Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 19(9): 8833–8865.
DOI: 10.3934/mbe.2022410
Received: 13 May 2022
Revised: 09 June 2022
Accepted: 13 June 2022
Published: 17 June 2022

Research article

Multiobjective particle swarm optimization with direction search and differential evolution for distributed flow-shop scheduling problem

Wenqiang Zhang[1,*], Chen Li[1], Mitsuo Gen[2], Weidong Yang[3], Zhongwei Zhang[4] and Guohui Zhang[5]

[1] College of Information Science and Engineering, Henan University of Technology, China
[2] Fuzzy Logic Systems Institute/Tokyo University of Science, Japan
[3] Henan Key Laboratory of Grain Photoelectric Detection and Control, Henan University of Technology, China
[4] School of Mechanical and Electrical Engineering, Henan University of Technology, China
[5] School of Management Engineering, Zhengzhou University of Aeronautics, China

* **Correspondence:** Email: zhangwq@haut.edu.cn.

Abstract: As a classic problem of distributed scheduling, the distributed flow-shop scheduling problem (DFSP) involves both the job allocation and the operation sequence inside the factory, and it has been proved to be an NP-hard problem. Many intelligent algorithms have been proposed to solve the DFSP. However, the efficiency and quality of the solution cannot meet the production requirements. Therefore, this paper proposes a bi-objective particle swarm optimization with direction search and differential evolution to solve DFSP with the criteria of minimizing makespan and total processing time. The direction search strategy explores the particle swarm in multiple directions of the Pareto front, which enhances the strong convergence ability of the algorithm in different areas of Pareto front and improves the solution speed of the algorithm. The search strategy based on differential evolution is the local search strategy of the algorithm, which can prevent the multiobjective particle swarm optimization from converging prematurely and avoid falling into local optimum, so that a better solution can be found. The combination of these two strategies not only increases the probability of particles moving in a good direction, but also increases the diversity of the particle swarm. Finally, experimental results on benchmark problems show that, compared with traditional multiobjective evolutionary algorithms, the proposed algorithm can accelerate the convergence speed of the algorithm while guaranteeing that the obtained solutions have good distribution performance and diversity.

Keywords: distributed flow-shop scheduling problem; multiobjective optimization; particle swarm optimization; differential evolution; Pareto front

1. Introduction

Production scheduling plays a vital role in many manufacturing systems, and effective production planning is an important factor that enables the industry to improve production efficiency and resource utilization [1]. Therefore, it is of great significance to study production scheduling problems, especially to develop effective solving algorithms. Production scheduling is the allocation of limited production resources to tasks within a specific time period to optimize one or more goals determined by decision makers [2]. With the rapid development of manufacturing technology and equipment, scheduling in production systems has become more and more complex. Under the trend of globalization, since the operation of a single factory has been unable to fully and flexibly respond to the high changes in market demand [3], the structure of the factory has changed from centralized to decentralized, and the multi-shop co-production pattern has become increasingly popular. Compared with a single factory production center, distributed manufacturing is conducive to resource sharing, which can reduce costs and risks, improve product quality [4], improve economic benefits, and respond quickly to market changes. Among all kinds of distributed manufacturing problems, the distributed flow-shop scheduling problem (DFSP) is one of the most widely studied scheduling problems and has strong application value [5].

To the best of our knowledge, the DFSP with the makespan criterion is currently the most studied because it is challenging [6]. The DFSP can be regarded as a generalized version of flow-shop scheduling problem (FSP), which involves two sub-problems [7]. The first is to assign jobs to factories, and the second is to sort jobs in each factory, and these two sub-problems affect each other, so they must be solved together [8]. However, the two sub-problems of job sequence and factory allocation are coupled with each other, and the optimization of each sub-problem does not necessarily mean that the coupled problem is optimized. Therefore, although there have been many research results on the single-shop scheduling problem, it cannot be directly applied to the DFSP. In addition, from a mathematical point of view, the DFSP is an NP-hard problem, and the increase of sub-problems increases the solution space, which in turn makes the problem more difficult to solve. To sum up, DFSP is much more difficult to solve than single-shop scheduling, and has strong research value, requiring researchers to develop new methods.

Today, there are more and more ways to solve DFSP. Traditional mathematical methods, such as enumeration, branch and bound, are suitable for small, simple problems due to their high computational complexity. Therefore, heuristics and meta-heuristics are used to solve complex combinatorial optimization problems. In recent years, existing optimization algorithms include genetic algorithm (GA) [9], particle swarm optimization algorithm (PSO) [10, 11], tabu search [12], distribution estimation algorithm [13], hybrid immune algorithm [14], scatter search [15], chemical reaction optimization [16] and iterative greedy algorithm [17], artificial bee colony algorithm (ABC) [18], water wave optimization algorithm [19], and cooperative optimization algorithm [20]. These algorithms work differently than traditional mathematical methods. The unique algorithm theory and operation mechanism guarantee that the exploration and utilization of the algorithm reach a certain balance in the iterative process. In addition, addressing other objectives of DFSP such as the total processing time criterion [21] is also considered.

To better address two closely related sub-problems of DFSP, two industrial indicators are proposed [21]. One indicator is the most studied makespan [22] and the other is total processing time.

Minimizing the total processing time can reduce the job in progress and lead to stable utilization of resources. The advantage is even higher in distributed environments where the imbalances among resources and jobs in progress may increase due to the existence of several factories. Total processing time has been pointed out as a more relevant and significant objective for today's dynamic manufacturing environment. Therefore, this study addresses the DFSP in order to minimize makespan and total processing time criteria. When solving the problem, one of these two indicators can be biased, so the problem model can be clarified, and it is very convenient to solve the DFSP. But this method can only take care of one indicator, it is difficult to optimize another indicator. Another method that can be used to solve these two closely related problems is to arrange the process sequence of each job completely randomly and assign it to the factory randomly [16]. This method can take into account both the makespan indicator and the total processing time indicator. Therefore, we use this method to solve DFSP.

The evolutionary algorithm may be not as accurate as traditional mathematics methods, but its calculation speed has been greatly improved. Therefore, evolutionary algorithm is more popular than traditional mathematical method. PSO and GA are more classic algorithms in evolutionary algorithm. Despite that PSO has some similar properties with GA, PSO does not apply evolution operators like mutation and crossover. PSO has a simple structure, is easy to calculate, and has obvious advantages in calculation speed. The scale of DFSP may be very large, and the calculation speed of PSO is still very fast in dealing with complex problems, so PSO is a good choice to solve DFSP.

PSO is a swarm intelligence optimization algorithm developed from bird foraging. PSO will first initialize a particle swarm, each particle will search the solution space according to its own historical optimal and global optimal, and finally find an excellent solution [11, 23]. In this process, the algorithm does not need much calculation, so its calculation speed has an advantage in many evolutionary algorithms. PSO was first proposed to solve the problem of continuous solution space. The solution space of DFSP is discontinuous, which needs to be solved with a discrete PSO [24–26]. The structure of discrete PSO is too simple, and the disadvantage of discrete PSO is that it is very easy to fall into local optimum. Therefore, it is necessary to add a mechanism to prevent the local optimum [27, 28].

Differential evolution (DE) algorithm is also a kind of evolutionary algorithm. The calculation quantity of the algorithm is small and the calculation speed is fast. DE is based on the difference between particles to evolve bad individual to excellent individual [29]. Therefore, DE can strengthen the connection between particles in a particle swarm and is suitable for improving the PSO.

DFSP usually have multiple constraints [30], and single-objective PSO is difficult to solve such problems. Therefore, a multiobjective particle swarm optimization algorithm (MoPSO) is studied. Since too many constraints on the DFSP will affect the performance of the algorithm, it is necessary to improve MoPSO [31].

In this study, a multiobjective particle swarm optimization (MoPSODS-DE) based on enhanced direction search and differential evolution is proposed. The enhanced direction search strategy explores the particle swarm in multiple directions of the Pareto front (PF), and speeds up the convergence of the upper, central and lower areas of the PF. The differential evolution strategy performs differential operations on the updated particle swarm to improve the quality of the particles and prevent falling into local optimum. Therefore, combining these two strategies can further improve the convergence performance and distribution performance of the algorithm, as well as improve the diversity of the particle swarm. In addition, a method of two-vector representation of particles is designed for DFSP for encoding and decoding, and the particles are updated using exchange sequences, crossover, and mutation. The proposed MoPSODS-DE are compared with NSGA-II, SPEA2, MoPSO, MOEA/D, a fast multiobjective hybrid evolutionary algorithm (MOHEA) and hybrid multiobjective evolutionary algorithm based on DE (MOHEA/DE). The results show that MoPSODS-DE not only has excellent convergence performance, but also can guarantee that the obtained solutions have good distribution performance and diversity.

The remainder of this study is organized as follows. In Section 2, related works about solving the DFSP is shown. In Section 3, the DFSP formulation is introduced. Section 4 describes the MoPSODS-DE algorithm. In Section 5, experimental data of MoPSODS-DE are presented and discussed. Finally, Section 6 concludes this study with some conclusions and future work.

2. Related work

The DFSP is a variant of FSP, and the characteristic of this type of problem is that if the scale of the problem becomes larger and larger, the difficulty of solving it will increase exponentially. Naderi and Ruiz as pioneers in introducing DFSP, established 6 mathematical models, developed 2 factory assignment rules, and proposed 2 variable neighborhood algorithms [32]. Furthermore, effective constructive heuristics and meta-heuristics are proposed to solve it. Since then, other authors also proposed many high-performance methods to deal with this problem.

In many realistic FSP, multiple interrelated objectives need to be considered, such as makespan, total processing time, machine idle time, and total delay [33]. Therefore, it becomes a trend for DFSP to consider multiple objective functions. For example, Rifai et al. [34] dealt with a distributed reentrant PFSP considering three objective functions: minimization of makespan, minimization of the total cost, and minimization of expected delays. These authors developed a multiobjective adaptive large neighborhood search algorithm to estimate the PF. In addition, Deng and Wang [35] proposed a competitive memetic algorithm for solving multiobjective DFSP considering the minimization of makespan and total delay. Lu et al. [36] established a new mathematical model for distributed energy-saving FSP with the objective of maximum lifetime and minimum total energy consumption, and designed a hybrid multiobjective optimization algorithm that combined iterative greedy and efficient local search , which provides a set of compromise solutions for this problem.

Furthermore, besides the classic DFSP, other types of distributed production scheduling problems have been studied in recent years. Most of them are extensions of the classic DFSP, such as distributed no-wait FSP [37], distributed no-idle permutation FSP [38], distributed hybrid no-idle FSP [39] and distributed finite buffer FSP [40] consider no-wait constraints, no-idle constraints, hybrid no-wait constraints, and finite buffer constraints in DFSP, respectively. Others come from extensions of other classic production scheduling problems in multi-factory production environments, such as distributed two-phase scheduling problems, distributed reentrant FSP [34], and distributed assembly permutation FSP [7]. For example, Zhao et al. proposed a two-stage cooperative evolutionary algorithm with problem-specific knowledge to solve the energy-efficient scheduling problem of the no-waiting FSP, whose criterion is to minimize the makespan and total energy consumption [41]. To solve the scheduling problem of distributed assembly without idle flow shop, Zhao et al. proposed a collaborative water wave optimization algorithm to solve the distributed assembly no-idle FSP, with the objective of minimizing the maximum assembly completion time [42]. In addition, Zhao et al. proposed a self-learning discrete Jaya algorithm to solve the energy-saving distributed no-idle FSP in heterogeneous factory systems with the criterion of maximizing minimize total tardiness, total energy consumption and factory load balancing [43].

GA is the most classic evolutionary algorithm, which is used in various complex problems. Jia et al. studied the DFSP under different criteria and employed a standard GA to solve the problem [44]. Zhang et al. proposed an enhanced GA for the distributed assembly permutation FSP, and a crossover strategy is designed based on local search to speed up convergence [45]. Wang and Wang proposed a mixed integer linear programming model and a novel iterative greedy algorithm to solve DFSP, and designed a local intensive method based on variable neighborhood search [46]. In this paper, the two operations of crossover and mutation in differential evolution algorithm are used to update the factory sequence of particles in the particle swarm.

PSO is a kind of swarm intelligence algorithm, which has a simple structure and fast calculation speed, and is often used in various NP-hard problems. For PSO, Tasgetiren et al. studied improved variable neighborhood search PSO, finding many optimal solutions for the top 90 Taillard benchmark instances [47]. On the other hand, Pan et al. [48] applied discrete PSO to solve the no-wait flow-shop scheduling problem. DFSP usually have multiple constraints [30], and single-objective PSO is difficult to solve such problems. To this end, a MoPSO is studied, which mainly solves multiobjective problems and is more practical than PSO. Since too many constraints on DFSP will affect the performance of the algorithm, it is necessary to improve MoPSO. Rahimi-Vahed and Mirghorbani [49] studied a PSO method targeting weighted average completion time and weighted average delay. The proposed MoPSO is compared with a multiobjective GA. The algorithm outperforms multiobjective GA on some specific performance metrics. Su and Chi [50] proposed a MoPSO that combines PSO with DE and utilizes a simulated annealing strategy to generate a scale factor to dynamically tune PSO and DE. Cui et al. [51] proposed a hybrid MoPSO, which uses different operators to generate new offspring, and then uses the environmental selection mechanism to screen out the next generation of individuals. To improve the efficiency of solving a bi-objective mixed no-idle FSP that minimizes the makespan and total processing time, Zhang et al. proposed a multiobjective particle swarm optimization based on multi-direction update [52].

ABC is a swarm intelligence algorithm like MoPSO, which is also widely used in solving DFSP. Li et al. proposed a hybrid ABC to solve DFSP [53]. In order to enable decision makers to conveniently choose the final solution according to actual needs, the purpose is to obtain a set of optimal solutions on PF, rather than finding a single solution [35]. Pan et al. used total processing time as the optimization goal of DFSP, designed different heuristic algorithms and improved ABC [54]. Huang et al. proposed three constructive heuristic methods and effective discrete ABC to solve the DFSP [55]. To solve the multiobjective distributed fuzzy permutation FSP, Emin et al. developed an artificial bee colony (ABC) algorithm [56].

However, whether it is MoPSO, ABC or other swarm intelligence algorithms, there is a common disadvantage, that is, it is difficult to find the global optimal solution. Because the DE has a single structure, simple operation, and can reflect the relationship between two individuals, so many researchers also use DE to solve scheduling problems. To address the electric vehicle charging scheduling (EVCS) problem and consider the dependencies between site selection, charging options at each site, and charging capacity settings. Liu et al. designed a mixed variable differential evolution as a scheduling algorithm for EVCS systems [57]. Zhou et al. developed an adaptive differential evolution algorithm for a single batch machine scheduling problem with varying release times and job sizes [58]. Zhang et al. proposed a HMOEA/DE to solve the FSP with the criteria of minimizing the maskspan and tardiness simultaneously. The DE strategy is used as a local search mechanism, which further improves the convergence and distribution performance of the algorithm on elite population [59]. For the FSP with blocking, Han et al. proposed a new discrete ABC, which was combined with DE and solved using the makespan criterion [60]. This paper adopts the combination of improved MoPSO (MoPSODS) and DE search to further improve the performance of the algorithm in all aspects.

The DFSP can be solved with the above algorithms. By comparison, the search speed of PSO algorithm is faster and the PSO algorithm based on enhanced direction search is better than other algorithms. Since the PSO uses *pbest* and *gbest* to tune search direction, while there is no operation between individuals. Therefore, combining the DE search with the particle swarm algorithm based on enhanced direction search can further improve the convergence performance of the algorithm and the distribution and diversity of solutions.

3. Problem formulation

The DFSP is described as follows: suppose the problem has n jobs, m machines, and F factories. The machines in the F factories are the same, and each job can be processed independently in each factory. When a job is assigned to a factory, all the processes of the job are processed within that factory without being transferred to other factories. Each job needs to be processed on m machines. Each processing of each job can only be completed on one machine, and each machine can only process one job at a time. It does not stop after each operation is started. No need to consider the preparation time before processing, and finally find a job order to arrange the production of the factory [54]. The mathematical model of DFSP is expressed in the following notations.

Indices

i: index of job, $(i = 1, 2, \dots, n)$;

j: index of machine, $(j = 1, 2, \dots, m)$;

f: index of factory, $(f = 1, 2, \dots, F)$.

Parameters

n: the number of jobs;

m: the number of machines;

F: the number of factories;

n_f: the number of jobs in the f-th factory;

$t_{(i,j,f)}$: the processing time of the job located on the i-th position in the processing sequence in the f-th factory, on the machine j;

$C_{(i,j,f)}$: the completion time of the job located on the i-th position in the processing sequence in the f-th factory, on the machine j.

$X_{(i,k,f)}$: the binary variable, if job i occupies the position k in the f-th factory $X_{(i,k,f)} = 1$. Otherwise, $X_{(i,k,f)} = 0$.

Mathematical model

The mathematical model for minimization of makespan and total processing time can be expressed as follows:

$$min_{(C_{\max}, TPT)}$$

$$\sum_{k=1,k\neq i}^{n}\sum_{f=1}^{F}X_{i,k,f}=1 \quad i\in\{1,2,3,\cdots,n\} \tag{3.1}$$

$$\sum_{i=1,k\neq i}^{n}\sum_{f=1}^{F}X_{i,k,f}=1 \quad i\in\{1,2,3,\cdots,n\} \tag{3.2}$$

$$X_{k,i,f}+X_{i,k,f}\leq 1 \quad \forall i\in\{1,2,3,\cdots,n-1\}, i<k \tag{3.3}$$

$$C_{(i,0,f)}=0 \quad \forall i,f \tag{3.4}$$

$$C_{(0,j,f)}=0 \quad \forall j,f \tag{3.5}$$

$$C_{1,1,f}=\sum_{i=1}^{n}\sum_{f=1}^{F}X_{i,1,f}\cdot t_{i,1,f} \tag{3.6}$$

$$C_{i,j,f}\geq C_{i,j-1,f}+\sum_{i=1}^{n}X_{i,k,f}\cdot t_{i,j,f} \quad \forall i,j,f \tag{3.7}$$

$$C_{i,j,f}\geq C_{i-1,j,f}+\sum_{i=1}^{n}X_{i,k,f}\cdot t_{i,j,f} \quad \forall i,j,f \tag{3.8}$$

$$C_{\max}\geq C_{(i,j,f)} \tag{3.9}$$

$$C_{(i,j,f)}=\max\{C_{(i,j-1,f)},C_{(i-1,j,f)}\}+\sum_{i=1}^{n}X_{i,k,f}\cdot t_{(i,j,f)} \quad \forall i,j,f \tag{3.10}$$

$$C_{\max}=\max\{C_{(n_f,m,f)}\} \quad f\in\{1,2,3,\cdots,F\} \tag{3.11}$$

$$TPT=\max\{\sum_{i=1}^{n_f}C_{(i,m,f)}\} \quad f\in\{1,2,3,\cdots,F\} \tag{3.12}$$

In this paper, two common objectives are used, one is the makespan and the other is the total processing time, and then minimize the two objectives. Constraint (3.1) guarantees that each job is only assigned to one factory and only appeared once in the factory. Constraint (3.2) explains that the possibility for n positions where all jobs should be allocated to each factory. Constraint (3.3) controls and guarantees that a job is both a predecessor and successor of another job at the same time. Constraints (3.4)–(3.6) illustrate that the completed time of the first job processed on the first machine in each factory, which is guaranteed that the start time of the first machine in each factory is 0. Constraints (3.7) and (3.8) are that each job is only start to process after the previous job processed on the machine in an identical factory, which guarantees that multiple jobs are not processed on one machine. Constraint (3.9) means that the makespan must be greater than or equal to the makespan of each job. Therefore, Eq (3.10) can be used to calculate the makespan of each job. The makespan and total processing time of DFSP can be obtained by Eqs (3.11) and (3.12), respectively. Figure 1 shows the Gantt chart of DFSP. Taking two factories and five jobs as an example, the values of the two objectives of DFSP can be obtained through the calculation of the above formula. The makespan of factory 1 is 20 and the makespan of factory 2 is 22, so the makespan of DFSP is 22. In addition, the total processing time of DFSP is determined by the makespan of the jobs in the factory. The total processing time of factory 1 is 12 + 20 = 32, and the total processing time of factory 2 is 12 + 20 + 22 = 54, so the total processing time of DFSP can be obtained as 54.

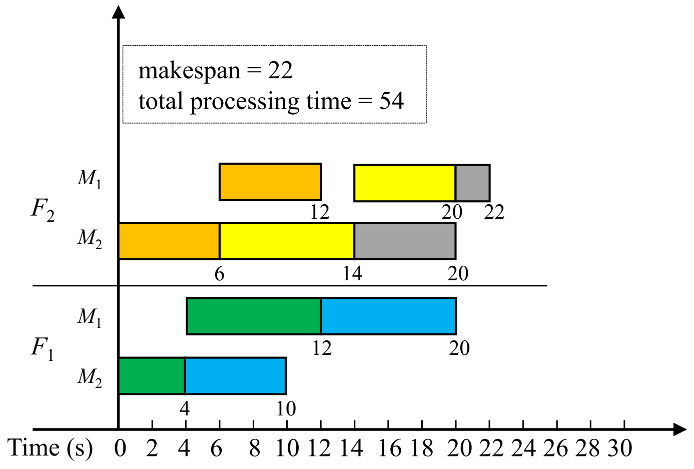


Figure 1. Gantt chart is used to solve example problem.

4. Solution algorithm

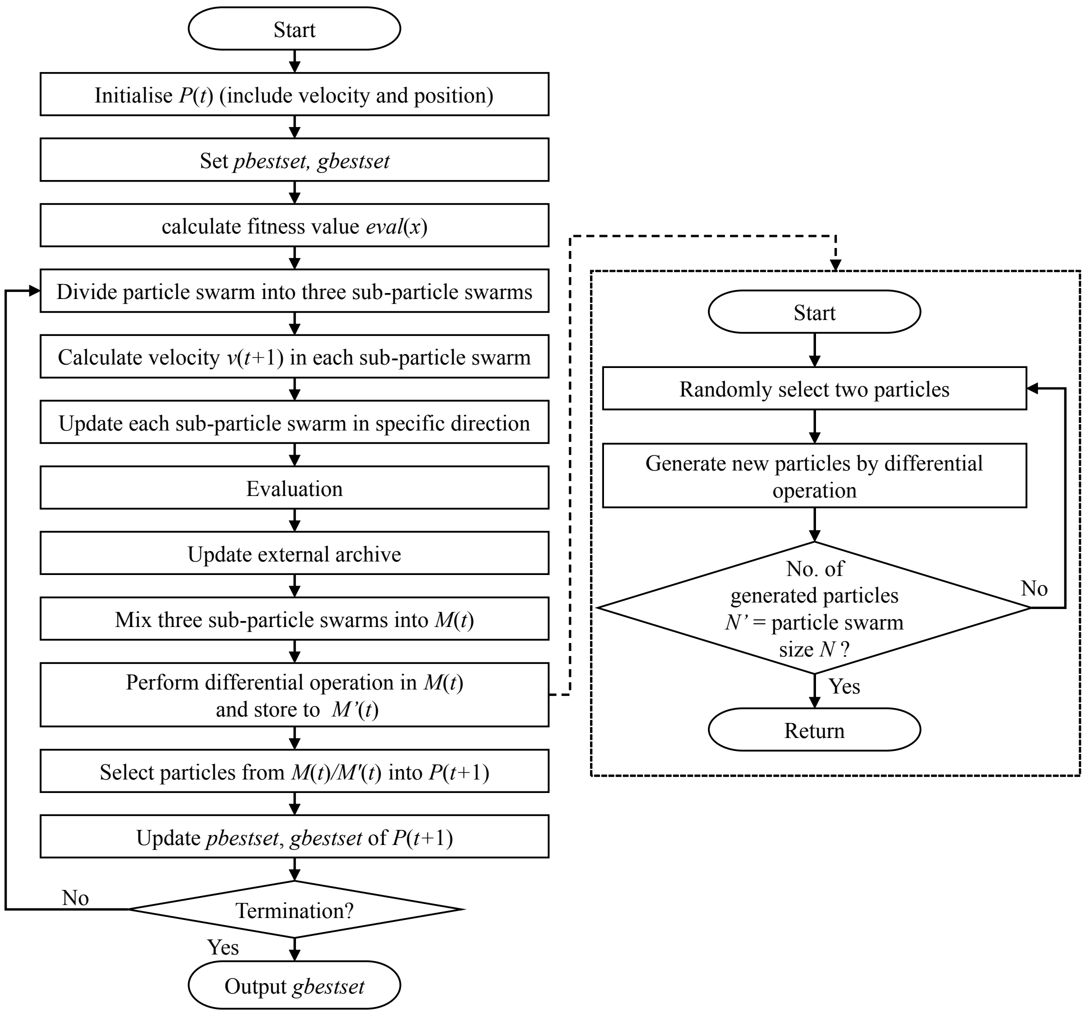
4.1. Overview of the MoPSODS-DE

The MoPSODS-DE algorithm is improved on the basis of the MoPSO. The first step is to consider the encoding and decoding strategies and adapt the MoPSO algorithm for the DFSP. Secondly, the enhanced direction search strategy is used to adapt the MoPSO to greatly improve the convergence performance without reducing the distribution performance. Third, the above improved algorithm, namely MoPSODS, is mixed with differential evolution search to improve the local convergence ability of the algorithm and prevent the algorithm from premature convergence. Then use the method of two-vector representation of particles to perform reasonable and effective update operations on the particles, which further enhances the adaptability of the algorithm. The algorithmic framework of MoPSODS-DE is shown in Figure 2.

MoPSO is the same as the ordinary PSO algorithm, without grouping the particle swarm. In order to strengthen the multi-direction search ability of MoPSO in PF, the PDDR-FF fitness function and VEGA strategy are introduced to group the particle swarm, thereby further improving the strong convergence ability of PF.

Using PDDR-FF fitness function and VEGA strategy, the particles are divided into three sub-particle swarms [61]. The VEGA strategy is used to select two single objectives, and the PDDR-FF is used to select particles in the central area of the PF. The three sub-particle swarms performed well on objective 1, objective 2 and central area of the PF respectively. The sum of the number of particles in these three sub-particle swarms is equal to the size of the particle swarm.

Then, use the PSO formula to update the velocity and position of the particles. Update particles use a combination of exchange sequence, crossover, and mutation to operate on two vectors respectively. The sub-particle swarm that perform well on objective 1 is updated towards objective 1, the sub-particle swarm that perform well on objective 2 is updated towards objective 2, and the sub-particle swarm near the central area of PF is updated towards the PF. In this way, each sub-particle swarm will be updated in a different direction. The updated particles are mixed into a new particle swarm, which is a temporary particle swarm, and its size is the same as that of the particle swarm.



Start
Initialise P(t) (include velocity and position)
Set pbestset, gbestset
calculate fitness value eval(x)
Divide particle swarm into three sub-particle swarms
Calculate velocity v(t+1) in each sub-particle swarm
Update each sub-particle swarm in specific direction
Evaluation
Update external archive
Mix three sub-particle swarms into M(t)
Perform differential operation in M(t) and store to M'(t)
Select particles from M(t)/M'(t) into P(t+1)
Update pbestset, gbestset of P(t+1)
Termination?
No
Yes
Output gbestset
Start
Randomly select two particles
Generate new particles by differential operation
No. of generated particles N' = particle swarm size N ?
No
Yes
Return

Figure 2. Basic framework of the MoPSODS-DE.

Apply the differential search strategy to the temporary particle swarm. Randomly select two particles to compare the advantages and disadvantages, and then perform the difference operation to generate a new particle. If the size of the particle swarm is N, then the difference operation is performed N times, and there are a total of N difference particles. Then select N excellent particles from the difference particles and the temporary particle swarm as the new generation particle swarm.

The MoPSODS-DE algorithm can be divided into the above steps. The pseudo-code of MoPSODS-DE is shown in the algorithm 1.

4.2. Encoding and decoding

Encoding and decoding is the first step in the algorithm, which is used to solve compatibility problems [62]. The DFSP requires two-vector to represent a solution. One is that the job sequence is represented by job sequence (JS) $\mathbf{v}_1$, and the other is that the factory vector is represented by factory

Algorithm 1 The pseudo-code of the MoPSODS-DE

```
Input: problem data, PSO parameters, $MaxGen$, DE probability;
Output: the best Pareto optimal solutions;
 1: $t$=0;
 2: initialize $P(t)$ (velocity $\mathbf{v}(t)$ and position $\mathbf{x}(t)$), by encoding routine;
 3: set $pbestset$ for every particles, add each particle in their $pbestset$;
 4: set $gbestset$, and $gbestset$ = argmin $\{eval(pbestset)\}$;
 5: while ($t < MaxGen$) do
 6:     calculate each objective $f_i(\mathbf{x})$ of $\mathbf{x}$, $i$=1,2, by decoding routine;
 7:     calculate fitness function $eval(\mathbf{x})$, by decoding routine;
 8:     divide the $P(t)$ to three sub-particle swarms by VEGA and PDDR-FF;
 9:     for each particle in each sub-particle swarm do
10:         get velocity $\mathbf{v}(t+1)$;
11:         update position $\mathbf{x}(t+1)$ by diretion search route;
12:     end for
13:     mix three sub-particle swarms into a temporary particle swarm $M(t)$;
14:     Perform differential operations on temporary particle swarms and store it in $M'(t)$;
15:     Select excellent particles from $M(t)$ and $M'(t)$ and store them in the next-generation particle
        swarm $P(t+1)$;
16:     update $pbestset$ = argmin $\{eval(pbestset), eval(P(t+1))\}$;
17:     update $gbestset$ = argmin $\{eval(pbestset), eval(gbestset)\}$;
18:     $t$=$t$+1;
19: end while
20: output the best Pareto optimal solutions $gbestset$;
```

allocation (FA) $\mathbf{v}_2$. As shown in Figure 3. The first vector contains a simple arrangement of 10 jobs. Each number in the vector is the job number. In the case of only 3 factories, there can only be 3 numbers in the second vector, each number is the number of the factory. These two vectors constitute a particle. This encoding method can effectively solve the two key problems of DFSP. At the same time, transforming these two vectors can not only achieve the assignment of jobs, but also achieve the sorting of jobs within the factory [6].

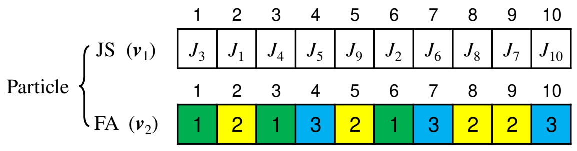


Figure 3. Particle representation.

In the case of using the above encoding scheme, the decoding scheme is very simple. The first vector is the total sequence of jobs. When decoding, first select the jobs of each factory according to the second vector, and then sort the jobs according to the first vector. After the sequence is determined, the entire processing flow is determined, and the processing schedule of the machine can be calculated by using the Eqs (3.6)–(3.12). As shown in Figure 4, three JS sequences will be obtained after decoding, each of which represents the JS of a factory, so the two objective values for the problem can finally be calculated.

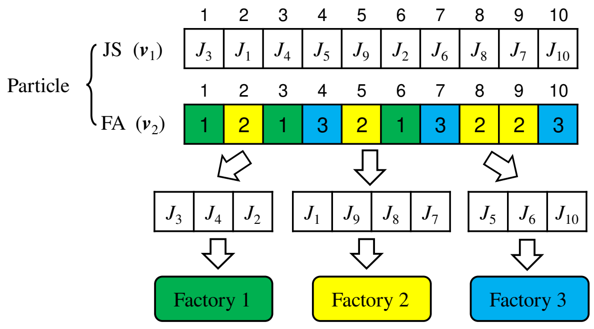


Figure 4. Decoding scheme.

4.3. *Two strategies of the MoPSODS-DE*

MoPSODS-DE is based on the MoPSO, which adopts an enhanced direction search strategy for mixed sampling of particle swarm, and a local search for the updated particle swarm through a differential evolution search strategy. The enhanced direction search strategy can speed up the problem solving, and the differential evolution search strategy can further improve the quality of the solution. This section describes two strategies in detail.

4.3.1. Direction search strategy

Enhanced direction search plays a very important role in this algorithm, which is an ingenious search strategy. It sets the search direction according to the PF, making the algorithm search in the specified direction, which can speed up the algorithm search. Enhanced direction search strategy requires two selection strategies, one selection strategy is PDDR-FF and the other is VEGA. PDDR-FF is a new particle evaluation method based on dominant and non-dominant relationships. PDDR-FF can analyze whether each particle in the current particle swarm is excellent, and its principle is also very simple and easy to implement [61].

In this algorithm, PDDR-FF is used to filter out the particles in the center area of the PF. These particles are excellent in each objective, and can be used to guide other particles to evolve toward the center area of the PF. The mathematical formula of PDDR-FF is as follows,

$$eval\,(s) = q\,(s) + \frac{1}{p\,(s) + 1} \tag{4.1}$$

where s is the evaluated particle, *q(s)* is the number of particles that dominate s, *p(s)* is the number of the particles which are dominated by s. When the particle is not dominated, the *q(s)* value is 0, and the *eval* value is always less than or equal to 1. When the particle is not dominated by other particles, the *p(s)* value is 0, and the *eval* value is always greater than or equal to 1. By comparing the *eval* value, the particles on the first PF can be selected.

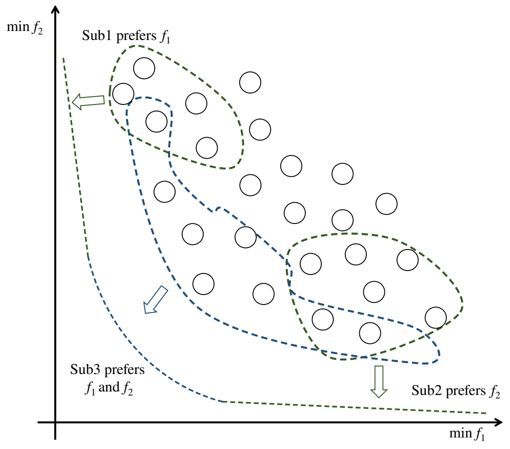


Figure 5. Illustration of particle swarm divided into three sub-particle swarms.

In most cases, the multiobjective problem will be decomposed into multiple single-objective problems to be solved. VEGA search technology is used to decompose multiobjective problems into multiple single-objective problems, and can find excellent solutions under one objective. An excellent solution is a sub-particle swarm under each objective. The role of VEGA is to find particles that satisfy a single objective, and form sub-particle swarm. When there are three objectives, there are three sub-particle swarms.

As shown in Figure 5, the particle swarms found by the VEGA search technology are in the edge area of the PF. The particles of these sub-particle swarms can guide the particle swarm towards the edge area of the PF, which guarantees that the searched solution meets each objective as much as possible. In general, the particle swarm is divided into different sub-particle swarms by the enhanced direction search strategy, and each sub-particle swarm is updated in specific directions, making the algorithm converge faster. The specific directions and corresponding sub-swarms are to divide sub-swarm 1 to the upper edge of the PF, sub-swarm 2 to the lower edge of the PF, and sub-swarm 3 to the central area of the PF.

4.3.2. Differential evolution search strategy

This section will describe the differential evolution search strategy in detail, and explain the combination of the differential evolution search strategy and the MoPSO.

The differential evolution search strategy is an inspiration from the DE. The DE is an evolution algorithm based on population, and the core of the algorithm is the differential idea. The difference idea is to find the difference between two particles, and then use this difference to evolve the less excellent particle to make it better. The point of differential evolution search strategy is also the idea of difference. The differential evolution search strategy applies to each generation of particles. First, two particles in the current particle swarm are randomly selected, then the difference is calculated, and finally the difference is used to evolve the non-excellent particle. Allow the current particle not to do the differential strategy, and allow the particle to be selected multiple times. Random selection is to not interfere excessively with the operation of the algorithm, making the algorithm more robust. When two identical particles are selected, in order to save time, no differential operation is performed, because there is no difference, and even if the differential operation is performed, it is the same. Allowing the particles to be selected multiple times is for a more random differential strategy to guarantee that the differential strategy can run efficiently.

The difference formula is as follows. $\mathbf{x}_1$ and $\mathbf{x}_2$ are the selected pair of particles, where $\mathbf{x}_1$ is better than $\mathbf{x}_2$, and the generated difference particle is $\mathbf{x}$. r is a random number between 0 and 1, this study makes r=1. P is the DE probability, rP is the difference probability, and its range is between 0–1, which is used to control the degree of difference. The larger the rP, the closer $\mathbf{x}_2$ is to $\mathbf{x}_1$.

$$\mathbf{x} = \mathbf{x}_2 + rP(\mathbf{x}_1 - \mathbf{x}_2) \tag{4.2}$$

The diagram of the differential evolution search strategy is shown in Figure 6. The characteristics of the PSO determine that the particles do not use the relationship between particles within the particle swarm when they are updated, so a differential evolution search strategy is needed to strengthen the connection between the particles. The differential evolution search strategy makes the particles not too scattered when searching for solutions, thus speeding up the search speed of the algorithm and improving the convergence performance of the algorithm.

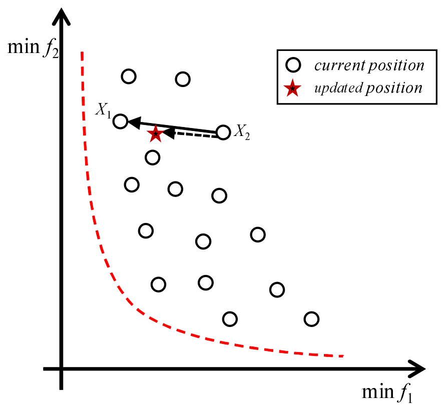


Figure 6. The diagram of the differential evolution search strategy.

The pseudo-code of the differential evolution search strategy is as follows.

Algorithm 2 The pseudo-code of the differential evolution search strategy

```
Input: particle swarm, swarmNumber, DE probability;
Output: the new particle swarm;
1: while ($t < swarmNumber$) do
2:     Randomly select a pair of particles;
3:     Compare the fitness of the particles to distinguish the good and bad of the two particles;
4:     Subtract the bad particles from the good particles to get the difference;
5:     Take part of the difference and add it to the bad particle;
6:     $t = t + 1$;
7: end while
8: output the new particle swarm;
```

4.4. Update operation of particles based on two-vector representation

Since the solution space of DFSP is not continuous, the sorting method is the simplest method to solve DFSP. The encoding and decoding methods are based on the sorting method. The proposed method of representation particles based on two-vector is to perform better encode and decode of the DFSP, and to update the individual more appropriately. In addition, the details of the algorithm updating the particles are described below.

Marco et al. proposed an algebraic PSO for the permutations search space [63]. This algorithm is improved from the classical PSO and is used to solve sequence problems. This algorithm rewrites the formula of the PSO algorithm to be suitable for solving the permutations search space, and gives some explanations. However, the algorithm formula remains basically unchanged, and the performance of the algorithm needs to be improved for the permutations search space.

Exchange sequence is a very important part here. The solution space of the permutation search space problem is a sequence, and the algorithm is always calculating the sequence. The exchange sequence plays an important role in the sequence calculation process, it can represent the difference between particles, and it is very convenient to use, and does not require other operations.

There are also some problems when using exchange sequences. The exchange sequence is the difference between the two sequences, and it represents the conversion step from one sequence to another sequence, so an exchange sequence is only related to the current two particles, and it cannot be applied to the third particle. Therefore, the use of the exchange sequence has the following restrictions. The order of the exchange elements in the exchange sequence cannot be changed, and it cannot be read at random. It can only be read in order.

In order to better solve the DFSP, the algebraic PSO algorithm and exchange sequence will be used in combination, and the algorithm formula will be improved. The improved PSO formula is as follows:

$$\mathbf{v}'(t+1) = \mathbf{rs} \tag{4.3}$$

$$\mathbf{x}_1(t+1) = \mathbf{x}(t) + \mathbf{v}'(t+1) \tag{4.4}$$

$$\mathbf{v}''(t+1) = c_1 r_1 (pbest - \mathbf{x}_1(t+1)) \tag{4.5}$$

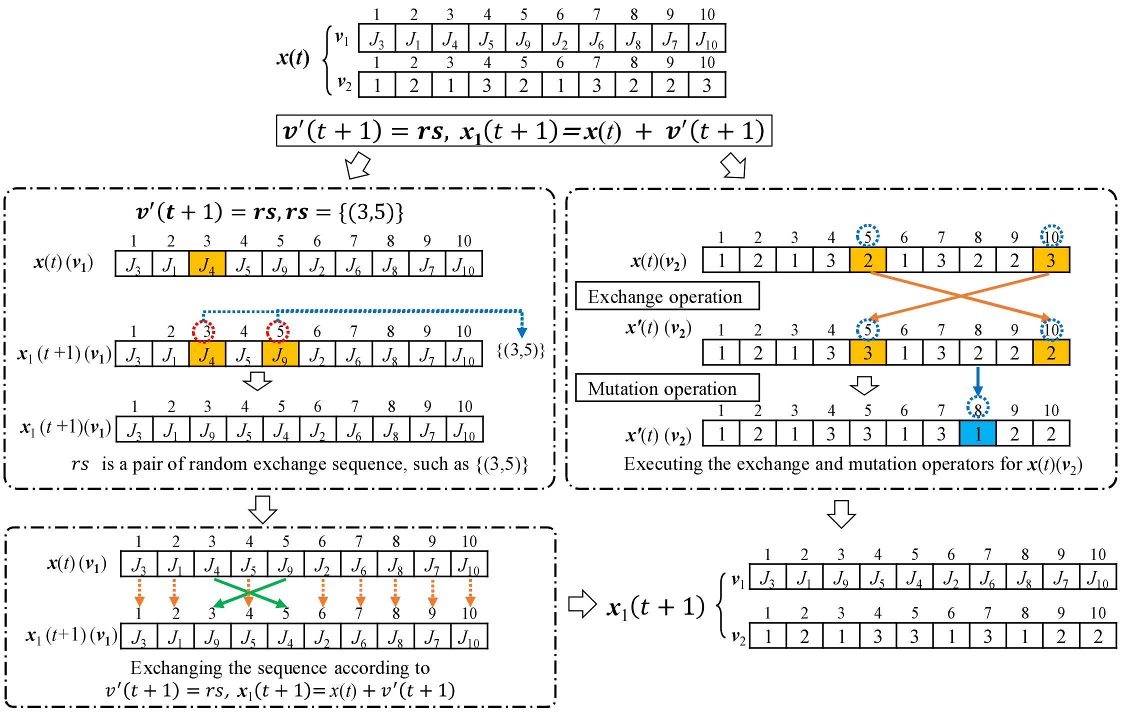
$$\mathbf{x}_2(t+1) = \mathbf{x}_1(t+1) + \mathbf{v}''(t+1) \tag{4.6}$$

$$\mathbf{v}'''(t+1) = c_2 r_2 (gbest - \mathbf{x}_2(t+1)) \tag{4.7}$$

$$\mathbf{x}(t+1) = \mathbf{x}_2(t+1) + \mathbf{v}'''(t+1) \tag{4.8}$$

The initial velocity of traditional PSO is determined by the inertia of the particles, but it is not applicable to the exchange sequence. Because of the existence of the exchange sequence, the previous speed is only applicable to the previous update process. In this study, the particle velocity update is divided into three parts, and **rs** is a randomly generated exchange sequence. **rs** is used to replace the initial speed in the original PSO formula, which can be expressed as Eq (4.3). Its role is to change the search direction of the particles and strengthen the search range of the particles. $pbest - \mathbf{x}_1(t)$ can get the exchange sequence of the current particle and *pbest*. $gbest - \mathbf{x}_2(t)$ can get the exchange sequence of the current particle and *gbest*. $\mathbf{v}'(t+1)$, $\mathbf{v}''(t+1)$, and $\mathbf{v}'''(t+1)$ are three exchange sequences corresponding to different reference particles for particle update, which can be represented by Eqs (4.3), (4.5) and (4.7) respectively. $\mathbf{x}_1(t+1)$ and $\mathbf{x}_2(t+1)$ are intermediate particles, which are used to update the current position of the particles, which can be represented by Eqs (4.4) and (4.6). Equation (4.8) shows that $\mathbf{x}(t+1)$ is the new generation particles.

In this study, aiming at the encoding of the problem, the particles are updated by crossover and mutation of the FA with a certain probability. There are three kinds of factory selection: 1, 2, and 3, which respectively represent three different factories, and then randomly generate a random decimal less than 1. If the random number is less than the crossover rate, a random single-point crossover will be performed in the current FA, otherwise, no crossover will be performed. At the same time, it is also necessary to randomly generate a decimal number less than 1. If this number is less than the mutation rate, the particles will be updated randomly by mutation, otherwise, no mutation will be performed. However, the update range of the serial number is still between 1, 2, and 3.



x(t)
v1
v2
v′(t + 1) = rs, x1(t + 1)=x(t) + v′(t + 1)
v′(t + 1) = rs, rs = {(3,5)}
x(t)(v1)
x1(t+1)(v1)
{(3,5)}
rs is a pair of random exchange sequence, such as {(3,5)}
x(t)(v2)
Exchange operation
x′(t)(v2)
Mutation operation
Executing the exchange and mutation operators for x(t)(v2)
Exchanging the sequence according to
v′(t + 1) = rs, x1(t + 1)=x(t) + v′(t + 1)
x1(t + 1)

Figure 7. The first step of PSO processing.

Therefore, the specific operation for the PSO update formula can be shown in Figures 7–9 as examples. Figure 7 corresponds to Eqs (4.3) and (4.4), which is also the first step of PSO. For the JS of the particle, suppose the number of exchange pairs in the exchange sequence is 1, and **rs** = (3, 5). That is, the actual executing sequence is (3, 5), the new sequence $\mathbf{x}_1(t+1)$ (v_1) is $\{J_3, J_1, J_9, J_5, J_4, J_2, J_6, J_8, J_7, J_{10}\}$. And then for the FA of the particle, perform a random exchange sequence of (5, 10) on the (v_2) sequence, and then perform random mutation, and perform mutation on the sequence number 8. Finally, the job and FA have been changed, $\mathbf{x}_1(t+1)$ (v_2) is {1, 2, 1, 3, 3, 1, 3, 1, 2, 2}.

Figure 8 corresponds to Eqs (4.5) and (4.6), which is also the second step of PSO. For the JS of the particle, suppose $c_1 = 1.0$, $r_1 = 0.4$, the difference between the particle in this state and the *pbest* particle is that the number of exchange pairs in the exchange sequence is 2, and $1.0 \times 0.4 \times 2 = 0.8$, so the number of exchange pairs of executing the exchange sequence is 0. That is, the actual executing sequence is empty, and the new sequence $\mathbf{x}_2(t+1)$ (v_1) is the same as the old individual $\mathbf{x}_1(t+1)$ (v_1). For the FA of the particle, suppose the starting sequence number of the single point crossover starts

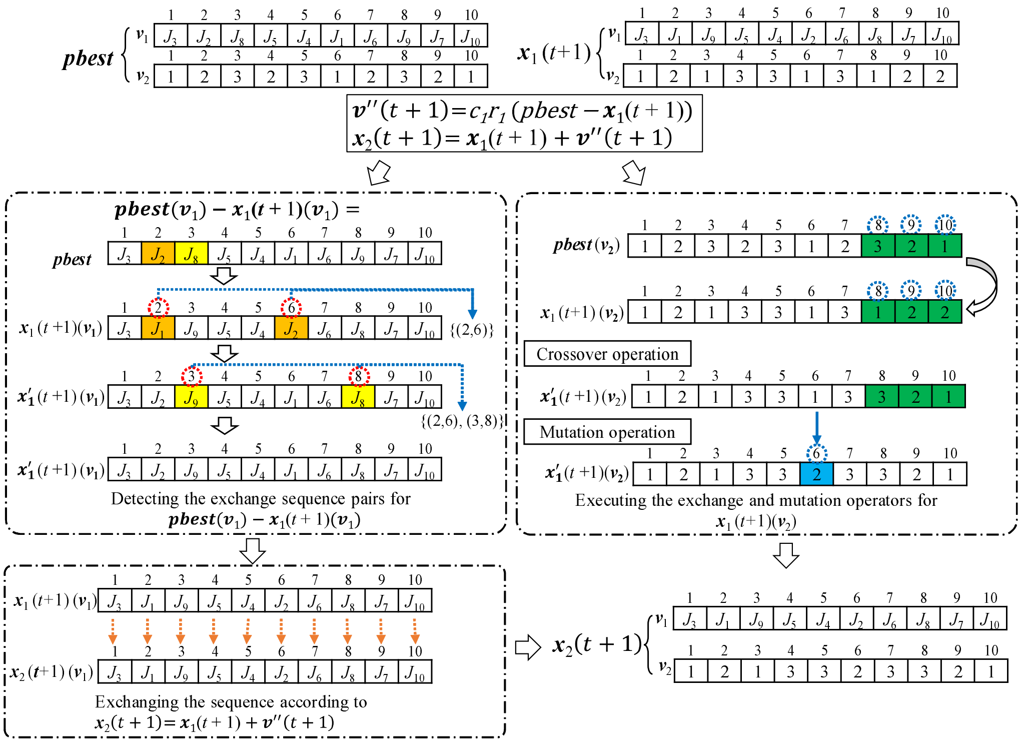


Figure 8. The second step of PSO processing.

from 8, and the random mutation is mutation at the sequence number 6, then the $\mathbf{x}_2(t+1)$ (v_2) sequence after the operation is completed is {1, 2, 1, 3, 3, 2, 3, 3, 2, 1}.

Figure 9 corresponds to Eqs (4.7) and (4.8), which is also the last step of PSO. Suppose c_2 = 1.0, r_2 = 0.4, for the difference between the JS in this state and the *gbest* sequence, the number of exchange pairs in the exchange sequence is 2, and 1.0 × 0.4 × 3 = 1.2, so the number of exchange pairs of executing the exchange sequence is 1. That is, the actual executing sequence is (1, 4). Finally, for the FA crossover and mutation, the sequence number for random single-point crossover is 6, and the sequence number for random mutation is 4, so that a new individual particle can be obtained. The two sequences $\mathbf{x}(t+1)$ (v_1) and $\mathbf{x}(t+1)$ (v_2) of the particle are $\{J_5, J_1, J_9, J_3, J_4, J_2, J_6, J_8, J_7, J_{10}\}$ and {1, 2, 1, 2, 3, 1, 2, 1, 2, 3}, respectively.

Since MoPSODS has updated two vectors of the particle, in order to guarantee the quality of the particle and the convergence performance of the algorithm, the difference operation does not operate on the factory vector of the particle. Therefore, the difference operation is only updated for the workpiece vector, and the factory sequence of the factory vector remains unchanged, so that the bad particles can move to the good particles more accurately.

The Figure 10 shows an example of an update operation on particles using the difference operation. On the one hand, the difference between particles x_2 and x_1 in the JA vector is (J_2, J_6), (J_5, J_8). Therefore, the JA vector of x_2 is improved in the direction of x_1 by way of exchange sequence, that is, the positions of J_2 and J_6, and the positions of J_5 and J_8 are exchanged. Suppose r = 1.0, P = 0.8, and the length of exchange sequence is 2, and 1.0 × 0.8 × 2 = 1.6, so the length of executing the exchange sequence is 1. That is, the actual executing sequence is (J_2, J_6), the new individual x (JS) is {J_3, J_2, J_4, J_8, J_9, J_1, J_6, J_5, J_{10}, J_7}

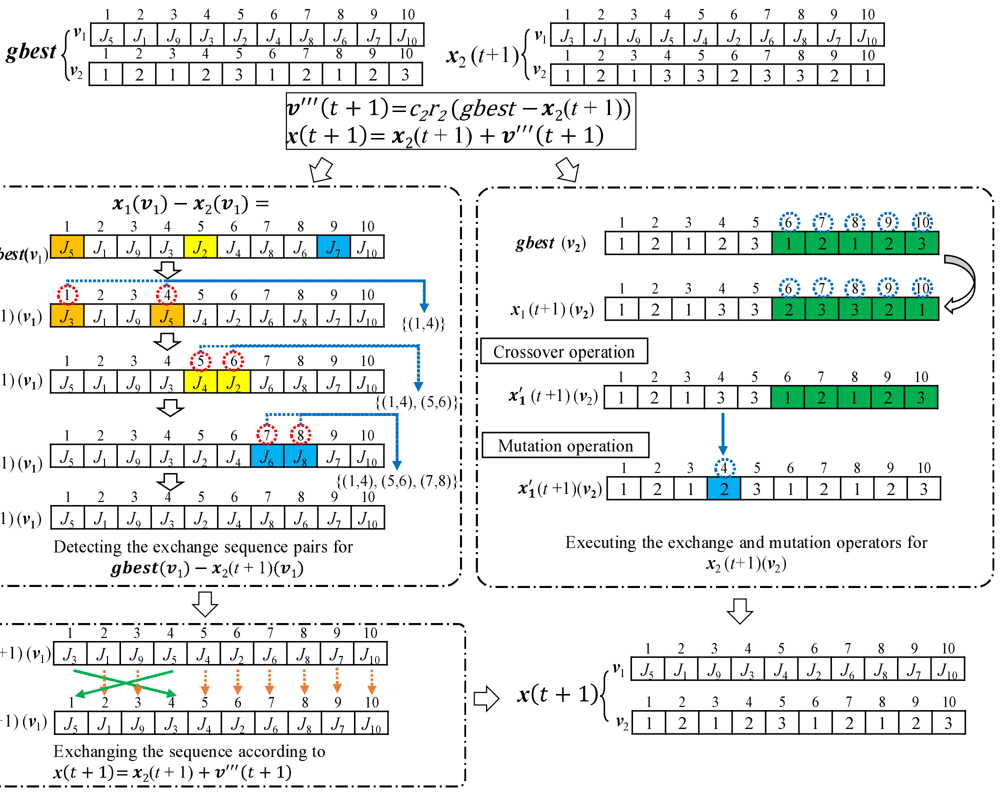


Figure 9. The third step of PSO processing.

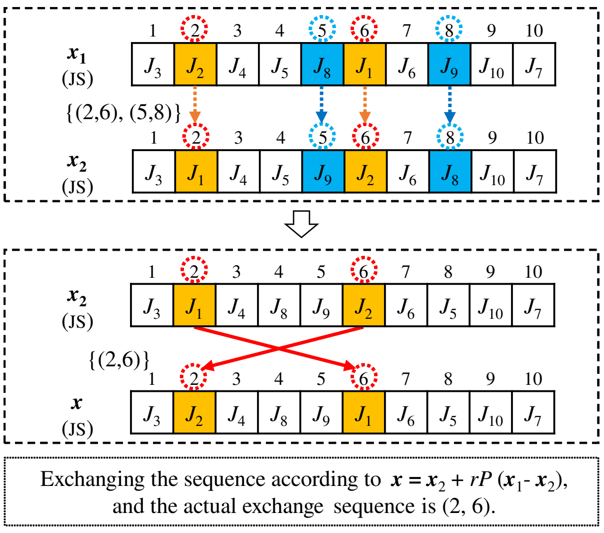


Figure 10. The differential search strategy.

5. Experimental results

This section analyzes the performance of the algorithm from an experimental perspective.

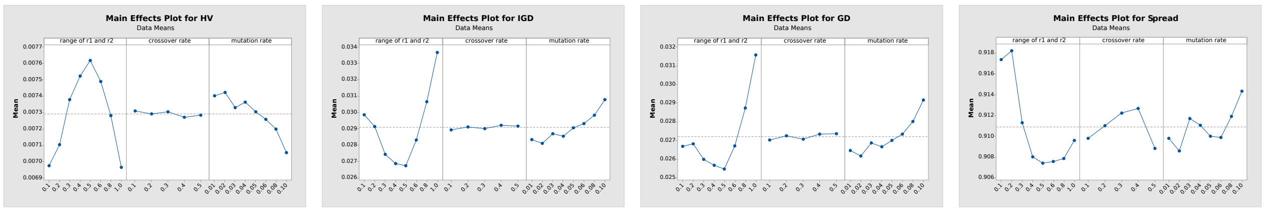
Naderi and Ruiz presented 720 large instances, increasing the number of factories F to the known 120 Taillard instances [64]. Data sets range from 20 jobs and 5 machines to 500 jobs and 20 machines. Small instances are easily solved by most existing DFSP meta-heuristics and are therefore not used in this work. This study will use large instances and choose data sets where the number of factories is 3. The name of the problem consists of the number of jobs, the number of machines and the number of factories. 20_5_3 is a problem that contains 20 jobs, 5 machines, and 3 factories, and 500_20_3 is a problem that contains 500 jobs, 20 machines, and 3 factories.

The proposed algorithm MoPSODS-DE will be compared with NSGA-II, SPEA2, MoPSO, MOEA/D, MOHEA, and MOHEA/DE. In the experimental part of this study, indicators hypervolume (*HV*), generational distance (*GD*), inverted generational distance (*IGD*), *Spread*, and coverage (*C*) are used to evaluate various performances of the algorithm. The results of significance analysis for algorithm performance are stored in the table, and excellent solutions will be highlighted in bold font style. In the table, "*" indicates that the comparison result is not significant, "+" indicates that our algorithm is better than other algorithms, and "-" indicates that our algorithm is worse than other algorithms.

The *HV* index evaluates the dominated space of the obtained solution set [65]. The larger the *HV* value, the better the convergence performance of the algorithm. The reference point of *HV* in this experiment is set as combining the worst fitness values in each objective, which found by all algorithms run multiple times. The *GD* index and the *IGD* index are very similar. They both evaluate the distance between the solution and the PF [66]. The smaller values of *GD* and *IGD* might mean that the algorithm has better convergence performance and distribution performance. *GD* is an index to evaluate the convergence performance of the algorithm, and the *IGD* index can evaluate the convergence and distribution performance of the algorithm at the same time. A smaller value of *GD* means that the algorithm has better convergence performance, while a smaller value of *IGD* means that the algorithm not only has better convergence performance but also better distribution performance. The *Spread* index is an index to evaluate the diversity of the solution set generated by the algorithm, and to judge whether the distribution of the solution is widespread [59]. The smaller the *Spread* value, the better the distribution performance and the more diverse the solution set.

5.1. Parameter settings

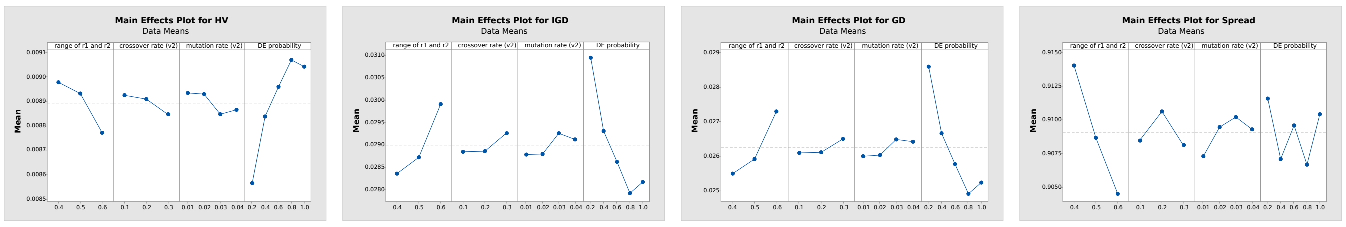
In this section, the main purpose is to find the optimal parameter settings for the velocity and position update formulas and the differential operation formulas in MoPSODS-DE. Because the algorithm is an improvement for MoPSO, so the parameters of MoPSO is first designed to provide a reliable parameter range for MoPSODS-DE parameters, and then the parameters of MoPSODS-DE are designed. The exchange sequence is used to represent the difference between particles, so the values of c_1 and c_2 are both 1. Three parameters of MoPSO is first considered: the range of r_1 and r_2, the crossover rate (v_2), and the mutation rate (v_2). The levels of these three parameters are set as follows: the range of r_1 and r_2 at eight levels (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, and 1.0), the crossover rate (v_2) at five levels (0.1, 0.2, 0.3, 0.4, 0.5), and Mutation rate (v_2) at eight levels (0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.08 and 0.1). There are 320 (8 x 5 x 8) different combinations of the three parameters. The design of experiments (DOE) is performed on instances with 100_10 (100 jobs and 10 machines), and the instance ran 30 independent replicates [67]. Therefore, a full factorial experimental design with 9600 (320 x 30) treatments is verified to decide parameter settings. Particularly, the size of the three subparticle swarms is set to one-third of the entire particle swarm size. Figure 11 shows the main effect of each parameter, where the best parameter settings are the ranges of r_1 and r_2 = 0.5, crossover rate (v_2) = 0.1, and mutation rate (v_2) = 0.02.



(a) Main effect on *HV* for problem 100x10. **(b)** Main effect on *IGD* for problem 100x10. **(c)** Main effect on *GD* for problem 100x10. **(d)** Main effect on *Spread* for problem 100x10.

Figure 11. Main effect of each parameter for different instances in MoPSO.

Then, design for the parameters of MoPSODS-DE. The levels of the four parameters related to the algorithm are set as follows: the range of r_1 and r_2 at three levels (0.4, 0.5, 0.6), the crossover rate (v_2) at three levels (0.1, 0.2, 0.3), the mutation rate (v_2) at eight levels (0.01, 0.02, 0.03, and 0.04), and five levels (0.2, 0.4, 0.6, 0.8 and 0.1) for DE probability P. There are 180 (3 x 3 x 4 x 5) different combinations of the four parameters. The full factorial experimental design parameters were still used. Figure 12 shows the main effect of each parameter, where the best parameter settings are the ranges of r_1 and r_2 = 0.4, crossover rate (v_2) = 0.1, mutation rate (v_2) = 0.01, and DE probability P = 0.8.



(a) Main effect on *HV* for problem 100x10. **(b)** Main effect on *IGD* for problem 100x10. **(c)** Main effect on *GD* for problem 100x10. **(d)** Main effect on *Spread* for problem 100x10.

Figure 12. Main effect of each parameter for different instances in MoPSODS-DE.

Because the parameters not only include the ranges of r_1 and r_2, crossover rate (v_2), mutation rate (v_2), and DE probability P, but also population size, elite population size, sub-particle swarm1 size, sub-particle swarm2 size, sub-particle swarm3 size, crossover rate (v_1) and mutation rate (v_1) required by other algorithms, and maximum generation. Therefore, this study uses the full factorial experiments to design the parameters for all comparison algorithms of NSGA-II, SPEA2, MoPSO, MOEA/D, MOHEA, and MOHEA/DE. After the full factorial experiments for each algorithm, the optimal parameter combination of each algorithm after optimization is obtained. The design of all parameters corresponding to all algorithms is shown in the Table 1.

For all algorithms in this study, the population size and the maximum generation are fixed, and the performance of each algorithm can be compared when the algorithm runs to the maximum generation. MOHEA and MOHEA/DE have sub-populations, and MoPSODS-DE have sub-particle swarms. The MOHEA and MOHEA/DE has two sub-populations and an elite population, while the MoPSODS-DE has three sub-particle swarms, and the sub-population size is set according to the population size, and the sum of the three sub-particle swarm sizes is equal to the particle swarm size.

Table 1. Parameter settings.

	NSGA-II	SPEA2	MoPSO	MOEA/D	MOHEA	MOHEA/DE	MoPSODS-DE
population size	100	100	100	100	100	100	100
elite population size	100	100	\	\	50	50	\
sub-particle swarm1 size	\	\	\	\	\	\	33
sub-particle swarm2 size	\	\	\	\	\	\	33
sub-particle swarm3 size	\	\	\	\	\	\	34
ranges of r_1 and r_2	\	\	0.5	\	\	\	0.4
crossover rate (v_1)	0.4	0.4	\	0.8	0.8	0.8	\
mutation rate (v_1)	0.4	0.4	\	0.4	0.2	0.1	\
crossover rate (v_2)	0.2	0.1	0.1	0.1	0.2	0.1	0.1
mutation rate (v_2)	0.04	0.02	0.02	0.04	0.02	0.01	0.01
DE probability P	0	0	0	0	0	0.8	0.8
max. generation	500	500	500	500	500	500	500

5.2. *Time complexity analysis of algorithms*

In the MoPSODS-DE algorithm, assuming the size of the particle swarm is n. Since the encoding method of two vectors is used, jobs and factories need to be one-to-one correspondence to calculate the makespan and the total processing time during decoding. Therefore, the computational complexity of calculating makespan and total processing time is $O(n^2)$ respectively. In the initial stage, the particle swarm is initialized, and the computational complexity of setting the historical optimal individual and the global optimal individual is $O(n^2)$ and $O(n)$, respectively. When updating the particle swarm, the time complexity of swarming the particle swarm is $O(n)$ and the time complexity of updating the particle position is $O(n^2)$. In the local search stage, two particles are randomly selected, and the time complexity of the differential evolution update of the particles is $O(n^2)$. Finally, when replacing old and new particles, the time complexity of quickly sorting the particle swarm according to the fitness function value of the particles and select the first n particles to enter the next iteration is $O(n^2{+}2nlog4n)$. In summary, in M iterations, the time complexity of MoPSODS-DE is shown as follows:

$$O(n) = O(n^2) + O(n^2) + O(n) + O(n) + O(n^2) + O(n^2) + O(n^2 + 2nlog4n) = O(n^2) \tag{5.1}$$

Also for DFSP, it can be known that the time complexities of NSGA-II, SPEA2, MoPSO, MOEA/D, MOHEA, and MOHEA/DE are $O(n^3)$, $O(n^2\text{logn})$, $O(n^2)$, $O(n^2)$, $O(n^2)$, and $O(n^2)$, respectively.

5.3. *Effectiveness of exchange sequence*

The encoding method based on two vectors to represent particles uses integer encoding. When using this encoding method to update particles, exchange sequence, crossover and mutation are used.

These update operations can not only improve the convergence performance of the algorithm but also increase the diversity of the particle swarm. To test the effectiveness of exchange sequence under integer encoding, this study encodes the following two discrete algorithms for the single-objective (makespan) DFSP: 1) PSO-noES based on position-order real number encoding and without the use of exchange sequence. 2) PSO-ES for updating particles based on integer encoding exchange sequence. The two algorithms are compared on 12 standard problems, and the comparison of the objective value (makespan) is used to illustrate the effectiveness of the exchange sequence. Figure 13 presents the comparison results, which show that it is more efficient to use the PSO of the exchange sequence to solve the DFSP.

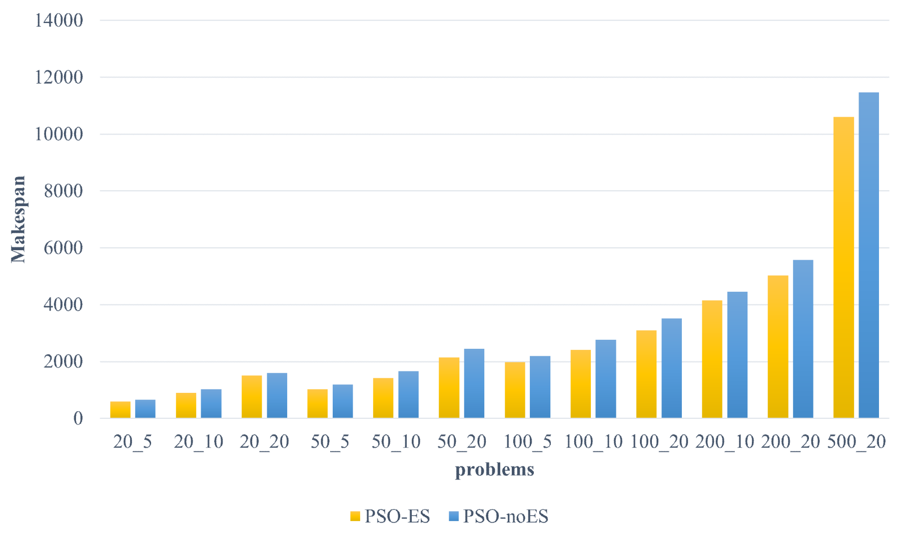


Figure 13. Results of makespan on algorithms.

In addition, this study codes the following two discrete algorithms for multiobjective DFSP to test the effectiveness of exchange sequence: (i) MoPSO-noES based on real-order encoding and without the use of exchange sequence. (ii) MoPSO-ES for updating particles based on integer encoding exchange sequence. The two algorithms are compared on 12 standard problems, and the performance of the algorithm is shown by two comprehensive evaluation indexes, *HV* and *IGD*. Table 2 presents the comparison results, which show that: 1) in the given 12 problems, MoPSO-ES can get 9 better results; and 2) the significance analysis results show that MoPSODS-ES has significant convergence and distribution performance compared with MoPSO-noES. It can be seen that after using exchange sequence, particles can be updated more efficiently, thereby improving the convergence performance and distribution performance of the algorithm.

It can be seen from the above that the position-order real number encoding method is suitable for the original particle swarm to solve the continuous optimization problem, and the individual obtained by encoding represents the direction vector, and this encoding method cannot well record the sequence relationship of the two-vector encoding method. Therefore, it is not suitable for solving discrete optimization problems. The integer encoding method is suitable for discrete PSO to solve discrete optimization problems. This encoding method can use exchange sequence to update individuals. The

Table 2. The index results for MoPSO-noES and MoPSO-ES.

problem	*HV*			*IGD*		
set	MoPSO-noES		MoPSO-ES	MoPSO-noES		MoPSO-ES
20_5_3	2.58E-02	+	**4.37E-02**	1.94E-01	+	**1.25E-01**
20_10_3	2.35E-02	+	**3.92E-02**	1.01E-01	+	**3.37E-02**
20_20_3	**3.73E-02**	-	2.02E-02	9.30E-02	-	1.54E-01
50_5_3	5.19E-02	+	**5.46E-02**	1.55E-01	+	**1.19E-01**
50_10_3	1.52E-02	+	**3.13E-02**	1.21E-01	+	**3.75E-02**
50_20_3	1.21E-02	+	**2.59E-02**	9.32E-02	+	**1.68E-02**
100_5_3	**4.28E-02**	-	3.20E-02	1.77E-01	*	1.87E-01
100_10_3	2.31E-02	+	**2.75E-02**	1.00E-01	+	**6.68E-02**
100_20_3	7.16E-03	+	**1.89E-02**	9.85E-02	+	**1.29E-02**
200_10_3	**3.22E-02**	-	1.99E-02	1.18E-01	*	1.51E-01
200_20_3	1.75E-02	+	**2.29E-02**	7.76E-02	+	**4.43E-02**
500_20_3	4.10E-03	+	**8.19E-03**	8.77E-02	+	**3.97E-02**
+/-/*		9/3/0			9/1/2	

flexible exchange and update of the encoded individual sequence can not only preserve the context of the sequence, so that the good job sorting and factory allocation remain unchanged, but also can move to the good individual more reasonably.

5.4. *Effectiveness of direction search strategy*

The direction search strategy explores the particle swarm in multiple directions of the PF, which can enhance the strong convergence ability of the algorithm in different areas of the PF. Therefore, it is necessary to test the effect of the direction search strategy on the performance of the algorithm. In this study, MoPSO and MoPSODS are compared on 12 standard problems, and the results of *HV* and *IGD* are used as the criterion for evaluating the performance of the algorithm, so that the effectiveness of the direction search strategy can be verified. Table 3 shows the *HV* and *IGD* values of MoPSO and MoPSODS on 12 problems. It can be seen that the index values of MoPSODS on all problems are significantly better than MoPSO, which fully proves the effectiveness of the direction search strategy. The direction search strategy can not only improve the convergence performance of the algorithm, but also improve the distribution performance.

5.5. *Effectiveness of DE search strategy*

The DE search strategy can strengthen the connection between particles in the particle swarm, and perform differential operations on the updated particle swarm to improve the quality of the particles and prevent falling into local optimum. Therefore, the effectiveness of DE search strategy needs to be tested. This study also compares MoPSODS and MoPSODS-DE on 12 standard problems, and uses the results of two frequently used comprehensive evaluation indexes *HV* and *IGD* as the criterion for evaluating whether the DE search strategy is effective. The mean *HV* and mean *IGD* values of MoPSODS and MoPSODS-DE after 30 runs on 12 problems are shown in Figure 4. The two indexes results in Figure 4 illustrate that MoPSODS-DE outperforms MoPSODS on 10 of the 12 problems, and the performance is significant on large-scale problems. Such results fully demonstrate the effectiveness of the DE search strategy. Therefore, it can be shown that the addition of DE search strategy further improves the convergence performance and distribution performance of the algorithm.

Table 3. The index results for MoPSO and MoPSODS.

problem	*HV*			*IGD*		
set	MoPSO		MoPSODS	MoPSO		MoPSODS
20_5_3	2.72E-02	+	**3.83E-02**	5.02E-02	+	**1.02E-02**
20_10_3	2.93E-02	+	**4.47E-02**	6.63E-02	+	**1.66E-02**
20_20_3	2.36E-02	+	**3.22E-02**	4.12E-02	+	**8.19E-03**
50_5_3	2.03E-02	+	**5.04E-02**	1.37E-01	+	**1.90E-02**
50_10_3	1.01E-02	+	**3.21E-02**	1.14E-01	+	**1.37E-02**
50_20_3	5.98E-03	+	**2.01E-02**	9.50E-02	+	**1.09E-02**
100_5_3	5.85E-03	+	**2.60E-02**	1.37E-01	+	**1.35E-02**
100_10_3	4.90E-03	+	**2.55E-02**	1.40E-01	+	**1.38E-02**
100_20_3	5.89E-03	+	**2.40E-02**	1.22E-01	+	**2.32E-02**
200_10_3	3.51E-03	+	**1.89E-02**	1.16E-01	+	**9.88E-03**
200_20_3	2.53E-03	+	**1.51E-02**	1.09E-01	+	**1.04E-02**
500_20_3	1.37E-03	+	**1.08E-02**	1.02E-01	+	**6.57E-03**
+/-/*	12/0/0			12/0/0		

Table 4. The index results for MoPSODS and MoPSODS-DE.

problem	*HV*			*IGD*		
set	MoPSODS		MoPSODS-DE	MoPSODS		MoPSODS-DE
20_5_3	9.12E-03	*	**9.58E-03**	1.30E-02	+	**1.11E-02**
20_10_3	8.62E-03	*	**8.72E-03**	1.90E-02	*	**1.88E-02**
20_20_3	2.09E-02	+	**2.16E-02**	1.91E-02	+	**1.56E-02**
50_5_3	1.53E-02	*	**1.56E-02**	2.83E-02	+	**2.14E-02**
50_10_3	1.04E-02	*	**1.04E-02**	3.38E-02	*	**3.35E-02**
50_20_3	**7.67E-03**	-	6.84E-03	**1.96E-02**	*	2.11E-02
100_5_3	6.09E-03	+	**6.57E-03**	2.02E-02	+	**1.87E-02**
100_10_3	**1.75E-03**	*	1.54E-03	**1.57E-02**	-	1.69E-02
100_20_3	2.56E-03	+	**3.01E-03**	1.91E-02	+	**1.29E-02**
200_10_3	5.60E-04	+	**1.10E-03**	1.98E-02	+	**9.05E-03**
200_20_3	5.15E-04	+	**5.77E-04**	1.15E-02	+	**1.01E-02**
500_20_3	1.93E-04	+	**5.03E-04**	1.95E-02	+	**7.88E-03**
+/-/*	6/1/5			8/1/3		

Table 5. The results of *HV* index.

problem set	NSGA-II		SPEA2		MoPSO		MOEA/D		MOHEA		MOHEA/DE		MoPSODS-DE
20_5_3	4.11E-02	+	4.15E-02	+	2.89E-02	+	4.04E-02	+	3.58E-02	+	4.21E-02	+	**4.31E-02**
20_10_3	4.47E-02	+	4.47E-02	+	3.04E-02	+	4.28E-02	+	3.52E-02	+	**4.67E-02**	*	4.66E-02
20_20_3	4.10E-02	+	**4.19E-02**	*	3.10E-02	+	3.98E-02	+	3.45E-02	+	4.10E-02	+	4.18E-02
50_5_3	5.74E-02	+	5.68E-02	+	3.03E-02	+	5.91E-02	+	5.19E-02	+	5.75E-02	+	**6.83E-02**
50_10_3	3.53E-02	+	3.57E-02	+	1.48E-02	+	3.70E-02	+	3.00E-02	+	3.42E-02	+	**4.06E-02**
50_20_3	2.06E-02	+	2.11E-02	+	7.80E-03	+	2.15E-02	+	1.75E-02	+	2.11E-02	+	**2.51E-02**
100_5_3	2.90E-02	+	2.99E-02	+	1.26E-02	+	3.26E-02	+	2.65E-02	+	2.97E-02	+	**3.81E-02**
100_10_3	1.92E-02	+	2.04E-02	+	5.23E-03	+	2.35E-02	+	1.66E-02	+	2.03E-02	+	**2.89E-02**
100_20_3	2.23E-02	+	2.30E-02	+	8.92E-03	+	2.59E-02	+	2.02E-02	+	2.34E-02	+	**3.16E-02**
200_10_3	1.94E-02	+	1.86E-02	+	6.37E-03	+	2.38E-02	+	1.72E-02	+	1.96E-02	+	**2.84E-02**
200_20_3	1.31E-02	+	1.34E-02	+	3.73E-03	+	1.74E-02	+	1.16E-02	+	1.34E-02	+	**2.06E-02**
500_20_3	8.34E-03	+	8.49E-03	+	2.34E-03	+	1.26E-02	+	8.08E-03	+	8.80E-03	+	**1.47E-02**
+/-/*	12/0/0		11/0/1		12/0/0		12/0/0		12/0/0		11/0/1		

5.6. Results and discussion

This section analyzes algorithm performance through various indexes. *HV*, *GD*, *IGD* and *Spread* mainly reflect the convergence performance and the distribution performance of the algorithm.

Table 6. The results of *GD* index.

problem set	NSGA-II		SPEA2		MoPSO		MOEA/D		MOHEA		MOHEA/DE		MoPSODS-DE
20_5_3	1.75E-02	+	1.63E-02	+	7.42E-02	+	1.57E-02	+	2.08E-02	+	1.47E-02	*	**1.18E-02**
20_10_3	2.22E-02	+	1.93E-02	+	7.24E-02	+	2.01E-02	+	2.72E-02	+	**1.43E-02**	*	1.51E-02
20_20_3	1.66E-02	+	1.47E-02	*	5.25E-02	+	1.45E-02	*	1.74E-02	+	**1.36E-02**	*	1.39E-02
50_5_3	4.77E-02	+	4.55E-02	+	1.48E-01	+	2.62E-02	+	5.17E-02	+	4.20E-02	+	**1.53E-02**
50_10_3	3.55E-02	+	3.74E-02	+	1.35E-01	+	3.35E-02	+	4.26E-02	+	3.83E-02	+	**1.98E-02**
50_20_3	3.19E-02	+	2.76E-02	+	1.09E-01	+	1.71E-02	+	3.67E-02	+	2.65E-02	+	**1.29E-02**
100_5_3	5.81E-02	+	5.43E-02	+	1.47E-01	+	2.87E-02	+	6.03E-02	+	5.20E-02	+	**1.59E-02**
100_10_3	5.64E-02	+	5.13E-02	+	1.53E-01	+	2.52E-02	+	6.50E-02	+	5.12E-02	+	**1.35E-02**
100_20_3	4.53E-02	+	4.16E-02	+	1.21E-01	+	1.61E-02	+	4.85E-02	+	3.86E-02	+	**1.12E-02**
200_10_3	5.70E-02	+	6.03E-02	+	1.45E-01	+	3.19E-02	+	6.27E-02	+	5.22E-02	+	**1.31E-02**
200_20_3	4.28E-02	+	4.22E-02	+	1.20E-01	+	1.24E-02	*	4.85E-02	+	4.10E-02	+	**9.72E-03**
500_20_3	4.28E-02	+	4.25E-02	+	1.04E-01	+	6.86E-03	*	4.36E-02	+	3.87E-02	+	**6.05E-03**
+/-/*	12/0/0		11/0/1		12/0/0		9/0/3		12/0/0		9/0/3		

The *HV* index results are shown in Table 5. It can be seen that in 10 of the 12 problems, the *HV* value of the MoPSODS-DE is the largest, which shows that the *HV* value of other algorithms is significantly smaller than that of the MoPSODS-DE algorithm. In addition, in the two problems that the main algorithm does not perform best, the significant difference between the algorithm is not obvious. Therefore, in terms of *HV* value results, MoPSODS-DE is better than all comparison algorithms, so its convergence performance is the most outstanding.

The results of the *GD* index are shown in Table 6, and MoPSODS-DE performs the best on 10 problems. On the other 2 problems, although the *GD* value of MOHEA is the largest, it is not obvious compared with the significant difference analysis of MoPSODS-DE. Table 7 shows the results of the *IGD* index, where MoPSODS-DE performs the best on all problems. Therefore, it can be seen from the results of *GD* index and *IGD* index that MoPSODS-DE has the best convergence performance and distribution performance among all the compared algorithms.

Table 7. The results of *IGD* index.

problem set	NSGA-II		SPEA2		MoPSO		MOEA/D		MOHEA		MOHEA/DE		MoPSODS-DE
20_5_3	1.86E-02	+	1.71E-02	+	6.31E-02	+	1.88E-02	+	3.25E-02	+	1.53E-02	+	**1.22E-02**
20_10_3	2.09E-02	+	2.09E-02	+	6.75E-02	+	2.42E-02	+	4.30E-02	+	1.63E-02	*	**1.58E-02**
20_20_3	1.70E-02	+	1.57E-02	*	4.84E-02	+	2.02E-02	+	3.32E-02	+	1.61E-02	*	**1.48E-02**
50_5_3	4.43E-02	+	4.35E-02	+	1.36E-01	+	3.04E-02	+	5.47E-02	+	4.13E-02	+	**1.75E-02**
50_10_3	3.40E-02	+	3.45E-02	+	1.30E-01	+	3.51E-02	+	4.74E-02	+	3.64E-02	+	**1.82E-02**
50_20_3	3.13E-02	+	2.93E-02	+	1.08E-01	+	2.46E-02	+	4.22E-02	+	2.80E-02	+	**1.35E-02**
100_5_3	6.15E-02	+	5.71E-02	+	1.45E-01	+	3.50E-02	+	6.79E-02	+	5.58E-02	+	**1.95E-02**
100_10_3	5.96E-02	+	5.40E-02	+	1.54E-01	+	3.68E-02	+	6.97E-02	+	5.37E-02	+	**1.66E-02**
100_20_3	4.92E-02	+	4.47E-02	+	1.26E-01	+	3.16E-02	+	5.42E-02	+	4.24E-02	+	**1.47E-02**
200_10_3	5.54E-02	+	5.86E-02	+	1.39E-01	+	3.23E-02	+	6.37E-02	+	5.19E-02	+	**1.24E-02**
200_20_3	4.81E-02	+	4.74E-02	+	1.25E-01	+	2.48E-02	+	5.59E-02	+	4.64E-02	+	**1.29E-02**
500_20_3	4.60E-02	+	4.51E-02	+	1.07E-01	+	1.71E-02	+	4.70E-02	+	4.21E-02	+	**1.08E-02**
+/-/*	12/0/0		11/0/1		12/0/0		12/0/0		12/0/0		10/0/2		

From the results of *Spread* index in Table 8, MoPSODS-DE performs best on 6 of the 12 questions. The only problems where MoPSODS-DE performs significantly worse are small-scale problems such as 20_5_3 and 50_5_3. For other problems, the distribution performance of MoPSODS-DE is still very outstanding. In conclusion, MoPSODS-DE has good convergence performance, and MoPSODS-DE can also make the distribution of the obtained solutions more diverse and extensive.

Table 8. The results of *Spread* index.

problem set	NSGA-II		SPEA2		MoPSO		MOEA/D		MOHEA		MOHEA/DE		MoPSODS-DE
20_5_3	8.04E-01	*	8.01E-01	*	8.13E-01	*	7.99E-01	*	8.28E-01	*	**7.98E-01**	-	8.23E-01
20_10_3	7.77E-01	*	7.30E-01	*	8.30E-01	+	**7.08E-01**	*	7.28E-01	*	8.18E-01	+	7.38E-01
20_20_3	8.67E-01	*	**8.55E-01**	*	8.62E-01	*	8.69E-01	*	8.62E-01	*	8.88E-01	-	8.80E-01
50_5_3	9.04E-01	*	8.94E-01	*	8.88E-01	*	**8.41E-01**	-	8.94E-01	*	8.65E-01	*	9.07E-01
50_10_3	8.53E-01	*	8.45E-01	*	9.00E-01	+	8.82E-01	*	8.91E-01	+	9.16E-01	+	**8.43E-01**
50_20_3	**8.90E-01**	*	9.09E-01	*	9.02E-01	*	9.20E-01	*	9.11E-01	*	9.44E-01	+	9.52E-01
100_5_3	9.16E-01	+	9.05E-01	*	9.15E-01	+	9.14E-01	+	9.29E-01	+	9.52E-01	+	**9.04E-01**
100_10_3	9.09E-01	*	9.08E-01	*	9.22E-01	+	9.24E-01	+	9.29E-01	+	9.48E-01	+	**9.07E-01**
100_20_3	9.28E-01	*	**9.02E-01**	*	9.21E-01	*	9.25E-01	*	9.23E-01	*	9.58E-01	+	9.36E-01
200_10_3	9.39E-01	+	9.52E-01	+	9.21E-01	+	9.42E-01	+	9.50E-01	+	9.72E-01	+	**8.88E-01**
200_20_3	9.34E-01	+	9.33E-01	+	9.31E-01	+	9.08E-01	+	9.42E-01	+	9.72E-01	+	**8.95E-01**
500_20_3	9.58E-01	+	9.53E-01	+	9.43E-01	+	9.24E-01	*	9.62E-01	+	9.78E-01	+	**9.20E-01**
+/-/*	4/0/8		3/0/9		7/0/5		4/1/7		6/0/6		9/2/1		

Besides, in order to clearly analyze the significant difference between the MoPSODS-DE and the comparison algorithms on all 12 problems, the experiment conducted the multi-problem Wilcoxon signed rank test [68], and the experimental results are listed in Table 9. Further, we set the significance level α equal to 0.05. Among them, "$w/t/l$" means that MoPSODS-DE is significantly better than, similar to and worse than comparison algorithms. R+ is the number of times the main algorithm was better than the comparison algorithms in each test, and $R-$ is the opposite. The *p-value* denotes the importance of the result of a test. The smaller the *p-value*, the greater the significance of the result.

From the experimental results in Table 9, it is clear in terms of the four indexes of *HV*, *GD*, *IGD*, and *Spread*, that all of MoPSODS-DE were significantly better than the compared algorithms, and they also had an absolute advantage in R+. The same result is also reflected in the significant difference α. It can be seen from these results that MoPSODS-DE has achieved excellent overall performance, and in terms of distribution performance, MoPSODS-DE also performed very well.

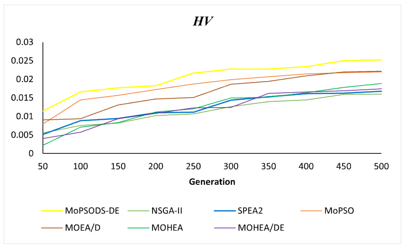
The C index is an index that measures the convergence of algorithm performance [20]. $C(S_1, S_2)$ is the percent of the individuals in S_2 who are weakly dominated by S_1. The larger $C(S_1, S_2)$ is, the better S_1 outperforms S_2 in C. In Table 10, the C index results show that MoPSODS-DE is definitely better than other comparison algorithms in 11 problems. The C index shows that MoPSODS-DE is well realized in terms of the convergence and distribution of the solution to the problem.

Finally, due to space limitations, only the convergence speed comparison of all algorithms for large-scale problems with more than 200 jobs is listed. Figure 14 shows the values of the comprehensive evaluation indexes *HV* and *IGD* obtained by the algorithm under different iterations. It can be seen from the Figure 14 that the MoPSODS-DE algorithm has the fastest convergence speed, and the convergence speed of MoPSO and MOEA/D is also a little faster than other algorithms. At the same time, it can be observed that the index values obtained by MoPSODS-DE are better. It fully illustrates the convergence performance and distribution performance of MoPSODS-DE due to all other compared algorithms.

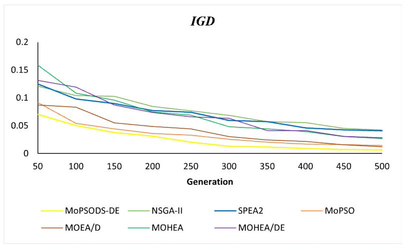
Table 9. Statistical results of MoPSO-DS and comparison algorithms.

	w/t/l	R+	R-	p-value	h (α = 0.05)
HV					
MoPSODS-DE VS NSGA-II	12/0/0	331	29	8.85E-10	1
MoPSODS-DE VS SPEA2	11/0/1	320	40	3.62E-08	1
MoPSODS-DE VS MoPSO	12/0/0	360	0	7.53E-65	1
MoPSODS-DE VS MOEA/D	12/0/0	339	21	1.50E-06	1
MoPSODS-DE VS MOHEA	11/0/1	328	32	4.27E-08	1
MoPSODS-DE VS MOHEA/DE	12/0/0	357	3	3.45E-23	1
GD					
MoPSODS-DE VS NSGA-II	12/0/0	331	29	7.04E-84	1
MoPSODS-DE VS SPEA2	11/0/1	322	38	2.37E-76	1
MoPSODS-DE VS MoPSO	12/0/0	360	0	4.43E-114	1
MoPSODS-DE VS MOEA/D	9/0/3	266	94	1.41E-29	1
MoPSODS-DE VS MOHEA	9/0/3	303	57	8.58E-61	1
MoPSODS-DE VS MOHEA/DE	12/0/0	326	34	2.35E-89	1
IGD					
MoPSODS-DE VS NSGA-II	12/0/0	336	24	1.19E-87	1
MoPSODS-DE VS SPEA2	11/0/1	327	33	1.22E-79	1
MoPSODS-DE VS MoPSO	12/0/0	360	0	3.02E-119	1
MoPSODS-DE VS MOEA/D	12/0/0	336	24	3.09E-72	1
MoPSODS-DE VS MOHEA	10/0/2	329	31	4.54E-74	1
MoPSODS-DE VS MOHEA/DE	12/0/0	359	1	9.14E-117	1
Spread					
MoPSODS-DE VS NSGA-II	4/0/8	203	157	3.80E-03	1
MoPSODS-DE VS SPEA2	3/0/9	192	169	9.33E-02	0
MoPSODS-DE VS MoPSO	7/0/5	231	129	1.35E-07	1
MoPSODS-DE VS MOEA/D	4/1/7	193	167	1.27E-01	0
MoPSODS-DE VS MOHEA	6/0/6	216	144	2.58E-05	1
MoPSODS-DE VS MOHEA/DE	9/2/1	284	76	2.47E-52	1

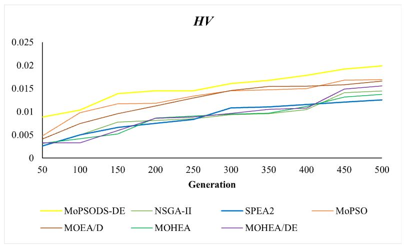
In conclusion, from the indicators of *HV*, *GD*, and *IGD*, MoPSODS-DE has obvious advantages, indicating that MoPSODS-DE has the best convergence performance and distribution performance. From the *Spread* index, it can be seen that MoPSODS-DE outperforms other algorithms in large-scale problems, and the solutions generated by MoPSODS-DE are more diverse and widely distributed. Besides, MoPSODS-DE generates solutions that outperform other algorithms on all problems in terms of *C* index. MoPSODS-DE uses the grouping strategy of enhanced direction search to make the particle swarm rapidly move to the PF in multiple directions, thereby accelerating the convergence of the particle swarm; at the same time, MoPSODS-DE uses the combination of exchange sequence, crossover and mutation to update the particles, which further improves the search ability of the algorithm and the diversity of the particle swarm. In addition, the algorithm uses the DE search strategy to perform local search on the particle swarm, which improves the quality of the solution and further enhances the convergence performance and distribution performance of the algorithm. Therefore, it can be proved that the MoPSODS algorithm combining the grouping operation, the exchange sequence operation, the crossover operation and the mutation operation performs well in the convergence performance and distribution performance, and with the addition of DE search, the MoPSODS-DE algorithm performs the best, which is superior to other compare algorithms.



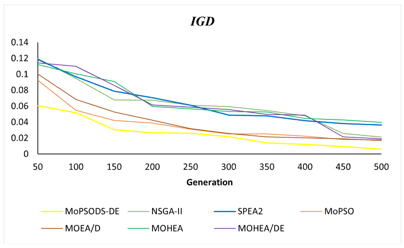
(a) 200-10-3



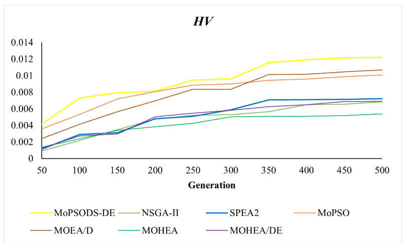
(b) 200-10-3



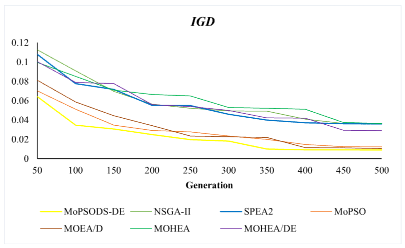
(c) 200-20-3



(d) 200-20-3



(e) 500-20-3



(f) 500-20-3

Figure 14. Convergence speed of all algorithms on large scale problems.

6. Conclusions

This study proposes the MoPSODS-DE to solve the DFSP. The objective is to minimize the makespan and total processing time. The MoPSODS-DE uses a combination of two strategies. One is the enhanced direction search strategy, which divides the particle swarm into different sub-particle swarms according to the upper area of the PF, the central area of the PF, and the lower area of the PF, so that individuals in the particle swarm can be in multiple directions on the PF at the same time. This strategy can specifically enhance the strong convergence ability of particle swarm optimization in different areas of PF, and improve the convergence of the algorithm. The second is the differential evolution search strategy, which can prevent MoPSODS from converging prematurely and avoid getting stuck in local optimum to find better solutions. Furthermore, an encoding-decoding method

Table 10. The results of *C* index.

A:MOPSO-DS, B:NSGA-II, C:SPEA2, D:MOPSO, E:MOEA/D, F:MOHEA, G:MOHEA/DE												
problem set	*C*(A,B)	*C*(B,A)	*C*(A,C)	*C*(C,A)	*C*(A,D)	*C*(D,A)	*C*(A,E)	*C*(E,A)	*C*(A,F)	*C*(F,A)	*C*(A,G)	*C*(G,A)
20_5_3	**4.00E-01**	0.00E+00	**3.00E-01**	0.00E+00	**1.00E+00**	0.00E+00	**3.00E-01**	3.33E-02	**1.33E-01**	0.00E+00	**8.67E-01**	0.00E+00
20_10_3	**3.33E-01**	6.67E-02	**1.67E-01**	6.67E-02	**1.00E+00**	0.00E+00	**2.67E-01**	3.33E-02	1.00E-01	1.00E-01	**8.00E-01**	0.00E+00
20_20_3	**2.33E-01**	0.00E+00	**1.00E-01**	3.33E-02	**1.00E+00**	0.00E+00	**1.00E-01**	0.00E+00	**6.67E-02**	0.00E+00	**7.33E-01**	0.00E+00
50_5_3	**9.00E-01**	0.00E+00	**9.00E-01**	0.00E+00	**1.00E+00**	0.00E+00	**3.67E-01**	0.00E+00	**9.33E-01**	0.00E+00	**1.00E+00**	0.00E+00
50_10_3	**7.00E-01**	0.00E+00	**7.33E-01**	0.00E+00	**1.00E+00**	0.00E+00	**3.00E-01**	0.00E+00	**8.33E-01**	0.00E+00	**1.00E+00**	0.00E+00
50_20_3	**8.33E-01**	0.00E+00	**7.00E-01**	3.33E-02	**1.00E+00**	0.00E+00	**4.00E-01**	0.00E+00	**7.67E-01**	0.00E+00	**1.00E+00**	0.00E+00
100_5_3	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**6.00E-01**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00
100_10_3	**1.00E+00**	0.00E+00	**9.67E-01**	0.00E+00	**1.00E+00**	0.00E+00	**5.33E-01**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00
100_20_3	**1.00E+00**	0.00E+00	**9.67E-01**	3.33E-02	**1.00E+00**	0.00E+00	**4.67E-01**	0.00E+00	**9.33E-01**	0.00E+00	**1.00E+00**	0.00E+00
200_10_3	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**4.67E-01**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00
200_20_3	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**3.67E-01**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00
500_20_3	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**1.67E-01**	0.00E+00	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00

based on two-vector representation is designed and uses exchange sequences, crossover, and mutation to operate on particles reasonably efficiently. On 12 benchmark problems, MoPSODS-DE is compared with other evolutionary algorithms, and these comparison algorithms are optimized using DOE, and the best parameter combination of each algorithm is obtained. The evaluation indicators of the evaluation algorithm in this study are *HV*, *GD*, *IGD*, *Spread*, and *C*. The proposed MoPSODS-DE has good distribution performance, convergence performance, and can maintain the diversity of the particle swarm.

The MoPSODS-DE also has shortcoming, that is, the CPU running time is not the shortest. The main computational time overhead is due to the additional computational time required for DE search, but experimental analysis shows that DE search can improve the convergence performance and diversity performance of MoPSODS-DE. Therefore, how to reduce the computation time of the optimization algorithm is a problem worth studying in the future. In addition, other extended problems related to the DFSP, such as more practical problems such as green energy and job assembly, are also worth investigating.

Acknowledgments

This research work is supported by the National Natural Science Foundation of China (62172141, 61772173, U1904107), Innovative Funds Plan of Henan University of Technology (2020ZKCJ02), Science and Technology Research Project of Henan Province (212102210357), Innovative Research Team (in Science and Technology) in University of Henan Province (21IRTSTHN018), Zhengzhou Science and Technology Collaborative Innovation Project (21ZZXTCX19), and the Grant-in-Aid for Scientific Research (C) of Japan Society of Promotion of Science (19K12148).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, *Comput. Oper. Res.*, **35** (2008), 3202–3212. https://doi.org/10.1016/j.cor.2007.02.014
2. Z. Shao, D. Pi, W. Shao, Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment, *Expert Syst. Appl.*, **145** (2020), 113147. https://doi.org/10.1016/j.eswa.2019.113147
3. J. Behnamian, S. Fatemi Ghomi, A survey of multi-factory scheduling, *J. Intell. Manuf.*, **27** (2016), 231–249. https://doi.org/10.1007/s10845-014-0890-y
4. M. Yazdani, S. Gohari, B. Naderi, Multi-factory parallel machine problems: Improved mathematical models and artificial bee colony algorithm, *Comput. Ind. Eng.*, **81** (2015), 36–45. https://doi.org/10.1016/j.cie.2014.12.023
5. C. Lu, Y. Huang, L. Meng, L. Gao, B. Zhang, J. Zhou, A pareto-based collaborative multi-objective optimization algorithm for energy-efficient scheduling of distributed permu-

tation flow-shop with limited buffers, *Rob. Comput. Integr. Manuf.*, **74** (2022), 102277. https://doi.org/10.1016/j.rcim.2021.102277

6. T. Meng, Q. K. Pan, L. Wang, A distributed permutation flowshop scheduling problem with the customer order constraint, *Knowl.-Based Syst.*, **184** (2019), 104894. https://doi.org/10.1016/j.knosys.2019.104894
7. S. Hatami, R. Ruiz, C. Andres-Romano, The distributed assembly permutation flowshop scheduling problem, *Int. J. Prod. Res.*, **51** (2013), 5292–5308. https://doi.org/10.1080/00207543.2013.807955
8. F. Xiong, M. Chu, Z. Li, Y. Du, L. Wang, Just-in-time scheduling for a distributed concrete precast flow shop system, *Comput. Oper. Res.*, **129** (2021), 105204. https://doi.org/10.1016/j.cor.2020.105204
9. M. Gen, R. Cheng, L. Lin, *Network models and optimization: Multiobjective genetic algorithm approach*, Springer Science & Business Media, 2008.
10. B. Jiao, S. Yan, A niche sharing scheme-based co-evolutionary particle swarm optimization algorithm for flow shop scheduling problem, *Syst. Cybernet. Inf.*, **15** (2017), 46–54.
11. X. Yu, M. Gen, *Introduction to Evolutionary Algorithms*, Springer Science & Business Media, 2010.
12. J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, *Int. J. Prod. Res.*, **51** (2013), 641–651. https://doi.org/10.1080/00207543.2011.644819
13. S. y. Wang, L. Wang, M. Liu, Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, *Int. J. Prod. Econ.*, **145** (2013), 387–396. https://doi.org/10.1016/j.ijpe.2013.05.004
14. Y. Xu, L. Wang, S. Wang, M. Liu, An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem, *Eng. Optim.*, **46** (2014), 1269–1283. https://doi.org/10.1080/0305215X.2013.827673
15. B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, *Eur. J. Oper. Res.*, **239** (2014), 323–334. https://doi.org/10.1016/j.ejor.2014.05.024
16. H. Bargaoui, O. B. Driss, K. Ghédira, A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion, *Comput. Ind. Eng.*, **111** (2017), 239–250. https://doi.org/10.1016/j.cie.2017.07.020
17. R. Ruiz, Q. K. Pan, B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem, *Omega*, **83** (2019), 213–222. https://doi.org/10.1016/j.omega.2018.03.004
18. A. Kaveh, T. Bakhshpoori, Artificial bee colony algorithm, in *Metaheuristics: Outlines, MATLAB Codes and Examples*, Springer, 2019, 19–30. https://doi.org/10.1007/978-3-030-04067-3_3
19. F. Zhao, H. Liu, Y. Zhang, W. Ma, C. Zhang, A discrete water wave optimization algorithm for no-wait flow shop scheduling problem, *Expert Syst. Appl.*, **91** (2018), 347–363. https://doi.org/10.1016/j.eswa.2017.09.028

20. J. f. Chen, L. Wang, Z. p. Peng, A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling, *Swarm Evol. Comput.*, **50** (2019), 100557. https://doi.org/10.1016/j.swevo.2019.100557

21. V. Fernandez-Viagas, P. Perez-Gonzalez, J. M. Framinan, The distributed permutation flow shop to minimise the total flowtime, *Comput. Ind. Eng.*, **118** (2018), 464–477. https://doi.org/10.1016/j.cie.2018.03.014

22. W. Shao, Z. Shao, D. Pi, Multi-objective evolutionary algorithm based on multiple neighborhoods local search for multi-objective distributed hybrid flow shop scheduling problem, *Expert Syst. Appl.*, **183** (2021), 115453. https://doi.org/10.1016/j.eswa.2021.115453

23. Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J. C. Hernandez, R. G. Harley, Particle swarm optimization: basic concepts, variants and applications in power systems, *IEEE Trans. Evol. Comput.*, **12** (2008), 171–195.

24. J. Kennedy, R. C. Eberhart, A discrete binary version of the particle swarm algorithm, in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, **5** (1997), 4104–4108. https://doi.org/10.1109/ICSMC.1997.637339

25. T. Jamrus, C. F. Chien, M. Gen, K. Sethanan, Hybrid particle swarm optimization combined with genetic operators for flexible job-shop scheduling under uncertain processing time for semiconductor manufacturing, *IEEE Trans. Semicond. Manuf.*, **31** (2017), 32–41. https://doi.org/10.1109/TSM.2017.2758380

26. T. Jamrus, C. F. Chien, M. Gen, K. Sethanan, Multistage production distribution under uncertain demands with integrated discrete particle swarm optimization and extended priority-based hybrid genetic algorithm, *Fuzzy Optim. Decis. Making*, **14** (2015), 265–287. https://doi.org/10.1007/s10700-014-9200-6

27. C. Moon*, J. Kim, M. Gen, Advanced planning and scheduling based on precedence and resource constraints for e-plant chains, *Int. J. Prod. Res.*, **42** (2004), 2941–2955. https://doi.org/10.1080/00207540410001691956

28. A. Okamoto, M. Gen, M. Sugawara, Integrated data structure and scheduling approach for manufacturing and transportation using hybrid genetic algorithm, *J. Intell. Manuf.*, **17** (2006), 411–421. https://doi.org/10.1007/s10845-005-0014-9

29. R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, *J. Global Optim.*, **11** (1997), 341–359.

30. E. Jiang, L. Wang, J. Wang, Decomposition-based multi-objective optimization for energy-aware distributed hybrid flow shop scheduling with multiprocessor tasks, *Tsinghua Sci. Technol.* , **26** (2021), 646–663. https://doi.org/10.26599/TST.2021.9010007

31. C. A. C. Coello, G. T. Pulido, M. S. Lechuga, Handling multiple objectives with particle swarm optimization, *IEEE Trans. Evol. Comput.*, **8** (2004), 256–279. https://doi.org/10.1109/TEVC.2004.826067

32. B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, *Comput. Oper. Res.*, **37** (2010), 754–768. https://doi.org/10.1016/j.cor.2009.06.019

33. J. J. Wang, L. Wang, A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop, *IEEE Trans. Syst. Man Cybern.: Syst.*, **50** (2018), 1805–1819. https://doi.org/10.1109/TSMC.2017.2788879

34. A. P. Rifai, H. T. Nguyen, S. Z. M. Dawal, Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling, *Appl. Soft Comput.*, **40** (2016), 42–57. https://doi.org/10.1016/j.asoc.2015.11.034

35. J. Deng, L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, *Swarm Evol. Comput.*, **32** (2017), 121–131. https://doi.org/10.1016/j.swevo.2016.06.002

36. C. Lu, L. Gao, J. Yi, X. Li, Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in china, *IEEE Trans. Ind. Inf.*, **17** (2020), 6687–6696. https://doi.org/10.1109/TII.2020.3043734

37. S. W. Lin, K. C. Ying, Minimizing makespan for solving the distributed no-wait flowshop scheduling problem, *Comput. Ind. Eng.*, **99** (2016), 202–209. https://doi.org/10.1016/j.cie.2016.07.027

38. K. C. Ying, S. W. Lin, C. Y. Cheng, C. D. He, Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems, *Comput. Ind. Eng.*, **110** (2017), 413–423. https://doi.org/10.1016/j.cie.2017.06.025

39. C. Y. Cheng, K. C. Ying, H. H. Chen, H. S. Lu, Minimising makespan in distributed mixed no-idle flowshops, *Int. J. Prod. Res.*, **57** (2019), 48–60. https://doi.org/10.1080/00207543.2018.1457812

40. G. Zhang, K. Xing, Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion, *Comput. Oper. Res.*, **108** (2019), 33–43. https://doi.org/10.1016/j.cor.2019.04.002

41. F. Zhao, X. He, L. Wang, A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem, *IEEE Trans. Cybern.*, **51** (2020), 5291–5303. https://doi.org/10.1109/TCYB.2020.3025662

42. F. Zhao, L. Zhang, J. Cao, J. Tang, A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem, *Comput. Ind. Eng.*, **153** (2021), 107082. https://doi.org/10.1016/j.cie.2020.107082

43. F. Zhao, R. Ma, L. Wang, A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system, *IEEE Trans. Cybern.*, 2021. https://doi.org/10.1109/TCYB.2021.3086181

44. H. Jia, J. Y. Fuh, A. Y. Nee, Y. Zhang, Web-based multi-functional scheduling system for a distributed manufacturing environment, *Concurrent Eng.*, **10** (2002), 27–39. https://doi.org/10.1177/1063293X02010001054

45. X. Zhang, X. T. Li, M. H. Yin, An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem, *Int. J. Bio-Inspired Comput.*, **15** (2020), 113–124.

46. J. J. Wang, L. Wang, An iterated greedy algorithm for distributed hybrid flowshop scheduling problem with total tardiness minimization, in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, (2019), 350–355. https://doi.org/10.1109/COASE.2019.8842885

47. M. F. Tasgetiren, Y. C. Liang, M. Sevkli, G. Gencyilmaz, A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, *Eur. J. Oper. Res.*, **177** (2007), 1930–1947. https://doi.org/10.1016/j.ejor.2005.12.024

48. Q. K. Pan, M. F. Tasgetiren, Y. C. Liang, A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem, *Comput. Oper. Res.*, **35** (2008), 2807–2839. https://doi.org/10.1016/j.cor.2006.12.030

49. A. Rahimi-Vahed, S. Mirghorbani, A multi-objective particle swarm for a flow shop scheduling problem, *J. Comb. Optim.*, **13** (2007), 79–102. https://doi.org/10.1007/s10878-006-9015-7

50. Y. X. Su, R. Chi, Multi-objective particle swarm-differential evolution algorithm, *Neural Comput. Appl.*, **28** (2017), 407–418. https://doi.org/10.1007/s00521-015-2073-y

51. Z. Cui, J. Zhang, D. Wu, X. Cai, H. Wang, W. Zhang, et al., Hybrid many-objective particle swarm optimization algorithm for green coal production problem, *Inf. Sci.*, **518** (2020), 256–271. https://doi.org/10.1016/j.ins.2020.01.018

52. W. Zhang, W. Hou, C. Li, W. Yang, M. Gen, Multidirection update-based multiobjective particle swarm optimization for mixed no-idle flow-shop scheduling problem, *Complex Syst. Modell. Simul.*, **1** (2021), 176–197. https://doi.org/10.23919/CSMS.2021.0017

53. J. Q. Li, M. X. Song, L. Wang, P. Y. Duan, Y. Y. Han, H. Y. Sang, et al., Hybrid artificial bee colony algorithm for a parallel batching distributed flow-shop problem with deteriorating jobs, *IEEE Trans. Cybern.*, **50** (2019), 2425–2439. https://doi.org/10.1109/TCYB.2019.2943606

54. Q. K. Pan, L. Gao, L. Wang, J. Liang, X. Y. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, *Expert Syst. Appl.*, **124** (2019), 309–324. https://doi.org/10.1016/j.eswa.2019.01.062

55. J. P. Huang, Q. K. Pan, Z. H. Miao, L. Gao, Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times, *Eng. Appl. Artif. Intell.*, **97** (2021), 104016. https://doi.org/10.1016/j.engappai.2020.104016

56. M. E. Baysal, A. Sarucan, K. Büyüközkan, O. Engin, Artificial bee colony algorithm for solving multi-objective distributed fuzzy permutation flow shop problem, *J. Intell. Fuzzy Syst.*, (2022), 1–11. https://doi.org/10.3233/JIFS-219202

57. W. L. Liu, Y. J. Gong, W. N. Chen, Z. Liu, H. Wang, J. Zhang, Coordinated charging scheduling of electric vehicles: A mixed-variable differential evolution approach, *IEEE Trans. Intell. Transp. Syst.*, **21** (2019), 5094–5109. https://doi.org/10.1109/TITS.2019.2948596

58. S. Zhou, L. Xing, X. Zheng, N. Du, L. Wang, Q. Zhang, A self-adaptive differential evolution algorithm for scheduling a single batch-processing machine with arbitrary job sizes and release times, *IEEE Trans. Cybern.*, **51** (2019), 1430–1442. https://doi.org/10.1109/TCYB.2019.2939219

59. W. Zhang, Y. Wang, Y. Yang, M. Gen, Hybrid multiobjective evolutionary algorithm based on differential evolution for flow shop scheduling problems, *Comput. Ind. Eng.*, **130** (2019), 661–670. https://doi.org/10.1016/j.cie.2019.03.019

60. Y. Y. Han, D. Gong, X. Sun, A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking, *Eng. Optim.*, **47** (2015), 927–946. https://doi.org/10.1080/0305215X.2014.928817

61. W. Zhang, M. Gen, J. Jo, Hybrid sampling strategy-based multiobjective evolutionary algorithm for process planning and scheduling problem, *J. Intell. Manuf.*, **25** (2014), 881–897. https://doi.org/10.1007/s10845-013-0814-2

62. Y. Li, C. Wang, L. Gao, Y. Song, X. Li, An improved simulated annealing algorithm based on residual network for permutation flow shop scheduling, *Complex Intell. Syst.*, **7** (2021), 1173–1183. https://doi.org/10.1007/s40747-020-00205-9

63. M. Baioletti, A. Milani, V. Santucci, Algebraic particle swarm optimization for the permutations search space, in *2017 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, (2017), 1587–1594. https://doi.org/10.1109/CEC.2017.7969492

64. E. Taillard, Benchmarks for basic scheduling problems, *Eur. J. Oper. Res.*, **64** (1993), 278–285. https://doi.org/10.1016/0377-2217(93)90182-M

65. A. P. Rifai, S. T. W. Mara, A. Sudiarso, Multi-objective distributed reentrant permutation flow shop scheduling with sequence-dependent setup time, *Expert Syst. Appl.*, **183** (2021), 115339. https://doi.org/10.1016/j.eswa.2021.115339

66. G. Wang, X. Li, L. Gao, P. Li, An effective multi-objective whale swarm algorithm for energy-efficient scheduling of distributed welding flow shop, *Ann. Oper. Res.*, **310** (2022), 223–255. https://doi.org/10.1007/s10479-021-03952-1

67. S. Yang, Z. Xu, The distributed assembly permutation flowshop scheduling problem with flexible assembly and batch delivery, *Int. J. Prod. Res.*, **59** (2021), 4053–4071. https://doi.org/10.1080/00207543.2020.1757174

68. J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, *Swarm Evol. Comput.*, **1** (2011), 3–18.