A Matrix-Cube-Based Estimation of Distribution Algorithm for No-Wait Flow-Shop Scheduling With Sequence-Dependent Setup Times and Release Times

Bin Qian, Zi-Qi Zhang, Rong Hu, Huai-Ping Jin, and Jian-Bo Yang

***Abstract*—The no-wait flow-shop scheduling problem (NFSSP) with sequence-dependent setup times (SDSTs) and release times (RTs) is applicable in many areas, such as steel production, food processing, and chemical processing. Estimation of the distribution algorithm (EDA) has recently been recognized as a prominent metaheuristic methodology in the field of evolutionary computation due to its excellent performance of global exploration. In this article, an innovative matrix-cube-based (i.e., 3-D) EDA (MCEDA) is first proposed to minimize the total earliness and tardiness (TET) of the NFSSP with SDSTs and RTs. This problem is NP-hard in the strong sense. First, a 3-D matrix cube is devised to learn the valuable information from promising solutions or excellent individuals. Second, an EDA model or probabilistic model based on the matrix cube and a special sampling method is presented to perform effective exploration in solution space and find promising regions. Third, based on a series of newly defined subneighborhoods, a new local search with both a speed-up scanning method and one search strategy is developed to execute exploitation from promising regions. Fourth, a speed-up evaluation method based on the problem's property is designed to reduce the computational complexity for calculating criterion and accelerate the search process. Owing to the reasonable hybridization of exploration and exploitation, MCEDA can perform very efficient search in solution space. Extensive test results on instances of such a just-in-time problem first show that MCEDA can achieve better solution than state-of-the-art algorithms in obviously less computation time. Additional experiments on instances of various NFSSPs further confirm the efficiency and robustness of MCEDA.**

***Index Terms*—Estimation of distribution algorithm (EDA), exploration and exploitation, fast local search, no-wait flow-shop scheduling problem (NFSSP), probabilistic model.**

Manuscript received 12 March 2022; accepted 10 August 2022. Date of publication 2 September 2022; date of current version 16 February 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62173169, Grant 61963022, and Grant 51665025; and in part by the Basic Research Key Project of Yunnan Province under Grant 202201AS070030. This article was recommended by Associate Editor S. Xie. *(Corresponding author: Rong Hu.)*

Bin Qian, Zi-Qi Zhang, Rong Hu, and Huai-Ping Jin are with the School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China (e-mail: ronghu@vip.163.com).

Jian-Bo Yang is with the Alliance Manchester Business School, The University of Manchester, Manchester M13 9SS, U.K.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TSMC.2022.3198829.

Digital Object Identifier 10.1109/TSMC.2022.3198829

I. Introduction

With the acceleration of economic globalization and the popularity of Internet, more and more enterprises are aware of the importance of quickly responding to customer needs and requirements. As an effective operation management approach, the just-in-time (JIT) system has been employed by many enterprises for maintaining customer satisfaction and reducing extra costs. In JIT environment, jobs which are completed earlier than their due dates may result in inventory holding costs, opportunity costs, and deterioration of perishable goods. Contrarily, jobs which are tardy may cause lost sales, contract penalties, and backlogging costs [1]. Nowadays, JIT management plays an important role in production, service, and manufacturing systems. Meanwhile, proper scheduling leads to increased efficiency and profitability [2], [3], [4], [5]. So, it is important to develop effective and efficient scheduling algorithms to minimize the total earliness and tardiness (TET).

The no-wait flow-shop scheduling problem (NFSSP) is a kind of broadly studied production scheduling problem with a strong engineering background. To fulfill the no-wait restrictions, each job is required to be processed continuously from start to end without waiting either on or between machines. In many real-life NFSSPs, sequence-dependent setup times (SDSTs) and release times (RTs) are two very common constraints, which have received increasing attention over the past decade [6], [7], [8]. An example of the NFSSP with SDSTs and RTs arises in the downstream process of some chinese steel enterprises, where ingots produced in the upstream plants are transported to a downstream plant (i.e., a hot rolling plant) and then undergo several uninterrupted operations, i.e., reheating, rough rolling, and precision rolling. The RT of each ingot is its transport arrival time. Each machine needs to be adjusted before processing different jobs. The adjusting time or setup time depends on material and final product of the current job and its previous one. Despite its important practical applications, there is less research considering the NFSSP with SDSTs and RTs [9], [10]. Thus, this article aims to minimize the TET for such problem.

Since the single machine scheduling problem with the TET criterion is *NP*-hard in the strong sense [11] and it reduces to the TET-NFSSP with SDSTs and RTs, the latter problem is also strongly NP-hard. The analyses of problem complexity can be found in Part 1 of the online supplementary material

(see the website address in Section IV-A). Standard mathematical algorithms, such as mixed integer programming and dynamic programming for NP-hard discrete problems are often of limited use because of their long running time. As a result, metaheuristic algorithms have been proposed, which provide satisfactory solutions in reasonable running time. However, when explaining why metaheuristic algorithms can obtain satisfactory solutions by using a small number of individuals to run tens or hundreds of generations, the existing literature only mentioned the mechanism and characteristics of each algorithm, but ignored the important influence of the permutation-based model on the solution quality. This leads many new researchers to pursue innovations in unimportant techniques too much. In fact, the effectiveness of metaheuristic algorithms is decided by the solution space of the permutation-based model and the mechanism of each algorithm.

In terms of the solution space of the permutation-based model, the variation range of the objective value of the scheduling problem is far smaller than the scale of the solution space. For example, for the FSSP with the objective of minimizing the makespan or Cmax (i.e., the most studied scheduling problem in the literature), if there are 60 jobs and 5 machines, and the processing time of each job is a random number evenly distributed between [1, 100], then the objective value is in the range (64, 6400), and the scale of the solution space is 60!. Here, 64 and 6400 are the objective values when each job's processing time is set to 1 and 100, respectively. In fact, since each job's processing time is a random number between [1, 100], the actual range (a, b) of the objective values of any concrete instance of the FSSP above is covered by (64, 6400). That is, each specific objective value corresponds to more than 1.31×10^{78} (i.e., $60!/(b\text{-}a)>60!/(6400\text{-}64)\approx1.31\times10^{78}$) different permutations or solutions on average. This indicates that a large number of different solutions has the same objective value. For other types of scheduling problems (including the problem considered in this article), the above situation also exists.

Because of the "extremely flat" property of the above solution space, any stochastic search only needs a short time to search very small regions in the entire space (similar to the area of a needle in football field), but it can reach a wide range of objective values. Each metaheuristic algorithm adds its own optimization mechanism on the basis of stochastic search, so that it can not only search a wide range of objective values in a short time, but also its inherent optimization mechanism can promote it to search different regions with smaller objective values. Metaheuristic algorithms can obtain smaller objective values in a wider range by searching very limited regions in the solution space of the permutation-based model. This is the essential reason for its effectiveness. However, it is difficult for mathematical algorithms that need to traverse or partially traverse the solution space to do this. Thus, it is reasonable to design metaheuristic algorithms for NP-hard problems.

Estimation of distribution algorithm (EDA) is one of the effective metaheuristic algorithms. This algorithm is based on a probability model constructed from a population of excellent individuals, which has a good ability of guiding the search to the promising regions of solution space [2]. Nowadays, EDA has already been extensively applied to deal with different kinds of optimization problems [12]. The literature review on EDA for scheduling problems can be found in Part 2 of the online supplementary material (see the website address in Section IV-A).

From the literature review, it can be seen that the existing EDA-based algorithms usually adopt a 2-D probabilistic model to accumulate the information of the blocks and the jobs from each excellent individual. Here, any two consecutive jobs in an individual constitute one block. However, the 2-D structure has no extra space to preserve the position information of each block and the whole order information of all jobs. As a result, when generating a new individual, it is difficult for the sampling procedure to place the blocks in their correct positions. Moreover, the solution space determined by some specific neighborhoods or operators (i.e., Insert, *Interchange*, *Swap*, etc.,) has a big-valley landscape, where a large number of local optima is relatively close to each other and surround global optima at the bottom part of big valley [2], [13], [14], [15]. This indicates excellent individuals usually have partially similar patterns. Obviously, it is crucial and challenging to build a more reasonable structure for learning the valuable information to guide the global search direction. Hence, we devise a novel matrix-cube-based EDA (MCEDA) for the TET-NFSSP with SDSTs and RTs.

The main contributions of this article are as follows.

1) The profound reason why metaheuristic algorithms are effective is pointed out, and the role of each part in MCEDA is deeply analyzed. This is conducive to the further development of metaheuristic algorithms for discrete optimization problems.
2) A novel probabilistic model constructed from a devised matrix cube and a special sampling method is presented to generate new population. This design can quickly guide the global search to promising solutions or regions.
3) Different from most of existing local searches that are based on several common neighborhoods, a new Insert-based local search with one strategy is proposed by using a set of subneighborhoods to execute a narrow but rich exploitation.
4) Based on the problem's properties, a speed-up evaluation method and a speed-up scanning method is designed. The former is used to reduce TET's computational complexity (CC), while the latter is used to accelerate the speed of scanning subneighborhoods in local search.
5) As for the theoretical analyses, the concept of Turing reduction is utilized to prove that the problem considered is strongly NP-hard, and the CC of MCEDA is analyzed in detail.

The remainder of this article is organized as follows. Section II introduces the model of the TET-NFSSP with SDSTs and RTs. Section III details the implementation of MCEDA. Extensive comparisons are presented and discussed in Section IV. Finally, Section V provides some concluding remarks.

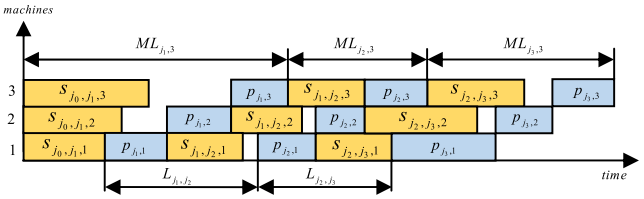

Fig. 1. Gantt chart of the NFSSP with SDSTs when $n = 3$ and $m = 3$.

II. Problem Descriptions

The permutation-based model of the NFSSP with SDSTs and RTs is given below. There are n jobs to be processed sequentially on m machines. At any moment, each machine can only process one job and preemption is prohibited. Each machine processes all jobs in the same order, and any two consecutive machines must process the same job without interruption. The setup times depend on the previous and the current jobs at each machine. In addition, an idle machine cannot process the job until it is released. The detailed notations adopted in this model are listed in Table I.

A. NFSSP With SDSTs

In the NFSSP with SDSTs, $ML_{j_i,l}$ can be calculated as follows:

$$ML_{j_i,l} = \begin{cases} \max\{s_{j_{i-1},j_i,1} + p_{j_i,1} - p_{j_{i-1},2}, s_{j_{i-1},j_i,2}\} + p_{j_i,2}, & l = 2 \\ \max\{ML_{j_i,l-1} - p_{j_{i-1},l}, s_{j_{i-1},j_i,l}\} + p_{j_i,l}, & l = 3, \ldots, m. \end{cases} \tag{1}$$

Then, C_{j_i} can be calculated as follows:

$$C_{j_i} = \sum_{k=1}^{i} ML_{j_k,m}, i = 1, \ldots, n. \tag{2}$$

Accordingly, L_{j_{i-1},j_i} can be calculated as follows:

$$L_{j_{i-1},j_i} = ML_{j_i,m} + sp_{j_{i-1}} - sp_{j_i}. \tag{3}$$

Fig. 1 shows a Gantt chart of the NFSSP with SDSTs when $n = 3$ and $m = 3$.

B. TET-NFSSP With SDSTs and RTs

In the NFSSP with SDSTs and RTs, St_{j_i} can be written as follows:

$$St_{j_i} = \begin{cases} \max\{ML_{j_i,m} - sp_{j_i}, r_{j_i}\}, & i = 1 \\ St_{j_{i-1}} + \max\{L_{j_{i-1},j_i}, r_{j_i} - St_{j_{i-1}}\}, & i = 2, \ldots, n. \end{cases} \tag{4}$$

Hence, C_{j_i}, E_{j_i}, T_{j_i}, and $TET(\boldsymbol{\pi})$ of the NFSSP with SDSTs and RTs can be calculated as follows:

$$C_{j_i} = St_{j_i} + sp_{j_i}, i = 1, \ldots, n \tag{5}$$

$$E_{j_i} = \max(d_{j_i} - C_{j_i}, 0), i = 1, \ldots, n \tag{6}$$

$$T_{j_i} = \max(C_{j_i} - d_{j_i}, 0), i = 1, \ldots, n \tag{7}$$

$$\text{TET}(\boldsymbol{\pi}) = \sum_{i=1}^{n} (E_{j_i} + T_{j_i}). \tag{8}$$

The criterion is to obtain a schedule $\boldsymbol{\pi}*$ in the set of all schedules $\prod$ such that

$$TET(\boldsymbol{\pi}^*) = \min_{\boldsymbol{\pi} \in \prod} TET(\boldsymbol{\pi}). \tag{9}$$

TABLE I
Notations Applied in Optimization Model

Symbol	Description of Symbol
$\boldsymbol{\pi}$	The schedule or permutation of jobs to be processed, $\boldsymbol{\pi} = [j_1, j_2, \ldots, j_n]$.
$p_{j_i,l}$	The processing time of job j_i on machine l ($p_{j_0,l} = 0$ for $l = 1, \ldots, m$).
sp_{j_i}	The total processing time of job j_i on all machines.
$ML_{j_i,l}$	The minimum delay on the machine l between the completion of job j_{i-1} and j_i.
C_{j_i}	The completion time of job j_i.
L_{j_{i-1},j_i}	The minimum delay on the first machine between the start of job j_{i-1} and j_i.
$s_{j_{i-1},j_i,l}$	The sequence-dependent setup time between job j_{i-1} and j_i on machine l.
r_{j_i}	The release time of job j_i ($r_{j_0} = 0$ for $i = 1, \ldots, n$).
St_{j_i}	The start processing time of job j_i on machine 1.
d_{j_i}	The due date of job j_i.
E_{j_i}	The earliness of job j_i on machine m.
T_{j_i}	The tardiness of job j_i on machine m.

Equations (1)–(9) give the model of the TET-NFSSP with SDSTs and RTs. The constraints are included in (1) and (4). More precisely, (1) requires that each job must be processed without interruptions between consecutive machines (no-wait constraint), and (4) ensures that the time for each job to start being processed on the first machine is not less than its RT (RT constraint). The decision variables are expressed as a job permutation $\boldsymbol{\pi}$. If the jobs in $\boldsymbol{\pi}$ are different and the number of jobs is n, $\boldsymbol{\pi}$ is a feasible solution.

III. MCEDA for TET-NFSSP With SDSTs and RTs

This section details the implementation of the proposed MCEDA, including the speed-up evaluation method, the matrix-cube-based global search, the problem-dependent local search, and the process of MCEDA. Meanwhile, the CC of MCEDA is analyzed. To better understand our contributions, the novelties and merits of its global and local search are also analyzed in depth.

A. Speed-Up Evaluation Method

In accordance with the no-wait property of the permutation-based model for the TET-NFSSP with SDSTs and RTs in Section II, it is clear that L_{j_{i-1},j_i} in (3) is only decided by both job j_{i-1} and job j_i. Therefore, a speed-up evaluation method can be designed to reduce the CC of evaluating TET($\boldsymbol{\pi}$). That is, L_{j_{i-1},j_i} and sp_{j_i} ($j_{i-1}, j_i \in \{1, \ldots, n\}$) can be calculated and reserved at the initial phase of the algorithm. In the global search and local search phases, they can be used as constant values. Based on the (3)–(8), the CC of calculating TET($\boldsymbol{\pi}$) can be reduced from $O(nm)$ to $O(n)$.

Actually, since the commonly used criterion to be minimized is always a function of C_{j_i} ($j_i \in \{1, \ldots, n\}$) and $C_{j_i}$of each kind of NFSSPs can be calculated by using L_{j_{i-1},j_i} and

sp_{j_i} [see (4) and (5)], the CC of calculating any common criterion (e.g., makespan, total completion time, total weighted ET, and total tardiness) for each specific NFSSP can also be reduced from $O(nm)$ to $O(n)$. Hence, the speed-up evaluation method is applied to all compared algorithms for solving different NFSSPs in Sections IV-B and IV-C.

B. Matrix-Cube-Based Global Search

A matrix-cube-based global search is presented to execute exploration in solution space. Two key components, i.e., a matrix cube and a matrix-cube-based probabilistic model, are designed at first. Then, the overall procedure of global search is described. The notations used in the presented global search but not defined in the text are provided in Table II.

1) Matrix Cube: A 3-D matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$ is devised to record the valuable information of excellent individuals. This matrix cube is important for designing a probabilistic model. The details of $\mathbf{MC^{gen}_{n\times n\times n}}$ are given as follows:

$$OneS\_MC^{\mathrm{gen,w}}_{n\times n\times n}(x,y,z)=\begin{cases}1, \text{ if } y=SubB\_j_x^{\mathrm{gen,w}}, z=SubB\_j_{x+1}^{\mathrm{gen,w}}\\ 0, \text{ else}\end{cases}$$
$$x=1,\ldots,n-1,y,z=1,\ldots,n \tag{10}$$

$$MC^{\mathrm{gen}}_{n\times n\times n}(x,y,z)=\sum_{w=1}^{\mathrm{sbpopsize}} OneS\_MC^{\mathrm{gen,w}}_{n\times n\times n}(x,y,z),$$
$$x=1,\ldots,n-1,y,z=1,\ldots,n \tag{11}$$

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

$$\mathbf{MC^{gen}_{n\times n\times n}}(\mathbf{x},\mathbf{y})=\left[MC^{\mathrm{gen}}_{n\times n\times n}(x,y,1),MC^{\mathrm{gen}}_{n\times n\times n}(x,y,2),\ldots MC^{\mathrm{gen}}_{n\times n\times n}(x,y,n)\right]_{1\times n},$$
$$x=1,2,\ldots,n-1,y=1,2,\ldots,n \tag{13}$$

$$\mathbf{MC^{gen}_{n\times n\times n}}=\left[\mathbf{MC^{gen}_{n\times n\times n}}(\mathbf{1}),\mathbf{MC^{gen}_{n\times n\times n}}(\mathbf{2}),\ldots,\mathbf{MC^{gen}_{n\times n\times n}}(\mathbf{n})\right]. \tag{14}$$

In (11), the element $MC^{\mathrm{gen}}_{n\times n\times n}(x,y,z)$ can save the occurrence frequency that the job $SubB\_j_{x+1}^{\mathrm{gen,w}}$ appears immediately after the job $SubB\_j_x^{\mathrm{gen,w}}$ when $SubB\_j_x^{\mathrm{gen,w}}$ is at the xth position of $\mathbf{SubB\_}\pi^{\mathbf{gen,w}}$, and its subscripts x and (y,z) record the ordinal number of job (i.e., the position of $SubB\_j_x^{\mathrm{gen,w}}$ in $\mathbf{SubB\_}\pi^{\mathbf{gen,w}}$) and the corresponding block (i.e., $[y = SubB\_j_x^{\mathrm{gen,w}}, z = SubB\_j_{x+1}^{\mathrm{gen,w}}]$), respectively. Thus, the subscripts of elements greater than 0 in the 2-D matrix $\mathbf{MC^{gen}_{n\times n\times n}}$ $(\mathbf{x})$ reserve all the ordinal numbers of jobs and the corresponding blocks at the xth positions of individuals in **SubBestPop**(**gen**). Moreover, the hierarchical structure of the 3-D matrix cube $\mathbf{MC^{gen}_{n\times n\times n}}$ reserves the whole order information of jobs in **SubBestPop**(**gen**) by means of a series of position-based submatrices, i.e., $\mathbf{MC^{gen}_{n\times n\times n}}(\mathbf{1})$, $\mathbf{MC^{gen}_{n\times n\times n}}(\mathbf{2}),\ldots,\mathbf{MC^{gen}_{n\times n\times n}}$ $(\mathbf{n})$. Therefore, the valuable information of excellent individuals at generation *gen* can be learned and reserved in an intuitive way. An example of the above explanations can be found in Part 3 of the online supplementary material (see the website address in Section IV-A).

2) Matrix-Cube-Based Probabilistic Model: The 3-D probability model $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ is designed to save the valuable information (i.e., the blocks with their corresponding positions and the total order of jobs) from $\mathbf{MC^{gen}_{n\times n\times n}}$. This model has a significant effect on the performance of the global exploration. The update equation of $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ is determined by the generation *gen*. When gen $=$ 0, $\mathbf{Pro\_MC^{0}_{n\times n\times n}}$ is updated by

$$Pro\_MC^{0}_{n\times n\times n}(x,y,z)=\begin{cases}0, & x=1;y,z=1,\ldots,n\\ 1/n^2, & x=2,\ldots,n-1;y,z=1,\ldots,n.\end{cases} \tag{15}$$

When $gen = 1$, $\mathbf{Pro\_MC^{1}_{n\times n\times n}}$ is updated by

$$Pro\_MC^{1}_{n\times n\times n}(x,y,z)=\frac{Pro\_MC^{0}_{n\times n\times n}(x,y,z)+MC^{0}_{n\times n\times n}(x,y,z)}{SumPro\_MC^{0}(x)+SumMC^{0}(x)},$$
$$x=1,2,\ldots,n-1 \text{ and } y,z=1,\ldots,n. \tag{16}$$

When gen $>$ 1, $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ is updated by

$$Pro\_MC^{\mathrm{gen}}_{n\times n\times n}(x,y,z)=(1-r)\times Pro\_MC^{\mathrm{gen}-1}_{n\times n\times n}(x,y,z)+r\times MC^{gen-1}_{n\times n\times n}(x,y,z)/\mathrm{SumMC}^{gen-1}(x),$$
$$x=1,2,\ldots,n-1, \text{ and } y,z=1,2,\ldots,n. \tag{17}$$

It is noteworthy that all elements in $\mathbf{Pro\_MC^{0}_{n\times n\times n}}(\mathbf{1})$ are set to 0 instead of $1/n^2$, which is helpful for increasing the selection probability of blocks in excellent individuals at initial stage. Equations (16) and (17) are used to accumulate valuable information from $\mathbf{MC^{gen}_{n\times n\times n}}$. An illustration on the update of $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$ can be found in Part 4 of the online supplementary material (see the website address in Section IV-A).

3) Global Search: MCEDA's global search is executed by sampling the probability model to generate new solutions in solution space. At the initial phase, all individuals in **Pop**(**0**) are randomly generated. Then, at the evolutionary phase,

TABLE II
SOME NOTATIONS APPLIED IN MATRIX-CUBE-BASED PROBABILISTIC MODEL

Symbol	Description of Symbol
Pop(gen)	The population of MCEDA at generation gen, $gen=0,\ldots,maxgen$.
popsize	The size of **Pop(gen)**.
SubBestPop(gen)	The superior sub-population or excellent individuals selected from **Pop(gen)**, i.e., **SubBestPop(gen)** = $\{\mathbf{SubB\_}\pi^{\mathbf{gen,1}},\ldots,\mathbf{SubB\_}\pi^{\mathbf{gen,sbpopsize}}\}$.
sbpopsize	The size of **SubBestPop(gen)**.
$\mathbf{SubB\_}\pi^{\mathbf{gen,w}}$	The wth individual in the superior sub-population **SubBestPop(gen)**, $\mathbf{SubB\_}\pi^{\mathbf{gen,w}}=[SubB\_j_1^{gen,w},\ldots,SubB\_j_n^{gen,w}]$, $w=1,\ldots,sbpopsize$.
$\mathbf{Pro\_MC^{gen}_{n\times n\times n}(x)}$	The xth layer or two-dimensional submatrix of $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$.
$\mathbf{Pro\_MC^{gen}_{n\times n\times n}(x,y)}$	The yth row vector of $\mathbf{Pro\_MC^{gen}_{n\times n\times n}(x)}$.
$SumPro\_MC^{gen}(x)$	The summation of all elements in $\mathbf{Pro\_MC^{gen}_{n\times n\times n}(x)}$, i.e., $SumPro\_MC^{gen}(x)=\sum_{y=1}^{n}\sum_{z=1}^{n}Pro\_MC^{gen}_{n\times n\times n}(x,y,z)$, $x=1,\ldots,n-1$.
$SumMC^{gen}(x)$	The summation of all elements in $\mathbf{MC^{gen}_{n\times n\times n}(x)}$, i.e., $SumMC^{gen}(x)=\sum_{y=1}^{n}\sum_{z=1}^{n}MC^{gen}_{n\times n\times n}(x,y,z)$, $x=1,\ldots,n-1$.

Algorithm 1 Job Selection Function

Step 1: Set $\mathbf{R} = \mathbf{Pro\_MC}^{\mathbf{gen}}_{\mathbf{n\times n\times n}}(\mathbf{i-1}, \mathbf{j}^{\mathbf{gen,q}}_{\mathbf{i-1}})$.
Step 2: Set $h = j_t^{gen,q}$ and $R_h = 0$, $t = 1, 2, \ldots, i-1$.
Step 3: Set $sumR = \sum_{h=1}^{n} R_h$ and $R_h = R_h/sumR$, $h = 1, \ldots, n$.
Step 4: Get a job CJ by using the roulette wheel selection rule on $\mathbf{R}$. Generate a random value r, $r \in [0, sumR)$.
Step 4.1: If $r \in [0, R_1)$, then set $CJ = 1$ and go to Step 5.
Step 4.2: If $r \in \left[\sum_{h=1}^{t} R_h, \sum_{h=1}^{t+1} R_h\right)$ and $t \in \{1, \ldots, n-1\}$, then set $CJ = t+1$.
Step 5: Return CJ.

Algorithm 2 First Position Limited Sampling Strategy

Step 1: Set $R1_y = \sum_{z=1}^{n} Pro\_MC^{\text{gen}}(1, y, z)$, $y = 1, \ldots, n$.
//Assign the cumulative probability of each row vector in
//$\mathbf{Pro\_MC}^{\mathbf{gen}}_{\mathbf{n\times n\times n}}(\mathbf{1})$ to the corresponding element in $\mathbf{R1}$.
Step 2: Generate a random value r, $r \in \left[0, \sum_{y=1}^{n} R1_y\right)$.
Step 2.1: If $r \in [0, R1_1)$, then set $CJ = 1$ and go to Step 3.
Step 2.2: If $r \in \left[\sum_{y=1}^{t} R1_y, \sum_{y=1}^{t+1} R1_y\right)$ and $t \in \{1, \ldots, n-1\}$, then set $CJ = t+1$.
Step 3: Set $j_1^{gen,q} = CJ$.
Step 4: Set $CJ = SelectJob(\boldsymbol{\pi}^{\mathbf{gen,q}}, 2)$. //**Algorithm 1**
Step 5: Set $j_2^{gen,q} = CJ$.

Algorithm 3 Population Generation

Step 1: Set $q = 1$ and

$$R1_y = \sum_{z=1}^{n} Pro\_MC^{gen}(1, y, z),\ y = 1, \ldots, n.$$

Step 2: Generate a new individual $\boldsymbol{\pi}^{\mathbf{gen,q}}$.
Step 2.1: Determine the job $j_1^{gen,q}$ and the job $j_2^{gen,q}$ by means of **FPLSS**. //**Algorithm 2 without its Step 1.**
Step 2.2: Set $i = 3$.
Step 2.3: Set $CJ = SelectJob(\boldsymbol{\pi}^{\mathbf{gen,q}}, i)$. //**Algorithm 1**
Step 2.4: Set $j_i^{gen,q} = CJ$.
Step 2.5: Set $i = i+1$.
Step 2.6: If $i \le n$, then go to Step 2.3.
Step 3: Set $q = q+1$.
Step 4: If $q \le popsize$, then go to Step 2.
Step 5: Output **Pop**(**gen**).

new individuals at generation *gen* are obtained by sampling $\mathbf{Pro\_MC}^{\mathbf{gen}}_{\mathbf{n\times n\times n}}$.

Let $\boldsymbol{\pi}^{\mathbf{gen,q}} = [j_1^{\text{gen,q}}, j_2^{\text{gen,q}}, \ldots, j_n^{\text{gen,q}}]$ denotes the qth individual in **Pop**(**gen**), $\mathbf{R} = [R_1, R_2, \ldots, R_n]$ denotes the temporary row vector, $\mathbf{R1} = [R_1, R_2, \ldots, R_n]^T$ denotes the temporary column vector, and $SelectJob(\boldsymbol{\pi}^{\mathbf{gen,q}}, i)$ $(i > 1)$ denotes the function of selecting a job CJ at the ith position of $\boldsymbol{\pi}^{\mathbf{gen,q}}$ by sampling $\mathbf{R}$. The procedure of $SelectJob(\boldsymbol{\pi}^{\mathbf{gen,q}}, i)$ is given in the following Algorithm 1.

Due to the virtuality of the job $j_0^{\text{gen,q}}$, the first job $j_1^{\text{gen,q}}$ of $\boldsymbol{\pi}^{\mathbf{gen,q}}$ cannot be chosen by using $SelectJob(\boldsymbol{\pi}^{\mathbf{gen,q}}, i)$. To reasonably select the first two jobs (i.e., the first block) of $\boldsymbol{\pi}^{\mathbf{gen,q}}$, the first position limited sampling strategy (**FPLSS**) is specially designed in the following Algorithm 2.

Steps 2 and 3 apply the roulette wheel selection rule to $\mathbf{R1}$ to choose the first job $j_1^{\text{gen,q}}$. The procedure for generating new individuals is given in the following Algorithm 3.

TABLE III
SOME NOTATIONS APPLIED IN THE LOCAL SEARCH

Symbol	Description of Symbol
$Insert(\boldsymbol{\pi},i,l)$	The insertion of the job j_i before the job j_l of $\boldsymbol{\pi}$ when $i>l$ and after j_l when $i<l$, and $\boldsymbol{\pi}=[j_1,...,j_s,...,j_n]$.
$N^1_{Insert}(\boldsymbol{\pi})$	The small and simple neighborhood of $\boldsymbol{\pi}$, i.e., $N^1_{Insert}(\boldsymbol{\pi})=\{\boldsymbol{\pi}^{n,i,l} = Insert(\boldsymbol{\pi},i,l)\,\vert\, l$ and i are randomly selected, $i,l \in \{1,2,...,n\}$ and $i \neq l\}$.
$\boldsymbol{\pi}_{better}$	The current better neighbor of $\boldsymbol{\pi}$, and $\boldsymbol{\pi}_{better} = [j_1^{better}, j_2^{better}, ..., j_s^{better}, ..., j_n^{better}]$.
$Interchange(\boldsymbol{\pi},i,l)$	The interchange of the job j_i at the ith position and the job j_l at the lth position of the permutation $\boldsymbol{\pi}$.

It should be pointed out that steps 2.3 and 2.4 are utilized to build the promising block (i.e., $[j_{i-1}^{\text{gen,q}}, j_i^{\text{gen,q}}]$) at positions $i-1$ and i via applying the roulette wheel selection rule on the row vector $\mathbf{Pro\_MC}^{\mathbf{gen}}_{\mathbf{n\times n\times n}}(\mathbf{i-1}, \mathbf{j}^{\mathbf{gen,q}}_{\mathbf{i-1}})$. Step 2 can link these promising blocks at different positions together to generate a new individual. Since the large values and their corresponding subscripts in $\mathbf{Pro\_MC}^{\mathbf{gen}}_{\mathbf{n\times n\times n}}$ are decided by the excellent individuals [see Section III-B2)], each new individual contains some promising blocks or patterns. This means that MCEDA's global search can be quickly driven to promising regions. The analyses of the CCs of Algorithms 1–3 can be found in Part 5 of the online supplementary material (see the website address in Section IV-A).

C. Problem-Dependent Local Search

It is well known that the neighborhood structures have a significant effect on the performance of local exploitation [2], [16], [17], [18], [19]. Although the Insert-based neighborhood has been widely studied and utilized in designing local search for different scheduling problems, most of existing studies focused on its common or general form and paid less attention to the problem's properties [2], [9], [20], [21], [22], [23]. This causes the proposed algorithms are effective, but not efficient.

In this section, we divide the common Insert-based neighborhood into a series of subsets and build a novel variable neighborhood search by means of these subsets or subneighborhoods. Moreover, the general property of the permutation-based model is utilized to design a speed-up scanning method to accelerate the neighbor search process. Then, a new Insert-based local search incorporating with this novel neighborhood search is proposed to perform exploitation efficiently in solution space. The notations used in the proposed local search but not defined in the text are provided in Table III.

1) Insert-Based Subneighborhoods: The *Insert*-based neighborhood of $\boldsymbol{\pi}$ can be presented as

$$N_{\text{Insert}}(\boldsymbol{\pi}) = \Big\{\pi^{n,i,l} = \text{Insert}(\boldsymbol{\pi}, i, l) | l \neq i, i-1 \text{ and } i, l = 1, 2, \ldots, n\Big\} \quad (18)$$

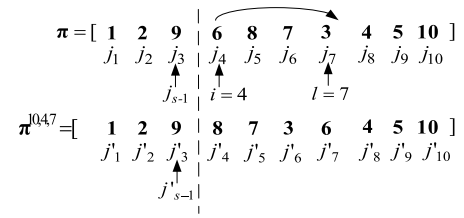

Fig. 2. Example of $\pi^{n,i,l}$ when $n = 10$, $i = 4$, $l = 7$, and $s = \min\{i, l\} = 4$.

where $\pi^{n,i,l}$ is a neighbor of π and $l \neq i-1$ is due to $\pi^{n,i-1,i} = \pi^{n,i,i-1}$. That is, $N_{\text{Insert}}(\pi)$ corresponds to the set of neighbors reachable through all possible insertions. Fig. 2 shows a small example of $\pi^{n,i,l}$ when $n = 10$, $i = 4$, $l = 7$, and $s = \min\{i, l\} = 4$.

Accordingly, for a fixed value of i ($i = 1, 2, \ldots, n$), the subneighborhood of $N_{\text{Insert}}(\pi)$ can be expressed as

$$N_{\text{Insert}}(\pi, i) = \left\{\pi^{n,i,l} = \text{Insert}(\pi, i, l) | l \neq i, i-1 \text{ and } l = 1, 2, \ldots, n\right\}. \quad (19)$$

Hence, it holds that

$$N_{\text{Insert}}(\pi) = N_{\text{Insert}}(\pi, 1) \cup N_{\text{Insert}}(\pi, 2) \cup \cdots \cup N_{\text{Insert}}(\pi, n). \quad (20)$$

Apparently, the neighbor size of $N_{\text{Insert}}(\pi)$ is $(n-1)^2$ and that of $N_{\text{Insert}}(\pi, i)$ is $n-1$ if $i = 1$ and $n-2$ otherwise.

Based on the above definitions, the neighborhood search of $N_{\text{Insert}}(\pi)$ can be implemented by searching the subneighbor- hoods in the sequence of $N_{\text{Insert}}(\pi, 1), N_{\text{Insert}}(\pi, 2), \ldots, N_{\text{Insert}}(\pi, n)$. In each $N_{\text{Insert}}(\pi, i)$ ($i = 1, \ldots, n$), the search is performed by visiting each neighbor $\pi^{n,i,l}$ for $l = 1, \ldots, n$ except $l = i, i-1$. This through search mode is important for designing an efficient neighborhood search.

2) Small Versus Large Neighborhood Search: The existing Insert-based local searches iterate between a perturbation phase and an exploitation phase. During their exploitation phases, these local searches repeat a greedy search process (GSP), which uses a kind of Insert-based neighborhood search to obtain one neighbor and replaces π with this neighbor if it is better than π. In this repeated GSP (RGSP), the total repeated times of the GSP are usually set to a polynomial function of the problem size n. The commonly used Insert-based neighborhood searches in GSP are the small neighborhood search based on $N^1_{\text{Insert}}(\pi)$ (see Table III) and the large neighborhood based on $N_{\text{Insert}}(\pi)$ [see (18)]. The neighborhood search of $N^1_{\text{Insert}}(\pi)$ is performed by visiting one randomly selected neighbor in $N_{\text{Insert}}(\pi)$. The large neighborhood search of $N_{\text{Insert}}(\pi)$ is performed by visiting all neighbors in $N_{\text{Insert}}(\pi)$ to find the local best neighbor π_{best}.

Since the landscape induced by Insert operator is smooth [15] and the distance between π and its best neighbor π_{best} is one, the difference between $TET(\pi)$ and $TET(\pi_{\text{best}})$ are usually small. This causes the search process of $N_{\text{Insert}}(\pi)$ is time consuming. Hence, if a better neighbor can be reached quickly after several random insertions [15], [24], it is suitable to execute the RGSP using the small neighborhood search [i.e., the neighborhood search of $N^1_{\text{Insert}}(\pi)$]. However, if the number of better neighbors in $N_{\text{Insert}}(\pi)$ is very limited or zero (i.e., the current π is a local optimum) [13], [14], [25], the RGSP using the small neighborhood search is less effective due to the randomness of insertions, but the RGSP using the large neighborhood search can restrict the insertion times for finding a better neighbor of π to no more than $(n-1)^2$ and can detect whether π is a local optimum.

To overcome the defects of both the small and large neighborhood searches and improve search efficiency, an efficient RGSP using a novel neighborhood search is devised in the upcoming three sections. In Sections III-C3 and III-C4, a novel neighborhood search based on the defined subneighborhoods and the problem's properties, namely, *FastS_*$N_{\text{Insert}}(\pi, KK)$, is designed. In Section III-C5, an efficient RGSP using *FastS_*$N_{\text{Insert}}(\pi, KK)$ is devised as the exploitation process in a new Insert-based local search.

3) Speed-Up Scanning Method: In *FastS_*$N_{\text{Insert}}(\pi, KK)$, a speed-up scanning method based on the general property of the permutation-based model is presented to accelerate the neighbor search process.

For $\pi = [j_1, \ldots, j_s, \ldots, j_n]$, $\pi^{n,i,l} = [j'_1, \ldots, j'_s, \ldots, j'_n]$ and $s = \min\{i, l\}$, it is obvious that we can obtain $j_k = j'_k$, $E_{j'_k} + T_{j'_k} = E_{j_k} + T_{j_k}$,and $\sum_{k=1}^{s-1}(E_{j'_k} + T_{j'_k}) = \sum_{k=1}^{s-1}(E_{j_k} + T_{j_k})$ when $s = 2, \ldots, n-1$ and $k = 1, \ldots, s-1$ (see Fig. 2). So, according to (8), it holds that

$$\begin{aligned} TET\left(\pi^{n,i,l}\right) &= \sum_{k=1}^{s-1}\left(E_{j'_k} + T_{j'_k}\right) + \sum_{k=s}^{n}\left(E_{j'_k} + T_{j'_k}\right) \\ &= \sum_{k=1}^{s-1}\left(E_{j_k} + T_{j_k}\right) + \sum_{k=s}^{n}\left(E_{j'_k} + T_{j'_k}\right), \\ & i, l = 1, \ldots, n \text{ and } l \neq i, i-1. \end{aligned} \quad (21)$$

So, in *FastS_*$N_{\text{Insert}}(\pi, KK)$, $St_{j_{s-1}}$, and $\sum_{k=1}^{s-1}(E_{j_k} + T_{j_k})$ ($s - 1 = 1, \ldots, n-2$) of π are directly calculated by utilizing L_{j_{i-1},j_i} and sp_{j_i} that are saved at the initial phase of MCEDA (see Sections II and III-A), and they should be reserved at first before scanning or visiting each neighbor $\pi^{n,i,l}$ in $N_{\text{Insert}}(\pi)$. Then, they are treated as constant when calculating $TET(\pi^{n,i,l})$. In other words, if $s > 1$, $St_{j'_{s-1}}$ and $\sum_{k=1}^{s-1}(E_{j'_k} + T_{j'_k})$ of $\pi^{n,i,l}$ need not to be computed and are replaced with $St_{j_{s-1}}$ and $\sum_{k=1}^{s-1}(E_{j_k} + T_{j_k})$ of π, respectively. Then, $E_{j'_s} + T_{j'_s}$ of $\pi^{n,i,l}$ is computed directly from $St_{j_{s-1}}$ of π by means of (4)–(7). As a result, the computing complexity of *FastS_*$N_{\text{Insert}}(\pi, KK)$ can be reduced to some extent. Note that this scanning method is easy to be implemented by adding a global array in the procedure to save $St_{j_{s-1}}$ and $\sum_{k=1}^{s-1}(E_{j_k} + T_{j_k})$ ($s - 1 = 1, \ldots, n-2$). The subneighborhood search with the speed-up scanning method is denoted by *FastSubS_*$N_{\text{Insert}}(\pi, i)$.

Algorithm 4 Novel Neighborhood Search

Step 1: Calculate and save $St_{j_{s-1}}$ and $\sum_{k=1}^{s-1}(E_{j_k}+T_{j_k})$ of $\boldsymbol{\pi}$ for $s-1=1,\ldots,n-2$.
Step 2: Set $i=KK$, $\boldsymbol{\pi}_{better}=\boldsymbol{\pi}$, $St_{j_{s-1}^{better}}=St_{j_{s-1}}$, and $\sum_{k=1}^{s-1}(E_{j_k^{better}}+T_{j_k^{better}})=\sum_{k=1}^{s-1}(E_{j_k}+T_{j_k})$ for $s-1=1,\ldots,n-2$.
Step 3: $\boldsymbol{\pi}_{temp}=FastSubS\_N_{Insert}(\boldsymbol{\pi}_{better},i)$.
Step 4: If $TET(\boldsymbol{\pi}_{temp})<TET(\boldsymbol{\pi}_{better})$, then $\boldsymbol{\pi}_{better}=\boldsymbol{\pi}_{temp}$, $St_{j_{s-1}^{better}}=St_{j_{s-1}^{temp}}$ and $\sum_{k=1}^{s-1}(E_{j_k^{better}}+T_{j_k^{better}})=\sum_{k=1}^{s-1}(E_{j_k^{temp}}+T_{j_k^{temp}})$ for $s-1=1,\ldots,n-2$.
Step 5: If $i<n$, then $i=i+1$ and go to Step 3.
Step 6: Output $\boldsymbol{\pi}_{better}$.

Although the speed-up scanning method presented here is for the TET-NFSSP with SDSTs and RTs, it is easy to see that by setting some processing restrictions (i.e., RTs, SDSTS, and SISTs) to 0 and changing the criterion, this method can also be applied to each kind of NFSSPs with any common criterion (e.g., makespan and total completion time). So, this method is still used in MCEDA's $FastSubS\_N_{\text{Insert}}(\boldsymbol{\pi},i)$ when comparing different algorithms in Section IV-C.

4) Novel Neighborhood Search With Subneighborhoods and the Promising Region Search Strategy: As mentioned in Section III-C2, if there are a certain number of better neighbors in $N_{\text{Insert}}(\boldsymbol{\pi})$, the small neighborhood search is more suitable, but if the number of better neighbors in $N_{\text{Insert}}(\boldsymbol{\pi})$ is very limited or zero, the large neighborhood search is a better choice. Thus, an important and difficult problem faced by the researchers is how to adaptively execute suitable neighborhood search during local search process. Here, a novel neighborhood search $FastS\_N_{\text{Insert}}(\boldsymbol{\pi},KK)$ is designed to cope with this problem. Its procedure is given in the following Algorithm 4.

KK ($KK\in\{1,\ldots,n\}$) is an input variable which is used to control the actual subneighborhoods searched in $N_{\text{Insert}}(\boldsymbol{\pi})$ when applying the speed-up scanning method. The default value of KK is set to 1. Since $St_{j_{s-1}^{\text{better}}}$ and $\sum_{k=1}^{s-1}(E_{j_k^{\text{better}}}+T_{j_k^{\text{better}}})$ of $\boldsymbol{\pi}_{\text{better}}$ are updated at steps 2–4, they can be used as constants in $FastSubS\_N_{\text{Insert}}(\boldsymbol{\pi}_{\text{better}},i)$. The designed $FastS\_N_{\text{Insert}}(\boldsymbol{\pi},KK)$ has threefold features.

The first feature is that the large neighborhood search mode that searches the subneighborhoods one by one is reserved. When there is no better neighbor of $\boldsymbol{\pi}$, $FastS\_N_{\text{Insert}}(\boldsymbol{\pi},KK)$ can perform a thorough search in $N_{\text{Insert}}(\boldsymbol{\pi})$ to identify $\boldsymbol{\pi}$ as a local optimum.

The second feature is that the subneighborhood $N_{\text{Insert}}(\boldsymbol{\pi},i)$ and the promising region search strategy are combined to reduce the search scope in $N_{\text{Insert}}(\boldsymbol{\pi})$ containing some better neighbors. The promising region search strategy is adopted for reaching more promising regions within a certain time limit. That is, the permutation $\boldsymbol{\pi}$ in $FastSubS\_N_{\text{Insert}}(\boldsymbol{\pi},i)$ is replaced by the current better neighbor $\boldsymbol{\pi}_{\text{better}}$ (see steps 3 and 4 in Algorithm 4). Fig. 3 gives the search process in $FastS\_N_{\text{Insert}}(\boldsymbol{\pi},KK)$ ($KK=1$), which is a key component in our proposed local search (see Algorithm 5).

It can be seen from ① in Fig. 3, that if the best neighbor $\boldsymbol{\pi}^{\mathbf{1}}$ in $N_{\text{Insert}}(\boldsymbol{\pi},1)$ is better than the current $\boldsymbol{\pi}$, the latter

Algorithm 5 Local Search

Step 1: Select an individual $\boldsymbol{\pi}_{i\_0}$ from the population.
Step 2: Perturbation Phase.
Set $\boldsymbol{\pi}_{i\_t}=\boldsymbol{\pi}_{i\_0}$.
For $k=1$ to 5
  Randomly select u and v, where $|u-v|>n/3$;
  $\boldsymbol{\pi}_i=Interchange(\boldsymbol{\pi}_{i\_t},u,v)$;
  $\boldsymbol{\pi}_{i\_t}=\boldsymbol{\pi}_i$;
End.
Step 3: Exploitation Phase. //The RGSP
Set $loop=0$;
Repeat
  $\boldsymbol{\pi}_{i\_1}=FastS\_N_{Insert}(\boldsymbol{\pi}_i,KK)$; //**Algorithm 4**
  If $TET(\boldsymbol{\pi}_{i\_1})<TET(\boldsymbol{\pi}_i)$
  $\boldsymbol{\pi}_i=\boldsymbol{\pi}_{i\_1}$;
  Else //A local optimum is detected.
  $loop++$;
  End;
Until $loop=1$. //When a local optimum is detected, the
  //RGSP is terminated.
Step 4: If $TET(\boldsymbol{\pi}_i)\le TET(\boldsymbol{\pi}_{i\_0})$, then $\boldsymbol{\pi}_{i\_0}=\boldsymbol{\pi}_i$.
Step 5: Output the updated job permutation $\boldsymbol{\pi}_{i\_0}$.

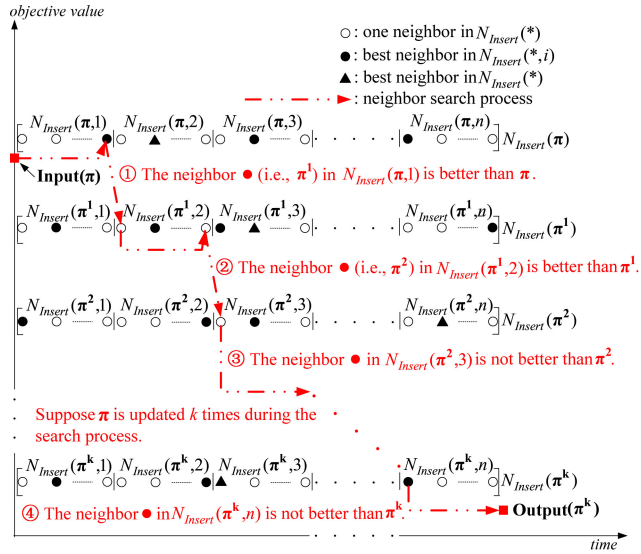

Fig. 3. Illustration on the search process in $FastS\_N_{\text{Insert}}(\boldsymbol{\pi},KK=1)$.

is updated by the former and the search in $N_{\text{Insert}}(\boldsymbol{\pi})$ stops and moves to $N_{\text{Insert}}(\boldsymbol{\pi}^{\mathbf{1}},2)$ in $N_{\text{Insert}}(\boldsymbol{\pi}^{\mathbf{1}})$. It can also be seen from ③ in Fig. 3, that if the best neighbor in $N_{\text{Insert}}(\boldsymbol{\pi}^{\mathbf{2}},3)$ is not better than the current $\boldsymbol{\pi}^{\mathbf{2}}$, the later keeps unchangeable, and the search in $N_{\text{Insert}}(\boldsymbol{\pi}^{\mathbf{2}})$ continues and executes in the next subneighborhood $N_{\text{Insert}}(\boldsymbol{\pi}^{\mathbf{2}},4)$. Unlike the large neighborhood search mode that always keeps the search in n subneighborhoods of one fixed neighborhood [i.e., $N_{\text{Insert}}(\boldsymbol{\pi})$], $FastS\_N_{\text{Insert}}(\boldsymbol{\pi},KK)$ (see Fig. 3 and steps 3–5 of Algorithm 4) performs the search in n subneighborhoods scattered in k+1 (k is larger than 2 in most cases) different neighborhoods [i.e., $N_{\text{Insert}}(\boldsymbol{\pi})$, $N_{\text{Insert}}(\boldsymbol{\pi}^{\mathbf{1}})$,..., $N_{\text{Insert}}(\boldsymbol{\pi}^{\mathbf{k}})$]. Since $\boldsymbol{\pi}^{\mathbf{1}}$ is better than $\boldsymbol{\pi}$ and $\boldsymbol{\pi}^l$ is better than $\boldsymbol{\pi}^{l-1}$ ($l=2,\ldots,k$), these rich neighborhoods can guide the search to more and more promising regions. That is, $FastS\_N_{\text{Insert}}(\boldsymbol{\pi},KK)$ can not only narrow

the search scope of single neighborhood to avoid unnecessary time consumption but also drive the local exploitation to many different and more promising neighborhoods or regions. This means that *FastS_N*$_{\text{Insert}}$ $(\boldsymbol{\pi}, KK)$ can perform a narrow but rich search.

The third feature is that the speed-up evaluation method in Section III-A and the speed-up scanning method in the above section are utilized to evaluate more neighbors in *FastS_N*$_{\text{Insert}}(\boldsymbol{\pi}, KK)$ within a limited time. By utilizing these methods, the efficiency of the local search can be improved.

In the existing Insert-based neighborhoods, the small neighborhood search [26], [27], [28], [29] (see $N^1_{\text{Insert}}(\boldsymbol{\pi})$ in Table III) and the large neighborhood search [18], [30], [31] have already got wide applications. *FastS_N*$_{\text{Insert}}(\boldsymbol{\pi}, KK)$ (see Fig. 3) is quite different from these two kinds of neighborhood searches. It will inherits the merits of both the small and large neighborhood searches, and can adaptively execute suitable neighborhood search throughout the exploitation phase.

5) New Insert-Based Local Search: The above designed *FastS_N*$_{\text{Insert}}(\boldsymbol{\pi}, KK)$ is utilized to construct a RGSP in the exploitation process of the following new Insert-based local search (i.e., MCEDA's problem- dependent local search). The procedure of this new local search is summarized in the following Algorithm 5.

In Algorithm 5, step 2 is a common perturbation phase, which is used to avoid cycling search and overcome local optima. Step 3 is a special devised exploitation phase, which constructs an efficient RGSP to perform exploitation. If a local optimum is detected, the RGSP in step 3 terminates. Then, the whole search of MCEDA resorts to EDA-based global search and step 2 to overcome the current local optimum and move closer to global optima. The analyses of the CCs of Algorithms 4 and 5 can be found in Part 5 of the online supplementary material (see the website address in Section IV-A).

D. MCEDA

The devised MCEDA is given in the following Algorithm 6.

In the above procedure, not only does MCEDA use the matrix-cube-based global search to execute exploration to find promising regions in solution space, but it also adopts the problem-dependent local search with both a speed-up scanning method and one strategy to perform exploitation in these promising regions. Moreover, a speed-up evaluation method based on the no-wait property of the considered problem is utilized to reduce TET's CC. In addition, the implementation of MCEDA is easy, because the detailed pseudocode of it is only one-third the length of this section. The detailed analyses of the CC of each step in Algorithm 6 can be found in Part 5 of the online supplementary material (see the website address in Section IV-A).

Let *TCC* denotes the total CC of Algorithm 6, $K_2(p, \text{gen})$ the repeat times of executing *FastS_N*$_{\text{Insert}}(\boldsymbol{\pi}_i, KK)$ in step 5 of Algorithm 6 at generation *gen* for the problem p, and $K_2(p) = \sum_{\text{gen}=1}^{\text{maxgen}} K_2(p, \text{gen})$. Because $K_2(p, gen)$ depends on the landscape of p and is always larger than or equal to one, we have $K_2(p) = \sum_{\text{gen}=1}^{\text{maxgen}} K_2(p, \text{gen}) \geq \text{maxgen}$. Moreover,

Algorithm 6 MCEDA

Step 0: Parameter Initialization. Set the start running time *Start_RT* to the current system time of the computer. Set the critical parameters *popsize*, φ ($sbpopsize = \varphi \times popsize$), and r.
Step 1: Calculate and save L_{j_{i-1},j_i} ($j_{i-1}, j_i \in \{1, \ldots, n\}$) and sp_{j_i} ($j_i \in \{1, \ldots, n\}$) for using the speed-up evaluation method to evaluation individuals.
Step 2: Population Initialization. Generate a population randomly, and then evaluate its individuals to obtain the global best individual $\boldsymbol{\pi}^g_{best}$ and **SubBestPop(0)**.
Step 3: Initialize $\mathbf{Pro\_MC^0_{n\times n\times n}}$ via Eq. (15) and set$gen = 1$.
Step 4: Global Exploration. //Subsection III-B
  Step 4.1: Update$\mathbf{Pro_MC^{gen}_{n\times n\times n}}$ via Eq. (16) when $gen = 1$ or Eq. (17) when $gen > 1$.
  Step 4.2: Generate population at generation *gen* by sampling $\mathbf{Pro\_MC^{gen}_{n\times n\times n}}$. **//Algorithm 3**
  Step 4.3: Evaluate the new individuals to update $\boldsymbol{\pi}^g_{best}$and

SubBestPop(gen).

Step 5: Local Exploitation. //Subsection III-C
Perform the *Insert*-based local search to $\boldsymbol{\pi}^g_{best}$ and update $\boldsymbol{\pi}^g_{best}$. **//Algorithm 5**
Step 6: Set $gen = gen + 1$. Set the end running time *End_RT* to the current system time of the computer. If $gen \leq maxgen$ or ($End\_RT - Start\_RT$) < the maximum running time, go to Step 4.
Step 7: Return the best solution $\boldsymbol{\pi}^g_{best}$ found so far.

n is usually larger than m. From Algorithm 6, *TCC* can be expressed as

$$\begin{aligned} TCC &= O\left(\text{maxgen}^*\left(n^2m + n^3 + \text{popsize}^*n^2\right) + \sum_{\text{gen}=1}^{\text{maxgen}} K_2(p, \text{gen})^*n^2\widetilde{\log}n\right) \\ &= O\left(\text{maxgen}^*\left(\text{n}^3 + \text{popsize}^*\text{n}^2\right) + \text{K}_2(\text{p})^*\text{n}^2\widetilde{\log}\text{n}\right) \end{aligned} \tag{22}$$

where $\widetilde{\log}n$ is less than n. From the above analyses, it can be found that the computational cost of MCEDA is not high since the highest degree of the polynomial $\text{maxgen}^*(n^3 + \text{popsize}*^*n^2) + K_2(p)^*n^2\widetilde{\log}n$ is three.

IV. Computational Comparisons and Statistical Analyses

A. Experimental Setup

Some randomly generated instances are adopted to test the performance of MCEDA. That is, fifteen different $\{n \times m\}$ in $\{20, 30, 50, 70, 100\} \times \{5, 10, 20\}$ are considered. The processing time $p_{j_i,l}$ and the setup time $s_{j_{i-1},j_i,l}$ are randomly generated from a uniform distribution in the range [1, 100] and a uniform distribution in the range [0, 100], respectively. The job RT r_{j_i} is an integer that is randomly generated from a uniform distribution in the range $[0, 150n\alpha]$, where the parameter α is used to control the speed of each job's RT and the interval between r_{j_i} and $r_{j_{i-1}}$. The level values of α are set as 0, 0.2, 0.4, 0.6, 0.8, 1, and 1.5, respectively. Then, we have a total of 105 different instances. The specific setting of each job's due

TABLE IV
COMPARISON RESULTS OF MCEDA AND SEVEN ALGORITHMS ON METRICS *ARPD* AND *SD* ($\rho = 2$, TET-NFSSP WITH SDSTS, AND RTS)

Instance	ρ	MCPSO		ISA_2		BIH		TMIIG		EEDA		IG_LS		HDTPL		MCEDA	
n, m		*ARPD*	*SD*	*ARPD*	*SD*	*ARPD*	*SD*	*ARPD*	*SD*	*ARPD*	*SD*	*ARPD*	*SD*	*ARPD*	*SD*	*ARPD*	*SD*
20,5	2	2.23	5.32	0.44	0.89	1.03	0.00	**0.23**	**0.31**	0.53	0.51	0.31	0.41	*0.27*	*0.37*	**0.12**	**0.28**
20,10		1.88	4.44	0.35	0.73	0.87	0.00	*0.17*	*0.21*	0.39	0.37	0.24	0.29	**0.15**	**0.17**	**0.09**	**0.16**
20,20		1.67	7.15	0.22	1.04	0.64	0.00	*0.13*	*0.15*	0.27	0.31	*0.13*	*0.15*	**0.12**	**0.14**	**0.06**	**0.11**
30,5		2.85	8.62	0.83	1.46	1.53	0.00	*0.57*	*0.62*	0.88	0.75	0.64	**0.58**	**0.47**	**0.58**	**0.18**	**0.27**
30,10		3.43	9.71	1.06	1.35	1.45	0.00	*0.84*	**0.67**	1.17	0.99	0.95	0.78	**0.69**	*0.71*	**0.27**	**0.34**
30,20		2.59	9.13	0.84	1.23	1.37	0.00	*0.41*	*0.47*	0.85	0.59	0.47	0.54	**0.34**	**0.34**	**0.16**	**0.28**
50,5		3.82	9.97	1.82	1.59	2.55	0.00	*1.25*	**0.81**	1.74	1.46	1.36	*1.16*	**0.92**	1.34	**0.43**	**0.53**
50,10		4.26	10.11	1.86	1.63	2.72	0.00	*1.34*	**0.92**	1.89	1.53	1.53	0.97	**0.96**	*0.95*	**0.47**	**0.79**
50,20		3.67	8.46	1.44	1.54	2.26	0.00	*1.02*	*0.88*	1.63	1.22	1.17	1.03	**0.77**	**0.76**	**0.31**	**0.52**
70,5		2.93	8.77	1.25	1.42	2.09	0.00	*0.71*	**0.75**	1.28	1.17	0.97	1.14	**0.55**	*0.87*	**0.28**	**0.41**
70,10		3.79	9.58	1.67	1.37	2.41	0.00	*1.07*	*0.67*	1.72	1.38	1.22	0.94	**0.84**	**0.55**	**0.35**	**0.45**
70,20		4.13	8.61	1.84	1.31	2.57	0.00	*1.32*	*0.72*	1.92	1.46	1.51	0.88	**1.15**	**0.64**	**0.59**	**0.56**
100,10		5.27	6.74	1.93	1.46	2.84	0.00	*1.75*	**0.77**	2.12	1.68	1.83	*0.98*	**1.43**	**0.68**	**0.75**	**0.68**
100,20		5.52	7.35	2.51	1.68	3.03	0.00	*1.82*	*0.94*	2.67	1.85	1.96	1.17	**1.74**	**0.83**	**0.96**	**0.74**
100,40		5.84	9.73	2.87	1.74	3.45	0.00	*1.92*	*1.12*	2.91	2.21	2.54	1.43	**1.77**	**0.94**	**1.14**	**0.92**
Average		3.59	8.25	1.40	1.36	2.05	0.00	*0.97*	*0.67*	1.46	1.17	1.12	0.83	**0.81**	**0.66**	**0.41**	**0.47**

date d_{p,j_i} and the reason why the values of α and d_{p,j_i} are set in such way can be found in Part 6 of the online supplementary material (see the following website). All of the testing instances and the supplementary material can be downloaded from the website (i.e., https://pan.baidu.com/s/1J_cqvhmi-0rCxkmBRkhjZg, password: TSMC).

The performance metrics are given as follows. Let $TET(\boldsymbol{\pi}(\alpha))$ denotes the total earliness and tardness of the permutation $\boldsymbol{\pi}(\alpha)$ at α level, $SD(\alpha)$ denotes the standard deviation of $TET(\boldsymbol{\pi}(\alpha))$ at α level, S_α denotes the set of all values of α, $|S_\alpha|$ denotes the number of different values in S_α, $TET_{\text{avg}}(\boldsymbol{\pi}(\alpha))$ denotes the average value of $TET(\boldsymbol{\pi}(\alpha))$, $TET^*(\boldsymbol{\pi}(\alpha))$ denotes the best $TET(\boldsymbol{\pi}(\alpha))$ obtained by all compared algorithms after running a long time ($\rho \geq 100$), and $ARPD(\alpha)$ denotes the average relative percentage deviation over $TET^*(\boldsymbol{\pi}(\alpha))$. The formulation of $ARPD(\alpha)$ is given as follows:

$$\text{ARPD}(\alpha) = \frac{TET_{\text{avg}}(\boldsymbol{\pi}(\alpha)) - TET^*(\boldsymbol{\pi}(\alpha))}{TET^*(\boldsymbol{\pi}(\alpha))} \times 100\%. \quad (23)$$

Then, two metrics are used to evaluate the average performance of all algorithms under different α, which are given as follows:

$$\text{ARPD} = \sum_{\alpha \in S\alpha} \text{ARPD}(\alpha)/|S_\alpha| \quad (24)$$

$$\text{SD} = \sum_{\alpha \in S\alpha} SD(\alpha)/|S_\alpha|. \quad (25)$$

The smaller the values of the above two metrics are, the better the performance of the corresponding algorithm is.

All algorithms have been coded in Delphi XE8 and experiments have been executed on an Intel 2.6 GHz PC server. Each algorithm is independently run 30 times on each instance and its maximum running time is set as $\rho \times n \times (m/2)$ milliseconds, where ρ is the running time factor. Hence, there are $30 \times |S_\alpha| = 30 \times 7 = 210$ results for each value of *ARPD* and also for each value of *SD*. In Tables IV and V, the best, the second-best, and the third-best values in each row are represented by using the bold, the bold and underlined, and the italic and underlined fonts, respectively.

Due to space limitations, the following Sections IV-B and IV-C only provide the brief comparison results. The extended Section IV has 23 double-column pages and is Part 6 of the online supplementary material, which can be downloaded from the above website. This extended version adds five new sections to calibrate MCEDA's parameters and manifest the effectivenesses of the speed-up evaluation method, MCEDA's global search strategies, global search ability, and local search components. It also gives the parameter settings of each algorithm and the detailed results of the following two sections.

B. Comparisons of MCEDA and Existing Algorithms

The algorithms for the considered problem are very limited. For new problems or less studied problems, the existing literature usually compares the designed algorithm with the existing algorithms for solving similar problems. Therefore, to show the effectiveness and efficiency of the proposed MCEDA, MCEDA is tested against six related state-of-the-art algorithms, i.e., MCPSO [21] for the Cmax-NFSSP with SDSTs, ISA_2 [26] for the TotalC-NFSSP with SISTs, BIH [9] for the Cmax-NFSSP with SDSTs and RTs, TMIIG [30] for the Cmax-NFSSP, IG_LS [23] for the Cmax-FSSP with SDSTs, and HDTPL [32] for the Cmax-NFSSP. MCEDA is also compared with a state-of-art EDA-based algorithm, i.e., effective EDA (EEDA) [33], which is presented for the distributed FSSP with the criterion of minimizing Cmax. According to the test results in [32], HDTPL is the most effective algorithm so far to deal with the NFSSP.

All steps of these compared algorithms are strictly implemented according to the literature, except that TMIIG's specific accelerating techniques for the Cmax-NFSSP are removed and EEDA's decoding scheme and search operators are only performed on one factory. We just replace their evaluation functions with the TET criterion presented in Section II. Since the speed-up evaluation method presented in Section III-A is a general one, all compared algorithms utilize it to calculate TET and keep their names unchangeable. The parameters of these compared algorithms are directly taken from the original literature and properly calibrated. The

TABLE V
COMPARISON RESULTS OF MCEDA AND SEVEN ALGORITHMS ON AVERAGE *ARPD* AND *SD* ($\rho = 2$, NINE TYPES OF PROBLEMS)

Problem	ρ	MCPSO		ISA_2		BIH		TMIIG		EEDA		IG_LS		HDTPL		MCEDA	
		$\overline{ARPD}$	$\overline{SD}$	$\overline{ARPD}$	$\overline{SD}$	$\overline{ARPD}$	$\overline{SD}$	$\overline{ARPD}$	$\overline{SD}$	$\overline{ARPD}$	$\overline{SD}$	$\overline{ARPD}$	$\overline{SD}$	$\overline{ARPD}$	$\overline{SD}$	$\overline{ARPD}$	$\overline{SD}$
P_1	2	2.21	0.14	0.81	0.03	1.01	0.00	*0.68*	*0.03*	0.88	0.05	0.70	**0.02**	**0.48**	0.11	**0.29**	**0.01**
P_4		4.29	0.09	2.10	0.03	2.83	0.00	1.64	**0.02**	*1.12*	0.03	1.88	*0.02*	**0.63**	0.06	**0.34**	**0.01**
P_5		0.76	0.02	0.30	0.01	0.48	0.00	*0.23*	**0.01**	0.33	0.01	0.27	*0.01*	**0.20**	0.01	**0.13**	**0.00**
P_6		2.30	0.14	0.85	0.03	1.11	0.00	*0.74*	**0.01**	0.97	0.11	0.79	*0.02*	**0.43**	0.03	**0.26**	**0.01**
P_9		4.28	0.09	2.16	0.03	2.81	0.00	1.73	*0.02*	*0.98*	0.13	1.84	**0.02**	**0.61**	0.03	**0.33**	**0.01**
P_{10}		0.59	0.03	0.30	0.01	0.39	0.00	*0.23*	**0.00**	0.32	0.01	0.25	*0.01*	**0.20**	0.01	**0.11**	**0.00**
P_{11}		2.90	0.38	1.08	0.04	1.65	0.00	*0.65*	**0.01**	0.97	0.24	0.73	**0.01**	**0.44**	0.04	**0.20**	*0.02*
P_{14}		3.66	0.25	2.00	0.06	2.53	0.00	1.50	**0.03**	*0.95*	0.30	1.64	*0.04*	**0.49**	0.07	**0.26**	**0.02**
P_{15}		0.69	0.04	0.30	0.01	0.36	0.00	*0.24*	*0.01*	0.32	0.01	0.25	**0.01**	**0.15**	**0.01**	**0.13**	**0.00**

Notes: P_1=TET-NFSSP with SDSTs, P_4=TotalC-NFSSP with SDSTs, P_5=Cmax-NFSSP with SDSTs, P_6=TET-NFSSP with SISTs, P_9=TotalC-NFSSP with SISTs, P_{10}=Cmax-NFSSP with SISTs, P_{11}=TET-NFSSP, P_{14}=TotalC-NFSSP, and P_{15}=Cmax-NFSSP. The results of P_2, P_3, P_7, P_8, P_{12}, and P_{13} can be found in Part 7 of the supplementary material on the website.

final parameters are listed in the extended Section IV on the website.

The statistical results under the running time factor $\rho = 2, 4$ and 6 are obtained and listed in the extended Section IV on the website. The means plots with 95% Tukey's HSD confidence intervals for MCEDA and seven compared algorithms are also shown in the extended Section IV on the website. Here, the results under $\rho = 2$ are reported in Table IV. From the corresponding results and plots on the website as well as Table IV, it can be easily seen that the *ARPD* values of MCEDA under each ρ are better than those of the other compared algorithms under the same ρ for all instances, and the *ARPD* values of MCEDA under $\rho = 2$ are better than all the *ARPD* values of the other compared algorithms under $\rho = 6$. Moreover, the *SD* values of MCEDA are still the best ones for all instances, from which it is concluded that MCEDA is also a robust one. In addition, from Tables VII and IX in the extended Section IV on the website, it is clear that MCEDA outperforms MCEDA_{nls} (i.e., MCEDA without local search). This means that the proposed local search can execute a very deep exploitation from the promising regions.

C. Comparisons of Algorithms for the NFSSP With SDSTs, the NFSSP With SISTs, and the NFSSP

Since the NFSSP with SDSTs, the NFSSP with SISTs, and the NFSSP have been widely studied in the compared algorithms in the above section, it is meaningful to test MCEDA for these problems. In this section, MCEDA is also compared with seven existing state-of-the-art algorithms in Section IV-B to testify its effectiveness for addressing the above three kinds of problems with five criteria, i.e., the TET, the TWET_(0.3*E+0.7*T), the TWET_(0.7*E+0.3*T), the total completion time (TotalC), and the Cmax. That is, there are fifteen types of problems for testing. All compared algorithms are used to solve each of these problems after replacing their original evaluation functions with the new evaluation function of the corresponding problem. The speed-up evaluation method for each type of problem is modified and used in the new evaluation functions of these algorithms (see Section III-A). Besides, MCEDA's speed-up scanning method is also used in its $FastSubS\_N_{\text{Insert}}(\boldsymbol{\pi}, i)$ by changing (21) to the corresponding criterion [see Section III-C3)]. TMIIG adds its accelerating techniques when solving the Cmax-NFSSP.

The comparison results under $\rho = 2, 4$, and 6 for fifteen different problems are listed in the corresponding fifteen tables of the extended Section IV on the website. The means plots with 95% Tukey's HSD confidence intervals for these algorithms under each problem are also given in the extended Section IV on the website. As here, the average results under $\rho = 2$ for nine problems are reported in Table V. The average results for these nine problems under $\rho = 4$ and 6 as well as the rest six problems under $\rho = 2, 4$, and 6 are omitted but provided in Part 7 of the online supplementary material. In Table V, P_i denotes the i-th problem (see the notes below Table V), $\overline{ARPD}$ is the average *ARPD* under the same ρ, and $\overline{SD}$ is the average *SD* under the same ρ. Specifically, each row in Table V is equal to the "Average" row in the corresponding nine tables of the extended Section IV on the website (like the Average row in Table IV). That is, there are a total of 15×210 results for each $\overline{ARPD}$ or $\overline{SD}$ in Table V. From the corresponding results and plots on the website as well as Table V, it is obvious that under the same ρ MCEDA statistically outperforms all the others except HDTPL for P_{15}. Moreover, except for HDTPL under $\rho = 6$ for P_2, P_7 (i.e., TWET_(0.3*E + 0.7*T)-NFSSP with SISTs) and P_{15}, MCEDA under $\rho = 2$ outperforms the others under any ρ. Hence, we can conclude that MCEDA has a powerful search engine for tackling these problems.

According to the test results in Sections IV-B and IV-C, MCEDA can achieve the best performance on most instances with one-third of the running time of the compared algorithms. The reason is that MCEDA's global search provides more effective guidance of search toward global optima and its local search uses the subsets of the Insert-based neighborhood and the problem's properties to perform more efficient exploitation. Furthermore, TMIIG, HDTPL, and IG_LS are better than MCPSO, BIH, and EEDA for most instances. The common characteristic of these better algorithms lies in the utilization of the Insert-based neighborhood for executing local search.

It is worth pointing out that all compared algorithms have similar ordinal relationships in their performance (i.e., MCEDA's ARPD value < HTDPL's ARPD value < TMIIG's ARPD value < $\cdots$) when solving the above sixteen types of problems. This verifies that the impact of the problem type on the performance of each above algorithm is not as big as we thought. Because the geometric structures of these scheduling problems (belonging to nonconvex problems) are

still unknown [34], and most of the above algorithms only use general operations and frameworks, these algorithms have very limited performance changes when solving different NFSSPs.

V. Conclusion and Future Research

This article proposes a novel MCEDA for addressing the NFSSP with SDSTs and RTs. The criterion is to minimize the TET. The test results validate the efficiency and robustness of MCEDA. This is the first report on the EDA-based algorithm for the NFSSPs.

Most of the existing literature on metaheuristic algorithms only gives a description of the specific steps of the algorithm and lacks a substantial analysis, which makes it difficult for other researchers, especially new researchers, to understand the reason for designing such algorithms. Therefore, many studies in recent years have merely combined lots of different existing steps or operations in a complicated way to obtain a so-called hybrid algorithm. This trend is not conducive to the further development of metaheuristic algorithms. This article gives an in-depth analysis of MCEDA, which helps readers to truly understand the role of each operation so that they can reasonably choose relevant operations or correctly design effective operations in their own algorithms.

In the future, we would like to design a suitable machine learning scheme to learn the dependency relationships among the variables in solution space and then add these relationships into the matrix cube to further improve the global search ability.

References

[1] C.-J. Liao and C.-C. Cheng, "A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date," *Comput. Ind. Eng.*, vol. 52, no. 4, pp. 404–413, 2007.

[2] Z. C. Li, B. Qian, R. Hu, L. L. Chang, and J. B. Yang, "An elitist nondominated sorting hybrid algorithm for multi-objective flexible job-shop scheduling problem with sequence-dependent setups," *Knowl. Based Syst.*, vol. 173, no. 1, pp. 83–112, 2019.

[3] B. Zhang, Q.-K. Pan, L. Gao, L.-L. Meng, X.-Y. Li, and K.-K. Peng, "A three-stage multiobjective approach based on decomposition for an energy-efficient hybrid flow shop scheduling problem," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 12, pp. 4984–4999, Dec. 2020.

[4] F. Yang *et al.*, "Efficient approach to scheduling of transient processes for time-constrained single-arm cluster tools with parallel chambers," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 10, pp. 3646–3657, Oct. 2020.

[5] X. Li, L. Gao, Q. Pan, L. Wan, and K.-M. Chao, "An effective hybrid genetic algorithm and variable neighborhood search for integrated process planning and scheduling in a packaging machine workshop," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 10, pp. 1933–1945, Oct. 2019.

[6] W. H. M. Raaymakersa and J. A. Hoogeveen, "Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing," *Eur. J. Oper. Res.*, vol. 126, no. 1, pp. 131–151, 2000.

[7] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *Eur. J. Oper. Res.*, vol. 187, no. 3, pp. 985–1032, 2008.

[8] A. Allahverdi, "A survey of scheduling problems with no-wait in process," *Eur. J. Oper. Res.*, vol. 255, no. 3, pp. 665–686, 2016.

[9] L. Bianco, P. Dell'Olmo, and S. Giordani, "Flow shop no-wait scheduling with sequence dependent setup times and release dates," *Inf. Syst.*, vol. 37, no. 1, pp. 3–18, 1999.

[10] P. M. França, G. Tin, Jr., and L. S. Buriol, "Genetic algorithms for the no-wait flowshop sequencing problem with time restrictions," *Int. J. Prod. Res.*, vol. 44, no. 5, pp. 939–957, 2006.

[11] L. Wan, and J. Yuan, "Single-machine scheduling to minimize the total earliness and tardiness is strongly NP-hard," *Oper. Res. Lett.*, vol. 41, no. 4, pp. 363–365, 2013.

[12] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano, "A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems," *Progr. Artif. Intell.*, vol. 1, no. 1, pp. 103–117, 2012.

[13] C. R. Reeves and T. Yamada, "Genetic algorithms, path relinking, and the flowshop sequencing problem," *Evol. Comput.*, vol. 6, no. 1, pp. 45–60, Mar. 1998.

[14] C. R. Reeves, "Landscapes, operators and heuristic search," *Ann. Oper. Res.*, vol. 86, no. 1, pp. 473–490, 1999.

[15] J. Humeau, A. Liefooghe, E.-G. Talbi, and S. Verel, "ParadisEO-MO: From fitness landscape analysis to efficient local search algorithms," *J. Heuristics*, vol. 19, no. 6, pp. 881–915, 2013.

[16] Y. Hou, Y.-S. Ong, L. Feng, and J. M. Zurada, "An evolutionary transfer reinforcement learning framework for multiagent systems," *IEEE Trans. Evol. Comput.*, vol. 21, no. 4, pp. 601–615, Aug. 2017.

[17] W. E. Hart, N. Krasnogor, and J. E. Smith, *Recent Advances in Memetic Algorithms*. Berlin, NY, USA: Springer, 2004.

[18] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong, "Classification of adaptive memetic algorithms: A comparative study," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 1, pp. 141–152, Feb. 2006.

[19] T. Schiavinotto and T. Stützle, "A review of metrics on permutations for search landscape analysis," *Comput. Oper. Res.*, vol. 34, no. 10, pp. 3143–3153, 2007.

[20] M. S. Nagano, A. A. Da Silva, and L. A. N. Lorena, "An evolutionary clustering search for the no-wait flow shop problem with sequence dependent setup times," *Expert Syst. Appl.*, vol. 41, no. 8, pp. 3628–3633, 2014.

[21] H. Samarghandi and T. Y. ElMekkawy, "Solving the no-wait flow-shop problem with sequence-dependent set-up times," *Int. J. Comput. Integr. Manuf.*, vol. 27, no. 3, pp. 213–228, 2014.

[22] H. Samarghandi, "Studying the effect of server side-constraints on the makespan of the no-wait flow-shop problem with sequence-dependent set-up times," *Int. J. Prod. Res.*, vol. 53, no. 9, pp. 2652–2673, 2015.

[23] R. Ruiz and T. Stützle, "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," *Eur. J. Oper. Res.*, vol. 187, no. 3, pp. 1143–1159, 2008.

[24] M.-E. Marmion, C. Dhaenens, L. Jourdan, A. Liefooghe, and S. Verel, "On the neutrality of flowshop scheduling fitness landscapes," in *Proc. Int. Conf. Learn. Intell. Optim.*, Berlin, Germany, 2011, pp. 238–252.

[25] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. Amsterdam, The Netherlands: Elsevier, 2004, pp. 61–112.

[26] A. Allahverdi and H. Aydilek, "Total completion time with makespan constraint in no-wait flowshops with setup times," *Eur. J. Oper. Res.*, vol. 238, no. 3, pp. 724–734, 2014.

[27] Q.-K. Pan and R. Ruiz, "An estimation of distribution algorithm for lot-streaming flow shop problems with setup times," *Omega Int. J. Manag. Sci.*, vol. 40, no. 2, pp. 166–180, 2012.

[28] B. Qian, L. Wang, R. Hu, W.-L. Wang, D.-X. Huang, and X. Wang, "A hybrid differential evolution method for permutation flow-shop scheduling," *Int. J. Adv. Manuf. Technol.*, vol. 38, nos. 7–8, pp. 757–777, 2008.

[29] B. Liu, L. Wang, Y. Liu, B. Qian, and Y.-H. Jin, "An effective hybrid particle swarm optimization for batch scheduling of polypropylene processes," *Comput. Chem. Eng.*, vol. 34, no. 4, pp. 518–528, 2010.

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

[31] B. Qian, L. Wang, R. Hu, D. X. Huang, and X. Wang, "A DE-based approach to no-wait flow-shop scheduling," *Comput. Ind. Eng.*, vol. 57, no. 3, pp. 787–805, 2009.

[32] W. Shao, D. Pi, and Z. Shao, "A hybrid discrete optimization algorithm based on teaching–probabilistic learning mechanism for no-wait flow shop scheduling," *Knowl. Based Syst.*, vol. 107, pp. 219–234, Sep. 2016.

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

[34] D. S. Chen, R. Batson, and Y. Dang, *Applied Integer Programming*. New York, NY, USA: Wiley, 2010.

Bin Qian received the B.Sc. degree in automation from Donghua University, Shanghai, China, in 1998, the M.Sc. degree in control theory and its applications from the Kunming University of Science and Technology (KMUST), Kunming, China, in 2004, and the Ph.D. degree in control science and engineering from Tsinghua University, Beijing, China, in 2009.

He is currently a Professor with KMUST. He has published over 120 referred papers. His current research interests include intelligent optimization and scheduling.

Zi-Qi Zhang received the M.Sc. degree in control theory and control engineering from the Kunming University of Science and Technology, Kunming, China, in 2017, where he is currently pursuing the Ph.D. degree.

His current research interests include evolutionary computation and scheduling.

Rong Hu received the B.Sc. degree in thermal energy and dynamic engineering from Southeast University, Nanjing, China, in 1997, and the M.Sc. degree in control science and engineering from Tsinghua University, Beijing, China, in 2004.

She is currently an Associate Professor with the Kunming University of Science and Technology, Kunming, China. She has published over 70 referred papers. Her current research interests include intelligent optimization and machine learning.

Huai-Ping Jin received the B.Sc. degree in process equipment and control engineering and the Ph.D. degree in control science and engineering from the Beijing Institute of Technology, Beijing, China, in 2010 and 2016, respectively.

He is currently an Associate Professor with the Kunming University of Science and Technology, Kunming, China. He has published over 30 referred papers. His current research interests include data-driven soft computing and scheduling.

Jian-Bo Yang received the B.Sc. and M.Sc. degrees in control and aerospace engineering from Northwestern Polytechnic University, Xi'an, China, in 1981 and 1984, respectively, and the Ph.D. degree in systems science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 1987.

He is currently a Chair Professor with the University of Manchester, Manchester, U.K., and the Visiting Changjian Chair Professor with the Hefei University of Technology, Hefei, China. He has published 4 books and over 400 papers in journals and conferences. His current research interests include artificial intelligence and decision systems.