Swarm and Evolutionary Computation 60 (2021) 100785

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem

Zi-Qi Zhang [a,b], Bin Qian [a,b,c,*], Rong Hu [a,c], Huai-Ping Jin [a], Ling Wang [d]

[a] School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China
[b] School of Mechanical and Electrical Engineering, Kunming University of Science and Technology, Kunming 650500, China
[c] Yunnan Key Laboratory of Artificial Intelligence, Kunming University of Science and Technology, Kunming 650500, China
[d] Department of Automation, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Keywords:
Scheduling
Matrix cube
Estimation of distribution algorithm
Distributed assembly permutation flow shop
Variable neighborhood descent

ABSTRACT

The distributed assembly permutation flow-shop scheduling problem (DAPFSP) is a typical NP-hard combinatorial optimization problem that has wide applications in advanced manufacturing systems and modern supply chains. In this work, an innovative three-dimensional matrix-cube-based estimation of distribution algorithm (MCEDA) is first proposed for the DAPFSP to minimize the maximum completion time. Firstly, a matrix cube is designed to learn the valuable information from elites. Secondly, a matrix-cube-based probabilistic model with an effective sampling mechanism is developed to estimate the probability distribution of superior solutions and to perform the global exploration for finding promising regions. Thirdly, a problem-dependent variable neighborhood descent method is proposed to perform the local exploitation around these promising regions, and several speedup strategies for evaluating neighboring solutions are utilized to enhance the computational efficiency. Furthermore, the influence of the parameters setting is analyzed by using design-of-experiment technique, and the suitable parameters are suggested for different scale problems. Finally, a comprehensive computational campaign against the state-of-the-art algorithms in the literature, together with statistical analyses, demonstrates that the proposed MCEDA produces better results than the existing algorithms by a significant margin. Moreover, the new best-known solutions for 214 instances are improved.

1. Introduction

The permutation flow-shop scheduling problem (PFSP) is one of the most extensively investigated combination optimization problems and widely exists in machinery factories, assembly lines, information service facilities, and petrochemical systems [1–4]. According to the research work of Garey et al. [5] and Gonzalez and Sahni [6], the PFSP is NP-hard in the strong sense. Due to its significance in both theory and practice, the PFSP has been studied by many researchers. The traditional PFSP assumes that all jobs have to be scheduled in a single factory. However, with the deepening of the global economy, many factories are now more actively involved in international and regional cooperation. As a consequence, the production scheduling in multi-factory mode, referred to as the distributed scheduling problem. Among all types of distributed scheduling problems, the distributed PFSP (DPFSP) is a generalization of the traditional PFSP. The DPFSP addressing both the allocation of jobs among factories and the scheduling of jobs in each factory has gained an increasing attention in the literature. Naderi and Ruiz [7] first introduced the DPFSP and presented six mixed integer linear programming (MILP) models for it. They also proposed 14 constructive heuristics and fast local search methods based on variable neighborhood descent (VND) to minimize the makespan of the problem. Afterward, some researchers proposed different algorithms for the DPFSP. These algorithms include one tabu search [8], three estimation of distribution algorithm [9–11], and three iterated greedy (IG) algorithms [12–14]. As for the multi-objective DPFSP, an adaptive large neighborhood search algorithm [15] and a competitive memetic algorithm [16] were developed.

In the past few decades, assembly systems have been widely applied in the flow-oriented manufacturing industry. The distributed assembly permutation flow-shop scheduling problem (DAPFSP) is a classical model where the components of a product are produced independently on flow-shop-type processing lines in the first stage and then all these components are assembled on one assembly machine in the second stage. In real world, many scheduling problems in multi-factory manufacturing companies can be modeled as DAPFSP. Typical examples can be found in automotive engine companies, personal computer manufacturing companies, and smartphone manufacturing companies. Since

* Corresponding authors.
E-mail address: bin.qian@vip.163.com (B. Qian).

https://doi.org/10.1016/j.swevo.2020.100785
Received 26 February 2020; Received in revised form 5 September 2020; Accepted 22 September 2020
Available online 1 October 2020

the PFSP is already strongly NP-hard and it reduces to the DAPFSP, the latter problem is also NP-hard.

Nowadays, the DAPFSP has received increasing attention from researchers due to its practical application and theoretical complexity. Since 2013, three types of the DAPFSP have been proposed in the literature. The first is the DAPFSP with makespan criterion. The existing research mainly focused on this problem. Hatami et al. [17] first considered this problem and generated 900 small-scaled instances and 810 large-scaled instances to evaluate the compared algorithms. They proposed three constructive algorithms and a fast variable neighborhood descent (VND) algorithm for solving it. Computational results indicated that the VND algorithm is quite effective and can offer good performance. Li et al. [18] developed a genetic algorithm (GA) by adopting an enhanced crossover operator and three local search strategies. Test results showed that the proposed GA can obtain good performance. Wang and Wang [19] presented an estimation of distribution algorithm-based memetic algorithm (EDAMA), which used a two-dimensional probabilistic model to guide exploration and designed a critical-path-based local search to perform exploitation. Experimental results demonstrated that EDAMA can update the best-known solutions of 181 instances. Lin and Zhang [20] designed a hybrid biogeography-based optimization (HBBO) algorithm that integrated the path-relinking based heuristic, the insertion-based heuristic, and the problem-based local search. Computational results showed that HBBO can find the best-known solutions of 181 instances. Lin et al. [21] presented a backtracking search hyper-heuristic (BS-HH) algorithm where the backtracking search algorithm was employed as the high-level heuristic strategy to manipulate a set of low-level heuristics (LLHs) to search solution space. Computational results indicated that the presented BS-HH can achieve better performance than the state-of-the-art algorithms. In addition, Hatami et al. [22] further generalized the first type of problem by adding sequence-dependent setup times and presented two constructive heuristics and two *meta*-heuristics (i.e., VND and IG) for solving it. Test results showed that IG_3 (one variant of IG) has better performance. Most recently, Pan et al. [23] considered an extended DAPFSP, in which each factory has its own assembly machine. They proposed three constructive heuristics, two variable neighborhood search methods, and an iterated greedy algorithm (IGA). Computational comparisons demonstrated that their proposed IGA outperforms the existing algorithms by a significant margin. The second is the DAPFSP with the total flowtime criterion. Sang et al. [24] presented a two-level discrete invasive weed optimization (TDIWO), a discrete invasive weed optimization with hybrid search operators (HDIWO), and a HDIWO with selection probability (HDIWOp) for this problem. Test results verified that the presented algorithms perform better than the other algorithms in the literature and HDIWO is the best one. The third is the DAPFSP with the total tardiness criterion. Yang et al. [25] proposed a scatter-search-based memetic algorithm (SS-MA), in which a subset generation mechanism and a solution combination method were designed to execute global search, and an enhanced scatter search was devised to perform local search. Experimental results showed that the proposed SS-MA outperforms several NEH-based heuristics. In the tests of the above algorithms, GA, EDAMA, HBBO, BS-HH and SS-MA used both the small-scaled and the large-scaled instances generated by Hatami et al. [17], IGA, TDIWO, HDIWO and HDIWOp only used these large-scaled instances, and the other algorithms used their own randomly generated instances. From the above literature review, it is clear that the DAPFSP with makespan criterion has been becoming a research hotspot in recent years. Thus, it is meaningful to design a new algorithm that can solve this problem more effectively.

Traditional mathematical algorithms use linear algebra and geometric analysis as basic tools. They try to utilize the structural information of the objective function and constraints of mixed integer programming model to narrow the search area, and execute the search that traverses or partially traverses solution space [26]. When the problem is NP-hard, the relationship between its inner geometric structure and the optimal solution is still an open problem [27,28]. As a consequence, for large-scaled or complex scheduling problems, mathematical algorithms are often of limited use due to their excessive computation time or poor results under reasonable computation time. To settle this issue, *meta*-heuristic algorithms based on permutation-based model have been presented. They do not depend on the structure of the model, but perform the search by using certain evolutionary mechanisms. This kind of algorithms can often obtain satisfactory solutions of various complex scheduling problems within seconds or tens of seconds. This is why almost all existing algorithms for solving the DAPFSP are *meta*-heuristic algorithms (see the pervious paragraph). So, it is necessary to design a *meta*-heuristic algorithm to deal with the DAPFSP.

As a novel *meta*-heuristic algorithm based on statistical learning technique, the estimation of distribution algorithm (EDA) establishes a probabilistic model to estimate the distribution of the promising or excellent solutions and generates new individuals through sampling from this model. Unlike the traditional evolutionary algorithms, the evolutionary mechanism of EDA can avoid the destruction of the blocks (i.e., the partial ordered patterns) in promising solutions [29]. Due to its outstanding global exploration, inherent parallelism and fast convergence, EDA has led to increasing studies and extensive applications during recent years [30]. In the field of intelligent production scheduling, EDA-based algorithms have been designed to address a variety of shop scheduling problems, such as the hybrid FSP [31], the PFSP [32], the lot-streaming FSP [33], the flexible job-shop scheduling problem (FJSP) [34], the DPFSP [9], the multi-objective PFSP [35], and the DAPFSP [19]. Hence, in this study, we select EDA as the basic framework to construct an effective algorithm for the DAPFSP.

The above currently existing EDA-based algorithms always tried to use a two-dimensional model to save the information of the blocks and the order of jobs from each promising solution or individual. Here one block consists of any two consecutive jobs in a solution. Nevertheless, owing to the limit of its own structure, a two-dimensional model cannot accurately record the position information of each block and the whole order information of jobs, which causes that the sampling procedure may misplace the blocks in new individuals. As a result, the search direction cannot be reasonably guided. To overcome the drawback of the existing two-dimensional models and solve the DAPFSP with makespan criterion more effectively, we develop a matrix-cube-based EDA (MCEDA) in this paper. The main features of MCEDA lie in two aspects: the global search based on the three-dimensional probabilistic model and the local search boosted by the speed-up strategies. As for the global search, a three-dimensional probabilistic model is built to save the valuable information of excellent individuals, and the new individuals are generated by sampling this model via a special sampling method. As for the local search, a VND framework coupled with rich neighborhood operators is constructed to execute exploitation, and two speed-up strategies based on the problem's property are utilized to boost search efficiency.

The remainder of this paper is organized as follows. The DAPFSP is described in the next section. The proposed MCEDA is elaborated in Section 3. Numerical results and comparisons are analyzed in Section 4. Finally, some concluding remarks and future research directions are provided in Section 5.

2. Problem description

As illustrated in Fig. 1, the considered DAPFSP contains two stages, i.e., a production stage and an assembly stage. The production stage consists of a series of production factories, and each of them can be regarded as a flowshop configuration. All operations of jobs are performed by the same route in any of the assigned factories. The assembly stage can be executed after all parts of each product have been finished and the single assembly machine is free.

The DAPFSP can be described as follows. There is a set of S products $\{P_1,P_2,...,P_S\}$ and each product is produced by assembling a set of its parts in terms of bill of material. The production stage consists of

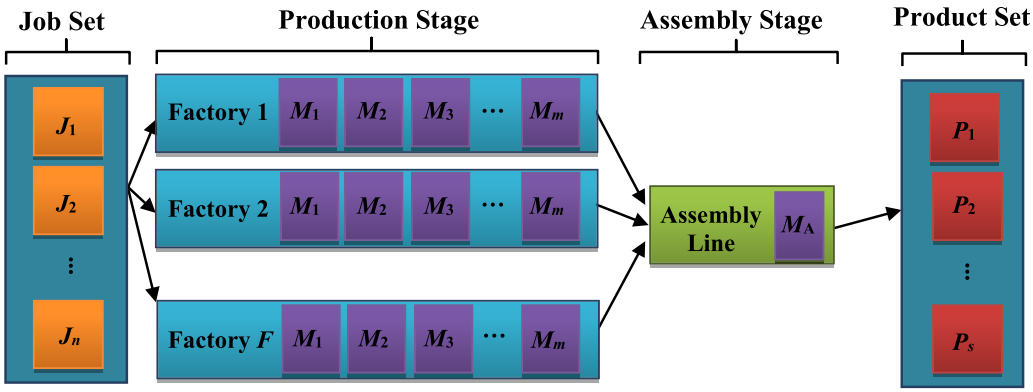


Fig. 1. Illustration of the distributed assembly flowshop.

Table 1
Notations applied in the model of DAPFSP.

Parameters	
N	Total number of jobs
m	Total number of machines
F	Total number of factories
S	Total number of products
$p_m(i,j)$	The processing time of operation $O_{i,j}$ on machine M_j
ω_h	The number of jobs belongs to product P_h
Π	The set of all feasible schedule
Indices	
i	Index for jobs where $i = 1,\ldots,n$
j	Index for machines where $j = 1,\ldots,m$
h	Index for products where $h = 1,\ldots,S$
f	Index for factories where $f = 1,\ldots,F$
Variables	
n_f	The total number of jobs assigned to factory f
$\boldsymbol{\pi}$	The total sequence of jobs
$\boldsymbol{\pi_f}$	The subsequence of job sets in factory f
$\boldsymbol{\pi_p}$	The sequence of assembled products
$C(i,j)$	The completion time of operation $O_{i,j}$ on machine M_j
λ	The product sequence $[\lambda(1),\lambda(2),\ldots,\lambda(S)]$
$p_a(h)$	The assembling time of product $\lambda(h)$ on machine M_A
$C_P(P_h)$	The latest completion time of jobs which belong to product P_h
$C_P(h)$	The latest completion time of jobs which belong to product $\lambda(h)$
$C_A(h)$	The completion time of assembling product $\lambda(h)$
$C_{\max}(\boldsymbol{\pi})$	The makespan for the schedule $\boldsymbol{\pi}$

a set F of f distributed factories that are responsible for processing the set of n jobs on the set of m machines and each factory has the same capacity for production. Each of these factories is regarded as the flowshop with a series of machines $M = [M_1, M_2, \ldots, M_m]$ ($|M_k| \geq 2$ for all factories and $|\bullet|$ is the cardinality of a set). All jobs must be assigned to any of the factories and processed in the same route, i.e., first on machine M_1, then on machine M_2, and so on until machine M_m. A series of operations $\{O_{i,1},O_{i,2},\ldots,O_{i,m}\}$ of job i are executed one after another and $O_{i,j}$ is performed on machine M_j during a period of processing time $p_m(i,j)$ without interruption. All operations of each job must be completed in the assigned factory which cannot be transferred to another factory. Each product P_h consists of ω_h jobs and each job only belongs to one product, i.e., $\sum_{h=1}^{S} \omega_h = n$. Once all parts of product P_h are finished, they are immediately delivered to an assembly machine M_A. At the assembly stage, all jobs are assembled to obtain a set of final products by a defined program. The assembly time that a product P_h requires to be assembled in M_A is referred to as $p_a(h)$ and the assembly operation of P_h cannot be started until its all parts have arrived and the assembly machine has released. For the convenience and readability, the detailed notation description including indices, parameters and variables are shown in Table 1. Additionally, a number of additional assumptions of PFSP are also met. Machines are always available and no breakdowns are considered and all the jobs are independent and available at time 0. The processing times are deterministic and the setup times and transportation times are considered to be included in processing times. At any time, each machine is capable of processing no more than one job and each job can only be processed on at most one machine and preemption is forbidden, i.e., each job should be processed sequentially on all machines as no intermingling is allowed. The goal of the DAPFSP is to determine the assignment of jobs to factories, the sequence of jobs in factories and the order of products in the assembly stage so that the maximum completion time (i.e., makespan) of the entire manufacturing process is minimized.

At the production stage, all n jobs assigned to F different factories can be represented as a set of subsequences $[\boldsymbol{\pi_1},\boldsymbol{\pi_2},\ldots,\boldsymbol{\pi_F}]$ where $\boldsymbol{\pi_f} = [\pi_f(1), \pi_f(2), \ldots, \pi_f(n_f)]$ is the sequence of jobs in factory f and $\sum_{f=1}^{F} n_f = n$. For a schedule $\boldsymbol{\pi}$ of the DAPFSP, the makespan $C_{\max}(\boldsymbol{\pi})$ can be calculated as follows:

$$C\left(\pi_f(1),1\right) = p_m\left(\pi_f(1),1\right), f = 1,2,\ldots,F \tag{1}$$

$$C\left(\pi_f(i),1\right) = C\left(\pi_f(i-1),1\right) + p_m\left(\pi_f(i),1\right), \quad i = 2,3,\ldots,n_f, f = 1,2,\ldots,F \tag{2}$$

$$C\left(\pi_f(1),j\right) = C\left(\pi_f(1),j-1\right) + p_m\left(\pi_f(1),j\right), \quad j = 2,3,\ldots,m, f = 1,2,\ldots,F \tag{3}$$

Table 2
Processing times, assembly times and job-ownership constraints of the given instance.

Product	Job	Processing time M_1	M_2	M_3	M_4	M_5	Assembly time M_A	Product	Job	Processing time M_1	M_2	M_3	M_4	M_5	Assembly time M_A
P_1	J_1	29	43	65	30	84	226	P_2	J_5	39	84	49	84	82	226
	J_2	84	43	5	9	74			J_6	80	52	93	52	32	
	J_3	1	48	46	62	26			J_7	11	63	79	15	30	
	J_4	2	69	34	73	74			J_8	90	15	70	16	39	
	J_{10}	84	60	7	45	28			J_9	53	95	85	14	75	
	J_{11}	21	29	59	5	4			J_{12}	40	28	96	79	40	
	J_{14}	2	58	42	30	6			J_{13}	57	34	33	6	32	
	J_{15}	54	20	7	66	85			J_{19}	41	75	44	43	82	
	J_{16}	63	32	1	2	33			J_{20}	25	47	97	28	87	
	J_{17}	76	62	22	80	91			J_{21}	63	79	73	98	37	
	J_{18}	49	90	6	59	43			J_{23}	99	29	29	99	94	
	J_{22}	59	27	27	23	27									
	J_{24}	94	4	19	21	91									

$$C\left(\pi_f(i), j\right) = max\left\{C\left(\pi_f(i-1), j\right), C\left(\pi_f(i), j-1\right)\right\} + p_m\left(\pi_f(i), j\right),$$
$$i = 2, 3, ..., n_f, j = 2, 3, ..., m, f = 1, 2, ..., F \quad (4)$$

At the assembly stage, all jobs are assembled to produce a set of S final products. The product sequence $\lambda = [\lambda(1), \lambda(2), ..., \lambda(S)]$ on the assembly machine can be determined according to the completion times of all jobs in the first stage, and then

$$C_P(h) = \max_{\pi_f(i)\in\lambda(h)} C\left(\pi_f(i), m\right),\ i=1,2,...,n_f, f=1,2,...,F \quad (5)$$

$$C_A(1) = C_P(1) + p_a(1) \quad (6)$$

$$C_A(h) = \max\{C_A(h-1), C_P(h)\} + p_a(h), h = 2, 3, ..., S. \quad (7)$$

Hence, the makespan of $\boldsymbol{\pi}$ is determined as

$$C_{\max}(\boldsymbol{\pi}) = C_A(S) \quad (8)$$

The aim of solving the DAPFSP with makespan criterion is to find a schedule $\boldsymbol{\pi}^*$ in the set of all permutations Π, such that

$$C_{\max}\left(\boldsymbol{\pi}^*\right) = \min_{\boldsymbol{\pi}\in\Pi} C_{\max}(\boldsymbol{\pi}). \quad (9)$$

Obviously, the model adopted in this section is a permutation-based model, which consists of several equations to calculate the complete time of each job or product on each machine. The optimization variables of this model are represented as a job sequence $\boldsymbol{\pi}$. Some existing studies on the *meta*-heuristics for the assembly flow-shop scheduling problems always establish the mixed-integer mathematical models. However, each individual or solution in these *meta*-heuristics actually corresponds to the decision variables of the unprovided permutation-based model. Many beginning researchers are easily confused by this phenomenon. Thus, this section only adopts a permutation-based model. If researchers are interested in the mathematical models of the DAPFSP, they can find them in [7].

The permutation-based models of production scheduling problems have no explicitly expressed constraints, which is different from the mathematical models of these problems. The operational constraints of DAPFSP's permutation-based model are implicitly included in its equations and a job-ownership table. There are two types of these constraints. The first type includes the general constraints for both the PFSP and the DPFSP. These general constraints are included in Eqs. (2)–(4), which require that each operation $O_{\pi_f(i),j}$ can only be performed after the operations $O_{\pi_f(i-1),j}$ and $O_{\pi_f(i),j-1}$ are completed. The second type includes the special job-ownership constraints for the DAPFSP. These special constraints are usually given in a job-ownership table (see Table 2), which determines the relations among jobs and products.

The constraints of the permutation-based model are easier to handle than those of the mathematical model. By using a job assignment rule, a solution or sequence $\boldsymbol{\pi}$ that contains different jobs can be decoded to a series of subsequences, each of which belongs to a corresponding factory. Then, the processing start times of the jobs on each machine of a factory are calculated through Eqs. (1)–(5), and no general constraints are violated. Moreover, since Eq. (5) utilizes the job-ownership constraints (see $\pi_f(i)\in\lambda(h)$) to calculate $C_P(h)$, no special constraints are violated. Hence, a solution $\boldsymbol{\pi}$ is feasible if and only if all the jobs in this solution are different from each other. In MCEDA, both the sampling method for generating new individuals (see Section 3.2.3) and the neighborhood searches for reaching new neighbors (see Section 3.3) can ensure that the jobs in each solution $\boldsymbol{\pi}$ (i.e., individual or neighbor) are different from each other. This means that any solution $\boldsymbol{\pi}$ obtained in MCEDA is feasible. From these analyses, it can be seen that by using the permutation-based model, researchers can focus more on algorithm design without spending too much time dealing with complex constraints. Therefore, most of the existing *meta*-heuristics choose to optimize the variables of the permutation-based model.

3. MCEDA for DAPFSP

3.1. Solution representation

The permutation-based representation has been widely used in the literature [7,19,33,23,36,37]. Therefore, regarding the DAPFSP with n jobs and F factories, a feasible solution is represented by a total job permutation or sequence $\boldsymbol{\pi} = [\pi(1), \pi(2), ..., \pi(n)]$ where the element $\pi(i)$ $(i = 1, ..., n)$ implies one job with the ith processing priority. The job assignment rule proposed by Hatami et al. [17] is adopted to convert $\boldsymbol{\pi}$ into a feasible scheduling scheme. This rule continually selects each job $\pi(i)$ in $\boldsymbol{\pi}$ according to its priority from high to low and assigns it to a factory which has the lowest makespan after including this job. Then, a set of subsequences $[\boldsymbol{\pi_1}, \boldsymbol{\pi_2}, ..., \boldsymbol{\pi_F}]$ in F factories are obtained and the makespan $C_{\max}(\boldsymbol{\pi})$ can be calculated by using Eqs. (1)–(9) and a preset λ in Section 2.

To facilitate understanding, Fig. 2 illustrates the Gantt chart of a high-quality schedule for the instance I_24_5_3_2_2 generated by Hatami et al. [17], where 24_5_3_2 represents the instance scale (n=24,m=5,F=3,S=2) and the last number 2 means that this instance is the second one of this scale. The relations among jobs and products, the processing times of jobs and the assembly times of products are provided in Table 2. As shown in Fig. 2, it is obvious that the product sequence is $\lambda = [1, 2]$ and the subsequences of jobs assigned into three factories are $\boldsymbol{\pi_1}$= [3, 18, 17, 16, 23, 12, 9], $\boldsymbol{\pi_2}$= [4, 11, 2, 15, 20, 5, 21, 6], and $\boldsymbol{\pi_3}$= [1, 14, 22, 10, 24, 19, 8, 7, 13], respectively. Moreover, it is clear from Fig. 2 that all jobs belonging to the same product are tied closely at the production stage and the assembly operation of the first product should be performed as early as possible if a smaller $C_{\max}(\boldsymbol{\pi})$ is expected. Thus, when decoding a solution of the proposed MCEDA, the products in λ are sorted in ascending order of their respective $C_P(P_h)$.

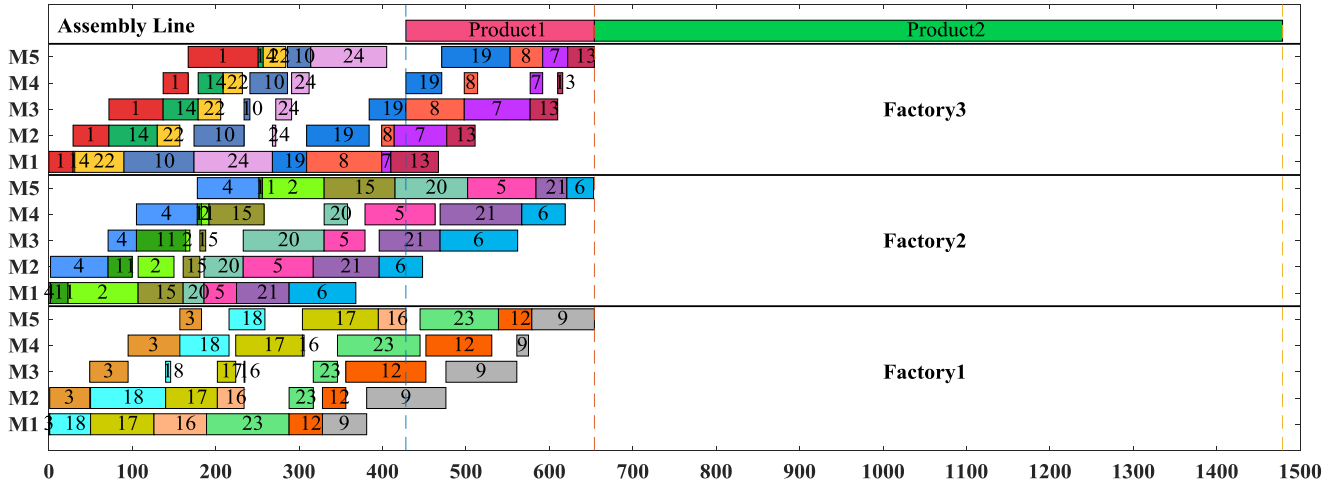


Fig. 2. A scheduling Gantt chart of instance I_24_5_3_2_2. (Makespan:1479).

3.2. Matrix-cube-based global search

A matrix-cube-based global search is proposed to find promising regions in solution space. In this section, a matrix-cube-based probabilistic model is developed to explicitly extract the distributing characteristic of the similar blocks and accurately accumulate the valuable information of superior solutions or individuals. It is expected to potentially guide the evolutionary direction toward promising regions in solution space. Then, the matrix-cube-based probabilistic model and the new population generation method are devised.

3.2.1. Matrix cube

In this subsection, we first introduce an effective data structure which is referred to as matrix cube. Then, the three-dimensional matrix cube is devised to reasonably reserve the valuable information of the order of jobs and the position of similar blocks of excellent individuals, which is essential for developing a probabilistic model. Let **Pop(gen)** be the population of MCEDA at *gen*-th $(gen = 0, ..., maxgen)$ generation, *popsize* the size of **Pop(gen)**, $\mathbf{SbestPop(gen)} = \{\boldsymbol{\pi}^{\mathbf{gen,1}}_{\mathbf{Sbest}}, \boldsymbol{\pi}^{\mathbf{gen,2}}_{\mathbf{Sbest}}, ..., \boldsymbol{\pi}^{\mathbf{gen,spsize}}_{\mathbf{Sbest}}\}$ the superior sub-population or excellent solutions selected from **Pop(gen)**, *spsize* the size of **SbestPop(gen)**, and $\boldsymbol{\pi}^{\mathbf{gen,k}}_{\mathbf{Sbest}} = [\pi^{gen,k}_{Sbest}(1), \pi^{gen,k}_{Sbest}(2), ..., \pi^{gen,k}_{Sbest}(n)]$ $(k = 1, ..., spsize)$ the *k*th individual in the **SbestPop(gen)**. Then, the details of the presented three-dimensional matrix or matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$ are given as follows:

$$N\_MC^{gen,k}_{n\times n\times n}(x,y,z) = \begin{cases} 1, \ if\ y = \pi^{gen,k}_{Sbest}(x)\ and\ z = \pi^{gen,k}_{Sbest}(x+1) \\ 0, \ else \end{cases},$$
$$x = 1, ..., n-1; y, z = 1, ..., n; k = 1, ..., spsize \quad (10)$$

$$MC^{gen}_{n\times n\times n}(x,y,z) = \sum_{k=1}^{spsize} N\_MC^{gen,k}_{n\times n\times n}(x,y,z),$$
$$x = 1, ..., n-1; y, z = 1, ..., n \quad (11)$$

$$\mathbf{MC^{gen}_{n\times n\times n}(x)} = \begin{bmatrix} \mathbf{MC^{gen}_{n\times n\times n}(x,\ 1)} \\ \vdots \\ \mathbf{MC^{gen}_{n\times n\times n}(x,\ n)} \end{bmatrix}_{n\times 1} = \begin{bmatrix} MC^{gen}_{n\times n\times n}(x,1,1) & \cdots & MC^{gen}_{n\times n\times n}(x,1,n) \\ \vdots & \ddots & \vdots \\ MC^{gen}_{n\times n\times n}(x,n,1) & \cdots & MC^{gen}_{n\times n\times n}(x,n,n) \end{bmatrix}_{n\times n} \quad (12)$$

$$\mathbf{MC^{gen}_{n\times n\times n}(y,z)} = \left[MC^{gen}_{n\times n\times n}(1,y,z), MC^{gen}_{n\times n\times n}(2,y,z), ..., MC^{gen}_{n\times n\times n}(n,y,z)\right]_{1\times n},$$
$$y, z = 1, 2, ..., n \quad (13)$$

In Eq. (10), $N\_MC^{gen,k}_{n\times n\times n}(x,y,z)$ is an indicator function that records the information of the order of jobs and the position of similar blocks for the *k*th individual in **SbestPop(gen)**. In Eq. (11), the element $MC^{gen}_{n\times n\times n}(x,y,z)$ of $\mathbf{MC^{gen}_{n\times n\times n}}$ can save the statistical information for all individuals in **SbestPop(gen)** according to the $N\_MC^{gen,k}_{n\times n\times n}(x,y,z)$ $(k = 1, ..., spsize)$. That is, the subscripts *x* and (*y*,*z*) of $MC^{gen}_{n\times n\times n}(x,y,z)$ record both the ordinal number of job (i.e., the position of $\pi^{gen,k}_{Sbest}(x)$ in $\boldsymbol{\pi}^{\mathbf{gen,k}}_{\mathbf{Sbest}}$) and the occurrence frequency of the corresponding similar block (i.e., $[\pi^{gen,k}_{Sbest}(x), \pi^{gen,k}_{Sbest}(x+1)]$), respectively. In Eq. (12), it is clear that each two-dimensional submatrix $\mathbf{MC^{gen}_{n\times n\times n}(x)}$ can learn the ordinal numbers of jobs and the distributing information of similar blocks at *x*th position of solutions in **SbestPop(gen)**. Moreover, in Eq. (13), the hierarchical structure of the proposed three-dimensional matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$ can reasonably reserve the total information of the order of jobs and the position of similar blocks by means of a series of the position-based two-dimensional submatrices, i.e., $\mathbf{MC^{gen}_{n\times n\times n}(1)}, \mathbf{MC^{gen}_{n\times n\times n}(2)}, ..., \mathbf{MC^{gen}_{n\times n\times n}(n)}$. Thus, the valuable information of the distinctive features of superior solutions at each generation can be learned and conserved in an appropriate and intuitive way. The matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$ is an important ingredient to construct an effective probabilistic model of the proposed MCEDA. In order to facilitate the understanding, a simple example with $n = 4$ and $spsize = 5$ is given as follows. In this example, five superior solutions are considered, i.e., $\boldsymbol{\pi}^{\mathbf{gen,1}}_{\mathbf{Sbest}} = [\pi^{gen,1}_{Sbest}(1), \pi^{gen,1}_{Sbest}(2), \pi^{gen,1}_{Sbest}(3), \pi^{gen,1}_{Sbest}(4)] = [1,2,3,4]$, $\boldsymbol{\pi}^{\mathbf{gen,2}}_{\mathbf{Sbest}} = [\pi^{gen,2}_{Sbest}(1), \pi^{gen,2}_{Sbest}(2), \pi^{gen,2}_{Sbest}(3), \pi^{gen,2}_{Sbest}(4)] = [2,3,1,4]$, $\boldsymbol{\pi}^{\mathbf{gen,3}}_{\mathbf{Sbest}} = [\pi^{gen,3}_{Sbest}(1), \pi^{gen,3}_{Sbest}(2), \pi^{gen,3}_{Sbest}(3), \pi^{gen,3}_{Sbest}(4)] = [3,2,1,4]$, $\boldsymbol{\pi}^{\mathbf{gen,4}}_{\mathbf{Sbest}} = [\pi^{gen,4}_{Sbest}(1), \pi^{gen,4}_{Sbest}(2), \pi^{gen,4}_{Sbest}(3), \pi^{gen,4}_{Sbest}(4)] = [4,3,2,1]$, and $\boldsymbol{\pi}^{\mathbf{gen,5}}_{\mathbf{Sbest}} = [\pi^{gen,5}_{Sbest}(1), \pi^{gen,5}_{Sbest}(2), \pi^{gen,5}_{Sbest}(3), \pi^{gen,5}_{Sbest}(4)] = [4,3,1,2]$. When $x = 1$, according to Eqs. (10) and (11), the similar blocks [1,2] (i.e., $y = 1$ and $z = 2$), [2,3] (i.e., $y = 2$ and $z = 3$), [3,2] (i.e., $y = 3$ and $z = 2$), and [4,3] (i.e., $y = 4$ and $z = 3$) at the first position of the five superior solutions can be recorded respectively. That is, we can get

$$\begin{aligned} MC^{gen}_{4\times4\times4}(1,1,2) &= \sum_{k=1}^{5} N\_MC^{gen,k}_{4\times4\times4}(1,1,2) \\ &= N\_MC^{gen,1}_{4\times4\times4}(1,1,2) + N\_MC^{gen,2}_{4\times4\times4}(1,1,2) \\ &\quad + N\_MC^{gen,3}_{4\times4\times4}(1,1,2) + N\_MC^{gen,4}_{4\times4\times4}(1,1,2) \\ &\quad + N\_MC^{gen,5}_{4\times4\times4}(1,1,2) = 1+0+0+0+0 = 1 \end{aligned}$$

$$\begin{aligned} MC^{gen}_{4\times4\times4}(1,2,3) &= \sum_{k=1}^{5} N\_MC^{gen,k}_{4\times4\times4}(1,2,3) \\ &= N\_MC^{gen,1}_{4\times4\times4}(1,2,3) + N\_MC^{gen,2}_{4\times4\times4}(1,2,3) \\ &\quad + N\_MC^{gen,3}_{4\times4\times4}(1,2,3) + N\_MC^{gen,4}_{4\times4\times4}(1,2,3) \\ &\quad + N\_MC^{gen,5}_{4\times4\times4}(1,2,3) = 0+1+0+0+0 = 1 \end{aligned}$$

$$MC^{gen}_{4\times4\times4}(1,3,2)=\sum_{k=1}^{5}N_c\_MC^{gen,k}_{4\times4\times4}(1,3,2)$$
$$=N_c\_MC^{gen,1}_{4\times4\times4}(1,3,2)+N_c\_MC^{gen,2}_{4\times4\times4}(1,3,2)$$
$$+N_c\_MC^{gen,3}_{4\times4\times4}(1,3,2)+N_c\_MC^{gen,4}_{4\times4\times4}(1,3,2)$$
$$+N_c\_MC^{gen,5}_{4\times4\times4}(1,3,2)=0+0+1+0+0=1$$

$$MC^{gen}_{4\times4\times4}(1,4,3)=\sum_{k=1}^{5}N_c\_MC^{gen,k}_{4\times4\times4}(1,4,3)$$
$$=N_c\_MC^{gen,1}_{4\times4\times4}(1,4,3)+N_c\_MC^{gen,2}_{4\times4\times4}(1,4,3)$$
$$+N_c\_MC^{gen,3}_{4\times4\times4}(1,4,3)+N_c\_MC^{gen,4}_{4\times4\times4}(1,4,3)$$
$$+N_c\_MC^{gen,5}_{4\times4\times4}(1,4,3)=0+0+0+1+1=2$$

The values of other cells in $\mathbf{MC^{gen}_{4\times4\times4}}(\mathbf{1})$ are set to 0. Similarly, when $x=2$, the four similar blocks of jobs [2,1] (i.e., $y=2$ and $z=1$), [2,3] (i.e., $y=2$ and $z=3$), [3,1] (i.e., $y=3$ and $z=1$) and [3,2] (i.e., $y=3$ and $z=2$) at the second position of these superior solutions can also be recorded respectively. Then, we have

$$MC^{gen}_{4\times4\times4}(2,2,1)=\sum_{k=1}^{5}N_c\_MC^{gen,k}_{4\times4\times4}(2,2,1)$$
$$=N_c\_MC^{gen,1}_{4\times4\times4}(2,2,1)+N_c\_MC^{gen,2}_{4\times4\times4}(2,2,1)$$
$$+N_c\_MC^{gen,3}_{4\times4\times4}(2,2,1)+N_c\_MC^{gen,4}_{4\times4\times4}(2,2,1)$$
$$+N_c\_MC^{gen,5}_{4\times4\times4}(2,2,1)=0+0+1+0+0=1$$

$$MC^{gen}_{4\times4\times4}(2,2,3)=\sum_{k=1}^{5}N_c\_MC^{gen,k}_{4\times4\times4}(2,2,3)$$
$$=N_c\_MC^{gen,1}_{4\times4\times4}(2,2,3)+N_c\_MC^{gen,2}_{4\times4\times4}(2,2,3)$$
$$+N_c\_MC^{gen,3}_{4\times4\times4}(2,2,3)+N_c\_MC^{gen,4}_{4\times4\times4}(2,2,3)$$
$$+N_c\_MC^{gen,5}_{4\times4\times4}(2,2,3)=1+0+0+0+0=1$$

$$MC^{gen}_{4\times4\times4}(2,3,1)=\sum_{k=1}^{5}N_c\_MC^{gen,k}_{4\times4\times4}(2,3,1)$$
$$=N_c\_MC^{gen,1}_{4\times4\times4}(2,3,1)+N_c\_MC^{gen,2}_{4\times4\times4}(2,3,1)$$
$$+N_c\_MC^{gen,3}_{4\times4\times4}(2,3,1)+N_c\_MC^{gen,4}_{4\times4\times4}(2,3,1)$$
$$+N_c\_MC^{gen,5}_{4\times4\times4}(2,3,1)=0+1+0+0+1=2$$

$$MC^{gen}_{4\times4\times4}(2,3,2)=\sum_{k=1}^{5}N_c\_MC^{gen,k}_{4\times4\times4}(2,3,2)$$
$$=N_c\_MC^{gen,1}_{4\times4\times4}(2,3,2)+N_c\_MC^{gen,2}_{4\times4\times4}(2,3,2)$$
$$+N_c\_MC^{gen,3}_{4\times4\times4}(2,3,2)+N_c\_MC^{gen,4}_{4\times4\times4}(2,3,2)$$
$$+N_c\_MC^{gen,5}_{4\times4\times4}(2,3,2)=0+0+0+1+0=1$$

The rest cells in $\mathbf{MC^{gen}_{4\times4\times4}}(\mathbf{2})$ are set to 0. There are four similar blocks [1,2], [1,4], [2,1] and [3,4] at the third position (i.e., $x=3$). Therefore, we can also get $MC^{gen}_{4\times4\times4}(3,1,2)=1$, $MC^{gen}_{4\times4\times4}(3,1,4)=2$, $MC^{gen}_{4\times4\times4}(3,2,1)=1$, and $MC^{gen}_{4\times4\times4}(3,3,4)=1$, respectively. Because the information of similar blocks in the last position has already been saved in the previous position, each element in $\mathbf{MC^{gen}_{4\times4\times4}}(\mathbf{4})$ can be set to 0. Therefore, according to Eq. (13), the hierarchical structure of the matrix cube $\mathbf{MC^{gen}_{4\times4\times4}}$ can be obtained, which is shown in Fig. 3.

Obviously, it can be seen that both the order of jobs and the job blocks with their exact positions (i.e., ordinal numbers) can be preserved in $\mathbf{MC^{gen}_{4\times4\times4}}$. That is, the similar blocks in the different positions of superior solutions can be learned and saved. Nevertheless, as for the two-dimensional EDAs, the position information of the similar blocks is ignored. For example, though the job block [2,3] appears in the second position in $\boldsymbol{\pi}^{\mathbf{gen,1}}_{\mathbf{Sbest}}$ and also in the first position in $\boldsymbol{\pi}^{\mathbf{gen,2}}_{\mathbf{Sbest}}$, this block information can only be saved in one element of a two-dimensional matrix. Hence, the shortcoming of the two-dimensional EDAs is that the sampling procedure cannot determine the suitable positions to place similar blocks. However, in the proposed matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$, all blocks can be reserved in the different layers of $\mathbf{MC^{gen}_{n\times n\times n}}$ in accordance with their respective positions. As can be seen from Fig. 3, the job block [2,3] in $\boldsymbol{\pi}^{\mathbf{gen,1}}_{\mathbf{Sbest}}$=[1, 2, 3, 4] is reserved in $\mathbf{MC^{gen}_{4\times4\times4}}(\mathbf{2})$ (i.e., the second layer of $\mathbf{MC^{gen}_{4\times4\times4}}$) and the same block [2,3] in $\boldsymbol{\pi}^{\mathbf{gen,2}}_{\mathbf{Sbest}}$=[2, 3, 1, 4] is recorded in $\mathbf{MC^{gen}_{4\times4\times4}}(\mathbf{1})$ (i.e., the first layer of $\mathbf{MC^{gen}_{4\times4\times4}}$). In this way, the job blocks or similar blocks with their corresponding positions and the total order information can be learned and reserved properly in different layers of the three-dimensional matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$, which is helpful for effectively guiding the search to promising regions and avoiding the promising patterns being destroyed or inappropriately fused.

$$\mathbf{MC^{gen}_{4\times4\times4}(1)}=\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&1&0&0\\0&0&2&0\end{bmatrix},\ \mathbf{MC^{gen}_{4\times4\times4}(2)}=\begin{bmatrix}0&0&0&0\\1&0&1&0\\2&1&0&0\\0&0&0&0\end{bmatrix},$$
$$\mathbf{MC^{gen}_{4\times4\times4}(3)}=\begin{bmatrix}0&1&0&2\\1&0&0&0\\0&0&0&1\\0&0&0&0\end{bmatrix},\ \mathbf{MC^{gen}_{4\times4\times4}(4)}=\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},$$

Fig. 3. Illustration of the hierarchical structure of $\mathbf{MC^{gen}_{4\times4\times4}}$.

3.2.2. Matrix-cube-based probabilistic model

Different from the existing research works, a three-dimensional probability model-based evolutionary mechanism is developed to accumulate the valuable information (i.e., the similar blocks with their corresponding positions and the total order information) from the superior solutions by the matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$. The proposed learning strategy not only achieves self-learning from the matrix cube, but also has a significant effect on the searching behavior of EDA. Let $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ be the matrix-cube-based probabilistic model, and $Pro\_MC^{gen}_{n\times n\times n}(x,y,z)$ the element in $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$. The xth layer of $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ is defined as follows:

$$\mathbf{Pro\_MC^{gen}_{n\times n\times n}}(\mathbf{x})=\begin{bmatrix}\mathbf{Pro\_MC^{gen}_{n\times n\times n}}(\mathbf{x},\mathbf{1})\\ \vdots \\ \mathbf{Pro\_MC^{gen}_{n\times n\times n}}(\mathbf{x},\mathbf{n})\end{bmatrix}_{n\times1}$$
$$=\begin{bmatrix}Pro\_MC^{gen}_{n\times n\times n}(x,1,1) & \cdots & Pro\_MC^{gen}_{n\times n\times n}(x,1,n)\\ \vdots & \ddots & \vdots \\ Pro\_MC^{gen}_{n\times n\times n}(x,n,1) & \cdots & Pro\_MC^{gen}_{n\times n\times n}(x,n,n)\end{bmatrix}_{n\times n} \quad (14)$$

Furthermore, let $Sum\_MC^{gen}(x)=\sum_{y=1}^{n}\sum_{z=1}^{n}MC^{gen}_{n\times n\times n}(x,y,z)$ and $Sum\_Pro\_MC^{gen}(x)=\sum_{y=1}^{n}\sum_{z=1}^{n}Pro\_MC^{gen}_{n\times n\times n}(x,y,z)$ $(x=1,\ldots,n-1)$ be the total summation of all elements in $\mathbf{MC^{gen}_{n\times n\times n}}(\mathbf{x})$ and $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}(\mathbf{x})$, respectively. Then the update procedure of $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ is designed as follows:

Step 0: Initialize the matrix-cube-based probabilistic model, i.e.,
$$Pro\_MC^{0}_{n\times n\times n}(x,y,z)=\begin{cases}1/n, & x=1,y,z=1,\ldots,n\\ 1/n^2, & x=2,3,\ldots,n-1,y,z=1,\ldots,n.\end{cases}$$

Step 1: Calculate $\mathbf{MC^{0}_{n\times n\times n}}$ and set

$$Pro\_MC^{1}_{n\times n\times n}(x,y,z)$$
$$=\begin{cases}MC^{0}_{n\times n\times n}(x,y,z)/Sum\_MC^{0}(x), & x=1,y,z=1,\ldots,n\\ \left(Pro\_MC^{0}_{n\times n\times n}(x,y,z)+MC^{0}_{n\times n\times n}(x,y,z)\right)/\\ \left(Sum\_Pro\_MC^{0}(x)+Sum\_MC^{0}(x)\right), & x=2,\ldots,n-1,y,z=1,\ldots,n.\end{cases}$$

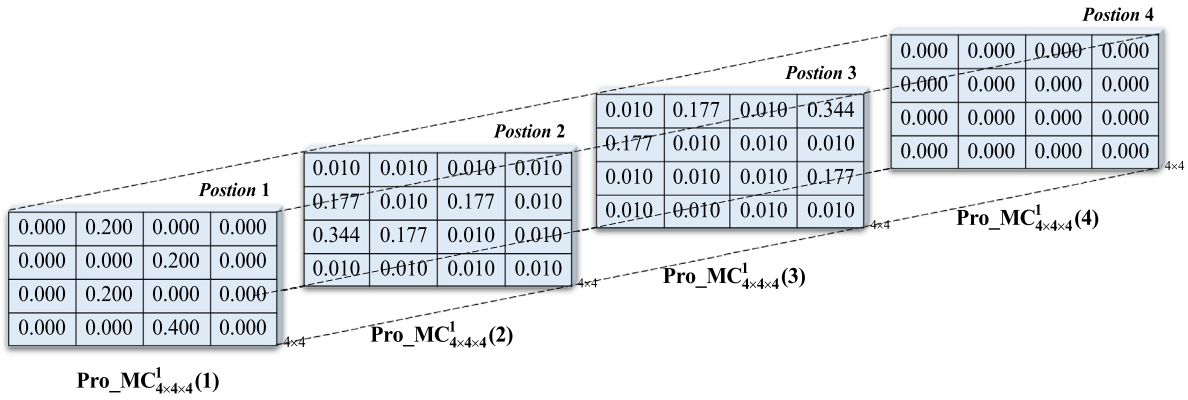


Fig. 4. Illustration for the hierarchical structure of $\mathbf{Pro\_MC^{1}_{4\times4\times4}}$.

Step 2: Set $gen = 1$.
Step 3: Calculate $\mathbf{MC^{gen}_{n\times n\times n}}$ and update $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ by using

$$Pro\_MC^{gen+1}_{n\times n\times n}(x,y,z) = (1-r)\times Pro\_MC^{gen}_{n\times n\times n}(x,y,z) + r \times MC^{gen}_{n\times n\times n}(x,y,z)/Sum\_MC^{gen}(x),$$
$$x = 1,...,n-1, y, z = 1,...,n.$$

Step 4: Set $gen = gen+1$. If $gen < maxgen$, then go to Step 3.

Step 0 is used to set all elements in $\mathbf{Pro\_MC^{0}_{n\times n\times n}}$ to $1/n$ instead of $1/n^2$, which can increase the selection probability of blocks in superior individuals at initial stage. Steps 1 and 3 are utilized to accumulate valuable information from $\mathbf{MC^{gen}_{n\times n\times n}}$, which is helpful for achieving a trade-off between the current and the historical information. Fig. 4 shows the update process of the probabilistic model at generation one by using the superior individuals considered in Section 3.2.1. It can be seen from Fig. 4 that the subscripts of the larger values in $\mathbf{Pro\_MC^{1}_{4\times4\times4}}$ correspond with the ordinal relationships and the similar blocks in these superior individuals. This means that $\mathbf{Pro\_MC^{gen}_{4\times4\times4}}$ can accumulate valuable information more accurately.

3.2.3. New population generation method

The global search is performed by sampling the probabilistic model to generate new individuals. In order to guarantee the initial population with certain quality and diversity, half of the individuals in the population are generated via the simple heuristics H_{21} and H_{22} presented by Hatami et al. [17], and the remaining individuals are randomly generated. Let $\boldsymbol{\pi}^{\mathbf{gen,k}}=[\pi^{gen,k}(1), \pi^{gen,k}(2), ..., \pi^{gen,k}(n)]$ be the kth ($k = 1, ..., popsize$) individual in **Pop(gen)**, $l(J_c,\boldsymbol{\pi}^{\mathbf{gen,k}})$ be the number of occurrences for J_c and $SelectJob(\boldsymbol{\pi}^{\mathbf{gen,k}},i)$ (i>1) be a selection function. Because the probability value of the occurrence of job block or similar block $[\pi^{gen,k}(i-1), \pi^{gen,k}(i)]$ is stored in the submatrix $\mathbf{Pro\_MC^{gen-1}_{n\times n\times n}(i-1)}$, $SelectJob(\boldsymbol{\pi}^{\mathbf{gen,k}},i)$ determines the candidate job J_c at the ith position in $\boldsymbol{\pi}^{\mathbf{gen,k}}$ by using $\mathbf{Pro\_MC^{gen-1}_{n\times n\times n}(i-1, \boldsymbol{\pi}^{gen,k}(i-1))}$. Then, the details of $SelectJob(\boldsymbol{\pi}^{\mathbf{gen,k}},i)$ is described as follows:

Step 1: Randomly generate a probability r for roulette selection, $r \in [0, \sum_{h=1}^{n} Pro\_MC^{gen-1}_{n\times n\times n}(i-1, \pi^{gen,k}(i-1), h))$.
Step 2: Get J_c by the roulette selection method.
Step 2.1: If $r \in [0, Pro\_MC^{gen-1}_{n\times n\times n}(i-1, \pi^{gen,k}(i-1), 1))$, then set J_c=1 and go to Step 3.
Step 2.2: If $r \in [\sum_{h=1}^{pos} Pro\_MC^{gen-1}_{n\times n\times n}(i-1, \pi^{gen,k}(i-1), h), \sum_{h=1}^{pos+1} Pro\_MC^{gen-1}_{n\times n\times n}(i-1, \pi^{gen,k}(i-1), h))$ and $pos \in \{1, ..., n-1\}$, then set J_c=pos+1.
Step 3: Return J_c.

Since $SelectJob(\boldsymbol{\pi}^{\mathbf{gen,k}},i)$ cannot be used to obtain the first job in the new individual, a special sampling method, namely the first position limited selection strategy (FPLSS), is provided to reasonably determine the first job. The details of FPLSS is given as follows:

Step 1: Calculate $SumPro\_MC^{gen-1}_{init}(y) = \sum_{z=1}^{n} Pro\_MC^{gen-1}(1,y,z)$; Obtain the cumulative probability of the yth row of $\mathbf{Pro\_MC^{gen-1}_{n\times n\times n}(1)}$.
Step 2: Determine the job J_c at the first position in $\boldsymbol{\pi}^{\mathbf{gen,k}}$, $k = 1, ..., popsize$ by roulette method.
Step 2.1: Randomly generate a r', $r' \in [0, \sum_{h=1}^{n} SumPro\_MC^{gen-1}_{init}(h))$.
Step 2.2: If $r' \in [0, SumPro\_MC^{gen-1}_{init}(1))$, then set J_c=1 and go to Step 3.
Step 2.3: If $r' \in [\sum_{h=1}^{pos} SumPro\_MC^{gen-1}_{init}(h), \sum_{h=1}^{pos+1} SumPro\_MC^{gen-1}_{init}(h))$ and $pos \in \{1, ..., n-1\}$, then set J_c=pos+1.
Step 3: Return J_c as the job in the first position.

Then, the new population generation method is given as follows:

Step 1: Set $k = 1$.
Step 2: Generate a new individual $\boldsymbol{\pi}^{\mathbf{gen,k}}$.
Step 2.1: Set $\boldsymbol{\pi}^{\mathbf{gen,k}}$ as an empty sequence. Select the first job J_c according to the above FPLSS.
Step 2.2: Set $\pi^{gen,k}(1) = J_c$ and $i = 2$.
Step 2.3: Set $J_c = SelectJob(\boldsymbol{\pi}^{\mathbf{gen,k}}, i)$.
Step 2.4: If $l(J_c, \boldsymbol{\pi}^{\mathbf{gen,k}}) = 2$, then go to Step 2.3.
Step 2.5: Set $\pi^{gen,k}(i) = J_c$.
Step 2.6: Set $i = i + 1$.
Step 2.7: If $i \le n$, then go to Step 2.3.
Step 3: Set $k = k + 1$.
Step 4: If $k \le popsize$, then go to Step 2.
Step 5: Output **Pop(gen)**.

It is noted that steps 2.3–2.5 are used to build the job block (i.e., $[\pi^{gen+1,k}(i-1), \pi^{gen+1,k}(i)]$) at positions $i-1$ and i by means of sampling from the probabilistic submatrix $\mathbf{Pro\_MC^{gen}_{n\times n\times n}(i-1)}$. Meanwhile, step 2 can link each job block or similar block located at different positions together to generate a new solution, which is a key step of global exploration of the proposed MCEDA.

3.3. Problem-dependent local search

It has been generally recognized that incorporating effective local exploitation into a *meta*-heuristic can increase its search depth in solution space, thereby enhance its performance [1,33,38,39]. In this subsection, four critical path-based neighborhood searches and eight problem-specific neighborhood searches are first designed, and then a problem-dependent variable neighborhood descent (VND) search is proposed based on these neighborhood searches to execute deep exploitation around the promising regions or superior solutions.

3.3.1. Neighborhood searches

For the combinatorial optimization problem, each global optimal solution is a local optimal solution under all neighborhood structures, and

high-quality solutions close to a global optimal solution are often local optimal solutions under multiple neighborhood structures. The repeated greedy search process (RGSP) with one single neighborhood operator is easy to reach and trap into a local optimal solution of the corresponding neighborhood structure, but the quality of this solution is usually not good enough. Meanwhile, the DAPFSP further considers multiple parallel flow shops and one assembly shop on the basis of the traditional flow shop, which makes its solution space huge and complex. That is, many high-quality local optimal solutions are scattered in the very deep regions of its solution space. It is difficult to find high-quality solutions for the DAPFSP. Therefore, we design twelve neighborhood searches based on different neighborhood structures to enrich search behavior and avoid the algorithm falling into a local optimal solution with common or low quality.

3.3.1.1. Four critical path-based neighborhood searches. The first four neighborhood searches are based on the so-called critical path in the solution's Gantt chart. The critical path refers to a continuous and longest path from the start to end of a schedule (i.e., a solution) without idle time between any two jobs [40]. For the DAPFSP, the makespan of a schedule is determined by the length of the critical path and it cannot be decreased without changing the scheduling of jobs in the critical path. In other words, it is possible to obtain better solutions by changing the jobs on the critical path. Let f_c be the critical factory, n_{f_c} the total number of jobs assigned to f_c, and n_c the number of the critical jobs in f_c. The details of these four critical path-based neighborhood searches are described as follows.

(1) $Critical\_Job\_N^{in}_{Swap}\_Search(\boldsymbol{\pi})$: Generate $n_{f_c} - n_c$ new solutions or neighbors by swaping a randomly selected critical job $\pi_{f_c}(u)$ ($u \in \{1,2,...,n_c\}$) and each non-critical job $\pi_{f_c}(v)$ ($v = n_c + 1, ..., n_{f_c}$) in $\boldsymbol{\pi}$, and then return the best neighbor.

(2) $Critical\_Job\_N^{in}_{Insert}\_Search(\boldsymbol{\pi})$: Generate $n_{f_c} - n_c + 1$ new neighbors by inserting a randomly selected critical job $\pi_{f_c}(u)$ ($u \in \{1,2,...,n_c\}$) before each non-critical job $\pi_{f_c}(v)$ ($v = n_c + 1, ..., n_{f_c}$) and after the non-critical job $\pi_{f_c}(n_{f_c})$ in $\boldsymbol{\pi}$, and then return the best neighbor.

(3) $Critical\_Job\_N^{bt}_{Swap}\_Search(\boldsymbol{\pi})$: Generate $F - 1$ new neighbors by swaping a randomly selected critical job $\pi_{f_c}(u)$ ($u \in \{1,2,...,n_c\}$) in $\boldsymbol{\pi}$ and a randomly selected job in each non-critical factory of $\boldsymbol{\pi}$, and then return the best neighbor.

(4) $Critical\_Job\_N^{bt}_{Insert}\_Search(\boldsymbol{\pi})$: Generate $F - 1$ new neighbors by inserting a randomly selected critical job $\pi_{f_c}(u)$ ($u \in \{1,2,...,n_c\}$) in $\boldsymbol{\pi}$ before or after a randomly selected job in each non-critical factory of $\boldsymbol{\pi}$, and then return the best neighbor.

The neighborhood operations performed in these critical path-based neighborhood searches are illustrated in Fig. 5, where the jobs 3, 18, 17, and 16 are on the critical path and the job 18 is selected to perform swap and insert operations within the critical factory or between the factories.

3.3.1.2. Eight non-critical path-based neighborhood searches. The remaining eight non-critical path-based neighborhood searches can be classified into two types: four product-based neighborhood searches and four job-based neighborhood searches. Let S be the total number of products, $\lambda(h)$ ($h = 1, 2, ..., S$) one product of $\boldsymbol{\pi}$, λ the product sequence of $\boldsymbol{\pi}$, and $\boldsymbol{\pi}_{\lambda(h)}$ the product subsequence $[\pi_{\lambda(h)}(1),\pi_{\lambda(h)}(2),...,\pi_{\lambda(h)}(\omega_h)]$ of $\boldsymbol{\pi}$.

Three product-based neighborhood searches and three job-based neighborhood searches are realized by Algorithm 1 and Algorithm 2, respectively. The input parameter *Neighborhood* determines the type of neighborhood search. Moreover, one product-based neighborhood search and one job-based neighborhood search are realized by Algorithm 3 and Algorithm 4, respectively. Each of these neighborhood searches consecutively scans new neighbors (i.e., solutions) of the current $\boldsymbol{\pi}$ via a special neighborhood operator until a local optimal solution under the corresponding neighborhood structure is found. During the search, the current $\boldsymbol{\pi}$ is updated by its best neighbor if this best neighbor is better than it, which is helpful in concentrating the search on more promising regions. Obviously, each neighborhood search works like a drill bit, constantly digging down to reach a local optimum in solution space.

3.3.2. Perturbation and exploitation

In the VND search, the perturbation is needed to escape from the current local optimal solution reached by the exploitation. Because each of the critical path-based neighborhood searches in Section 3.3.1 can quickly find a promising but different solution near the local optimal one, all of them are utilized to execute perturbation in our VND search (see Section 3.3.3). The details of the designed perturbation are provided in Algorithm 5.

The non-critical path-based neighborhood searches in Section 3.3.1 can reach a larger number of neighbors in the quite different regions of solution space, which is beneficial to guiding the search toward high-quality local optima. Thus, the exploitation in our VND search is performed via these production-based and job-based neighborhood searches. The details of the designed exploitation are given in Algorithms 6 and 7, respectively. To accelerate the search process in Algorithms 5–7, the speed-up strategies based on the general property of the permutation-based model for scheduling problem [41] are utilized to calculate the makespan of each neighbor.

3.3.3. Variable neighborhood descent search

The variable neighborhood descent (VND) search is used as the local search in MCEDA. The VND method is proposed by Hansen and Mladenovic [42], which has been widely utilized to detect high-quality solutions or potential regions in solution space. The VND method systematically exploits a set of predefined neighborhood structures both in the descent to get different local optimal solutions and in the perturbation to escape from these local optimums. Obviously, the VND method provides a general framework for addressing different combinatorial optimization problems. The key to ensuring VND's performance on the specific problem is to design the problem-dependent neighborhood searches to effectively execute exploitation.

Based on the above designed neighborhood searches (Algorithms 1–7), the VND search is presented in Algorithm 8, where Line 4 executes perturbation to escape from the current local optimal solution, Lines 11 and 15 perform exploitation or neighborhood searches, Lines 12 and 16 ensure that the search direction is descent during the exploitation phase (Lines 10–25), and Lines 28–30 update the best neighbor found at the exploitation phase. Moreover, $\boldsymbol{\pi}' = \boldsymbol{\pi}''$ at Line 13 and it at Line 17 guarantee that the perturbation at Line 4 always performs on the best neighbor obtained at the last exploitation phase. The VND search process (Lines 3–34) consists of a series of repeated greedy search processes (RGSPs). Each RGSP (Lines 4–25) is that $\boldsymbol{\pi}''$ driven by the production-based neighborhood searches at Line 11 and the job-based neighborhood searches at Line 15 continuously searches down from the perturbation region generated at Line 4 until it reaches a common local optimal solution of the designed neighborhood structures. Since there are four critical path-based neighborhood searches for perturbation (Line 4) and eight non-critical path-based neighborhood searches for exploitation (Lines 11 and 15), the proposed VND search has the ability to perform deep exploitation from different promising regions.

3.4. Outline of MCEDA

With the aforementioned design, we give the main framework of MCEDA shown in Fig. 6. It can be seen that the presented scheme can not only apply the matrix-cube-based probabilistic model to estimate the probability distribution of superior solutions and to guide the global exploration for finding promising regions in search space, but also adopt the problem-dependent VND search to perform in-depth local exploitation around these promising regions. Since the global and local searches

Algorithm 1
Product_Search($\boldsymbol{\pi}$, *Neighborhood*).

Algorithm 1 *Product_Search*($\boldsymbol{\pi}$, *Neighborhood*)

Input: $\boldsymbol{\pi}$, *Neighborhood*
Output: $\boldsymbol{\pi}$
1. Set $h=1$
2. **While** ($h \le S$)
3. Extract each product $\lambda(h)$ from product sequence $\boldsymbol{\lambda}$
4. **Case** *Neighborhood*
5. *Swap*: **Swap** $\lambda(h)$ with every other product in $\boldsymbol{\lambda}$
6. *Insert*: **Insert** $\lambda(h)$ at every possible position of $\boldsymbol{\lambda}$
7. *Interchange*: **Interchange** $\lambda(h)$ with its adjacent product in $\boldsymbol{\lambda}$
8. **End Case**
9. $\boldsymbol{\pi}^*$ = The best one among the solutions or neighbors generated at lines 4-8
10. **If** $C_{\max}(\boldsymbol{\pi}^*) < C_{\max}(\boldsymbol{\pi})$ **then**
11. $\boldsymbol{\pi} = \boldsymbol{\pi}^*$, $h=1$
12. **Else**
13. $h = h+1$
14. **End If**
15. **End While**
16. **Return** $\boldsymbol{\pi}$

Algorithm 2
Job_Search($\boldsymbol{\pi}$, *Neighborhood*).

Algorithm 2 *Job_Search*($\boldsymbol{\pi}$, *Neighborhood*)

Input: $\boldsymbol{\pi}$, *Neighborhood*
Output: $\boldsymbol{\pi}$
1. Set $h=1$
2. **While** ($h \le S$)
3. Set $j=1$
4. **While** ($j \le \omega_h$)
5. Extract each job $\pi_{\lambda(h)}(j)$ from $\boldsymbol{\pi}_{\boldsymbol{\lambda(h)}}$
6. **Case** *Neighborhood*
7. *Swap*: **Swap** $\pi_{\lambda(h)}(j)$ with every other job in $\boldsymbol{\pi}_{\boldsymbol{\lambda(h)}}$
8. *Insert*: **Insert** $\pi_{\lambda(h)}(j)$ at every possible position of $\boldsymbol{\pi}_{\boldsymbol{\lambda(h)}}$
9. *Interchange*: **Interchange** $\pi_{\lambda(h)}(j)$ with its adjacent job $\pi_{\lambda(h)}(j+1)$ in $\boldsymbol{\pi}_{\boldsymbol{\lambda(h)}}$
10. **End Case**
11. $\boldsymbol{\pi}^*$ = The best one among the solutions or neighbors generated at lines 6-10
12. **If** $C_{\max}(\boldsymbol{\pi}^*) < C_{\max}(\boldsymbol{\pi})$ **then**
13. $\boldsymbol{\pi} = \boldsymbol{\pi}^*$, $h=1$, $j=1$
14. **Else**
15. $j = j+1$
16. **End If**
17. **End While**
18. $h=h+1$
19. **End While**
20. **Return** $\boldsymbol{\pi}$

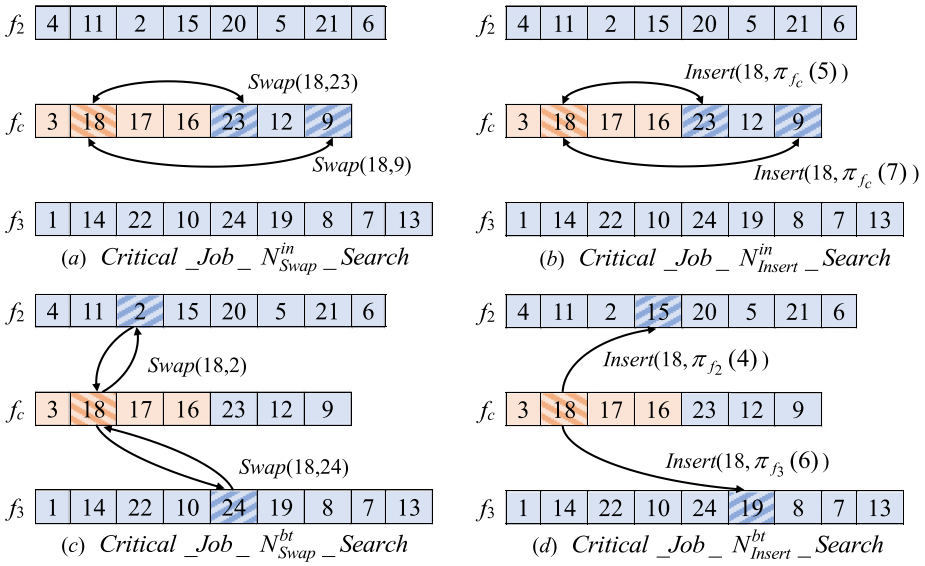


Fig. 5. An example of four critical path-based neighborhood searches.

Algorithm 3
Product_Search_Inverse($\boldsymbol{\pi}$).

Algorithm 3 *Product_Search _Inverse*($\boldsymbol{\pi}$)

Input: $\boldsymbol{\pi}$
Output: $\boldsymbol{\pi}$
1. **Inverse** the product subsequence between every two different product s in $\boldsymbol{\lambda}$
2. $\boldsymbol{\pi}^*$ = The best one among the solutions or neighbors generated at line 1
3. **If** $C_{\max}(\boldsymbol{\pi}^*) < C_{\max}(\boldsymbol{\pi})$ **then**
4. $\quad \boldsymbol{\pi} = \boldsymbol{\pi}^*$
5. **End If**
6. **Return** $\boldsymbol{\pi}$

Algorithm 4
Job_Search_Inverse($\boldsymbol{\pi}$).

Algorithm 4 *Job _Search _Inverse* ($\boldsymbol{\pi}$)

Input: $\boldsymbol{\pi}$
Output: $\boldsymbol{\pi}$
1. Set h=1
2. **While** ($h \le S$)
3. $\quad$ **Inverse** the job subsequence between every two different jobs in $\boldsymbol{\pi}_{\boldsymbol{\lambda(h)}}$
4. $\quad \boldsymbol{\pi}^*$ = The best one among the solutions or neighbors generated at line 3
5. $\quad$ **If** $C_{\max}(\boldsymbol{\pi}^*) < C_{\max}(\boldsymbol{\pi})$ **then**
6. $\quad\quad \boldsymbol{\pi} = \boldsymbol{\pi}^*$, $h = 1$
7. $\quad$ **Else**
8. $\quad\quad h$=h+1
9. $\quad$ **End If**
10. **End While**
11. **Return** $\boldsymbol{\pi}$

Algorithm 5
Perturbation($\boldsymbol{\pi}$, *Neighborhood_Num*).

Algorithm 5 *Perturbation*($\boldsymbol{\pi}$, *Neighborhood _ Num*)

Input: $\boldsymbol{\pi}$, *Neighborhood _ Num*
Output: $\boldsymbol{\pi}$
Case *Neighborhood _ Num*
  1: $Critical\_Job\_N^{in}_{Swap}\_Search(\boldsymbol{\pi})$ // (1) in Subsubsection 3.3.1
  2: $Critical\_Job\_N^{in}_{Insert}\_Search(\boldsymbol{\pi})$ // (2) in Subsubsection 3.3.1
  3: $Critical\_Job\_N^{bt}_{Swap}\_Search(\boldsymbol{\pi})$ // (3) in Subsubsection 3.3.1
  4: $Critical\_Job\_N^{bt}_{Insert}\_Search(\boldsymbol{\pi})$ // (4) in Subsubsection 3.3.1
End Case
Return $\boldsymbol{\pi}$

Algorithm 6
Exploitation_Product_Search($\boldsymbol{\pi}$, *Neighborhood_Num*).

Algorithm 6 *Exploitation _ Product _ Search*($\boldsymbol{\pi}$, *Neighborhood _ Num*)

Input: $\boldsymbol{\pi}$, *Neighborhood _ Num*
Output: $\boldsymbol{\pi}$
Case *Neighborhood _ Num*
  1: $Product\_Search(\boldsymbol{\pi}, Swap)$ // **Algorithm 1**
  2: $Product\_Search(\boldsymbol{\pi}, Insert)$ // **Algorithm 1**
  3: $Product\_Search(\boldsymbol{\pi}, Interchange)$ // **Algorithm 1**
  4: $Product\_N_{Inverse}\_Search(\boldsymbol{\pi})$ // **Algorithm 2**
End Case
Return $\boldsymbol{\pi}$

Algorithm 7
Exploitation_Job_Search($\boldsymbol{\pi}$, *Neighborhood_Num*).

Algorithm 7 *Exploitation _ Job _ Search*($\boldsymbol{\pi}$, *Neighborhood _ Num*)

Input: $\boldsymbol{\pi}$, *Neighborhood _ Num*
Output: $\boldsymbol{\pi}$
Case *Neighborhood _ Num*
  1: $Job\_Search(\boldsymbol{\pi}, Swap)$ // **Algorithm 3**
  2: $Job\_Search(\boldsymbol{\pi}, Insert)$ // **Algorithm 3**
  3: $Job\_Search(\boldsymbol{\pi}, Interchange)$ // **Algorithm 3**
  4: $Job\_N_{Inverse}\_Search(\boldsymbol{\pi})$ // **Algorithm 4**
End Case
Return $\boldsymbol{\pi}$

are well stressed and balanced, the proposed MCEDA can be expected to achieve better results for addressing the DAPFSP. In next section, we will investigate the performance of MCEDA.

3.5. Convergence analysis of MCEDA

Although empirical evidence indicates that EDA has effectiveness and efficiency for solving complex optimization problems [30–36,38], there are few theoretical results concerning its convergence properties. The theoretical investigation on evolutionary algorithms can provide more insight into the principles and mechanism of the optimizer [43–45]. In this subsection, the convergence properties of our proposed MCEDA is theoretically analyzed by means of homogeneous finite Markov model. For convenience, some preliminaries are provided as follows.

Definition 1. The set of N finite and feasible individuals or solutions constitutes the population $\mathbf{Pop} = \{I_1, I_2, \ldots, I_N\}$. Individual state is de-

Algorithm 8
VND Search.

Algorithm 8 VND Search

Input: $\boldsymbol{\pi}_{\mathbf{best}}$, $l^p_{max}=4$, $l^J_{max}=4$, $N^c_{max}=4$

Output: $\boldsymbol{\pi}_{\mathbf{best}}$

Obtain the corresponding product sequence $\boldsymbol{\lambda}_{\mathbf{best}}$ according to the $\boldsymbol{\pi}_{\mathbf{best}}$
$\boldsymbol{\pi}'=\boldsymbol{\pi}_{\mathbf{best}}$, $k=1$, $flag=true$
While $k<N^c_{max}$ **do**
  $\boldsymbol{\pi}''=Perturbation(\boldsymbol{\pi}',k)$, $l^p=1$, $l^J=1$ // Perturbation **Algorithm 5**
  $ExploitBest\_C_{max}=C_{\max}(\boldsymbol{\pi}'')$, $Perturbation\_C_{max}=C_{\max}(\boldsymbol{\pi}'')$
  If $flag=true$ **then**
    $TotalExploitBest\_C_{max}=C_{\max}(\boldsymbol{\pi}'')$, $\boldsymbol{\pi}_{\mathbf{TotalExploitBest}}=\boldsymbol{\pi}''$
  End If
  $flag=flase$
  While $l^p<l^p_{max}$ **do** //Exploitation phase
    $\boldsymbol{\pi}''=Exploitation\_Product\_Search(\boldsymbol{\pi}'',l^p)$ //**Algorithm 6**
    If $C_{\max}(\boldsymbol{\pi}'')<ExploitBest\_C_{max}$ **then**
      $ExploitBest\_C_{max}=C_{\max}(\boldsymbol{\pi}'')$, $\boldsymbol{\pi}'=\boldsymbol{\pi}''$, $l^p=1$, $l^J=1$
      While $l^J<l^J_{max}$ **do**
        $\boldsymbol{\pi}''=Exploitation\_Job\_Search(\boldsymbol{\pi}'',l^J)$ //**Algorithm 7**
        If $C_{\max}(\boldsymbol{\pi}'')<ExploitBest\_C_{max}$ **then**
          $ExploitBest\_C_{max}=C_{\max}(\boldsymbol{\pi}'')$, $\boldsymbol{\pi}'=\boldsymbol{\pi}''$, $l^J=1$
        Else
          $l^J=l^J+1$
        End If
      End While
    Else
      $l^p=l^p+1$
    End If
  End While
  If $ExploitBest\_C_{max}<Perturbation\_C_{max}$ **then**
    $k=1$
    If $ExploitBest\_C_{max}<TotalExploitBest\_C_{max}$ **then**
      $TotalExploitBest\_C_{max}=ExploitBest\_C_{max}$, $\boldsymbol{\pi}_{\mathbf{TotalExploitBest}}=\boldsymbol{\pi}'$
    End If
  Else
    $k=k+1$
  End If
End While
If $TotalExploitBest\_C_{max}<C_{\max}(\boldsymbol{\pi}_{\mathbf{best}})$ **then**
  $\boldsymbol{\pi}_{\mathbf{best}}=\boldsymbol{\pi}_{\mathbf{TotalExploitBest}}$
End If
Return $\boldsymbol{\pi}_{\mathbf{best}}$

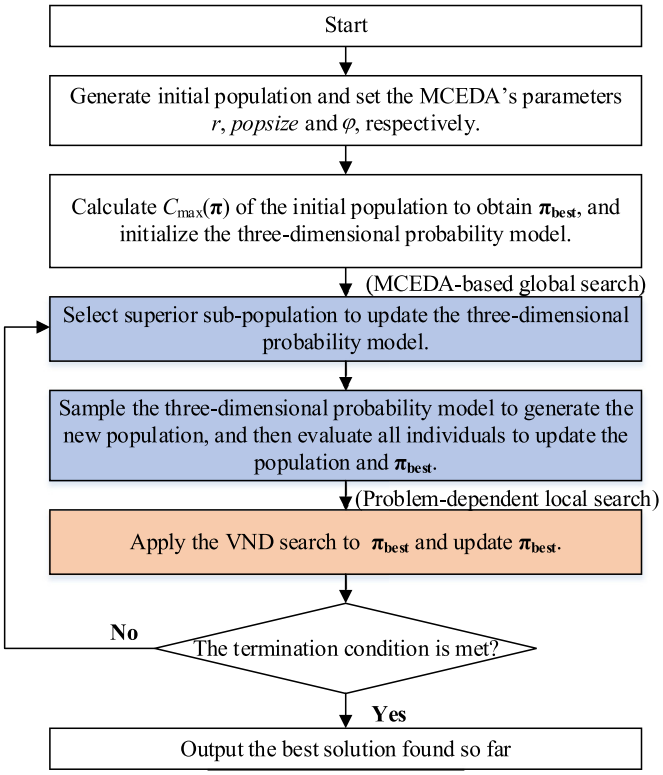


Fig. 6. The main framework of MCEDA for DAPFSP.

termined by the vector **x** and all individual states of the population form the population state $\mathbf{S^p} = (x_1, x_2, \dots, x_N)$, where x_k is the state of I_k $(k = 1, 2, ..., N)$.

Definition 2. The individual state space is denoted as Φ. The population state space Ω consists of all possible states of the population, i.e., $\Omega = \mathbf{S_1^p} \cup \mathbf{S_2^p} \cup \dots \cup \mathbf{S_{N'}^p}$, where $\mathbf{S_i^p} = (x_{i1}, x_{i2}, \dots, x_{iN})$ $(i = 1, 2, \dots, N')$, $\mathbf{S_i^p} \cap \mathbf{S_j^p} = \emptyset$ for $\forall i, j \in N' \wedge i \neq j$, and $\Omega = \Phi^N$.

Definition 3. In MCEDA, the state transition probability of individuals or solutions from two randomly selected states x_i to x_j in one step is denoted by $p(T_\Phi(x_i) = x_j)$, where T_Φ is the transition function in the state space Φ. Similarly, the state transition probability of the population from $S_i^p = (x_{i1}, x_{i2}, ..., x_{iN})$ to $S_j^p = (x_{j1}, x_{j2}, ..., x_{jN})$ is represented as $p(T_\Omega(S_i^p) = S_j^p)$, where T_Ω is the transition function in the population state space Ω.

Definition 4. In MCEDA, the optimal population state set is defined as $\mathbf{S} *= (x_1, x_2, ..., x_N)$, where $x_i^* \in \mathbf{B}^*$ and $\mathbf{B}^*$ is the set of the global optima of the considered optimization problem.

Definition 5. Let $\{\mathbf{S^p(t)}, t = 0, 1, 2, ...\}$ be a sequence of solutions generated by the population-based *meta*-heuristic algorithm. The stochastic sequence $\{\mathbf{S^p(t)}, t = 0, 1, 2, ...\}$ weakly converges in the sense of probability to the global optimum, if and only if $\lim_{t\to\infty} p\{\mathbf{S^p(t)} \cap \mathbf{B}* \neq \emptyset\} = 1$.

Lemma 1. *In MCEDA, the individual state x_i is essentially shifted in one step to another state x_j, and its state transition probability is given by*

$$p\left(T_\Phi(x_i) = x_j\right) = \prod_{l=1}^{n} P_{x_i \to x_j}^{(l)} \tag{15}$$

where $P_{x_i \to x_j}^{(l)} = \begin{cases} \sum_{l'=1}^{n} \left(\frac{\sum_{z=1}^{n} Pro\_MC(1,l',z)}{\sum_{y=1}^{n}\sum_{z=1}^{n} Pro\_MC(1,y,z)} \right) \times Sel(l'), l = 1 \\ \sum_{l'=1}^{n} \left(\frac{Pro\_MC(l-1,\pi_{x_j}(l-1),l')}{\sum_{z=1}^{n} Pro\_MC(l-1,\pi_{x_j}(l-1),z)} \right) \times Sel(l'), l > 1 \end{cases}$, and $Sel(x) = \begin{cases} 1, \text{ if } x \text{ is selected} \\ 0, \text{ else} \end{cases}$.

Proof. According to the Section 3.2, our proposed three-dimensional probability model employs an incremental learning mechanism to accumulate the valuable information from the superior solutions during the searching process. The valuable information can be accumulated into the three-dimensional probability model of MCEDA if and only if the individual corresponding to the state x_i is a superior solution. The next offspring is generated by sampling from this probability model. Let $\pi_{\mathbf{x_j}}$ be the job sequence of the jth individual state x_j, and the element $\pi_{x_j}(l)$ $(l = 1, 2, ..., n)$ the lth job in $\pi_{\mathbf{x_j}}$. For the selection function $SelectJob(\pi_{\mathbf{x_j}}, l)$ with roulette selection method, the probability to determine the l'th job selected and placed in the first position $(l = 1)$ of sequence $\pi_{\mathbf{x_j}}$ is $\sum_{z=1}^{n} Pro\_MC(1, l', z) / \sum_{y=1}^{n}\sum_{z=1}^{n} Pro\_MC(1, y, z)$. The selection event of all jobs is a mutually exclusive event. The selection probability is independent of each job. Then the selection probability $\sum_{l'=1}^{n} \left(\frac{\sum_{z=1}^{n} Pro\_MC(1,l',z)}{\sum_{y=1}^{n}\sum_{z=1}^{n} Pro\_MC(1,y,z)} \right) \times Sel(l')$ of the selected job at the first position in the sequence $\pi_{\mathbf{x_j}}$ can be calculated. Similarly, the probability of all possible selected jobs located in the lth $(l > 1)$ position of $\pi_{\mathbf{x_j}}$ is given by $\sum_{l'=1}^{n} \left(\frac{Pro\_MC(l-1,\pi_{x_j}(l-1),l')}{\sum_{z=1}^{n} Pro\_MC(l-1,\pi_{x_j}(l-1),z)} \right) \times Sel(l')$. Therefore, the state transition probability from x_i to x_j can be expressed in the form of joint probability $p(T_\Phi(x_i) = x_j) = \prod_{l=1}^{n} P_{x_i \to x_j}^{(l)}$. This completes the proof.

Lemma 2. *In the iterative process of MCEDA, the state transition probability of the population from $\mathbf{S_i^p}$ to $\mathbf{S_j^p}$ in one step is the joint probability*

$$p\left(T_\Omega\left(\mathbf{S_i^p}\right) = \mathbf{S_j^p}\right) = \prod_{k=1}^{N} p\left(T_\Phi\left(x_{ik}\right) = x_{jk}\right) \tag{16}$$

where N is the number of the population.

Proof. $T_\Omega(\mathbf{S_i^p}) = \mathbf{S_j^p}$ indicates that all individual states of population state $\mathbf{S_i^p}$ are simultaneously transferred to the corresponding individual states of population state $\mathbf{S_j^p}$, i.e., $T_\Phi(x_{ik}) = x_{jk}$ for $k = 1, 2, ..., N$. Since the state transfer processes among individuals are just independent events that aren't related to each other, Eq. (16) holds. This completes the proof.

Lemma 3. *In MCEDA, the population state sequence $\{\mathbf{S^p(t)}, t \geq 0\}$ is a finite homogeneous Markov chain on the state space Ω.*

Proof. Because both the population size and the number of iterations are finite, the search space of any *meta*-heuristic algorithms is finite during the whole evolutionary process. According to Definition 2, the MCEDA's population state $\mathbf{S^p} = (x_1, x_2, ..., x_N)$ at each generation consists of the state of N individuals and the state of each individual is finite, which leads to the fact that the population state space Ω is finite. Moreover, the next population state of MCEDA is independent from the past generation or state and only depends on the current state. Hence, the population state sequence $\{\mathbf{S^p(t)}, t \geq 0\}$ is a finite homogeneous Markov chain on the state space Ω.

According to the Lemma 2, the state transition probability in one step for any two states transforming from $\mathbf{S^p(t)}$ to $\mathbf{S^p(t+1)}$ in $\{\mathbf{S^p(t)}, t \geq 0\}$ is the joint probability

$$p\left(T_\Omega\left(\mathbf{S^p(t)}\right) = \mathbf{S^p(t+1)}\right) = \prod_{i=1}^{N} p\left(T_\Phi\left(x_i^t\right) = x_i^{t+1}\right) \tag{17}$$

From Lemma 1, for each individual state x_i of the population state $\mathbf{S^p}$, the state transition probability of going from time t to $t+1$ is determined

by $p(T_\Phi(x_i^t) = x_i^{t+1}) = \prod_{l=1}^{n} P^{(l)}_{x_i^t \to x_i^{t+1}}$. Since the state transition probability is only related to the population state at time t and independent of the state before time t, Lemma 3 holds.

Theorem 1. *MCEDA can converge to global optimum with the probability 1,* i.e., $\lim_{t\to\infty} p\{\mathbf{S^p(t)} \cap \mathbf{B*} \neq \emptyset\} = 1$.

Proof. Without loss of generality, we assume that the ith population state at generation or iteration times t is $\mathbf{S_i^p(t)} \in \mathbf{S^p(t)}$, and then the state $\mathbf{S_i^p(t)}$ is transferred to the jth state $\mathbf{S_j^p(t+1)} \in \mathbf{S^p(t+1)}$ at time $t+1$. For the tth generation, let $p_i(t)$ be the probability of ith state of the population state set $\mathbf{S^p(t)}$. If **Pop(t)** does not contain the optimal solution, then the probability of all states of the population is $\hat{p}_t = \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} p_i(t)$.

According to the Lemma 3, we have

$$\begin{aligned}\hat{p}_{t+1} &= \sum_{\mathbf{S_i^p(t)}} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) \\ &= \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} \neq \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) + \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t)\end{aligned} \tag{18}$$

According to the non-aftereffect property of Markov chain, we have

$$\begin{aligned}&\sum_{\mathbf{S_i^p}(t) \cap \mathbf{B*} \neq \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) + \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) \\ &= \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} p_i(t) = \hat{p}_t.\end{aligned} \tag{19}$$

After performing conversion in Eq. (19), we have

$$\sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) = \hat{p}(t) - \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} \neq \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t). \tag{20}$$

Hence, we can get

$$\begin{aligned}0 \le \hat{p}_{t+1} &= \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) + \hat{p}_t \\ &\quad - \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} \neq \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) \\ &< \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) + \hat{p}_t.\end{aligned} \tag{21}$$

In the above formula, it is obvious that $\sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} \sum_{\mathbf{S_j^p(t+1)} \cap \mathbf{B*} = \emptyset} p_i(t) p_{ij}(t) = 0$. Since $0 \le \hat{p}_{t+1} < \hat{p}_t$, we have $\lim_{t\to\infty} \hat{p}_t = 0$. That is, $\lim_{t\to\infty} p\{\mathbf{S^p(t)} \cap \mathbf{B*} \neq \emptyset\} = 1 - \lim_{t\to\infty} \sum_{\mathbf{S_i^p(t)} \cap \mathbf{B*} = \emptyset} p_i(t) = 1 - \lim_{t\to\infty} \hat{p}_t = 1$. According to the above analyses, it is concluded that MCEDA can converge to the global optimum with the probability 1.

4. Experimental comparisons and statistical analysis

This section describes the experimental settings and results obtained by MCEDA and other compared algorithms for solving the DAPFSP with the objective of minimizing makespan. The experimental setup and the tested data sets are provided in Section 4.1. The computational results are organized according to the following aspects: calibration parameters of the proposed MCEDA, assessment of the contribution of components in MCEDA, and comprehensive comparison of the effectiveness of MCEDA with some existing well-performing algorithms.

4.1. Experimental setup

In order to reasonably demonstrate the performance of the proposed MCEDA for the DAPFSP, we use two benchmark sets (the data are derived from http://soa.iti.es) presented by Hatami et al. [17] to make experimental comparisons among MCEDA and other algorithms. The first set consists of 900 small-scaled instances, where $n = \{8, 12, 16, 20, 24\}$, $m = \{2, 3, 4, 5\}$, $F = \{2, 3, 4\}$, and $S = \{2, 3, 4\}$. The second set is composed of 810 large-scaled instances, where $n = \{100, 200, 500\}$, $m = \{5, 10, 20\}$, $F = \{4, 6, 8\}$, and $S = \{30, 40, 50\}$. For each combination of $\{n, m, F, S\}$, there are five different test instances in the first set and ten different test instances in the second set. To make a fair comparison, we use the maximum elapsed CPU time $n\times(m/2)\times\rho$ (milliseconds) as the stopping criterion for each compared algorithm. The runtime factor ρ is set as three values: 1, 5 and 10. Considering the problem scale, each compared algorithm is executed 30 and 5 times independently for the first and second sets, respectively. All algorithms have been coded in Delphi XE8 and experiments have been executed on an Intel 2.6 GHz PC server with 64 GB RAM.

To verify the efficiency and effectiveness of the compared algorithms, the experimental results are evaluated by using a statistical metric, namely the average relative percentage deviation (ARPD), which is calculated as follows:

$$\text{ARPD} = \frac{1}{R} \sum_{i=1}^{R} \left(\frac{C_i - C_{opt}}{C_{opt}} \right) \times 100\% \tag{22}$$

where C_{opt} is the best-known solution provided in SOA website, and C_i is the solution obtained by a specific algorithm in the ith experiment for each instance. The ARPD is widely used to measure the gap between the solution obtained by the algorithm and the best solution found so far [19,20,33]. The lower the value of ARPD is, the higher the performance of algorithm is. Obviously, if the value of ARPD is less than zero, it means that the best-known solution is improved.

For saving space, the test results in the following Tables 6–9 and 12–14 are grouped according to the main instance characteristics, i.e., n, m, F and S. Each result in these tables is the average of the ARPD values of some corresponding instances. That is, there are 270 ARPD values ($3 \times 3 \times 3 \times 10$=270 instances) for each result in these tables except Table 7, and 60 ARPD values ($4 \times 3 \times 5$=60 instances) for each result in Table 7. In addition, the best, the second-best, and the third-best values in each row of Tables 7–8 and 12–14 are represented by using the bold, the bold and underlined, and the underlined fonts, respectively.

To verify whether the compared algorithms in Section 4.5 were reprogrammed correctly, we ran some of our reprogramming algorithms (i.e., EDAMA [19], HBBO [20], BS-HH [21], IG_3 [22], IGA [23] and HDIWO [24]) on the test sets in [19–24]. These test sets were chosen because only their download links are given in the literature. The test results of both the algorithms in [18–24] and our reprogramming algorithms were summarized in the attached tables A-F, which can be downloaded from https://pan.baidu.com/s/16cWZJFuwALMOvXEpexXmQQ (password: c6qv). It can be seen from each of these attached tables that the two types of test results are very similar. This manifests that our reprogramming algorithms correctly retain the performance of the algorithms in the literature.

4.2. Parameter calibration

It's known that the appropriate parameter calibration has a remarkable effect on the performance of the stochastic *meta*-heuristics, in terms of solution quality and computational efficiency. In this subsection, to decide desirable parameter configurations, the parameters of MCEDA and other compared algorithms are calibrated by means of design of experiment (DOE) technique [46]. We only detail the calibration of MCEDA because of the space limitation, and all experimental results are available upon request from the authors. According to the procedure of MCEDA described in Section 3.2.2, three parameters are included in MCEDA, i.e., the population size (*popsize*), the percentage of superior sub-population (φ), and the learning rate (γ). In consideration of the essential differences of the two benchmark sets, the calibrated experiments are separately executed on two different scale sets. Besides, to avoid the risk of over-fitting or biased results, our experiments are conducted by using randomly selected instances as the test bed [33]. For the

Table 3
Combinations of parameter values for MCEDA.

Parameters	Factor level 1	2	3	4
popsize	50	100	150	200
φ	0.1	0.2	0.3	0.4
γ	0.05	0.1	0.3	0.5

Table 4
Orthogonal array and RV values.

Experiment Number	Factor level			RV	
	popsize	φ	γ	Small-scaled	Large-scaled
1	1	1	1	1.42	0.11
2	1	2	2	1.48	0.14
3	1	3	3	1.41	0.16
4	1	4	4	1.27	0.18
5	2	1	2	1.58	0.15
6	2	2	1	1.49	0.17
7	2	3	4	1.47	0.16
8	2	4	3	1.46	0.15
9	3	1	3	1.35	0.18
10	3	2	4	1.71	0.14
11	3	3	1	2.09	0.18
12	3	4	2	1.87	0.19
13	4	1	4	1.75	0.19
14	4	2	3	1.82	0.15
15	4	3	2	1.67	0.21
16	4	4	1	2.18	0.22

small-scaled instances, one instance I_16_5_3_2_1 is used for calibration, where 16_5_3_2 indicates the combination of $\{n = 16, m = 5, F = 3, S = 2\}$ and the last number 1 means that it is the first instance under this combination. Similarly, another case I_200_5_6_40_7 is selected from the large-scaled instances. After the preliminary experiments, four potential levels of the three factors or parameters are explicitly listed in Table 3, and the orthogonal array $L_{16}(4^3)$ is selected to arrange these different combinations. That is, the number of parameters is 3, the number of factor levels is 4, and the total number of treatments is 16. Each configuration is conducted 30 independent replications with a predefined elapsed CPU time $n\times(m/2)\times 10$ (milliseconds). The ARPD value under each configuration is set as a response variable (RV). The orthogonal array and the obtained results are given in Table 4. According to Table 4, the values of average RV (ARV) for all factor levels are figured out to analyze the significance

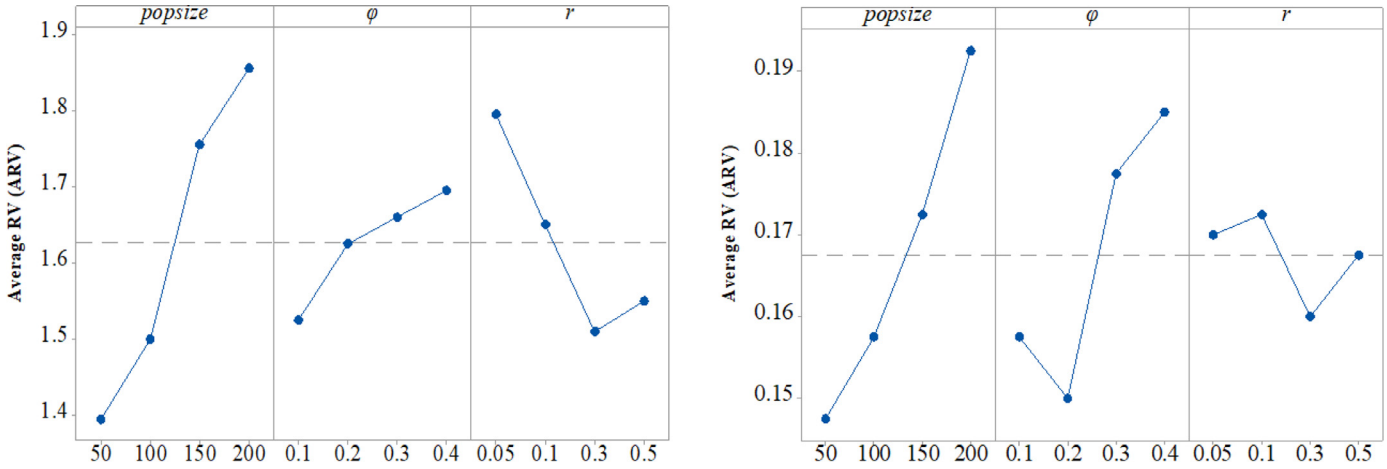
Table 5
The ARV and significant rank of each parameter.

Level	Small-scaled *popsize*	φ	γ	Large-scaled *popsize*	φ	γ
1	**1.395**	**1.525**	1.795	**0.148**	0.158	0.170
2	1.500	1.625	1.650	0.158	**0.150**	0.173
3	1.755	1.660	**1.510**	0.173	0.178	**0.160**
4	1.855	1.695	1.550	0.193	0.185	0.168
Delta	0.460	0.170	0.285	0.045	0.035	0.013
Rank	1	3	2	1	3	2

rank, where the *Delta* value as well as the significance rank are reported in Table 5. To investigate the effects of the key factors for different scale cases, the trend plots of each factor are illustrated in Fig. 7.

From Table 5, it can be seen that *popsize* results in the largest *Delta* values, which indicates that *popsize* has the most important significant impact on the performance of MCEDA. It can also be clearly observed from Fig. 7 that MCEDA with the smallest population size can obtain the lowest ARV for both the small-scaled instances and the large-scaled instances. The performance of the MCEDA degrades significantly with the increase of *popsize*. This finding suggests that we should set a relatively small value for the population size (i.e., $popsize$=50). The reason is that a large value of *popsize* may be helpful to increase the population diversity, but meanwhile the high computational efforts are needed in the global exploration. When the total running time is limited, the depth of local exploitation cannot be guaranteed. The learning rate γ is the second most significant parameter. The learning rate controls the learning speed of the probabilistic model. To be specific, a small value of γ slows down the updating process of the probability model and reduces the speed of convergence, while a large value may lead to premature convergence. Therefore, the value of γ should be determined by considering the trade-off among the convergence, prematurity and solution quality.

It is clear from Fig. 7(a) and Fig. 7(b) that γ=0.3 provides the best results. The significance of φ ranks the third. Based on the trend curves of φ in Fig. 7(a) and Fig. 7(b), φ=0.1 and φ=0.2 are suitable for the small-scaled and large-scaled instances, respectively. This indicates that the smaller value of φ can focus the search on better regions, and it should be appropriately increased as the problem scale increases to keep a suitable search range. Considering both computational efficiency and solution quality, the suggested parameter values of MCEDA are determined as follows: $popsize$= 50, φ=0.1, γ=0.3 for the small-scaled cases, and $popsize$= 50, φ=0.2, γ=0.3 for the large-scaled instances.



(a) Factor level trends for small-scaled instances

(b) Factor level trends for large-scaled instances

Fig. 7. Factor level trends plots for parameters.

Table 6
Comparison results of MCEDA and its two variants on large-scaled instances.

		ρ=1			ρ=5			ρ=10		
		$MCEDA_{v1}$	$MCEDA_{v2}$	MCEDA	$MCEDA_{v1}$	$MCEDA_{v2}$	MCEDA	$MCEDA_{v1}$	$MCEDA_{v2}$	MCEDA
F	4	0.047	0.162	**−0.009**	0.042	0.153	**−0.012**	0.031	0.146	**−0.014**
	6	0.105	0.216	**−0.001**	0.094	0.207	**−0.003**	0.086	0.191	**−0.005**
	8	0.109	0.211	**−0.002**	0.101	0.201	**−0.004**	0.091	0.194	**−0.005**
S	30	0.038	0.159	**−0.008**	0.031	0.151	**−0.011**	0.023	0.142	**−0.014**
	40	0.073	0.185	**−0.004**	0.065	0.178	**−0.006**	0.056	0.171	**−0.008**
	50	0.118	0.253	**−0.001**	0.106	0.246	**−0.002**	0.094	0.233	**−0.004**
n	100	0.062	0.193	**−0.003**	0.053	0.184	**−0.006**	0.045	0.175	**−0.008**
	200	0.083	0.206	**−0.003**	0.077	0.193	**−0.004**	0.063	0.188	**−0.006**
	500	0.029	0.141	**−0.012**	0.024	0.136	**−0.015**	0.011	0.124	**−0.018**
Average		0.074	0.192	**−0.005**	0.066	0.183	**−0.007**	0.056	0.174	**−0.009**

Table 7
Comparison results of MCEDA and twelve heuristics on small-scaled instances.

$F \times n$	H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	VND_{H11}	VND_{H12}	VND_{H21}	VND_{H22}	VND_{H31}	VND_{H32}	MCEDA
2 × 8	14.62	13.61	6.91	5.99	13.55	12.17	1.00	**0.76**	1.00	**0.76**	1.02	0.78	**0.00**
2 × 12	13.70	12.78	5.74	5.17	11.58	11.05	0.93	**0.87**	0.93	**0.87**	0.93	**0.87**	**0.00**
2 × 16	12.52	11.40	5.77	5.10	10.00	9.16	0.73	0.55	0.72	**0.53**	1.09	**0.53**	**−0.05**
2 × 20	10.23	9.59	4.55	3.78	8.96	8.46	0.53	**0.36**	0.51	0.37	0.57	0.37	**−0.33**
2 × 24	8.71	8.34	5.00	4.74	7.54	7.15	0.54	**0.21**	0.54	**0.21**	0.54	**0.21**	**−0.50**
3 × 8	11.35	9.96	4.57	3.15	8.92	7.79	1.09	**0.70**	1.15	0.76	1.15	0.76	**0.00**
3 × 12	9.96	9.13	3.03	2.55	8.72	7.50	0.44	**0.28**	0.44	**0.28**	0.44	**0.28**	**0.00**
3 × 16	10.10	9.16	3.77	3.14	9.59	8.73	0.86	**0.56**	0.91	**0.56**	0.91	**0.56**	**−0.01**
3 × 20	9.86	8.93	2.72	2.19	8.53	7.84	**0.43**	**0.43**	**0.43**	**0.43**	**0.43**	**0.43**	**−0.01**
3 × 24	7.77	6.48	3.11	2.52	7.24	6.32	0.64	**0.33**	0.64	**0.33**	0.64	**0.33**	**−0.19**
4 × 8	9.03	8.01	2.16	1.25	6.41	5.25	1.08	**0.63**	0.99	**0.63**	0.99	**0.63**	**−0.01**
4 × 12	5.63	4.53	1.82	1.38	4.58	3.58	0.74	**0.47**	0.74	**0.47**	0.74	0.56	**−0.05**
4 × 16	7.21	6.34	2.86	2.27	6.14	5.18	0.59	**0.28**	0.59	**0.28**	0.59	**0.28**	**0.01**
4 × 20	6.80	6.00	2.96	2.61	5.66	5.04	1.10	**0.63**	1.10	**0.63**	1.10	**0.63**	**−0.01**
4 × 24	5.14	4.43	2.02	1.60	4.87	4.19	0.57	**0.26**	0.57	**0.26**	0.57	**0.26**	**−0.05**
Average	9.51	8.58	3.80	3.16	8.15	7.29	0.75	**0.49**	0.75	**0.49**	0.78	*0.50*	**−0.08**

Table 8
Comparison results of MCEDA and twelve heuristics on large-scaled instances.

		H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	VND_{H11}	VND_{H12}	VND_{H21}	VND_{H22}	VND_{H31}	VND_{H32}	MCEDA
F	4	5.57	5.09	0.32	0.19	2.96	2.56	0.06	0.03	0.05	**0.01**	0.05	**0.01**	**−0.014**
	6	3.77	3.29	0.11	0.06	1.64	1.31	0.03	0.01	0.02	**0.00**	0.02	**0.00**	**−0.005**
	8	3.09	2.66	0.04	0.02	1.21	0.93	0.02	**0.00**	0.01	**0.00**	0.01	**0.00**	**−0.005**
S	30	3.78	3.34	0.21	0.11	2.23	1.86	0.03	**0.01**	0.04	**0.01**	0.04	**0.01**	**−0.014**
	40	3.30	3.85	0.15	0.10	1.94	1.62	0.04	0.02	0.02	**0.01**	0.02	**0.01**	**−0.008**
	50	3.36	3.85	0.11	0.05	1.65	1.32	0.04	0.01	0.02	**0.00**	0.02	**0.00**	**−0.004**
n	100	6.30	5.61	0.17	0.08	2.02	1.58	0.05	0.02	0.03	**0.01**	0.03	**0.01**	**−0.008**
	200	3.76	3.28	0.15	0.07	1.92	1.55	0.03	0.01	0.02	**0.00**	0.02	**0.00**	**−0.006**
	500	2.37	2.16	0.14	0.10	1.87	1.67	0.03	**0.01**	0.03	**0.01**	0.03	**0.01**	**−0.018**
Average		4.14	3.68	0.16	0.09	1.94	1.60	0.04	**0.01**	0.03	**0.01**	0.03	**0.01**	**−0.009**

4.3. Performance analysis of key components in mceda

In this subsection, the infiuence of the speedup strategy and the VND method proposed in Section 3.3 on MCEDA are investigated to demonstrate their contributions. For this purpose, two variants $MCEDA_{v1}$ and $MCEDA_{v2}$ are compared with MCEDA under three stopping criteria. $MCEDA_{v1}$ removes both the speedup strategies and the perturbation phase from MCEDA, while $MCEDA_{v2}$ eliminates both the speedup strategies and the VND method from MCEDA. The test results given in Table 6 show that MCEDA is superior to its two variants on most instances in terms of both ARPD and SD. For the runtime factor ρ=1, 5, 10, MCEDA yields the lowest ARPD for all instances.

Furthermore, the analysis of variance (ANOVA) technique is used to analyze the results. The three hypotheses (i.e., normality, homogeneity of variance and independence of the residuals) of ANOVA are accepted in the experiments. Following Pan and Ruiz [33] and many others, we conduct a multifactor ANOVA to examine whether the differences between MCEDA and its variants are statistically significant. The means plots with 95% Tukey's HSD confidence intervals for the three compared algorithms are drawn in Fig. 8. As shown in Fig. 8, MCEDA significantly outperforms its two variants, which testifies the effectiveness and efficiency of both the proposed VND method and the speedup strategies.

4.4. Comparisons of mceda with twelve effective heuristics

This subsection provides the comparison results of MCEDA with twelve state-of-the-art constructive heuristics (i.e., H_{11}-H_{32}, and VND_{H11}-VND_{H32}) developed by Hatami et al. [17] on two sets of instances. Each of H_{11}–H_{32} constructs a complete job sequence by using the priority rule or heuristic rule, i.e. the smallest processing time (SPT) rule or the Framinan and Leisten (FL) heuristic, and then all jobs in the sequence are assigned to the specific factories by using the NR1 or NR2 rules. In addition, each of VND_{H11}-VND_{H32} further employs some local searches to improve the performance of the corresponding heuristic. The comparison results are listed in Tables 7–9, where Tables 7 and 8 provides the results on small-scaled and large-scaled instances respectively

Table 9
CPU times (second) of MCEDA and twelve heuristics on large-scaled instances.

		H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	VND_{H11}	VND_{H12}	VND_{H21}	VND_{H22}	VND_{H31}	VND_{H32}	MCEDA
F	4	0.01	0.01	0.01	0.01	0.01	0.01	3.39	6.79	2.90	7.67	2.55	42.87	26.26
	6	0.01	0.01	0.01	0.01	0.01	0.01	3.49	7.73	2.85	8.94	1.95	6.11	25.35
	8	0.01	0.01	0.01	0.01	0.01	0.01	3.26	9.56	1.86	10.21	1.83	20.64	28.36
S	30	0.01	0.01	0.02	0.02	0.01	0.01	3.64	8.05	3.14	11.00	2.70	45.20	55.42
	40	0.01	0.01	0.01	0.01	0.01	0.01	3.59	7.12	2.45	8.05	1.96	5.54	53.38
	50	0.01	0.01	0.01	0.01	0.01	0.01	3.91	8.91	2.02	7.77	1.66	18.88	53.54
n	100	0.01	0.01	0.01	0.01	0.01	0.01	1.09	2.84	0.27	0.72	0.24	0.43	5.88
	200	0.01	0.01	0.01	0.01	0.01	0.01	2.02	3.85	0.58	2.22	0.66	1.37	15.46
	500	0.03	0.02	0.03	0.04	0.03	0.02	8.03	17.39	6.76	23.88	5.41	67.81	75.89
Average		0.01	0.01	0.01	0.01	0.01	0.01	3.71	8.03	2.54	8.94	2.11	23.20	37.73

Table 10
Pair-wise *t*-test results of MCEDA and three best heuristics on two scaled sets.

Instances	Algorithm	Mean	Std. deviation	Std. Error Mean	Lower	Upper	*t*	*df*	Significance (2-tailed)
Small-scaled instances	*t*-test (VND_{H12}, MCEDA)	.56438	.17663	.04416	.47025	.65850	12.781	15	**.000**
	t-test (VND_{H22}, MCEDA),	.56750	.18068	.04517	.47122	.66378	12.563	15	**.000**
	t-test (VND_{H32}, MCEDA)	.57500	.18192	.04548	.47806	.67194	12.643	15	**.000**
	t-test (VND_{H12}, VND_{H22})	−0.00313	.01621	.00405	−0.01177	.00552	−0.771	15	.453
	t-test (VND_{H12}, VND_{H32})	−0.01063	.02695	.00674	−0.02499	.00374	−1.577	15	.136
	t-test (VND_{H22}, VND_{H32})	−0.00750	.02266	.00566	−0.01957	.00457	−1.324	15	.205
Large-scaled instances	*t*-test (VND_{H12}, MCEDA)	.01920	.01057	.00334	.01164	.02676	5.744	9	**.000**
	t-test (VND_{H22}, MCEDA)	.01220	.00777	.00246	.00664	.01776	4.964	9	**.001**
	t-test (VND_{H32}, MCEDA)	.01220	.00777	.00246	.00664	.01776	4.964	9	**.001**
	t-test (VND_{H12}, VND_{H22})	.00700	.00675	.00213	.00217	.01183	3.280	9	.010
	t-test (VND_{H12}, VND_{H32})	.00700	.00675	.00213	.00217	.01183	3.280	9	.010

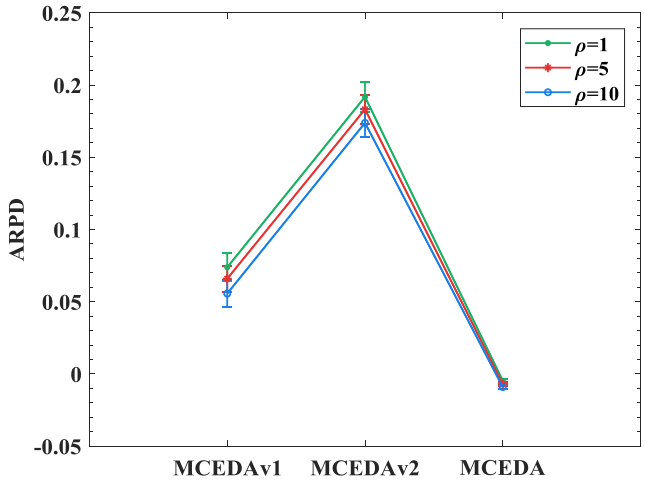


Fig. 8. Means plots with 95% Tukey's HSD confidence intervals for MCEDA and its two variants on large-scaled instances.

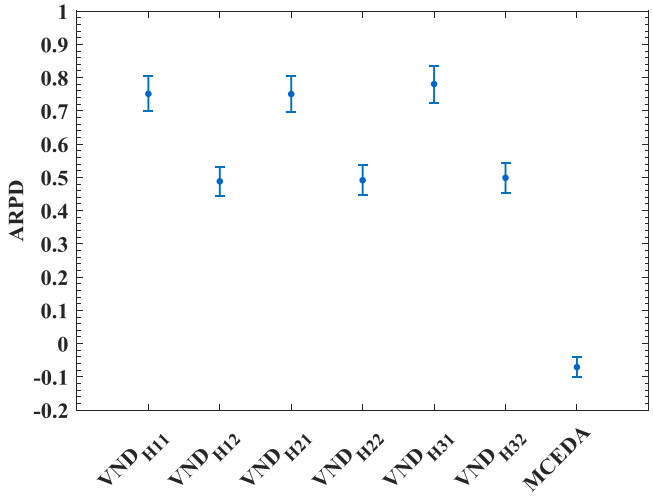


Fig. 9. Means plots with 95% Tukey's HSD confidence intervals for MCEDA and several better heuristics on small-scaled instances.

and Table 9 shows the CPU time of each algorithm. It can be seen from Tables 7 and 8 that MCEDA outperforms all the others by a considerable margin and VND_{H12}, VND_{H22} and VND_{H32} perform better than the other heuristics. The common feature of these better algorithms is the rational use of VND-based local search to perform further exploitation in solution space. Note that the average CPU time of MCEDA is 37.73 s which is acceptable in real-life environments (see Table 9).

To further check whether the differences between the test results obtained in Tables 7 and 8 are indeed statistically significant, the hypothesis tests (*t*-test) with 95% Tukey's HSD confidence interval are executed between MCEDA and each of three best heuristics, respectively. The statistic results are given in Table 10, where if the value of "Significance" in Table 10 is smaller than 0.05, the difference between the two algorithms is significant. Furthermore, the ANOVA statistical tests are carried out to analyze the results of MCEDA and several better heuristics on small-scaled and large-scaled instances, respectively. The means plots with 95% Tukey's HSD confidence intervals are shown in Figs. 9 and 10, where the non-overlapping intervals indicate that the two algorithms are significantly different in statistical significance. From Table 10 and Figs. 9 and 10, it is clear that MCEDA is statistically better than the compared heuristics.

4.5. Comparisons of mceda with eleven state-of-the-art meta-heuristics

In this subsection, to further investigate the efficiency and effectiveness of the proposed MCEDA, comprehensive comparison experiments are conducted on MCEDA and eleven state-of-the-art *meta*-heuristics, i.e., GA [18], IG_3 [22], EDAMA [19], HBBO [20], BS-HH [21], HDIWO [24], IGA [23], HVNS [47], HGA [47], HDDE [47], and CMA [48]. The parameters of these compared algorithms are properly calibrated by using the DOE technique and their setting values are listed in Table 11.

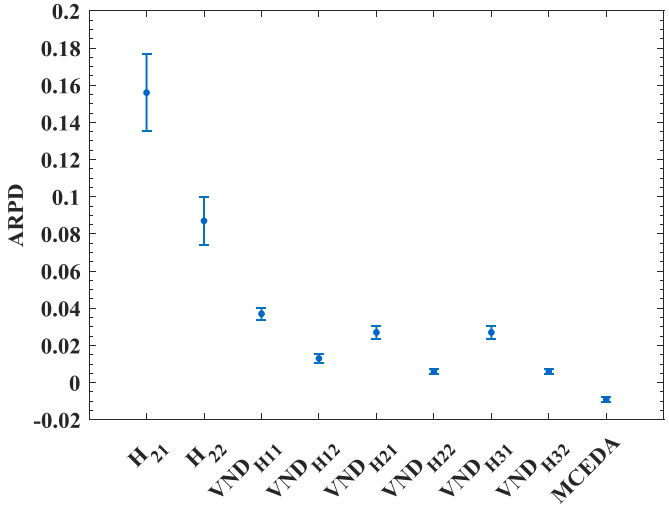


Fig. 10. Means plots with 95% Tukey's HSD confidence intervals for MCEDA and several better heuristics on large-scaled instances.

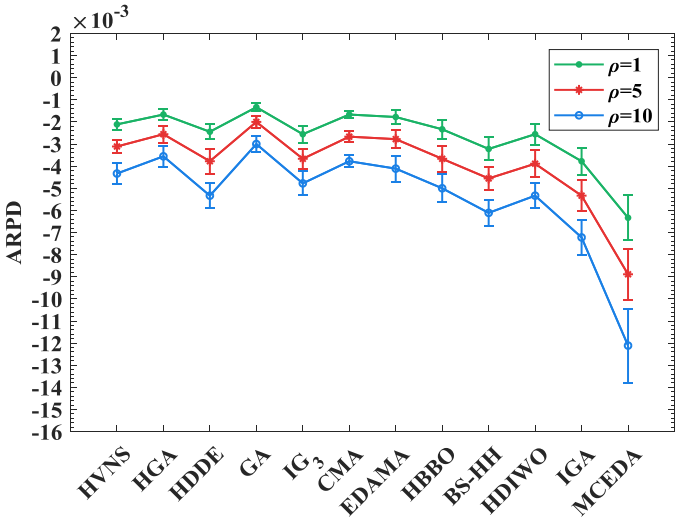


Fig. 11. Means plots with 95% Tukey's HSD confidence intervals for MCEDA and the compared algorithms under ρ=1, 5, 10.

Table 11
The parameter settings of the compared algorithms.

Algorithm	Parameters setting
HGA	$popsize = 30, P_c = 0.8, P_m = 0.1, PM = 0.1$
HDDE	$popsize = 20, F = 0.6, CR = 0.1, PM = 0.25$
GA	$popsize = 100, nkeep = 0.2, mateP = 10$
IG_3	$PR = 2, NR = 2, INI = CH_{21}, d = 5\%$
CMA	$popsize = 50, LS = 50$
EDAMA	$popsize = 50, \eta = 10, \alpha = 0.3, \gamma = 0.25$
HBBO	$popsize = 50, m_{\max} = 0.1, \lambda_{\max} = 1, \mu_{\max} = 1$
BS-HH	$popsize = 50, \lambda = 5, r_{mix} = 1.0, \xi = 0.8$
HDIWO	$popsize = 10, \delta = 5, PS_{\max} = 50, S_{\max} = 20, \rho = 0.9$
IGA	T= 0.5,d= 4

The test results under ρ=1, 5, 10 (see Section 4.1) are obtained and reported in Tables 12–14, respectively. It can be observed from Tables 12–14 that the proposed MCEDA achieves the best results for all instances. The reason is that MCEDA's global search guides the search direction more effectively and its local search further utilizes twelve specific neighborhoods to execute more efficient exploitation.

To make the test results more convincing, a multifactor ANOVA is executed to verify whether the observed differences are statistically significant. The controlled factor are the type of algorithms and the running time factor ρ. The means plots with 95% Tukey's HSD confidence intervals for MCEDA and the compared algorithms under ρ=1, 5, 10 are shown in Fig. 11. These confidence intervals in Fig. 11 suggest that MCEDA is statistically better than the other algorithms regardless of the different values of ρ.

The Gantt chart of the obtained best solution for the large-scaled instance I_100_10_4_30_5 is depicted in Fig. 12. As can be seen from Fig. 12, the assembly operation of the first product is performed as early as possible and the assembly operations of all products on the assembly machine M_A are seamlessly connected, which manifests that MCEDA can obtain high-quality solutions. Moreover, MCEDA has updated the best-known solutions of 91 small-scaled instances and 123 large-scaled instances (see Appendix A). Based on the test results of Sections 4.4 and 4.5, it can be concluded that the proposed MCEDA is a new state-of-the-art algorithm for tackling the DAPFSP.

5. Conclusions

This paper proposed a novel matrix-cube-based estimation of distribution algorithm (MCEDA) to address the DAPFSP with makespan criterion. The considered problem has many real-practice applications in modern industries. To the best of our knowledge, this is the first work on the application of multidimensional EDA to such an important problem. Firstly, a matrix cube was designed to reserve the valuable information of excellent individuals. Each two-dimensional matrix in it was utilized to record the ordinal numbers of jobs and the corresponding blocks at each position, and the hierarchical structure of it was used to save the whole order information. This design is more reasonable than those in the existing literature. Secondly, a probabilistic model based on the designed matrix cube and a special sampling method were proposed to generate new individuals and guide global search to promising regions. Thirdly, a fast VND constructed with twelve neighborhood operators and two speed-up strategies was devised to perform rich and efficient

Table 12
Comparison results of MCEDA and eleven state-of-the-art *meta*-heuristics on large-scaled instances (n×(m/2)×1 milliseconds).

		HVNS[47]	HGA[47]	HDDE[47]	GA[18]	IG_3[22]	CMA[48]	EDAMA[19]	HBBO[20]	BS-HH[21]	HDIWO[24]	IGA[23]	MCEDA
F	4	−0.003	−0.002	−0.004	−0.002	−0.004	−0.002	−0.003	−0.005	−0.006	−0.005	**−0.007**	**−0.010**
	6	−0.002	−0.001	−0.002	−0.001	−0.003	−0.001	−0.002	−0.002	**−0.004**	−0.002	−0.003	**−0.006**
	8	**−0.001**	**−0.002**	**−0.001**	**−0.001**	**−0.001**	**−0.002**	**−0.001**	**−0.002**	**−0.002**	**−0.001**	**−0.002**	**−0.002**
S	30	−0.002	−0.001	−0.003	−0.002	−0.002	−0.002	−0.003	−0.003	**−0.004**	−0.002	**−0.004**	**−0.008**
	40	−0.003	−0.002	−0.003	−0.001	−0.003	−0.002	−0.001	−0.003	**−0.004**	−0.002	**−0.004**	**−0.009**
	50	−0.001	−0.001	−0.001	−0.001	−0.001	−0.001	−0.001	−0.001	**−0.002**	**−0.002**	−0.001	**−0.004**
n	100	**−0.002**	**−0.002**	**−0.003**	−0.001	**−0.002**	**−0.002**	−0.001	−0.001	**−0.002**	**−0.002**	**−0.003**	**−0.003**
	200	−0.002	−0.001	−0.002	−0.001	−0.003	−0.001	−0.001	−0.001	−0.001	−0.002	**−0.004**	**−0.005**
	500	−0.003	−0.003	−0.003	−0.002	−0.004	−0.002	−0.003	−0.003	−0.004	−0.005	**−0.006**	**−0.010**
Average		−0.002	−0.002	−0.002	−0.001	−0.003	−0.002	−0.002	−0.002	−0.003	−0.003	**−0.004**	**−0.006**

Table 13
Comparison results of MCEDA and eleven state-of-the-art *meta*-heuristics on large-scaled instances ($n\times(m/2)\times5$ milliseconds).

		HVNS[47]	HGA[47]	HDDE[47]	GA[18]	IG_3[22]	CMA[48]	EDAMA[19]	HBBO[20]	BS-HH[21]	HDIWO[24]	IGA[23]	MCEDA
F	4	−0.004	−0.003	−0.006	−0.003	−0.005	−0.003	−0.005	−0.007	−0.007	−0.007	**−0.008**	**−0.013**
	6	−0.003	−0.002	−0.003	−0.002	−0.004	−0.002	−0.002	−0.003	−0.004	−0.003	**−0.005**	**−0.008**
	8	−0.002	−0.002	−0.002	−0.001	−0.002	**−0.003**	−0.002	**−0.003**	**−0.004**	−0.002	**−0.003**	**−0.004**
S	30	−0.003	−0.003	−0.005	−0.003	−0.003	−0.004	−0.004	−0.005	**−0.006**	−0.004	**−0.006**	**−0.011**
	40	−0.003	−0.002	−0.005	−0.002	−0.004	−0.002	−0.002	−0.004	−0.005	−0.003	**−0.007**	**−0.012**
	50	**−0.003**	−0.002	−0.002	−0.001	−0.002	−0.002	−0.002	−0.002	**−0.003**	**−0.003**	**−0.003**	**−0.006**
n	100	**−0.003**	**−0.003**	**−0.003**	−0.002	**−0.003**	**−0.003**	−0.002	−0.002	**−0.003**	**−0.003**	**−0.003**	**−0.005**
	200	−0.002	−0.001	−0.003	−0.001	−0.004	−0.002	−0.002	−0.002	−0.003	−0.003	**−0.005**	**−0.008**
	500	−0.005	−0.005	−0.006	−0.003	−0.006	−0.003	−0.004	−0.005	−0.006	−0.007	**−0.008**	**−0.013**
Average		−0.003	−0.003	−0.004	−0.002	−0.004	−0.003	−0.003	−0.004	**−0.005**	−0.004	**−0.005**	**−0.009**

Table 14
Comparison results of MCEDA and eleven state-of-the-art *meta*-heuristics on large-scaled instances ($n\times(m/2)\times10$ milliseconds).

		HVNS[47]	HGA[47]	HDDE[47]	GA[18]	IG_3[22]	CMA[48]	EDAMA[19]	HBBO[20]	BS-HH[21]	HDIWO[24]	IGA[23]	MCEDA
F	4	−0.006	−0.004	−0.007	−0.005	−0.006	−0.005	−0.007	−0.008	−0.009	−0.008	**−0.010**	**−0.019**
	6	−0.003	−0.003	−0.004	−0.003	−0.005	−0.003	−0.003	−0.004	−0.005	−0.004	**−0.007**	**−0.012**
	8	−0.004	−0.003	−0.004	−0.002	−0.003	−0.003	−0.002	−0.003	**−0.005**	−0.004	**−0.005**	**−0.006**
S	30	−0.005	−0.004	−0.007	−0.004	−0.005	−0.005	−0.006	−0.007	**−0.008**	−0.006	**−0.008**	**−0.015**
	40	−0.004	−0.003	−0.006	−0.002	−0.005	−0.003	−0.004	−0.005	−0.006	−0.005	**−0.008**	**−0.015**
	50	−0.004	−0.003	−0.004	−0.002	−0.003	−0.003	−0.003	−0.004	**−0.005**	**−0.005**	**−0.005**	**−0.007**
n	100	−0.003	−0.003	**−0.004**	−0.003	−0.003	**−0.004**	−0.003	−0.003	**−0.004**	−0.003	**−0.004**	**−0.006**
	200	−0.003	−0.002	−0.004	−0.002	−0.005	−0.004	−0.003	−0.004	−0.005	−0.005	**−0.007**	**−0.011**
	500	−0.007	−0.007	−0.008	−0.004	−0.008	−0.004	−0.006	−0.007	−0.008	−0.008	**−0.011**	**−0.019**
Average		−0.004	−0.004	−0.005	−0.003	−0.005	−0.004	−0.004	−0.005	−0.006	−0.005	**−0.007**	**−0.012**

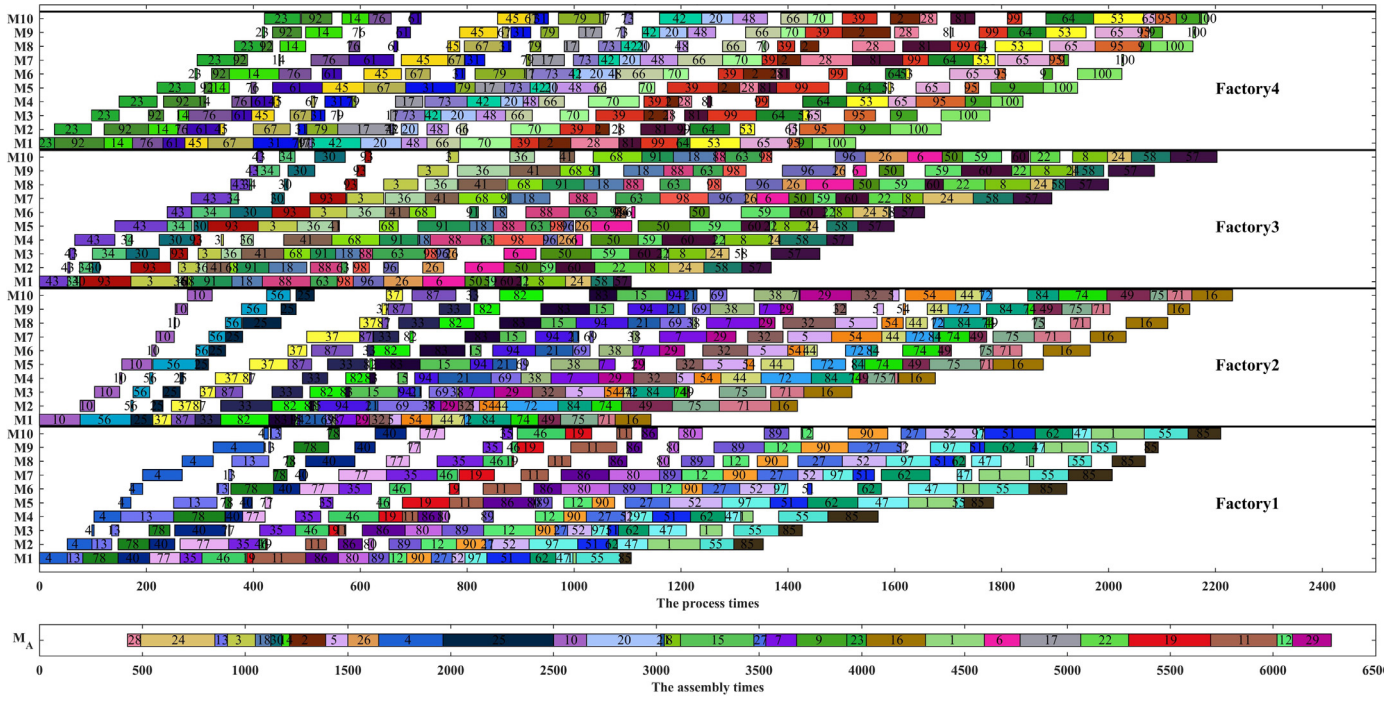


Fig. 12. Gantt chart of the new best solution obtained by MCEDA for instance I_100_10_4_30_6.

exploitation from the promising regions found by global search, which can improve the performance of MCEDA. Finally, computational comparisons and numerical results on 1710 benchmark instances demonstrated that our proposed MCEDA not only statistically outperforms the existing state-of-the-art algorithms but also can improve the current best solutions of 91 small-scaled instances and 123 large-scaled instances. Our future research is to improve the global search ability of MCEDA by considering additional machine learning schemes and to apply MCEDA to solve the energy-saving assembly scheduling problem and other kinds of scheduling problems.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Bin Qian: Funding acquisition, Investigation, Writing - review & editing, Writing - original draft, Supervision, Project administration. **Rong Hu:** Investigation, Writing - review & editing, Supervision, Project administration. **Huai-Ping Jin:** Writing - review & editing. **Ling Wang:** Supervision, Project administration.

Acknowledgment

This research is partially supported by National Natural Science Foundation of China (51665025, 61963022, 60904081), and National Natural Science Fund for Distinguished Young Scholars of China (61525304). In addition, the authors would also like to thank the anonymous reviewers for their insightful comments and suggestions that substantially improved the quality of the paper.

Appendix A

The new best-known solutions obtained by the MCEDA for two sets of benchmark instances are as follows.

Tables A1 and A2.

Table A1
New best solutions for 91 small-scaled instances.

Instance	Best known	MCEDA	ARPD	Instance	Best known	MCEDA	ARPD
I_16_2 _2_2_3	1058	1057	−0.09	I_24_3 _2_2_2	1538	1537	−0.07
I_16_3 _2_4_2	626	625	−0.16	I_24_3 _2_2_3	879	870	−1.02
I_16_4 _2_2_3	1146	1143	−0.26	I_24_3 _2_2_5	1685	1674	−0.65
I_16_4 _3_2_3	858	853	−0.58	I_24_3 _2_3_1	1077	1075	−0.19
I_16_5 _2_3_5	1334	1326	−0.60	I_24_3 _2_3_2	1417	1388	−2.05
I_16_5 _2_4_4	836	832	−0.48	I_24_3 _2_4_1	1323	1320	−0.23
I_16_5 _4_2_3	919	915	−0.44	I_24_3 _2_4_2	755	736	−2.52
I_20_2 _2_2_2	859	857	−0.23	I_24_3 _3_2_2	1232	1227	−0.41
I_20_2 _2_2_3	1462	1460	−0.14	I_24_3 _3_2_3	1999	1995	−0.20
I_20_2 _2_3_2	1206	1198	−0.66	I_24_3 _3_2_4	584	577	−1.20
I_20_2 _3_2_2	1627	1626	−0.06	I_24_3 _3_2_5	856	854	−0.23
I_20_2 _4_2_2	369	363	−1.63	I_24_3 _3_3_2	1078	1067	−1.02
I_20_2 _4_2_3	553	551	−0.36	I_24_3 _3_4_1	918	916	−0.22
I_20_3 _2_2_1	968	960	−0.83	I_24_3 _3_4_2	1042	1031	−1.06
I_20_3 _2_2_3	997	982	−1.50	I_24_3 _3_4_3	1738	1728	−0.58
I_20_3 _2_2_4	1722	1718	−0.23	I_24_3 _4_2_3	1753	1749	−0.23
I_20_3 _2_3_1	1756	1749	−0.40	I_24_4 _2_2_1	1350	1335	−1.11
I_20_3 _2_3_2	1121	1108	−1.16	I_24_4 _2_2_2	1365	1358	−0.51
I_20_3 _2_3_4	1482	1449	−2.23	I_24_4 _2_2_5	999	974	−2.50
I_20_3 _3_2_2	1561	1560	−0.06	I_24_4 _2_3_2	1225	1214	−0.90
I_20_4 _2_2_1	1229	1220	−0.73	I_24_4 _2_3_3	1711	1673	−2.22
I_20_4 _2_2_2	862	844	−2.09	I_24_4 _2_3_5	2046	2045	−0.05
I_20_4 _2_2_4	1675	1673	−0.12	I_24_4 _2_4_2	1205	1198	−0.58
I_20_4 _2_2_5	1394	1381	−0.93	I_24_4 _3_2_2	1694	1684	−0.59
I_20_4 _2_3_2	858	820	−4.43	I_24_4 _3_2_3	2458	2448	−0.41
I_20_4 _2_3_4	1725	1716	−0.52	I_24_4 _3_2_4	1844	1841	−0.16
I_20_4 _2_4_2	1282	1277	−0.39	I_24_4 _3_4_5	1480	1473	−0.47
I_20_4 _3_3_5	1137	1129	−0.70	I_24_4 _4_2_5	1356	1355	−0.07
I_20_5 _2_2_1	1478	1474	−0.27	I_24_5 _2_2_1	1625	1605	−1.23
I_20_5 _2_2_2	1258	1250	−0.64	I_24_5 _2_2_2	2136	2118	−0.84
I_20_5 _2_2_3	1500	1479	−1.40	I_24_5 _2_2_3	2049	2028	−1.02
I_20_5 _2_2_5	1589	1587	−0.13	I_24_5 _2_2_4	1764	1751	−0.74
I_20_5 _2_3_2	1181	1171	−0.85	I_24_5 _2_2_5	2100	2095	−0.24
I_20_5 _2_3_3	1499	1494	−0.33	I_24_5 _2_3_1	2333	2328	−0.21
I_20_5 _3_3_5	1127	1126	−0.09	I_24_5 _2_3_3	1340	1333	−0.52
I_20_5 _4_2_1	1012	1002	−0.99	I_24_5 _2_3_5	914	900	−1.53
I_24_2 _2_2_2	947	944	−0.32	I_24_5 _2_4_2	1054	994	−5.69
I_24_2 _2_2_3	1986	1985	−0.05	I_24_5 _2_4_3	1691	1687	−0.24
I_24_2 _2_3_2	1514	1491	−1.52	I_24_5 _3_2_2	1518	1479	−2.57
I_24_2 _2_3_3	1495	1492	−0.20	I_24_5 _3_2_3	1890	1871	−1.01
I_24_2 _2_3_5	1861	1855	−0.32	I_24_5 _3_2_4	2287	2277	−0.44
I_24_2 _3_2_4	1706	1705	−0.06	I_24_5 _3_3_3	1761	1759	−0.11
I_24_2 _3_2_5	1084	1082	−0.18	I_24_5 _3_3_4	2318	2308	−0.43
I_24_2 _4_2_2	1264	1258	−0.47	I_24_5 _4_2_2	1771	1757	−0.79
I_24_2 _4_2_5	563	554	−1.60	I_24_5 _4_2_5	1792	1791	−0.06
I_24_3 _2_2_1	1705	1692	−0.76				

Table A2
New best solutions for 123 large-scaled instances.

Instance	Best known	MCEDA	ARPD	Instance	Best known	MCEDA	ARPD
l_100_5 _4_40_8	4825	4820	−0.10	l_500_10 _4_30_7	26,650	26,649	0.00
l_100_10 _4_30_5	6296	6284	−0.19	l_500_10 _4_30_8	26,670	26,644	−0.10
l_100_20 _4_30_1	5614	5603	−0.20	l_500_10 _4_30_9	28,040	28,031	−0.03
l_100_20 _4_30_7	5626	5611	−0.27	l_500_10 _4_40_4	22,904	22,892	−0.05
l_100_20 _4_40_1	5563	5562	−0.02	l_500_10 _4_40_5	25,341	25,333	−0.03
l_100_20 _4_40_3	5907	5855	−0.88	l_500_10 _4_40_7	23,335	23,296	−0.17
l_100_20 _4_40_8	5393	5365	−0.52	l_500_10 _4_40_8	24,650	24,642	−0.03
l_100_20 _4_50_1	5417	5391	−0.48	l_500_10 _4_40_9	22,928	22,885	−0.19
l_100_20 _6_40_9	6239	6225	−0.22	l_500_10 _4_40_10	25,598	25,593	−0.02
l_200_5 _4_30_4	9309	9308	−0.01	l_500_10 _4_50_1	28,887	28,837	−0.17
l_200_5 _4_30_9	9585	9576	−0.09	l_500_10 _4_50_4	24,349	24,348	0.00
l_200_5 _4_30_10	10,759	10,754	−0.05	l_500_10 _4_50_10	21,220	21,201	−0.09
l_200_5 _4_40_1	11,214	11,211	−0.03	l_500_10 _6_30_1	28,319	28,317	−0.01
l_200_5 _4_40_2	9204	9203	−0.01	l_500_10 _6_30_2	25,022	24,997	−0.10
l_200_5 _6_40_7	9516	9510	−0.06	l_500_10 _6_30_3	28,516	28,515	0.00
l_200_5 _6_50_6	9400	9377	−0.24	l_500_10 _6_30_5	31,658	31,657	0.00
l_200_5 _8_30_9	7612	7607	−0.07	l_500_10 _6_30_6	23,584	23,569	−0.06
l_200_5 _8_40_10	9785	9768	−0.17	l_500_10 _6_30_7	22,654	22,637	−0.08
l_200_10 _4_30_10	10,904	10,903	−0.01	l_500_10 _6_30_9	30,512	30,479	−0.11
l_200_10 _4_40_3	10,717	10,716	−0.01	l_500_10 _6_40_1	26,234	26,217	−0.06
l_200_10 _6_30_9	9082	9079	−0.03	l_500_10 _6_40_5	23,791	23,784	−0.03
l_200_20 _4_30_1	11,639	11,635	−0.03	l_500_10 _6_50_7	25,590	25,553	−0.14
l_200_20 _4_30_4	10,033	10,029	−0.04	l_500_10 _6_50_10	26,910	26,909	0.00
l_200_20 _4_30_10	11,152	11,137	−0.13	l_500_10 _8_30_1	20,965	20,952	−0.06
l_200_20 _4_40_6	9320	9289	−0.33	l_500_10 _8_30_5	21,922	21,886	−0.16
l_200_20 _4_50_4	10,926	10,916	−0.09	l_500_10 _8_30_10	17,054	17,031	−0.13
l_200_20 _4_50_5	11,939	11,918	−0.18	l_500_10 _8_40_3	24,971	24,968	−0.01
l_200_20 _4_50_8	9532	9525	−0.07	l_500_10 _8_40_6	27,705	27,702	−0.01
l_200_20 _6_30_2	10,983	10,951	−0.29	l_500_10 _8_40_10	24,055	24,050	−0.02
l_200_20 _6_30_10	11,873	11,866	−0.06	l_500_20 _4_30_10	25,343	25,342	0.00
l_200_20 _6_40_4	10,793	10,681	−1.04	l_500_20 _4_30_3	23,096	23,069	−0.12
l_200_20 _6_50_5	10,408	10,394	−0.13	l_500_20 _4_30_4	26,810	26,791	−0.07
l_200_20 _8_30_3	9586	9577	−0.09	l_500_20 _4_30_8	27,519	27,499	−0.07
l_200_20 _8_30_7	12,692	12,678	−0.11	l_500_20 _4_30_9	24,783	24,769	−0.06
l_500_5 _4_30_1	27,845	27,841	−0.01	l_500_20 _4_30_10	24,395	24,377	−0.07
l_500_5 _4_30_3	24,811	24,794	−0.07	l_500_20 _4_40_1	25,222	25,202	−0.08
l_500_5 _4_30_5	28,052	28,050	−0.01	l_500_20 _4_40_3	24,585	24,583	−0.01
l_500_5 _4_30_6	29,111	29,094	−0.06	l_500_20 _4_40_8	21,272	21,267	−0.02
l_500_5 _4_30_7	27,785	27,772	−0.05	l_500_20 _4_40_10	28,057	28,020	−0.13
l_500_5 _4_30_8	23,787	23,770	−0.07	l_500_20 _4_50_2	23,868	23,840	−0.12
l_500_5 _4_30_9	20,595	20,576	−0.09	l_500_20 _4_50_3	25,697	25,687	−0.04
l_500_5 _4_30_10	30,191	30,189	−0.01	l_500_20 _4_50_4	26,082	26,074	−0.03
l_500_5 _4_40_3	25,925	25,916	−0.03	l_500_20 _4_50_5	23,666	23,663	−0.01
l_500_5 _4_40_6	29,700	29,693	−0.02	l_500_20 _4_50_6	24,438	24,431	−0.03
l_500_5 _4_40_7	28,543	28,538	−0.02	l_500_20 _4_50_8	25,599	25,582	−0.07
l_500_5 _4_50_6	24,512	24,492	−0.08	l_500_20 _4_50_10	27,350	27,317	−0.12
l_500_5 _6_30_2	28,170	28,165	−0.02	l_500_20 _6_30_1	24,512	24,481	−0.13
l_500_5 _6_30_6	26,723	26,715	−0.03	l_500_20 _6_30_3	26,019	25,998	−0.08
l_500_5 _6_40_1	22,655	22,650	−0.02	l_500_20 _6_30_6	26,381	26,341	−0.15
l_500_5 _6_40_2	22,361	22,357	−0.02	l_500_20 _6_30_8	26,169	26,138	−0.12
l_500_5 _6_40_4	24,017	24,010	−0.03	l_500_20 _6_30_9	27,023	26,984	−0.14
l_500_5 _6_50_10	25,661	25,647	−0.05	l_500_20 _6_30_10	28,621	28,608	−0.05
l_500_5 _8_30_1	24,383	24,377	−0.02	l_500_20 _6_40_2	25,595	25,589	−0.02
l_500_5 _8_30_3	29,878	29,875	−0.01	l_500_20 _6_40_9	24,398	24,385	−0.05
l_500_5 _8_30_4	29,782	29,772	−0.03	l_500_20 _6_50_4	30,028	29,944	−0.28
l_500_5 _8_30_5	26,461	26,453	−0.03	l_500_20 _6_50_6	27,009	27,000	−0.03
l_500_5 _8_30_8	27,737	27,731	−0.02	l_500_20 _8_30_5	25,654	25,552	−0.40
l_500_5 _8_50_7	24,103	24,099	−0.02	l_500_20 _8_30_7	22,579	22,568	−0.05
l_500_10 _4_30_1	26,749	26,741	−0.03	l_500_20 _8_30_8	22,575	22,571	−0.02
l_500_10 _4_30_2	22,034	22,016	−0.08	l_500_20 _8_40_5	24,129	24,120	−0.04
l_500_10 _4_30_4	22,961	22,945	−0.07	l_500_20 _8_40_8	24,699	24,667	−0.13
l_500_10 _4_30_5	29,194	29,163	−0.11				

References

[1] B. Qian, L. Wang, R. Hu, W.L. Wang, D.X. Huang, X. Wang, A hybrid differential evolution method for permutation flow-shop scheduling, Int. J. Adv. Manuf. Tech. 38 (7–8) (2008) 757–777.

[2] Q.K. Pan, R. Ruiz, A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime, Comput. Oper. Res. 40 (1) (2013) 117–128.

[3] J.S. Neufeld, J.N.D. Gupta, U. Buscher, A comprehensive review of flowshop group scheduling literature, Comput. Oper. Res. 70 (1) (2016) 56–74.

[4] V. Fernandez-Viagas, R. Ruiz, J.M. Framinan, A new vision of approximate methods for the permutation flowshop to minimise makespan: state-of-the-art and computational evaluation, Eur. J. Oper. Res. 257 (3) (2017) 707–721.

[5] M.R. Garey, J.R. Sethi, The complexity of flowshop and jobshop scheduling, Math. Oper. Res. 1 (2) (1976) 117–129.

[6] T. Gonzalez, S. Sahni, Flowshop and jobshop schedules: complexity and approximation, Oper. Res. 26 (1) (1978) 36–52.

[7] B. Naderi, R. Ruiz, The Distributed Permutation Flowshop Scheduling Problem, Comput. Oper. Res. 37 (4) (2010) 754–768.

[8] J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (3) (2013) 641–651.

[9] 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 (1) (2013) 387–396.
[10] W.Z. Duan, Z.Y. Li, M.C. Ji, Y.X. Yang, S.Y. Wang, B. Liu, A hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling with flowline eligibility, in: Proceeding of the IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 2581–2587.
[11] W.Z. Duan, Z.Y. Li, Y.X. Yang, B. Liu, K.Y. Wang, EDA based probabilistic memetic algorithm for distributed blocking permutation flowshop scheduling with sequence dependent setup time, in: Proceeding of the IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 992–999.
[12] S.W. Lin, K.C. Ying, C.Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, Int. J. Prod. Res. 51 (16) (2013) 5029–5038.
[13] V. Fernandez-Viagas, J.M. Framinan, A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 53 (4) (2015) 1111–1123.
[14] Z.Y. Li, W.Z. Duan, M.C. Ji, Y.X. Yang, S.Y. Wang, B. Liu, The distributed permutation flowshop scheduling problem with different transport timetables and loading capacities, in: Proceeding of the IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 2433–2437.
[15] 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.
[16] J. Deng, L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, Swarm. Evol. Comput. 32 (2017) 121–131.
[17] S. Hatami, R. Ruiz, C. Andrés-Romano, The distributed assembly permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (17) (2013) 5292–5308.
[18] X. Li, X. Zhang, M. Yin, J. Wang, A genetic algorithm for the distributed assembly permutation flowshop scheduling problem, Evolutionary Computation (CEC), in: Proceeding of the 2015 IEEE Congress on IEEE, 2015, pp. 3096–3101.
[19] S.Y. Wang, L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, IEEE Trans. Syst., Man, Cybern., Syst. 46 (1) (2016) 139–149.
[20] J. Lin, S. Zhang, An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem, Comput. Ind. Eng. 97 (2016) 128–136.
[21] J. Lin, Z.J. Wang, X. Li, A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem, Swarm. Evol. Comput. 36 (2017) 124–135.
[22] S. Hatami, R. Ruiz, C. Andrés-Romano, Heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times, Int. J. Prod. Econ. 169 (2015) 76–88.
[23] Q.K. Pan, L. Gao, X.Y. Li, J. Framinan, Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem, Appl. Soft. Comput. 81 (2019).
[24] H.Y. Sang, Q.K. Pan, J.Q. Li, P. Wang, Y.Y. Han, K.Z. Gao, P. Duan, Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion, Swarm. Evol. Comput. 44 (2019) 64–73.
[25] Y.X. Yang, P. Li, S.Y. Wang, B. Liu, Y.L. Luo, Scatter search for distributed assembly flowshop scheduling to minimize total tardiness, in: Proceeding of the IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 861–868.
[26] L.A. Wolsey, Integer Programming, Wiley Press, New York, 1998.
[27] A. Schrijver, Combinatorial Optimization: Polyhedra and Efficiency, Springer, 2003.
[28] D.S. Chen, R. Batson, Y. Dang, Applied Integer Programming, Wiley Press, New York, 2010.
[29] P. Larranaga, J.A. Lozano, in: Estimation of Distribution algorithms: A new Tool For Evolutionary Computation, Springer Science & Business Media, 2002, p. 2.
[30] J. Ceberio, E. Irurozki, A. Mendiburu, J.A. Lozano, A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems, Prog. Artif. Intell. 1 (1) (2012) 103–117.
[31] A. Salhi, José Antonio Vázquez Rodríguez, Q. Zhang, An estimation of distribution algorithm with guided mutation for a complex flow shop scheduling problem, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, pp. 570–576.
[32] B. Jarboui, M. Eddaly, P. Siarry, An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems, Comput. Oper. Res. 36 (9) (2009) 2638–2646.
[33] Q.K. Pan, R. Ruiz, An estimation of distribution algorithm for lot-streaming flow shop problems with setup times, Omega 40 (2) (2012) 166–180.
[34] L. Wang, S. Wang, Y. Xu, G. Zhou, M. Liu, A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem, Comput. Ind. Eng. 62 (4) (2012) 917–926.
[35] A. Tiwari, P.C. Chang, M.K. Tiwari, N.J. Kollanoor, A Pareto block-based estimation and distribution algorithm for multi-objective permutation flow shop scheduling problem, Int. J. Prod. Res. 53 (3) (2015) 793–834.
[36] W. Shao, D. Pi, Z. Shao, A hybrid discrete optimization algorithm based on teaching-probabilistic learning mechanism for no-wait flow shop scheduling, Knowl-Based Syst 107 (2016) 219–234.
[37] Y.M. Chen, M.C. Chen, P.C. Chang, S.H. Chen, Extended artificial chromosomes genetic algorithm for permutation flowshop scheduling problems, Comput. Ind. Eng. 62 (2) (2012) 536–545.
[38] B. Qian, Z.C. Li, R. Hu, A copula-based hybrid estimation of distribution algorithm for m-machine reentrant permutation flow-shop scheduling problem, Appl. Soft. Comput. 61 (2017) 921–934.
[39] Y. Han, J. Li, H. Sang, Y. Liu, K. Gao, Q.K. Pan, Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time, Appl. Soft. Comput. 93 (2020) 106343.
[40] J. Grabowski, M. Wodecki, A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion, Comput. Oper. Res. 31 (11) (2004) 1891–1909.
[41] E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, Eur. J. Oper. Res. 47 (1) (1990) 65–74.
[42] P. Hansen, N. Mladenović, Variable neighborhood search: principles and applications, Eur. J. Oper. Res. 130 (3) (2001) 449–467.
[43] Q. Zhang, H. Muhlenbein, On the convergence of a class of estimation of distribution algorithms, IEEE Trans. Evolut. Comput. 8 (2) (2004) 127–136.
[44] F. Zhao, F. Xue, Y. Zhang, W. Ma, C. Zhang, H. Song, A hybrid algorithm based on self-adaptive gravitational search algorithm and differential evolution, Expert. Syst. Appl. 113 (2018) 515–530.
[45] S. Chen, G.H. Peng, X.S. He, X.S. Yang, Global convergence analysis of the bat algorithm using a markovian framework and dynamical system theory, Expert. Syst. Appl. 114 (2018) 173–182.
[46] D.C. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, 2008.
[47] F. Xiong, K. Xing, F. Wang, H. Lei, L. Han, Minimizing the total completion time in a distributed two stage assembly system with setup times, Comput. Oper. Res. 47 (2014) 92–105.
[48] J. Deng, L. Wang, S.Y. Wang, X.L. Zheng, A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem, Int. J. Prod. Res. 54 (12) (2016) 3561–3577.