in $[1, n]$, and then get next integer point r_2 randomly in $[1, n - 1]$. If $r_2 \geq r_1$, let $r_2 = r_2 + 1$.
- Step 2 For two parents P_1 and P_2 , the assignments of jobs are $A_1(A_2)$ for $P_1(P_2)$.
- Step 3 Crossover A_1 and A_2 between the r_1 th position and the r_2 th position.

4.4 Mutation

According to the characteristics of the DPFSSP with eligibility constraints, two mutation operators are designed to modify the factory with largest completion time or the job with maximum lateness time in the schedule. For a given schedule, one job j is chosen by one of two methods with the same probability 1/2:

Method 1 The job with maximum lateness time (denoted by j);

Method 2 A job which is selected randomly from the jobs which are assigned to the factory with the latest completion time.

Then the total processing time of job j in its available factories is calculated. Suppose that the available factory of job j is f_1, f_2, \dots, f_{a_j} . For f_i , the total processing time of job j in factory f_i is $t_{j,i} = \sum_{k=1}^m p_{j,k}^{f_i}$. Without lose of generality, the original assignment of job j is f_1 , we get the probability of assignment to factory f_i of job j

$$p_i = \frac{1/t_{j,i}}{\sum_{k=2}^{a_j} 1/t_{j,k}}, \quad i = 2, 3, \dots, a_j.$$

The mutation is very similar with the method of roulette wheel selection, and the implementation is the same.

4.5 Non-dominated Sorting and Crowding Distance

In this paper, we use the method of NSGA-II to get a permutation of a group of individuals (a population). The individuals (chromosomes) in the population will be assigned a rank based on non-domination sorting: All non-dominated individuals are classified into one category (its rank is denoted by 1). Deleting the individuals with rank 1, the non-dominated individuals among the remaining jobs belong to a new category with rank 2 and so on. The fast sort algorithm is used in our paper.

Once the non-dominated sort is completed, the crowding distance is assigned. The crowding distance is calculated and the larger distance is chosen to keep a diverse front by making sure that each member stays a crowding distance apart. This keeps the population diverse and helps the algorithm to explore the fitness landscape. Since the individuals are selected based on rank and crowding distance, all individuals in the population should be assigned a crowding distance value. The crowding distance is calculated as below.

For each front F_i (the individuals set with rank i), let n_i be the number of individuals. Initialize the distance to be zero for all the individuals, i.e., $d_h(F_i(j)) = 0$, where j

is corresponding to the j th individual in front F_i and h is corresponding to the h th scheduling objective.

Step 1 For each objective function h , sort the individuals in front F_i based on objective h , i.e., $I_h = \text{sort}(d_h(F_i))$, here I_h represents a permutation (1 to n_i) of individuals in front F_i .

Step 2 Assign infinite distance to boundary values for each individual in F_i , i.e., $d_h(F_i(I_h(1))) = \infty$ and $d_h(F_i(I_h(n_i))) = \infty$.

Step 3 For $k = 2$ to $n_i - 1$.

$$d_h(F_i(I_h(k))) = \frac{f^h(F_i(I_h(k+1))) - f^h(F_i(I_h(k-1)))}{f_{\max}^h - f_{\min}^h}.$$

Here f_{\max}^h (f_{\min}^h) is the maximum (minimum) h th objective value of all individuals, $f^h(j)$ is the h th objective value of individuals j .

Step 4 As we have three objectives, i.e., $h = 1, 2, 3$,

$$d_j = \sqrt{(d_1(j))^2 + (d_2(j))^2 + (d_3(j))^2}.$$

Here d_j is the crowding distance for individual (chromosome) j .

The basic idea behind the crowding distance is to obtain the Euclidian distance based on three objectives in the three-dimensional hyper space. The individuals in the boundary are always selected since they have infinite distance assignment.

When there are two individuals which have the same objectives, it may be the case that they are the same individual and one of them should be eliminated. But the above algorithm keeps both of them when they have boundary objective values. Here we give an eliminating algorithm after getting the rank and crowding distance of each individual.

Suppose the maximum rank is r , n_i is the number of individuals in rank i and n is the number of jobs.

Step 1: For $i = 1 : r$,

Step 2: If $n_i > 1$

Step 3: For $j = 1 : (n_i - 1)$,

Step 4: For $k = 2 : n_i$, $j < k$

Step 5: If the objectives of job j and job k are the same,

Step 6: assign rank $r + 1$ to individual j , break;

Step 7: end if.

Step 8: end for

Step 9: end for

Step 10: end if

Step 11: end for.

Non-dominated sorting is performed after local search operators are completed in the proposed algorithm and P_{size} individuals are selected into the next generation. The population individuals will be compared with the archive individuals by using non-dominated sorting and the best N individuals are selected into the archives.

4.6 Local Search

It is widely accepted that local search operators are efficient in improving the quality of solutions when using meta-heuristic algorithm. In this paper, two kinds of local search operators are designed based on the scheduling objectives makespan and maximum lateness.

1. Operators for the makespan criterion

Since the makespan of a solution can be reduced by modifying the schedule in the factory (denote as f_c) with the latest completion time, four operators are proposed as follows:

Job-swap randomly select two jobs in factory f_c and swap the positions of two jobs.

Job-insert randomly select two jobs in factory f_c and move the latter job immediately before the former one.

Job-inverse randomly select two jobs in the factory f_c and inverse the jobs between the selected two jobs.

Factory-insert randomly select a job in factory f_c and insert it to a randomly selected position in one of its available factories.

2. Operators for the maximum lateness criterion

Since the maximum lateness of a solution can be reduced by modifying the job (denote as J_l in factory f_l) with the maximum lateness time, four operators are proposed as follows:

Job-swap randomly select a job which has the front position compared with J_l in factory f_l and swap the positions of these two jobs.

Job-insert randomly select a job which has the front position compared with J_l in factory f_c and insert J_l to the front one.

Job-inverse randomly select a job which has the front position compared with J_l in the factory f_c and inverse the jobs between the selected two jobs.

Factory-insert delete job J_c in factory f_c and insert it to a randomly selected position in each available factory of the other $F - 1$ factories.

4.7 Selection

For every generation, after crossover and mutation, individuals of the population are selected from both parents and children, and then local search is carried out. After local search, we have to update the next population. The principle of our selection is based on the rank and crowding distance of each individual. After the update of

generation, we put the original archives and the new population together. Then, the archives are updated by choosing the lower rank and the larger crowding distance.

4.8 Re-/initialization

If the objectives of the archives are not improved several times, two kinds of re-/initialization have been used:

- Case 1 If the total times reach n_1 , we update half of the population by half of the randomly selected archives.
- Case 2 If the total times reach n_2 ($n_2 > n_1$), we update half of the population by new individuals which are generated by the following method: The job priority is obtained by random and all jobs are randomly assigned to available factories for getting $P/2$ individuals.

4.9 Procedure of the Improved NSGA-II

With the above design, the procedure of this algorithm for solving the considered multi-objective problem is illustrated in Fig. 3.

Step 1 Initialize the population by the method of Sect. 4.2 and then copy the individuals to archives. Calculate the objectives of population (archives).

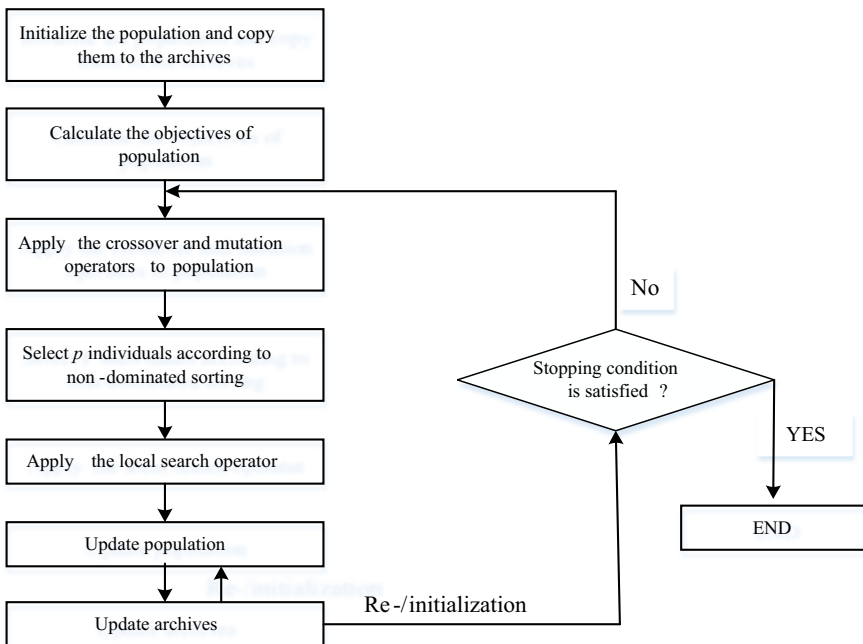


Fig. 3 Framework of the proposed algorithm procedure

- Step 2 Apply crossover and mutation operators to each population without selection.
- Step 3 Select P individuals from the parents and children according to non-dominated sorting in Sect. 4.5.
- Step 4 Apply one kind of local search operators to P individuals.
- Step 5 Update the population from the generated individuals from Step 4 and the original P individuals.
- Step 6 Update the archives from the original archives and new population.
- Step 7 If stopping condition is satisfied, end with the results of new archives; else do the re-initialization operator.
- Step 8 Update the sequence of population into a substitution of the original sequence randomly and go to Step 2.

It can be seen that when the population go to the steps of crossover and mutation, we do not select parts of individuals to crossover and mutation as they are usually non-dominated solutions. All individuals will be parents, and they all have children. Of course, before the individuals of population go to the step of crossover, the sequence of individuals in population will be updated into a substitution of the original sequence randomly.

4.10 Computational Complexity Analysis

In our proposed algorithm, the initial solutions are generated based on the three proposed heuristic algorithms. According to Sect. 4.2, there are three parts of individuals and the time complexity of them are $O(P_{\text{size}}F + P_{\text{size}}^2m)$, $O(P_{\text{size}}^2m)$ and $O(P_{\text{size}}^2m)$, respectively, so the total time complexity for initialization is $O(P_{\text{size}}F + P_{\text{size}}^2m)$.

At each generation of the population excluding initialization, the computational complexity can be analyzed as follows.

First, we need the crossover and mutation operators for each parent when updating, so the time complexity is $O(P_{\text{size}} \log P_{\text{size}})$. The time complexity for non-dominated sorting and crowding distance is $O(P_{\text{size}}^2)$ and $O(P_{\text{size}} \log P_{\text{size}})$. When doing local search operators, all the eight methods have at most $O(P_{\text{size}} \log P_{\text{size}} + P_{\text{size}}F)$ time complexity. The selection and Re/-initialization both have $O(P_{\text{size}})$ time complexity. The time complexity for updating the archives is $O(P_{\text{size}}^2)$.

From the above analysis, we can conclude that the computational complexity is not large. The time complexity of the proposed algorithm is almost the same as both the original NSGA-II and SPEA2 (time complexity $O(P_{\text{size}}^2)$).

5 Experiment of the Proposed NSGA-II Method

In order to test the performance of the new algorithm, lots of instances are carried out based on two sets of benchmarks (Naderi and Ruize [15]). The first set includes 150 small-sized instances, and the other one includes 250 larger-sized instances. We will test the improved NSGA-II with two algorithms, NSGA-II and SPEA2, which are the most efficient multi-objective algorithms. The experiments have been conducted

in a PC with 2.8 GHz processor and 1 GB memory. MATLAB 2014a has been used to program the proposed algorithm, the original NSGA-II and SPEA2.

5.1 The Evaluation of Results of the Experiments

The proposed algorithm is a meta-heuristic algorithm, and we expect to find the approximate optimal solution (Pareto frontier). Therefore, we give four measures to evaluate the performance of results, and there are ten runs for each instance and each algorithm. The first measure is the minimum and average values of three objectives among the final results of an algorithm ([39]). The second measure was the average number of non-dominated solutions obtained after ten runs of each instance ([55]). Next one is a quality indicator that creates a Pareto frontier composed of the non-dominated solutions from all three algorithms and calculates the percentage of solutions in each algorithm ([56]). The last measure is expressed by the equation below, which is called spacing, that reflects the uniformity of the spread of solutions in ten runs (the total number of results of ten runs is $Q=10 \times 40=400$).

$$S = \sqrt{\frac{1}{Q} \sum_{i=1}^Q (d_i - \bar{d})^2}, \quad (5.1)$$

where d_i and \bar{d} is calculated by the following Eqs. (5.2) and (5.3)

$$d_i = \min_{k \neq i} \left\{ \sum_{h=1}^3 |f_h^k - f_h^i| \right\}, \quad (5.2)$$

$$\bar{d} = \frac{1}{Q} \sum_{i=1}^Q d_i. \quad (5.3)$$

Here f_h^k represents the h th objective of the k th result (three objectives).

5.2 Parameters Setting

The proposed algorithm contains several key parameters:

P_{size} (population size), \bar{P}_{size} (archive size), P_c (crossover probability), P_m (mutation probability), E_t (experimentation time of algorithm).

In order to get high-quality solutions of the considered problem, we need to give an appropriate value to each parameter. As it is difficult to find the best results of each instance, we set the experiment time (seconds) $E_t = 120$ according to [55]. Here we implement an experiment by using a moderate-sized instance ($n=14, m=5, F=4$).

For each parameter combination, we run 10 times of our proposed algorithm and use the volume of three objective deviations (deviation i is calculated by $(\bar{f}^i - f_{\min}^i) / (f_{\max} - f_{\min})$, where \bar{f}^i represents the average i th objective value) as the evaluation value (EV). Here we set $P_{\text{size}} = \bar{P}_{\text{size}}$.

Table 2 Parameter setting

Experiment number	P_{size}	P_c	P_m	EV
1	40	0.9	0.1	0.049
2	40	0.8	0.2	0.046
3	40	0.7	0.3	0.047
4	40	0.6	0.4	0.046
5	40	0.5	0.5	0.044
6	40	0.4	0.6	0.045
7	40	0.3	0.7	0.046
8	40	0.2	0.8	0.051
9	40	0.1	0.9	0.052
10	20	0.5	0.5	0.043
11	40	0.5	0.5	0.0436
12	60	0.5	0.5	0.046

As $\{P_c, P_m\}$ (put P_c and P_m together as $P_c + P_m = 1$) and P_{size} are independent parameters, according to the data of Table 2, we choose the value of $P_{\text{size}} = \bar{P}_{\text{size}} = 40$, $P_c = 0.5$, $P_m = 0.5$, $E_t = 120$.

5.3 Comparison with NSGA-II and SPEA 2 in Small-Sized Instances

The set of small-sized instances which are randomly generated are as follows:

$n = 4, 6, 8, 12, 14, 16$, $m = 2, 3, 4, 5$, $F = 2, 3, 4$ and M is a 0-1 matrix generated randomly which make sure that the sum of each line is at least one. We randomly selected 30 combinations of $n \times m \times F$ and for each combination, 5 different M are generated, so the total number of small-sized instances is 150. The transportation times and costs are generated by the following expressions $T_{i,f} = U([1, 50m]) + 10m$ and $c_{i,f} = U([1, 10])$. The setup cost of each factory is $c_f^s = U([1, 10F]) + 10$. Due date of each job is calculated by the expression $D_j = U([1, 50nm]) + 100m$. Here $U([a, b])$ is the uniform distribution of integers between a and b . The processing times are uniformly distributed over the range (1, 99) as usual in the scheduling literature. The results of the experiments are presented in Tables 3 and 4.

Table 3 shows the minimum and average values of objectives among the three algorithms. The minimum and average results of each algorithm are calculated by the following equations:

$$\text{ARD}_{j,m,i,k} = \left| V_{j,m,i,k} / \min_{1 \leq i \leq 3} V_{j,m,i,k} - 1 \right|, \quad (5.4)$$

$$\text{ARD}_{j,a,i,k} = \left| V_{j,a,i,k} / \min_{1 \leq i \leq 3} V_{j,a,i,k} - 1 \right|. \quad (5.5)$$

Here $V_{j,m,i,k}$ ($V_{j,a,i,k}$) represents the minimum (average) objective k of algorithm i of instance j , $\text{ARD}_{j,m,i,k}$ ($\text{ARD}_{j,a,i,k}$) represents the deviation compared with the best results of three algorithms. Table 3 shows that the proposed algorithm obtained the best results with respect to the minimum and average objective values.

Table 4 shows the number of Pareto optimal solutions of the three algorithms, the percentage of Pareto optimal solutions when putting solutions together and the

Table 3 Minimum and average values of objectives for small-sized instances (problem 1–150)

	Objective 1		Objective 2		Objective 3	
	Minimum	Average	Minimum	Average	Minimum	Average
Improved NSGA-II						
Pro 1–50	0.0006	0.0003	0	0.0003	0.0923	0.0112
Pro 51–100	0.0052	0.0049	0.0114	0.0029	0.0472	0.0101
Pro 101–150	0	0.0132	0	0.0179	0.0234	0.0071
Original NSGA-II						
Pro 1–50	0.3711	0.1966	0.1321	0.4207	0.6244	0.2910
Pro 51–100	0.2368	0.1416	0.2325	0.9329	0.5007	0.2598
Pro 101–150	0.2538	0.1954	0.0452	0.2939	0.2745	0.1972
SPEA2						
Pro 1–50	0.1384	0.2356	0.0275	0.4194	0.0464	0.0862
Pro 51–100	0.1027	0.1876	0.0323	0.8696	0.0328	0.1083
Pro 101–150	0.0947	0.3175	0.0152	0.4273	0.0200	0.1549

Table 4 Results of 2–4 measures for small-sized instances (problem 1–150)

	Avg.ND solutions	Pareto frontier	Spacing
Improved NSGA-II			
Pro 1–50	60.2800	0.8928	32.0920
Pro 51–100	57.6000	0.8175	29.6650
Pro 101–150	23.8000	0.7398	4.6432
Original NSGA-II			
Pro 1–50	51.0200	0.0009	24.6645
Pro 51–100	50.9000	0.0299	25.0465
Pro 101–150	15.1200	0.0596	5.0479
SPEA2			
Pro 1–50	64.3800	0.1064	39.3355
Pro 51–100	59.4400	0.1526	37.2296
Pro 101–150	19.2400	0.2006	11.8551

last measure value (spacing), respectively. It shows that the proposed algorithm has the best results comparing the second and third measures with other two algorithms. For the last measure spacing, smaller values usually represent better solutions. Our algorithm outperforms SPEA2 for all problems in average. The original NSGA-II has a smaller spacing than the proposed algorithm with problem 1-100 on average, which may partly because the results are distributed more concentrated.

It can be concluded that the improved NSGA-II performs better than the original NSGA-II and SPEA2 in the small-sized instances although the fourth measurement is not perfect based on all obtained results (SPEA2 performs better than the original NSGA-II, which may be because that the NSGA-II cannot eliminate the repeated boundary solutions).

5.4 Comparison with NSGA-II and SPEA2 in Large-Sized Instances

The set of large-sized instances which are randomly generated as follows:

$n = 20, 50, 100, 200, 500, m = 5, 10, 20, F = 2, 3, 4, 5, 6, 7$ and M is a 0–1 matrix generated randomly which make sure that the sum of each line is at least one. We randomly selected 50 combinations of $n \times m \times F$ and for each combination, 5 different M are generated, so the total number of large-sized instances is 250. The remaining data of each instance have the same generating method as the small-sized instances. The results of experiments are presented in Tables 5 and 6.

Table 5 Minimum and average values of objectives for large-sized instances (problem 151–400)

	Objective 1		Objective 2		Objective 3	
	Minimum	Average	Minimum	Average	Minimum	Average
Improved NSGA-II						
Pro 151–200	0	0	0	0	0.0047	0.0019
Pro 201–250	0	0	0	0	0.0606	0.0049
Pro 251–300	0	0	0.0300	0.0006	0.0731	0.0036
Pro 301–350	0.0135	0.0111	0.0056	0.0155	0.0628	0.0139
Pro 351–400	0	0.0177	0	0.0124	0.0131	0.0070
Original NSGA-II						
Pro 151–200	0.2954	0.2292	0.9377	4.6701	0.2139	0.1204
Pro 201–250	0.3132	0.2242	0.5208	14.0704	0.4447	0.1905
Pro 251–300	0.2289	0.1969	0.2241	13.2520	0.3245	0.1435
Pro 301–350	0.2332	0.1871	0.0780	0.3326	0.3221	0.2185
Pro 351–400	0.4482	0.1999	0.0650	0.4327	0.2273	0.2168
SPEA2						
Pro 151–200	0.1974	0.2195	0.3369	3.5094	0.0195	0.0488
Pro 201–250	0.1799	0.2233	0.1955	23.7542	0.0281	0.0586
Pro 251–300	0.1434	0.2011	0.0595	9.2033	0.0193	0.0388
Pro 301–350	0.0864	0.2870	0.0151	0.4294	0.0185	0.1540
Pro 351–400	0.2132	0.2688	0.0213	0.4992	0.0243	0.1758

Table 6 Results of 2–4 measures for large-sized instances (problem 151–400)

	Avg. ND solutions	Pareto frontier	Spacing
Improved NSGA-II			
Pro 151–200	66.0800	0.7162	41.8728
Pro 201–250	68.4200	0.7119	44.3907
Pro 251–300	66.3800	0.7229	54.9186
Pro 301–350	25.2800	0.7074	13.5292
Pro 351–400	30.7000	0.7606	6.5048
Original NSGA-II			
Pro 151–200	51.0200	0.0921	26.2635
Pro 201–250	51.7600	0.0647	34.7067
Pro 251–300	60.3000	0.1109	38.0558
Pro 301–350	22.9400	0.0723	12.9998
Pro 351–400	19.7600	0.0581	5.8014
SPEA2			
Pro 151–200	58.3400	0.1917	48.2831
Pro 201–250	60.3000	0.2235	58.9183
Pro 251–300	73.9000	0.1662	68.6901
Pro 301–350	26.9600	0.2203	23.4833
Pro 351–400	23.1400	0.1813	11.8724

Table 5 includes the minimum and average values of objectives among three algorithms, and it is the same with Table 3. It shows that the proposed algorithm obtains the best results with respect to the minimum and average objective values, which has the same analysis with the small-sized instances, and the proposed algorithm has the best results compared with the original NSGA-II and SPEA2.

Table 6 shows the number of Pareto optimal solutions of the three algorithms, the percentage of Pareto optimal solutions when putting solutions together and the last measure value (spacing), respectively, which is the same with Table 4.

It can be primarily ascertained that the improved NSGA-II performs better than the original NSGA-II and SPEA2 for both small-sized instances and large-sized instances (SPEA 2 also has a better performance than the original NSGA-II).

5.5 Statistical Analysis

To show the statistical significance of the experiment results among the three algorithms, we conduct two nonparametric pairwise comparisons (Sign test and Wilcoxon test, Zhang et al. [57]) on the results achieved by 4 measures of 400 instances (Table 7).

5.5.1 Sign Test

As the first measure is the minimum and average values of three objectives, every algorithm has $400 \times 3 \times 2 = 2\,400$ data after putting these values together. Every other measure has 400 results for each algorithm. The average performance of the proposed algorithm is compared separately with the other two algorithms. Wins (Loses) count the times when the proposed algorithm performs superior (inferior) to its counterpart algorithm. The detected difference indicates that the improved NSGA-II outperforms the original NSGA-II and SPEA 2 with a significance level of 0.05 when using the first, the third and the last measures. The detected differences of Sign test cannot support

Table 7 Results of sign test

Improved NSGA-II	Original NSGA-II	SPEA2
First measure		
Wins (+)	2 005	1 663
Loses (-)	80	198
Detected differences	0	$1.925\,2 \times 10^{-252}$
Second measure		
Wins (+)	266	187
Loses (-)	119	158
Detected differences	$1.000\,4 \times 10^{-13}$	-
Third measure		
Wins (+)	400	400
Loses (-)	0	0
Detected differences	$1.498\,9 \times 10^{-88}$	$1.498\,9 \times 10^{-88}$
Fourth measure		
Wins (+)	136	346
Loses (-)	262	54
Detected differences	$3.711\,9 \times 10^{-10}$	$5.840\,3 \times 10^{-48}$

Table 8 Results of Wilcoxon test

Improved NSGA-II	Original NSGA-II	SPEA 2
First measure		
R^+	2.7313×10^6	2.3497×10^6
R^-	52 261	1.9889×10^5
p value	0	1.3150×10^{-231}
Second measure		
R^+	5.7141×10^4	4.0287×10^4
R^-	20 969	3.0449×10^4
p value	1.0166×10^{-14}	0.073 6
Third measure		
R^+	8.0068×10^4	7.2050×10^4
R^-	132	7 680
p value	6.3925×10^{-67}	6.3913×10^{-43}
Fourth measure		
R^+	20 373	72 869
R^-	59 827	7 331
p value	1.4039×10^{-17}	1.5555×10^{-45}

the proposed algorithm which has better performance compared with SPEA2 when using the second measure.

5.5.2 Wilcoxon Test

Compared with Sign test, a more powerful pairwise test tool, Wilcoxon test (Derrac et al. [58]) is utilized which could take into consideration the degree of difference among searching performances. Specifically, the difference between two algorithms will be ranked according to its absolute value among all test instances. The sum of ranks that the proposed algorithm is superior (inferior) to the other algorithms is indicated by R^+ (R^-) in Table 8. As observed from Table 8, the p values show that the improved NSGA-II performs better than the original NSGA-II and SPEA2 are at significance level of 0.05 in the first and third measure, and better than SPEA2 and the original NSGA-II at significance 0.1 in the second measure. For the fourth measure, the original NSGA-II outperforms the improved NSGA-II and the improved NSGA-II outperforms SPEA2 both at significance 0.05.

6 Conclusions and Future Research

In this paper, we proposed a more general DPFSSP (the processing time of jobs in different factory can be different and not all factories can process each job) with transportation conditions. Three objectives are considered (makespan, maximum lateness, total costs). The mixed integer linear programming model has been formulated based on the sequence of jobs in each factory. For each objective, we proposed a new heuristic algorithm. With objective makespan or maximum lateness, the heuristics is proposed based on Taillard method. The greedy algorithm is used to solve the problem with objective total cost. An effective MOEA known as NSGA-II has been modified in order to improve the results of the considered problem in this paper. The

proposed algorithm employs heuristics for each objective as an initialization. A total of 400 instances, consisting of 150 small-sized instances and 250 large-sized instances, are designed to compare the results with the original NSGA-II and SPEA 2. Running results and statistical analysis show that the proposed algorithm has better performance than both the original NSGA-II and SPEA 2. To the best of our knowledge, there is no published work on multi-objective optimization of the DPFFSSP with transportation and eligibility constraints.

As for the future work, there is some room for improvement in both the model and the algorithm. Firstly, the considered problem in this paper assumes the transportation time or cost of job i to any factory which is fixed which is a strong assumption that can be generalized to limited transportation tools or quantity discounts. Secondly, more elaborated meta-heuristics can be designed to obtain better solutions and more complex local search methods can be used to search better neighborhoods.

Acknowledgements The authors wish to thank the anonymous referees for their constructive comments. The authors would like to thank Professor Bo Liu for his discussions and constant encouragement.

References

- [1] Pinedo, M.: *Scheduling: Theory, Algorithms, and Systems*. Springer, Berlin (2015)
- [2] Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1**(2), 117–129 (1976)
- [3] Johnson, S.M.: Optimal two-and three-stage production schedules with setup times included. *Nav. Res. Logist. (NRL)* **1**(1), 61–68 (1954)
- [4] Palmer, D.S.: Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. *J. Oper. Res. Soc.* **16**(1), 101–107 (1965)
- [5] Campbell, H.G., Dudek, R.A., Smith, M.L.: A heuristic algorithm for the n job, m machine sequencing problem. *Manag. Sci.* **16**(10), B-630 (1970)
- [6] Nawaz, M., Enscore, E.E., Ham, I.: A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem. *Omega* **11**(1), 91–95 (1983)
- [7] Jia, H.Z., Fuh, J.Y., Nee, A.Y., Zhang, Y.F.: Web-based multi-functional scheduling system for a distributed manufacturing environment. *Concurr. Eng.* **10**(1), 27–39 (2002)
- [8] Jia, H.Z., Fuh, J.Y., Nee, A.Y., Zhang, Y.F.: Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. *Comput. Ind. Eng.* **53**(2), 313–320 (2007)
- [9] Jia, H.Z., Nee, A.Y., Fuh, J.Y., Zhang, Y.F.: A modified genetic algorithm for distributed scheduling problems. *J. Intell. Manuf.* **14**(3–4), 351–362 (2003)
- [10] Chan, F.T., Chung, S.H., Chan, L.Y., Finke, G., Tiwari, M.K.: Solving distributed FMS scheduling problems subject to maintenance: genetic algorithms approach. *Robot. Comput. Integr. Manuf.* **22**(5), 493–504 (2006)
- [11] Chan, F.T., Chung, S.H., Chan, P.L.Y.: An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Syst. Appl.* **29**(2), 364–371 (2005)
- [12] De Giovanni, L., Pezzella, F.: An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. *Eur. J. Oper. Res.* **200**(2), 395–408 (2010)
- [13] Deng, J., Wang, L., Wang, S.Y., Zheng, X.L.: A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem. *Int. J. Prod. Res.* **54**(12), 3561–3577 (2016)
- [14] Wang, S.Y., Wang, L.: An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem. *IEEE Trans. Syst. Man Cybern. Syst.* **46**(1), 139–149 (2016)
- [15] Naderi, B., Ruiz, R.: The distributed permutation flowshop scheduling problem. *Comput. Oper. Res.* **37**(4), 754–768 (2010)

- [16] Liu, H., Gao, L.: A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem. In: 2010 International Conference on Manufacturing Automation (ICMA), pp. 156–163. IEEE (2010)
- [17] Gao, J., Chen, R.: A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *Int. J. Comput. Intell. Syst.* **4**(4), 497–508 (2011)
- [18] Gao, J., Chen, R.: An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems. *Sci. Res. Essays* **6**(14), 3094–3100 (2011)
- [19] Gao, J., Chen, R., Liu, Y.: A knowledge-based genetic algorithm for permutation flowshop scheduling problems with multiple factories. *Int. J. Adv. Comput. Technol.* **4**(7), 121–129 (2012)
- [20] Gao, J., Chen, R., Deng, W.: An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *Int. J. Prod. Res.* **51**(3), 641–651 (2013)
- [21] Lin, S.W., Ying, K.C., Huang, C.Y.: Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *Int. J. Prod. Res.* **51**(16), 5029–5038 (2013)
- [22] Wang, S.Y., Wang, L., Liu, M., Xu, Y.: An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *Int. J. Prod. Econ.* **145**(1), 387–396 (2013)
- [23] Xu, Y., Wang, L., Wang, S., Liu, M.: An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem. *Eng. Optim.* **46**(9), 1269–1283 (2014)
- [24] Naderi, B., Ruiz, R.: A scatter search algorithm for the distributed permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **239**(2), 323–334 (2014)
- [25] Wang, J., Wang, L., Shen, J.: A hybrid discrete cuckoo search for distributed permutation flowshop scheduling problem. In: 2016 IEEE Congress on Evolutionary Computation, pp. 2240–2246. IEEE (2016)
- [26] Wang, K., Huang, Y., Qin, H.: A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown. *J. Oper. Res. Soc.* **67**(1), 68–82 (2016)
- [27] Pareto, V.: *Oeuvres Complètes: Tome 7, Manuel d'économie Politique*. Librairie Droz, Geneva (1981)
- [28] Ciavotta, M., Minella, G., Ruiz, R.: Multi-objective sequence dependent setup times permutation flowshop: a new algorithm and a comprehensive study. *Eur. J. Oper. Res.* **227**(2), 301–313 (2013)
- [29] Ishibuchi, H., Murata, T.: A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)* **28**(3), 392–403 (1998)
- [30] Varadharajan, T.K., Rajendran, C.: A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *Eur. J. Oper. Res.* **167**(3), 772–795 (2005)
- [31] Murata, T., Ishibuchi, H., Tanaka, H.: Multi-objective genetic algorithm and its applications to flowshop scheduling. *Comput. Ind. Eng.* **30**(4), 957–968 (1996)
- [32] Yagmahan, B., Yenisey, M.M.: Ant colony optimization for multi-objective flow shop scheduling problem. *Comput. Ind. Eng.* **54**(3), 411–420 (2008)
- [33] Gelders, L.F., Sambandam, N.: Four simple heuristics for scheduling a flow-shop. *Int. J. Prod. Res.* **16**(3), 221–231 (1978)
- [34] Ponnambalam, S.G., Jagannathan, H., Kataria, M., Gadicherla, A.: A TSP-GA multi-objective algorithm for flow-shop scheduling. *Int. J. Adv. Manuf. Technol.* **23**(11–12), 909–915 (2004)
- [35] Yenisey, M.M., Yagmahan, B.: Multi-objective permutation flow shop scheduling problem: literature review, classification and current trends. *Omega* **45**, 119–135 (2014)
- [36] Deng, J., Wang, L.: A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm Evol. Comput.* **32**, 121–131 (2017)
- [37] Zitzler, E., Laumanns, M., Thiele, L.: *SPEA 2: improving the strength Pareto evolutionary algorithm*. Tik-report (2001)
- [38] Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: *International Conference on Parallel Problem Solving From Nature*, pp. 849–858. Springer, Berlin, Heidelberg (2000)
- [39] Bandyopadhyay, S., Bhattacharya, R.: Solving multi-objective parallel machine scheduling problem by a modified NSGA-II. *Appl. Math. Model.* **37**(10), 6718–6729 (2013)
- [40] Bolaños, R., Echeverry, M., Escobar, J.: A multiobjective non-dominated sorting genetic algorithm (NSGA-II) for the Multiple Traveling Salesman Problem. *Decis. Sci. Lett.* **4**(4), 559–568 (2015)
- [41] Sahu, D.P., Singh, K., Prakash, S.: Maximizing availability and minimizing makespan for task scheduling in grid computing using NSGA II. In: *Proceedings of the Second International Conference on Computer and Communication Technologies*, pp. 219–224. Springer, India (2016)

- [42] Long, J., Zheng, Z., Gao, X., Pardalos, P.M.: A hybrid multi-objective evolutionary algorithm based on NSGA-II for practical scheduling with release times in steel plants. *J. Oper. Res. Soc.* **67**(9), 1184–1199 (2016)
- [43] Autuori, J., Hnaien, F., Yalaoui, F., Hamzaoui, A., Essounbouli, N.: Comparison of solution space exploration by NSGA 2 and SPEA 2 for Flexible Job Shop Problem. In: 2013 International Conference on Control, Decision and Information Technologies, pp. 750–755. IEEE (2013)
- [44] Blumenfeld, D.E., Burns, L.D., Daganzo, C.F., Frick, M.C., Hall, R.W.: Reducing logistics costs at General Motors. *Interfaces* **17**(1), 26–47 (1987)
- [45] Wang, W.F., Yun, W.Y.: Scheduling for inland container truck and train transportation. *Int J. Prod. Econ.* **143**(2), 349–356 (2013)
- [46] Siddiqui, A.W., Verma, M.: A bi-objective approach to routing and scheduling maritime transportation of crude oil. *Transp. Res. Part D Transp. Environ.* **37**, 65–78 (2015)
- [47] Chen, Z.L., Lee, C.Y.: Machine scheduling with transportation considerations. *J. Sched.* **4**, 3–24 (2001)
- [48] Naderi, B., Zandieh, M., Balagh, A.K.G., Roshanaei, V.: An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert Syst. Appl.* **36**(6), 9625–9633 (2009)
- [49] Tang, L., Liu, P.: Flowshop scheduling problems with transportation or deterioration between the batching and single machines. *Comput. Ind. Eng.* **56**(4), 1289–1295 (2009)
- [50] Zhu, H., Leus, R., Zhou, H.: New results on the coordination of transportation and batching scheduling. *Appl. Math. Model.* **40**(5), 4016–4022 (2016)
- [51] Zabihzadeh, S.S., Rezaeian, J.: Two meta-heuristic algorithms for flexible flow shop scheduling problem with robotic transportation and release time. *Appl. Soft Comput.* **40**, 319–330 (2016)
- [52] Low, C., Li, R.K., Wu, G.H.: Ant colony optimization algorithms for unrelated parallel machine scheduling with controllable processing times and eligibility constraints. In: Proceedings of the Institute of Industrial Engineers Asian Conference 2013, pp. 79–87. Springer, Singapore (2013)
- [53] Soltani, S.A., Karimi, B.: Cyclic hybrid flow shop scheduling problem with limited buffers and machine eligibility constraints. *Int. J. Adv. Manuf. Technol.* **76**(9–12), 1739–1755 (2015)
- [54] Taillard, E.: Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Oper. Res.* **47**(1), 65–74 (1990)
- [55] Baesler, F., Palma, C.: Multiobjective parallel machine scheduling in the sawmill industry using memetic algorithms. *Int. J. Adv. Manuf. Technol.* **74**(5–8), 757–768 (2014)
- [56] Hyun, C.J., Kim, Y., Kim, Y.K.: A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Comput. Oper. Res.* **25**(7), 675–690 (1998)
- [57] Zhang, H., Li, B., Zhang, J., Qin, Y., Feng, X., Liu, B.: Parameter estimation of nonlinear chaotic system by improved TLBO strategy. *Soft Comput.* **20**(12), 4965–4980 (2016)
- [58] Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **1**(1), 3–18 (2011)