A Hybrid Estimation of Distribution Algorithm for Unrelated Parallel Machine Scheduling with Sequence-Dependent Setup Times

Ling Wang, Shengyao Wang, and Xiaolong Zheng

***Abstract*—A hybrid estimation of distribution algorithm (EDA) with iterated greedy (IG) search (EDA-IG) is proposed for solving the unrelated parallel machine scheduling problem with sequence-dependent setup times (UPMSP-SDST). For makespan criterion, some properties about neighborhood search operators to avoid invalid search are derived. A probability model based on neighbor relations of jobs is built in the EDA-based exploration phase to generate new solutions by sampling the promising search region. Two types of deconstruction and reconstruction as well as an IG search are designed in the IG-based exploitation phase. Computational complexity of the algorithm is analyzed, and the effect of parameters is investigated by using the Taguchi method of design-of-experiment. Numerical tests on 1640 benchmark instances are carried out. The results and comparisons demonstrate the effectiveness of the EDA-IG. Especially, the best-known solutions of 531 instances are updated. In addition, the effectiveness of the properties is also demonstrated by numerical comparisons.**

***Index Terms*—Unrelated parallel machine scheduling, sequence-dependent setup time (SDST), estimation of distribution algorithm (EDA), iterated greedy search.**

I. Introduction

PARALLEL machine scheduling problem (PMSP)[1] is a typical combinatorial optimization problem existing in many manufacturing systems[2−4]. According to the characteristic of machines, PMSP can be classified into three types: identical PMSP[5], where each job has an identical processing time on every machine; uniform PMSP[6], where the processing time of a job is inversely proportional to the processing speed of a machine; and unrelated PMSP (UPMSP)[7], where the processing times of a job on different machines are unrelated. Identical PMSP and uniform PMSP can be regarded as the special cases of UPMSP. As a complex scheduling problem, PMSP is NP-hard even in a two-machine case[8]. Therefore, it is important to study PMSP and to develop effective solution algorithms, especially for the most general one, UPMSP.

Manuscript received February 29, 2016; accepted May 17, 2016. This work was supported by the National Science Fund for Distinguished Young Scholars of China (61525304). Recommended by Associate Editor Yanjun Liu.

Citation: Ling Wang, Shengyao Wang, Xiaolong Zheng. A hybrid estimation of distribution algorithm for unrelated parallel machine scheduling with sequence-dependent setup times. *IEEE/CAA Journal of Automatica Sinica*, 2016, **3**(3): 235−246

Ling Wang, Shengyao Wang, and Xiaolong Zheng are with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: wangling@mail.tsinghua.edu.cn; wangshengyao@tsinghua.org.cn; zhengxl11@mails.tsinghua.edu.cn).

In the literature about the UPMSP, it is common to assume that there is no additional time between any two processing operations[7]. However, setup times widely exist in practice when preparing machines for the coming jobs[9]. Setup times are relevant to the job sequences on machines. During recent years, the UPMSP with sequence-dependent setup times (UPMSP-SDST) has been a focus in the area of PMSP. According to the reported research work before 2008, a brief review of the UPMSP-SDST was presented in [10]. Here, we focus on the typical literature after 2008.

To minimize the makespan of the UPMSP-SDST, Behnamian et al.[11] presented a hybrid algorithm based on ant colony optimization (ACO), simulated annealing (SA), and variable neighborhood search (VNS). The population was initialized by ACO and evolved by SA and VNS. Chang and Chen[12] presented a set of dominance properties as necessary conditions for inter- and intra-machine switching of job sequencing orders in a schedule. Besides, they proposed a genetic algorithm (GA) by integrating these dominance properties. Vallada and Ruiz[10] proposed a GA with a fast local search and a local-search-enhanced crossover operator. They also developed sets of benchmark instances to test the performances of algorithms in solving the UPMSP-SDST. It was shown that the proposed GA outperformed other evaluated methods in solving those benchmark instances. Besides, Fleszar et al.[13] hybridized the variable neighborhood descent (VND) with mathematical programming. Lin and Ying[14] presented a hybrid artificial bee colony algorithm. Yilmaz et al.[15] developed a GA with local search. Arnaout et al.[16] proposed an ACO and then enhanced the ACO by using a pheromone re-initialization method. Avalos-Rosales et al.[17] proposed a metaheuristic algorithm based on the multi-start algorithm and VND.

Considering the additional due-date constraints, some work aimed at minimizing the total tardiness of jobs. Chen[18] presented a hybrid heuristic based on the modified apparent-tardiness-cost-with-setup procedure, SA method, and improvement procedures. Lin et al.[19] presented an iterated greedy (IG) heuristic, which was shown to be more effective and efficient than the state-of-the-art algorithms.

To minimize the total completion time of all jobs, Hsu et al.[20] showed that there existed a polynomial time method for the problem with the past-sequence-dependent setup time and learning effects on machines. Then, they derived similar conclusions when minimizing the total absolute deviation of job completion times and the total load on all machines[21].

By supposing that a machine had a production availability, Joo and Kim[22] proposed a hybrid GA with three dispatching rules to minimize the total completion time.

For other objectives, Chen and Chen[23] developed several hybrid metaheuristics by integrating VND approach and tabu search to minimize the weighted number of tardy jobs. Lin and Ying[4] proposed a multi-point multi-objective SA algorithm to simultaneously minimize makespan, total weighted completion time, and total weighted tardiness.

As a population-based evolutionary algorithm, estimation of distribution algorithm (EDA) employs a probability model for optimization[24]. By sampling a promising area of the search space to generate new solutions, the EDA has a strong ability of global exploration. Due to its merits, the EDA has already been successfully applied to solve a variety of academic and engineering optimization problems[25]. However, to the best of our knowledge, it has never been applied to the UPMSP-SDST. Meanwhile, the EDA often shows good global exploration capability while its exploitation capability is relatively weak. Therefore, it motivates us to propose a hybrid EDA with the iterated greedy search[26] and especially some problem-specific properties, named EDA-IG, to balance the global exploration and the local exploitation for the UPMSP-SDST with makespan criterion. To be specific, some properties for judging the effectiveness of neighborhood search operators are analyzed to avoid invalid search operators. Based on neighbor relations of the jobs, a probability model is built to generate new solutions for global exploration. In addition, two types of deconstruction and reconstruction are designed to develop an IG search for local exploitation. With 1640 benchmark instances, numerical tests are carried out to show the effectiveness of the proposed EDA-IG and the properties.

The remainder of the paper is organized as follows: First, the UPMSP-SDST is briefly described in Section II. Then, after analyzing the properties of the neighborhood search operators in Section III, the EDA-IG for solving the UPMSP-SDST is presented in Section IV. In Section V, the influence of parameter setting is investigated and numerical results are provided. Finally the paper is ended with some conclusions and future work in Section VI.

II. Problem Description

A. Notation

n: the number of jobs to be processed;

m: the number of machines;

J_i: the i-th job;

M_k: the k-th machine;

Cmax: the maximum completion time of jobs, i.e., makespan;

C_i: the completion time of J_i;

$P_{i,k}$: the processing time of J_i on M_k;

$ST_{i,j,k}$: the setup time of processing J_j right after J_i on M_k;

$ST_{0,j,k}$: the setup time of processing J_j first on M_k;

$X_{i,j,k}$: a binary variable that is equal to 1 if J_j is right after J_i on M_k, 0 otherwise;

$X_{0,j,k}$: a binary variable that is equal to 1 if J_j is the first one on M_k, 0 otherwise;

$X_{j,0,k}$: a binary variable that is equal to 1 if J_j is the last one on M_k, 0 otherwise;

V: a very large constant;

R_k: the release (completion) time of M_k;

N_k: the total number of jobs on M_k;

$P_{[j],k}$: the processing time of the j-th job on M_k;

$ST_{[i],[j],k}$: the setup time of processing the j-th job right after the i-th one on M_k;

$AP_{i,j,k}$: the total needed (setup and processing) time for the j-th job right after the i-th one on M_k;

$ST_{[0],[1],k}$: the setup time of processing the first job on M_k;

$AP_{0,1,k}$: the total needed (setup and processing) time for the first job on M_k.

B. Mathematical Model

The UPMSP-SDST with makespan criterion can be described as follows: There are n jobs to be processed by m machines. Each job has only one operation, which can be executed on any machine. Setup time is needed before processing a job. Setup time is job-sequence-dependent and machine-dependent. The objective is to determine the job processing sequences on all the machines to minimize the maximum completion time of all jobs. Usually, it assumes that: All the jobs and machines are available at the initial time; and each machine can process only one job, and each job can be processed on only one machine at a time; and once an operation starts, it cannot be interrupted before its completion. Mathematically, the UPMSP-SDST with makespan criterion can be formulated as follows[27]:

$$\min C_{\max} = \max\{C_i | i = 1, 2, \ldots, n\}, \tag{1}$$

Subject to:

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} \sum_{k=1}^{m} X_{i,j,k} = 1, \ \forall j = 1, 2, \ldots, n, \tag{2}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{n} X_{i,h,k} - \sum_{\substack{j=0 \\ j \neq h}}^{n} X_{h,j,k} = 0,$$
$$\forall h = 1, 2, \ldots, n; \forall k = 1, 2, \ldots, m, \tag{3}$$

$$C_j \geq C_i + \sum_{k=1}^{m} X_{i,j,k}(ST_{i,j,k} + P_{j,k}) + V(\sum_{k=1}^{m} X_{i,j,k} - 1),$$
$$\forall i = 0, 1, \ldots, n; \forall j = 1, 2, \ldots, n, \tag{4}$$

$$\sum_{j=0}^{n} X_{0,j,k} = 1, \ \forall k = 1, 2, \ldots, m, \tag{5}$$

$$X_{i,j,k} \in \{0, 1\}, \forall i, j = 0, 1, \ldots, n; \forall k = 1, 2, \ldots, m, \tag{6}$$

$$C_0 = 0, \tag{7}$$

$$C_j \geq 0, \forall j = 1, \ldots, n. \tag{8}$$

Equation (1) is the objective function. Equation (2) ensures that every job should be processed only once. Equation (3) implies that each job has a predecessor and a successor. Note that, a dummy job J_0 is used to present the first job on a machine. Equation (4) ensures that the processing of a

job should be started after the completion of its predecessor. Equation (5) ensures that each machine has only one dummy job. Equation (6) defines the binary variables. Equation (7) implies that the completion time of a dummy job is zero. Equation (8) ensures that the completion time of a regular job is non-negative.

III. Properties of Neighborhood Search Operators

For shop scheduling problems, some typical neighborhood search operators are often used to find better solutions. Regarding the UPMSP-SDST, five operators are usually employed, including three inter-machine operators and two intra-machine operators.

A. Typical Operators for UPMSP-SDST

The inter-machine operators are as follows (Let $j > i$ without loss of generality):

1) SWS(i, j, k): Swap the i-th job and the j-th one on machine M_k.

2) ISS(i, j, k): Insert the i-th job after the j-th one on machine M_k.

3) RVS(i, j, k): Reverse the sub-sequence between the i-th job and the j-th one on M_k.

Note that, ISS(i, j, k) and RVS(i, j, k) are SWS(i, j, k) when $j = i + 1$. Hence, for ISS(i, j, k) and RVS(i, j, k), only case $j > i + 1$ is considered.

The intra-machine operators are as follows ($k1 \neq k2$):

1) SWD(i, j, $k1$, $k2$): Swap the i-th job on M_{k1} and the j-th job on M_{k2}.

2) ISD(i, j, $k1$, $k2$): Insert the i-th job on M_{k1} after the j-th job on M_{k2}.

B. Properties of the Operators

To calculate the makespan of the new solution after applying a search operator, the computational effort can be reduced with a speed-up evaluation procedure[13]. In addition, it is possible to identify invalid operators by some necessary conditions of the corresponding operators[15]. Inspired by the above, we derive some necessary and sufficient conditions for the neighborhood search operators.

Definition 1. If $R_{k^*} = \max\{R_k | k = 1, 2, \ldots, m\}$, then M_{k^*} is called a critical machine, where $R_k = \max\{X_{i,j,k} C_j | i = 0, 1, \ldots, n; j = 1, 2, \ldots, n\}$.

Obviously, the release times of the critical machines are equal to the makespan of a schedule. With makespan criterion, a neighborhood search operator is effective if the completion times of all the critical machines can be shortened.

Property 1. The makespan can be reduced by SWS(i, j, k) iff $R_k > R_{k'}$ ($\forall k' \neq k$) and:

1) When $j = i + 1$, $ST_{[i-1],[j],[k]} + ST_{[j],[i],[k]} + ST_{[i],[j+1],[k]} < ST_{[i-1],[i],[k]} + ST_{[i],[j],[k]} + ST_{[j],[j+1],[k]}$;

2) When $j > i + 1$, $ST_{[i-1],[j],[k]} + ST_{[j],[i+1],[k]} + ST_{[j-1],[i],[k]} + ST_{[i],[j+1],[k]} < ST_{[i-1],[i],[k]} + ST_{[i],[i+1],[k]} + ST_{[j-1],[j],[k]} + ST_{[j],[j+1],[k]}$.

Proof.

1) Case $j = i + 1$.

Before execution:

$$R_k = \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,i,k} + AP_{i,j,k} + AP_{j,j+1,k} + \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k}.$$

After execution:

$$R'_k = \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,j,k} + AP_{j,i,k} + AP_{i,j+1,k} + \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k},$$

$$\begin{aligned}
\triangle_k &= R'_k - R_k \\
&= AP_{i-1,j,k} + AP_{j,i,k} + AP_{i,j+1,k} - AP_{i-1,i,k} \\
&\quad - AP_{i,j,k} - AP_{j,j+1,k} \\
&= (ST_{[i-1],[j],[k]} + P_{[j],k}) + (ST_{[j],[i],[k]} + P_{[i],k}) \\
&\quad + (ST_{[i],[j+1],[k]} + P_{[j+1],k}) - (ST_{[i-1],[i],[k]} + P_{[i],k}) \\
&\quad - (ST_{[i],[j],[k]} + P_{[i],k}) - (ST_{[j],[j+1],[k]} + P_{[j+1],k}) \\
&= ST_{[i-1],[j],[k]} + ST_{[j],[i],[k]} + ST_{[i],[j+1],[k]} \\
&\quad - ST_{[i-1],[i],[k]} - ST_{[i],[j],[k]} - ST_{[j],[j+1],[k]}.
\end{aligned}$$

2) Case $j > i + 1$.

Before execution:

$$R_k = \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,i,k} + AP_{i,i+1,k} + \sum_{a=i+1}^{j-2} AP_{a,a+1,k} + AP_{j-1,j,k} + +AP_{j,j+1,k} + \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k}.$$

After execution:

$$R'_k = \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,j,k} + AP_{j,i+1,k} + \sum_{a=i+1}^{j-2} AP_{a,a+1,k} + AP_{j-1,i,k} + +AP_{i,j+1,k} + \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k},$$

$$\begin{aligned}
\triangle_k &= R'_k - R_k \\
&= AP_{i-1,j,k} + AP_{j,i+1,k} + AP_{j-1,i,k} \\
&\quad + AP_{i,j+1,k} - (AP_{i-1,i,k} + AP_{i,i+1,k} \\
&\quad + AP_{j-1,j,k} + AP_{j,j+1,k}) \\
&= (ST_{[i-1],[j],[k]} + P_{[j],k}) + (ST_{[j],[i+1],[k]} + P_{[i+1],k}) \\
&\quad + (ST_{[j-1],[i],[k]} + P_{[i],k}) + (ST_{[i],[j+1],[k]} + P_{[j+1],k}) \\
&\quad - (ST_{[i-1],[i],[k]} + P_{[i],k}) - (ST_{[i],[i+1],[k]} + P_{[i+1],k}) \\
&\quad - (ST_{[j-1],[j],[k]} + P_{[j],k}) - (ST_{[j],[j+1],[k]} + P_{[j+1],k}) \\
&= ST_{[i-1],[j],[k]} + ST_{[j],[i+1],[k]} + ST_{[j-1],[i],[k]} \\
&\quad + ST_{[i],[j+1],[k]} - ST_{[i-1],[i],[k]} - ST_{[i],[i+1],[k]} \\
&\quad - ST_{[j-1],[j],[k]} - ST_{[j],[j+1],[k]}.
\end{aligned}$$

Because $R_{k'}$ ($\forall k' \neq k$) remains unchanged after operator SWS(i, j, k), the makespan can be reduced by SWS(i, j, k)

iff M_k is the only critical machine and R_k is smaller, i.e., $R_k > R_{k'}\ (\forall k' \neq k)$ and $\triangle_k < 0$. □

Property 2. The makespan can be reduced by ISS(i, j, k) iff $R_k > R_{k'}(\forall k' \neq k)$, $ST_{[i-1],[i+1],[k]} + ST_{[j],[i],[k]} + ST_{[i],[j+1],[k]} < ST_{[i-1],[i],[k]} + ST_{[i],[i+1],[k]} + ST_{[j],[j+1],[k]}$.

Proof.

$$\begin{aligned} R_k =& \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,i,k} + AP_{i,i+1,k} \\ &+ \sum_{a=i+1}^{j-1} AP_{a,a+1,k} + AP_{j,j+1,k} \\ &+ \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k}, \end{aligned}$$

$$\begin{aligned} R'_k =& \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,i+1,k} \\ &+ \sum_{a=i+1}^{j-1} AP_{a,a+1,k} + AP_{j,i,k} + AP_{i,j+1,k} \\ &+ \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k}, \end{aligned}$$

$$\begin{aligned} \triangle_k &= R'_k - R_k \\ &= AP_{i-1,i+1,k} + AP_{j,i,k} + AP_{i,j+1,k} \\ &\quad - (AP_{i-1,i,k} + AP_{i,i+1,k} + AP_{j,j+1,k}) \\ &= (ST_{[i-1],[i+1],[k]} + P_{[i+1],k}) + (ST_{[j],[i],[k]} \\ &\quad + P_{[i],k}) + (ST_{[i],[j+1],[k]} + P_{[j+1],k}) \\ &\quad - (ST_{[i-1],[i],[k]} + P_{[i],k}) - (ST_{[i],[i+1],[k]} + P_{[i+1],k}) \\ &\quad - (ST_{[j],[j+1],[k]} + P_{[j+1],k}) \\ &= ST_{[i-1],[i+1],[k]} + ST_{[j],[i],[k]} + ST_{[i],[j+1],[k]} \\ &\quad - ST_{[i-1],[i],[k]} - ST_{[i],[i+1],[k]} - ST_{[j],[j+1],[k]}. \end{aligned}$$

Because $R_{k'}\ (\forall k' \neq k)$ remains unchanged after operator ISS(i, j, k), the makespan can be reduced by ISS(i, j, k) iff M_k is the only critical machine and R_k is smaller, i.e., $R_k > R_{k'}\ (\forall k' \neq k)$ and $\triangle_k < 0$. □

Property 3. The makespan can be reduced by RVS(i, j, k) iff $R_k > R_{k'}\ (\forall k' \neq k)$, $ST_{[i-1],[j],[k]} + ST_{[i],[j+1],[k]} + \sum_{a=i}^{j-1} ST_{[a+1],[a],[k]} < ST_{[i-1],[i],[k]} + ST_{[j],[j+1],[k]} + \sum_{a=i}^{j-1} ST_{[a],[a+1],[k]}$.

Proof.

$$\begin{aligned} R_k =& \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,i,k} \\ &+ \sum_{a=i}^{j-1} AP_{a,a+1,k} + AP_{j,j+1,k} + \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k}, \end{aligned}$$

$$\begin{aligned} R'_k =& \sum_{a=1}^{i-2} AP_{a,a+1,k} + AP_{i-1,j,k} \\ &+ \sum_{a=i}^{j-1} AP_{a+1,a,k} + AP_{i,j+1,k} + \sum_{a=j+1}^{N_k-1} AP_{a,a+1,k}, \end{aligned}$$

$$\begin{aligned} \triangle_k &= R'_k - R_k \\ &= AP_{i-1,j,k} + AP_{i,j+1,k} + \sum_{a=i}^{j-1} AP_{a+1,a,k} \\ &\quad - (AP_{i-1,i,k} + AP_{j,j+1,k} + \sum_{a=i}^{j-1} AP_{a,a+1,k}) \\ &= (ST_{[i-1],[j],[k]} + P_{[j],k}) + (ST_{[i],[j+1],[k]} + P_{[j+1],k}) \\ &\quad + \sum_{a=i}^{j-1} (ST_{[a+1],[a],[k]} + P_{[a],k}) \\ &\quad - (ST_{[i-1],[i],[k]} + P_{[i],k}) - (ST_{[j],[j+1],[k]} + P_{[j+1],k}) \\ &\quad - \sum_{a=i}^{j-1} (ST_{[a],[a+1],[k]} + P_{[a+1],k}) \\ &= ST_{[i-1],[j],[k]} + ST_{[i],[j+1],[k]} + \sum_{a=i}^{j-1} ST_{[a+1],[a],[k]} \\ &\quad - ST_{[i-1],[i],[k]} - ST_{[j],[j+1],[k]} - \sum_{a=i}^{j-1} ST_{[a],[a+1],[k]}. \end{aligned}$$

Because $R_{k'}\ (\forall k' \neq k)$ remains unchanged after operator RVS(i, j, k), the makespan can be reduced by RVS(i, j, k) iff M_k is the only critical machine and R_k is smaller, i.e., $R_k > R_{k'}\ (\forall k' \neq k)$ and $\triangle_k < 0$. □

Property 4. The makespan can be reduced by SWD(i, j, $k1$, $k2$) iff $R_{k1} = C_{\max}$, $R_{k'} < C_{\max}\ (\forall k' \neq k1, k2)$, $AP_{[i-1],[j],[k1]} + ST_{[j],[i+1],[k1]} < AP_{[i-1],[i],[k1]} + ST_{[i],[i+1],[k1]}$, $R_{k2} + AP_{[j-1],[i],[k2]} + ST_{[i],[j+1],[k2]} < R_{k1} + AP_{[j-1],[j],[k2]} + ST_{[j],[j+1],[k2]}$.

Proof.

$$\begin{aligned} R_{k1} =& \sum_{a=1}^{i-2} AP_{a,a+1,k1} + AP_{i-1,i,k1} + AP_{i,i+1,k1} \\ &+ \sum_{a=i+1}^{N_{k1}-1} AP_{a,a+1,k1}, \end{aligned}$$

$$\begin{aligned} R'_{k1} =& \sum_{a=1}^{i-2} AP_{a,a+1,k1} + AP_{i-1,j,k1} + AP_{j,i+1,k1} \\ &+ \sum_{a=i+1}^{N_{k1}-1} AP_{a,a+1,k1}, \end{aligned}$$

$$\begin{aligned} R_{k2} =& \sum_{a=1}^{j-2} AP_{a,a+1,k2} + AP_{j-1,j,k2} + AP_{j,j+1,k2} \\ &+ \sum_{a=j+1}^{N_{k2}-1} AP_{a,a+1,k2}, \end{aligned}$$

$$R'_{k2} = \sum_{a=1}^{j-2} AP_{a,a+1,k2} + AP_{j-1,i,k2} + AP_{i,j+1,k2} + \sum_{a=j+1}^{N_{k2}-1} AP_{a,a+1,k2},$$

$$\begin{aligned}\triangle_{k1} &= R'_{k1} - R_{k1} \\ &= AP_{i-1,j,k1} + AP_{j,i+1,k1} - (AP_{i-1,i,k1} + AP_{i,i+1,k1}) \\ &= AP_{i-1,j,k1} + (ST_{[j],[i+1],[k1]} + P_{[i+1],k1}) \\ &\quad - AP_{i-1,i,k1} - (ST_{[i],[i+1],[k1]} + P_{[i+1],k1}) \\ &= AP_{i-1,j,k1} + ST_{[j],[i+1],[k1]} - AP_{i-1,i,k1} \\ &\quad - ST_{[i],[i+1],[k1]},\end{aligned}$$

$$\begin{aligned}\triangle_{k2} &= R'_{k2} - R_{k2} \\ &= AP_{j-1,i,k2} + AP_{i,j+1,k2} - (AP_{j-1,j,k2} + AP_{j,j+1,k2}) \\ &= AP_{i-1,j,k2} + (ST_{[i],[j+1],[k2]} + P_{[j+1],k2}) \\ &\quad - AP_{j-1,j,k2} - (ST_{[j],[j+1],[k2]} + P_{[j+1],k2}) \\ &= AP_{i-1,j,k2} + ST_{[i],[j+1],[k2]} - AP_{j-1,j,k2} \\ &\quad - ST_{[j],[j+1],[k2]}.\end{aligned}$$

Because $R_{k'}$ $(\forall k' \neq k1, k2)$ remains unchanged after operator SWD(i, j, $k1$, $k2$), the makespan can be reduced by SWD(i, j, $k1$, $k2$) iff there is no other critical machine except M_{k1} and M_{k2} and $R'_{k1}, R'_{k2} < C_{\max}$.

Without loss of generality, suppose M_{k1} is a critical machine, i.e., $R'_{k1} = C_{\max}$. The above conditions are equivalent to $\triangle_{k1} < 0$ and $R_{k2} + \triangle_{k2} < R_{k1}$. □

Property 5. The makespan can be reduced by ISD(i, j, $k1$, $k2$) iff $R_{k1} = C_{\max}$, $R_{k'} < C_{\max}$ $(\forall k' \neq k1, k2)$, $ST_{[i-1],[i+1],[k1]} < AP_{[i-1],[i],[k1]} + ST_{[i],[i+1],[k1]}$, $R_{k2} + AP_{[j],[i],[k2]} + ST_{[i],[j+1],[k2]} < R_{k1} + ST_{[j],[j+1],[k2]}$.

Proof.

$$R_{k1} = \sum_{a=1}^{i-2} AP_{a,a+1,k1} + AP_{i-1,i,k1} + AP_{i,i+1,k1} + \sum_{a=i+1}^{N_{k1}-1} AP_{a,a+1,k1},$$

$$R'_{k1} = \sum_{a=1}^{i-2} AP_{a,a+1,k1} + AP_{i-1,i+1,k1} + \sum_{a=i+1}^{N_{k1}-1} AP_{a,a+1,k1},$$

$$R_{k2} = \sum_{a=1}^{j-1} AP_{a,a+1,k2} + AP_{j,j+1,k2} + \sum_{a=j+1}^{N_{k2}-1} AP_{a,a+1,k2},$$

$$R'_{k2} = \sum_{a=1}^{j-1} AP_{a,a+1,k2} + AP_{j,i,k2} + AP_{i,j+1,k2} + \sum_{a=j+1}^{N_{k2}-1} AP_{a,a+1,k2},$$

$$\begin{aligned}\triangle_{k1} &= R'_{k1} - R_{k1} \\ &= AP_{i-1,i+1,k1} - (AP_{i-1,i,k1} + AP_{i,i+1,k1}) \\ &= (ST_{[i-1],[i+1],[k1]} + P_{[i+1],k1}) - AP_{i-1,i,k1} \\ &\quad - (ST_{[i],[i+1],[k1]} + P_{[i+1],k1}) \\ &= ST_{[i-1],[i+1],[k1]} - AP_{i-1,i,k1} - ST_{[i],[i+1],[k1]},\end{aligned}$$

$$\begin{aligned}\triangle_{k2} &= R'_{k2} - R_{k2} \\ &= AP_{j,i,k2} + AP_{i,j+1,k2} - AP_{j,j+1,k2} \\ &= AP_{j,i,k2} + (ST_{[i],[j+1],[k2]} + P_{[j+1],k2}) \\ &\quad - (ST_{[j],[j+1],[k2]} + P_{[j+1],k2}) \\ &= AP_{j,i,k2} + ST_{[i],[j+1],[k2]} - ST_{[j],[j+1],[k2]}.\end{aligned}$$

Because $R_{k'}$ $(\forall k' \neq k1, k2)$ remains unchanged after operator ISD(i, j, $k1$, $k2$), the makespan can be reduced by ISD(i, j, $k1$, $k2$) iff there is no other critical machine except M_{k1} and M_{k2} and $R'_{k1}, R'_{k2} < C_{\max}$.

Without loss of generality, suppose M_{k1} is a critical machine, i.e., $R'_{k1} = C_{\max}$. The above conditions are equivalent to $\triangle_{k1} < 0$ and $R_{k2} + \triangle_{k2} < R_{k1}$. □

With a corresponding property, the effectiveness of an operator can be identified. It can be used to avoid the invalid neighborhood search and the waste of the computation budget. Besides, after applying a valid operator, the makespan of the new solution can be calculated by a speed-up strategy based on the variation of machine release times.

IV. EDA-IG For UPMSP-SDST

In this section, the EDA-IG for solving the UPMSP-SDST is presented in details. First, the solution representation, initialization mechanism, probability model and its updating mechanism, and IG search are proposed. Then, the flowchart of the EDA-IG is provided. Besides, the computational complexity of the EDA-IG is analyzed.

A. Solution Representation

The population is composed by a set of solutions for the UPMSP-SDST. A solution is represented by m job sequences as illustrated in Fig. 1. Each sequence represents the order of processing jobs on the corresponding machine. For example, element $\pi_k(i)$ implies that $J_{\pi_k(i)}$ is the i-th job on M_k.

$$\pi = \begin{cases} \pi_1(1), \pi_1(2), \cdots, \pi_1(n_1) \\ \pi_2(1), \pi_2(2), \cdots, \pi_2(n_2) \\ \vdots \\ \pi_m(1), \pi_m(2), \cdots, \pi_m(n_m) \end{cases}$$

Fig. 1. Solution representation in the EDA-IG.

B. Initialization Mechanism

Considering both the quality and diversity of initial solutions, the smallest release-time and earliest completion (SR_EC) rule is employed for initialization. To be specific, to determine the job sequences on the machines, it selects a machine with the smallest release-time and assigns an unprocessed job which results in the earliest completion time of the machine. If more than one machine or job meets the requirement, it randomly selects one from them. A new solution is obtained after all the jobs are assigned. The pseudo code to generate an initial solution is shown in Fig. 2.

INPUT: Job set $\Omega = \{J_1, \ldots, J_n\}$
OUTPUT: Solution $\boldsymbol{\pi}$
FOR k = 1 to m DO
  $N_k = 0$;
  $\pi_k(0) = 0$;
  $R_k = 0$;
END FOR
FOR j = 1 to n DO
  $l = \arg\min_k \{R_k\}$;
  $h = \arg\min_{J_l \in \Omega} \{R_l + ST_{\pi_l(N_l),i,l} + P_{i,l}\}$;
  $N_l = N_l + 1$;
  $\pi_l(N_l) = h$;
  $\Omega = \Omega \setminus J_h$;
END FOR

Fig. 2. The procedure to generate an initial solution.

With the above procedure, a total of $Psize$ initial solutions are generated. Because the SR_EC rule is a greedy assignment, it is helpful to obtain a solution with a small makespan. Meanwhile, when generating a new solution, initial release times of all the machines are the same, i.e., zero. Hence, the order of the first m assigned machines is random, which ensures the diversity of initial solutions. Therefore, the initialization mechanism is beneficial to both the quality and diversity of initial solutions.

C. Probability Model and Updating Mechanism

The crucial factor of the EDA is the probability model that describes the distribution of the search space. Usually, the probability model is built according to the characteristics of the elite solutions. For the UPMSP-SDST, the neighboring relations of the jobs on the machines affect the makespan of the solution. Therefore, a probability model is designed as m probability matrices as follows:

$$P^k(g) = \begin{bmatrix} p^k_{01}(g) & p^k_{02}(g) & \ldots & p^k_{0n}(g) \\ 0 & p^k_{12}(g) & \ldots & p^k_{1n}(g) \\ p^k_{21}(g) & 0 & \ldots & p^k_{2n}(g) \\ \vdots & \vdots & \ddots & \vdots \\ p^k_{n1}(g) & p^k_{n2}(g) & \ldots & 0 \end{bmatrix}, k = 1, 2, \ldots, m, \tag{9}$$

where $p^k_{ij}(g)$ represents the probability that J_j is right after J_i on M_k at the g-th generation. Besides, $p^k_{0j}(g)$ represents the probability that J_j is the first job on M_k . Note that, $\forall j, g, p^k_{jj}(g) = 0$.

The matrices are initialized uniformly as follows:

$$P^k(0) = \begin{bmatrix} 1/n & 1/n & \ldots & 1/n \\ 0 & 1/(n-1) & \ldots & 1/(n-1) \\ 1/(n-1) & 0 & \ldots & 1/(n-1) \\ \vdots & \vdots & \ddots & \vdots \\ 1/(n-1) & 1/(n-1) & \ldots & 0 \end{bmatrix}, \forall k. \tag{10}$$

At each generation, the probability matrices are updated by the information of some elite solutions of the population. The elite sub-population consists of the best $SPsize$ solutions, where $SPsize = \eta\%Psize$. To be specific, the updating process can be regarded as the following incremental learning:

$$p^k_{ij}(g+1) = (1-\alpha)p^k_{ij}(g) + \frac{\alpha}{SPsize} \sum_{s=1}^{SPsize} I^s_{ijk}(g), \tag{11}$$

where $\alpha \in (0, 1)$ is the learning rate, and $I^s_{ijk}(g)$ is the indicator function corresponding to the s-th solutions of the elite sub-population.

$$I^s_{ijk}(g) = \begin{cases} 1, & \text{if } J_j \text{ is right after } J_i \text{ on } M_k, \\ 0, & \text{else.} \end{cases} \tag{12}$$

Then, new solutions are generated by sampling the probability matrices. To generate a solution, the jobs are assigned to machines one by one. For a machine with the smallest release-time, a job is selected by using the roulette wheel method. The procedure to generate a solution by sampling the probability model is illustrated in Fig. 3.

INPUT: Job set $\Omega = \{J_1, \ldots, J_n\}$
OUTPUT: Solution $\boldsymbol{\pi}$
FOR k = 1 to m DO
  $N_k = 0$;
  $\pi_k(0) = 0$;
  $R_k = 0$;
END FOR
FOR j = 1 to n DO
  $l = \arg\min_k \{R_k\}$;
  Select J_h with a probability of $p^l_{\pi_l(N_l),h}$ by using the roulette wheel method;
  $N_l = N_l + 1$;
  $\pi_l(N_l) = h$;
  Set the h-th column of $\boldsymbol{P}^l$ as zero;
END FOR

Fig. 3. The procedure to generate a solution by sampling.

D. IG Search

The EDA pays much attention to global exploration while its exploitation capability is relatively limited. Therefore, an IG search is developed to enhance the local exploitation of the EDA.

The IG method was introduced by Ruiz and Stützle[26] and has been applied for the UPMSP without SDST[7]. The IG method starts from a schedule and iterates three phases: deconstruction, reconstruction, and improvement. First, some jobs are randomly removed in the deconstruction phase. Then,

these jobs are reassigned one by one in a greedy way in the reconstruction phase. Afterwards, local search operators are implemented in the improvement phase. If the new solution is better, it will replace the old one; else, the next iteration starts from the old solution.

For the UPMSP-SDST, two types of IG method (named IG1 and IG2 respectively) are developed. The three phases of IG1 and IG2 are introduced as follows:

1) Deconstruction phase.

IG1: For each machine, randomly select a job and remove it.

IG2: For each machine, randomly select a job and remove all the jobs after it.

2) Reconstruction phase.

IG1: Suppose to insert each removed job into every position on all the machines. Record the job and the position which result in the smallest makespan. Then, insert the job into the position. Repeat the procedure until all the removed jobs are reassigned.

IG2: Reassign all the removed jobs based on the SR_EC rule.

3) Improvement phase.

The improvement phases of IG1 and IG2 are the same. If the solution obtained in the reconstruction phase has only one critical machine, neighborhood search operators are executed one after another. Besides, Properties 1-5 are used to avoid invalid operators. The procedure of the improvement phase is illustrated in Fig. 4.

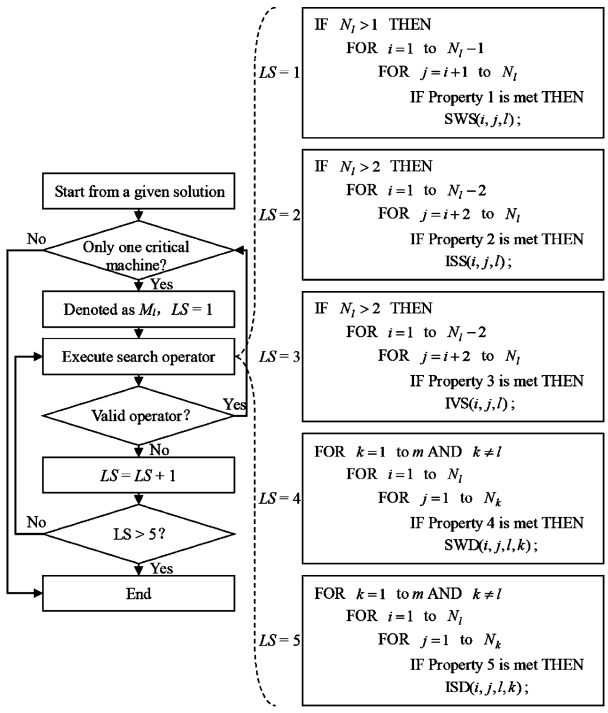

Fig. 4. The procedure of the improvement phase.

At each generation of the EDA-IG, IG search is implemented on the best solution of the population. Considering search sufficiency and computation budget, IG search iterates until the solution has not been updated in γ continuous iterations. Parameter γ implies the intensity of the local exploitation. The parameter setting for γ will be investigated later, and a rule for selecting a type of IG according to the job and machine numbers will be presented as well.

E. Flowchart of EDA-IG

With the above design, the flowchart of the EDA-IG for solving the UPMSP-SDST is illustrated in Fig. 5.

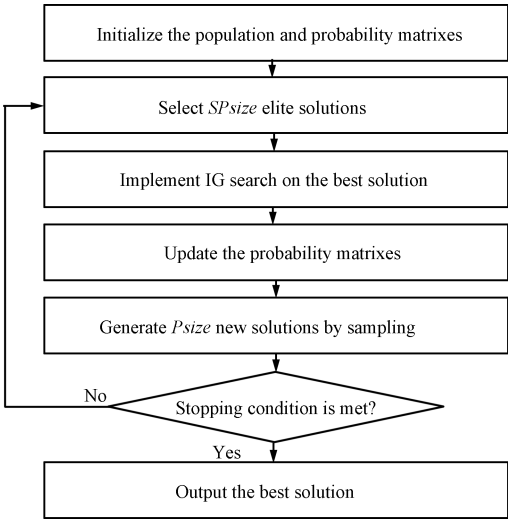

Fig. 5. The flowchart of the EDA-IG.

At the initial stage of evolution process, the whole search space is sampled uniformly. Then, the algorithm uses the EDA-based evolutionary search mechanism to sample a potential area. Moreover, IG search is implemented in a promising region, aiming to obtain better solutions. With the benefit of combining EDA and IG search, global exploration and local exploitation are balanced. The stopping criterion is set with a maximum elapsed CPU time of $n \times (m/2) \times t$ ms.

F. Computational Complexity Analysis

For each generation of the EDA-IG, the computational complexity can be roughly analyzed as follows:

1) EDA: For the updating process, first it is with a computational complexity of O($Psize\cdot \log Psize$) by using the quick sorting method to sort and select elite solutions; then, it is with a computational complexity of O$[n \cdot (Psize + m \cdot n)]$ to update the probability matrices. For the sampling process, it generates a new solution with a computational complexity of O$[n \cdot (m + n)]$.

2) IG: The deconstruction phase is with a computational complexity of O(m). The reconstruction phases of IG1 and IG2 are with computational complexities of O$(m^2 \cdot n)$ and O$[n \cdot (m + n)]$, respectively. The improvement phase is with a computational complexity of O(n^2). Therefore, the computational complexities of IG1 and IG2 are O$(m^2 \cdot n + n^2)$ and O$(m \cdot n + n^2)$, respectively. Note that, IG2 needs less computation budget than IG1.

V. Numerical Results And Comparisons

To test the performance of the EDA-IG, numerical tests are carried out with two sets of benchmark instances[10], which are

available at http://soa.iti.es. The first set consists of 640 small-scale instances, where n = {6, 8, 10, 12} and m= {2, 3, 4, 5}. The second set consists of 1000 large-scale instances, where n = {50, 100, 150, 200, 250} and m = {10, 15, 20, 25, 30}. The processing times are uniformly distributed between 1 and 99. The setup times consist of four sub-sets, which are uniformly distributed between 1-9, 1-49, 1-99 and 1-124, respectively.

To evaluate the performance of the EDA-IG, same as the literature[10], the experimental results are evaluated by relative percentage deviation (RPD) as follows:

$$\text{RPD} = \frac{alg - opt}{opt} \times 100. \tag{13}$$

A. Parameters Setting

The EDA-IG has four key parameters to be set: $Psize$ (population size), η (percentage of elite solutions), α (learning rate), and γ (intensity of IG search). To investigate the influence of these parameters on the performance of the EDA-IG, the Taguchi method of design-of-experiment (DOE)[28] is implemented by using a moderate-scale instance I_100_10_S_1-124_1. Besides, the stopping criterion is set with a maximum elapsed CPU time of $n \times (m/2) \times 30$ ms, and IG2 method is used in the EDA-IG.

Four factor levels are employed. Different values of these parameters are listed in Table I. Accordingly, the orthogonal array $L_{16}(4^4)$ is chosen. For each parameter combination, the EDA-IG is run 20 times independently and the average RPD value of 20 runs is obtained as the response variable (RV) value. The orthogonal array and RV values are listed in Table II.

TABLE I
FACTOR LEVEL OF PARAMETERS

Parameters	Factor level			
	1	2	3	4
$Psize$	20	30	40	50
η	5	10	15	20
α	0.1	0.2	0.3	0.4
γ	500	1000	1500	2000

According to the orthogonal table, the influence trend of each parameter is illustrated in Fig. 6. The average RV values of each parameter are figured out to analyze the significance rank. The results are listed in Table III.

From Fig. 6 and Table III, it can be seen that all the parameter values should be neither too small nor too large. Besides, the intensity of IG search has the most significant impact. A small value of γ could not provide sufficient local exploitation while a large value may waste the computation budget. The significance of η ranks the second. A proper value is helpful to update the probability model efficiently. The significance of α ranks the third. A small value of α could lead to slow convergence while a large value could lead to premature convergence. Although $Psize$ has the slightest influence, a small value cannot provide enough search breadth while a large value may cause insufficient evolution due to fewer generations. According to the analysis, the suggested values are as follows: $Psize = 40, \eta = 10, \alpha = 0.2, \gamma = 1000$. This setting is also used in the following experiments.

TABLE II
THE RV VALUE

Experiment No	Factor				RV
	$Psize$	η	α	γ	
1	1	1	1	1	243.25
2	1	2	2	2	240.80
3	1	3	3	3	242.70
4	1	4	4	4	243.20
5	2	1	1	1	241.15
6	2	2	2	2	242.30
7	2	3	3	3	243.90
8	2	4	4	4	242.20
9	3	1	1	1	242.55
10	3	2	2	2	242.30
11	3	3	3	3	242.75
12	3	4	4	4	244.20
13	4	1	1	1	242.50
14	4	2	2	2	243.85
15	4	3	3	3	242.90
16	4	4	4	4	243.10

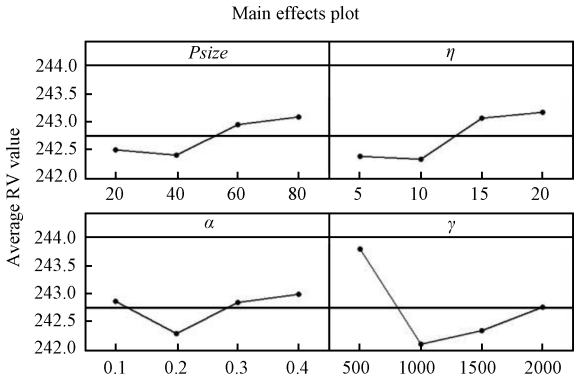

Fig. 6. The influence trend of each parameter.

TABLE III
RESPONSE TABLE

Factor	Level				Delta	Rank
	1	2	3	4		
$Psize$	242.49	242.39	242.95	243.09	0.70	4
η	242.36	242.31	243.06	243.18	0.86	2
α	242.85	242.26	242.83	242.98	0.71	3
γ	243.80	242.06	242.31	242.74	1.74	1

B. Effectiveness of Hybrid Strategy

To show the effectiveness of hybridizing EDA and IG search, the performances of EDA, IG1, IG2, EDA-IG1, and EDA-IG2 are compared by using the first set of benchmark instances. EDA has no IG search while the IG1 and IG2 have no EDA-based search, i.e., the learning rate is zero. EDA-IG1 and EDA-IG2 denote the EDA with IG1 and IG2, respectively. For fair comparison, the parameter values are the same and the stopping criterion is set with a maximum elapsed CPU time of $n \times (m/2) \times 10$ ms for all the algorithms. The algorithms

are run 5 times independently and the smallest RPD values are obtained. Table IV summarizes the results grouped by n (160 data per average).

TABLE IV
COMPARISON OF EDA, IG1, IG2, EDA-IG1 AND EDA-IG2

n	EDA	IG1	IG2	EDA-IG1	EDA-IG2
6	4.00	0.02	0.94	0.00	0.78
8	2.38	0.10	0.86	0.08	0.31
10	1.79	0.19	0.51	0.11	0.31
12	2.96	0.26	0.78	0.15	0.39
Ave	2.78	0.14	0.77	0.09	0.45

From Table IV, it can be seen that EDA-IG1 outperforms EDA and IG1 while EDA-IG2 outperforms EDA and IG2. Therefore, it can be concluded that EDA-IG is better than both EDA and IG, which shows that the hybrid strategy is effective. Besides, EDA-IG1 is the best one of the five algorithms for the small-scale instances.

C. Selection Rule of IG Methods

To further investigate the effect of different IG methods, performances of EDA-IG1 and EDA-IG2 are compared based on the second set of benchmark instances. The stopping criterion is set with a maximum elapsed CPU time of $n \times (m/2) \times 30$ ms. The two algorithms are run 5 times independently and the smallest RPD values are obtained. Table V summarizes the results grouped by each combination of n and m (40 data per average).

From Table V, it can be seen that EDA-IG1 outperforms EDA-IG2 when $n \times m$ is small, and EDA-IG2 outperforms EDA-IG1 when $n \times m$ is large. The reasons are as follows: In the reconstruction phase, IG1 searches every possible position for each removed job while IG2 assigns a job to the last position of a machine. Hence, IG1 explores more search space than IG2. Accordingly, the local exploitation ability of EDA-IG1 is stronger than that of EDA-IG2. Meanwhile, the computational complexity of IG1 is larger than that of IG2. When the scale of instance is larger, EDA-IG2 can evolve much more generations than EDA-IG1. Therefore, EDA-IG1 is better for small-scale instances and EDA-IG2 is better for large-scale ones.

To investigate the selection rule of IG methods, the average result for each combination of $n \times m$ is illustrated in Fig. 7. For example, when $n \times m = 1500$, the value is an average result of $\{n = 50, m = 30\}$, $\{n = 100, m = 15\}$, and $\{n = 150, m = 10\}$.

The linear least square method is utilized to determine the relationship between average RPD (y-axis) and the value of $n \times m$ (x-axis). The tropic equations for EDA-IG1 and EDA-IG2 are $y = 0.0035x - 8.5972$ and $y = -0.0006x + 5.039$, respectively. The abscissa of the intersection point is about 3325.9. Therefore, the selection rule of IG methods is as follows: If $n \times m < 3326$, IG1 is employed in EDA-IG; else, IG2 is employed. Such a selection rule will be used in the following numerical tests.

TABLE V
COMPARISON OF EDA, IG1, IG2, EDA-IG1 AND EDA-IG2 FOR LARGE-SCALED INSTANCES

n	m	$n \times m$	EDA-IG1	EDA-IG2
50	10	500	–2.39	2.15
50	15	750	–6.13	–0.03
50	20	1000	–8.45	2.33
50	25	1250	–9.64	5.98
50	30	1500	–8.69	16.09
100	10	1000	1.12	5.36
100	15	1500	–1.65	4.31
100	20	2000	–2.63	4.41
100	25	2500	–4.64	3.75
100	30	3000	–5.07	6.11
150	10	1500	0.97	5.02
150	15	2250	2.14	5.06
150	20	3000	3.20	4.22
150	25	3750	3.75	3.05
150	30	4500	5.39	3.98
200	10	2000	0.64	4.28
200	15	3000	2.63	3.47
200	20	4000	5.72	3.20
200	25	5000	9.64	1.39
200	30	6000	14.61	1.98
250	10	2500	0.67	3.32
250	15	3750	1.93	2.03
250	20	5000	8.73	1.79
250	25	6250	12.35	–0.61
250	30	7500	18.84	–0.12

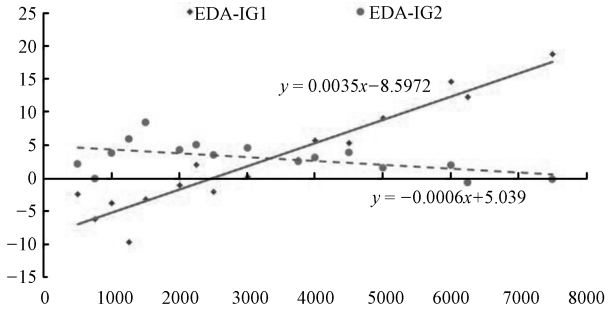

Fig. 7. Comparison of EDA-IG1 and EDA-IG2 for different values of $n \times m$.

D. Comparison with Existing Algorithms and EDA-IG Without the Properties

Next, the EDA-IG is compared with two GAs[10]. Moreover, to verify the effectiveness of proposed five properties by numerical experiments, the EDA-IG is compared with the EDA-IG without using the properties (denoted as EDA-IG_np). For the 640 small-scaled and 1000 large-scale benchmark instances, Vallada and Ruiz provided best-known solutions accordingly[10]. Same as [10], EDA-IG and EDA-IG_np are

TABLE VI
COMPARISON AMONG GAs, EDA-IG, AND EDA-IG_NP FOR SMALL-SCALED INSTANCES

n, m	t = 10				t = 30				t = 50			
	GA1	GA2	EDA-IG	EDA-IG_np	GA1	GA2	EDA-IG	EDA-IG_np	GA1	GA2	EDA-IG	EDA-IG_np
6, 2	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6, 3	0.26	0.07	0.00	0.00	0.06	0.08	0.00	0.00	0.15	0.08	0.00	0.00
6, 4	0.30	0.16	0.00	0.00	0.57	0.37	0.00	0.00	0.40	0.27	0.00	0.27
6, 5	0.11	0.10	0.00	0.04	0.21	0.12	0.00	0.00	0.23	0.21	0.00	0.00
8, 2	0.00	0.03	0.24	0.16	0.02	0.00	0.00	0.10	0.07	0.03	0.00	0.06
8, 3	0.41	0.32	0.00	0.03	0.34	0.31	0.00	0.00	0.20	0.24	0.00	0.00
8, 4	0.75	0.50	0.09	0.12	0.77	0.41	0.00	0.03	0.66	0.39	0.00	0.00
8, 5	0.58	0.23	0.00	0.00	0.65	0.11	0.00	0.00	0.49	0.20	0.00	0.00
10, 2	0.24	0.19	0.25	0.20	0.13	0.07	0.13	0.02	0.19	0.17	0.00	0.02
10, 3	0.30	0.15	0.11	0.19	0.40	0.18	0.06	0.10	0.26	0.20	0.00	0.00
10, 4	0.46	0.26	0.00	0.05	0.53	0.30	0.00	0.05	0.45	0.32	0.00	0.00
10, 5	1.38	1.15	0.08	0.23	1.51	1.03	0.05	0.20	1.16	1.15	0.00	0.15
12, 2	0.18	0.15	0.44	0.80	0.16	0.10	0.11	0.18	0.21	0.09	0.00	0.16
12, 3	0.54	0.15	0.08	0.12	0.20	0.12	0.00	0.00	0.22	0.08	0.00	0.00
12, 4	1.44	1.00	0.11	0.15	1.73	0.89	0.00	0.04	1.29	0.75	0.00	0.04
12, 5	2.32	1.54	–0.05	0.21	1.95	1.49	–0.13	–0.13	1.98	1.50	–0.16	–0.13
Ave	0.58	0.37	0.09	0.14	0.58	0.35	0.01	0.04	0.50	0.36	–0.01	0.04

TABLE VII
COMPARISON AMONG GAs, EDA-IG, AND EDA-IG_NP FOR LARGE-SCALED INSTANCES

n, m	t = 10				t = 30				t = 50			
	GA1	GA2	EDA-IG	EDA-IG_np	GA1	GA2	EDA-IG	EDA-IG_np	GA1	GA2	EDA-IG	EDA-IG_np
50, 10	13.56	7.79	–1.56	6.86	12.31	6.92	–2.39	3.65	11.66	6.49	–2.66	2.39
50, 15	13.87	12.25	–4.94	1.43	13.95	8.92	–6.13	–0.86	12.74	9.80	–6.50	–2.03
50, 20	12.92	11.08	–6.84	0.66	12.58	8.04	–8.45	–2.75	13.44	9.57	–8.91	–4.24
50, 25	13.18	10.48	–7.87	–1.06	12.39	8.49	–9.64	–4.27	11.87	8.10	–10.16	–5.73
50, 30	15.16	10.98	–6.74	0.24	14.42	10.19	–8.69	–2.93	14.42	9.40	–9.27	–3.80
100, 10	13.11	15.72	1.97	9.45	10.46	6.76	1.12	7.63	9.68	5.54	0.63	7.02
100, 15	15.41	22.15	0.15	8.66	13.95	8.36	–1.65	6.09	12.94	7.32	–1.86	5.01
100, 20	15.34	22.02	–0.60	7.82	13.65	9.79	–2.63	5.31	13.60	8.59	–3.17	3.58
100, 25	12.47	16.71	–3.39	4.21	11.30	7.86	–4.64	0.56	11.29	8.07	–4.99	–0.76
100, 30	11.11	15.69	–3.32	3.43	11.31	8.69	–5.07	1.72	10.98	7.90	–6.10	–1.26
150, 10	10.95	18.40	1.97	8.96	8.19	5.75	0.97	7.03	7.69	5.28	0.41	6.71
150, 15	14.51	24.89	4.12	10.72	11.93	8.09	2.14	9.23	11.78	6.80	1.62	8.52
150, 20	13.82	22.63	5.48	11.15	12.66	9.53	3.20	9.74	12.49	7.40	2.78	8.85
150, 25	11.74	17.16	4.01	14.08	10.98	7.89	3.05	11.90	10.12	7.05	2.70	10.35
150, 30	10.74	14.85	5.09	15.46	10.06	8.03	3.98	13.02	9.72	7.17	3.43	11.92
200, 10	9.75	8.40	1.17	7.28	7.56	6.01	0.64	6.24	6.17	4.24	0.33	5.81
200, 15	12.68	10.95	3.98	11.72	10.66	7.20	2.63	10.65	9.82	6.21	2.06	10.21
200, 20	12.59	11.07	3.80	10.70	10.77	8.36	3.20	10.00	10.37	6.71	2.71	9.48
200, 25	11.05	9.83	2.57	10.01	9.86	7.47	1.39	8.92	9.51	6.82	1.16	7.68
200, 30	10.17	9.49	2.92	11.01	9.49	7.09	1.98	9.09	8.65	7.09	1.70	8.44
250, 10	9.64	8.20	1.52	7.23	7.13	5.99	0.67	6.76	5.88	4.38	0.23	6.45
250, 15	11.64	9.85	2.77	8.33	8.97	6.70	2.03	7.27	8.05	5.12	1.74	6.75
250, 20	11.28	10.73	2.12	8.35	10.04	7.72	1.79	7.37	9.17	6.92	1.25	6.95
250, 25	10.37	9.55	0.28	6.93	9.05	7.49	–0.61	6.09	7.88	6.02	–0.98	5.56
250, 30	9.08	8.95	0.50	8.03	8.10	6.80	–0.12	6.71	7.18	5.91	–0.73	6.24
Ave	12.25	13.59	0.37	7.67	10.87	7.77	–0.85	5.77	10.28	6.93	–1.30	4.80

run 5 times independently for each instance. Besides, the best RPD values are obtained for three t values, which represent the performances of different CPU times. Tables VI and VII summarize the results grouped by each combination of n and m (40 data per average) as in [13], where the results of the GAs are directly from the literature.

From Tables VI and VII, it can be seen that EDA-IG outperforms EDA-IG_np for solving both the small- and large-scale instances. As the problem scale increases, the superiority of EDA-IG becomes more obvious. For the small-scale instances, the average result of EDA-IG is better than that of EDA-IG_np for each CPU time. For the large-scale instances, the average result of EDA-IG_np within a relatively large amount of CPU time (t = 50) is even worse than that of EDA-IG within a relatively short amount of CPU time (t = 10). Thus, the results demonstrate that the proposed properties can truly save the computational time to a great extent.

Compared with the GAs, EDA-IG performs better for solving both the small- and large-scale instances. For the small-scale instances, the average result of EDA-IG is 0.09 within a relatively short amount of CPU time (t = 10). After a relatively large amount of CPU time (t = 50), the average result of EDA-IG is -0.01 while the one of GA1 is 0.50 and the one of GA2 is 0.36. Within such amount of CPU time, EDA-IG updates the best-known solution of one small-scale instance. For the large-scale instances, the superiority of EDA is more obvious. The average result of EDA-IG is 0.37 within a relatively short amount of CPU time (t = 10). After a relatively large amount of CPU time (t = 50), the average result of EDA-IG is -1.30 while the one of GA1 is 10.28 and the one of GA2 is 6.93. Within such amount of CPU time, EDA-IG updates the best-known solutions of 530 large-scaled instances.

With above numerical tests, it can be concluded that the proposed EDA-IG and the derived properties are effective for solving the UPMSP-SDST.

VI. Conclusion

In this paper, an EDA-IG is proposed to solve the UPMSP-SDST with the objective of minimizing the makespan. The contributions of this work can be summarized as follows:

1) Some properties for judging the effectiveness of neighborhood search operators are derived. These properties can be used to avoid invalid search and to speed up the evaluation of new solutions in the EDA-IG. Besides, these properties can be employed to develop effective strategies for local search in designing other algorithms.

2) Based on the neighbor relations of the jobs, a probability model is built to describe the distribution of the search space. It helps the algorithm to generate new solutions by sampling a promising search region.

3) To enhance the exploitation, a special IG search is developed to improve the best solution of the population. Two types of deconstruction and reconstruction are designed, and a selection rule is presented.

4) Numerical tests and comparisons with 1640 benchmark instances show that the EDA-IG outperforms the existing GAs. Meanwhile, the best-known solutions of 531 instances are updated by the EDA-IG. Besides, the effectiveness of the derived properties is also verified by comparisons.

Future work could focus on developing the learning-based EDA-IG algorithm by adaptively using multiple search operators. It is also interesting to study the specific EDA-IG algorithm for the distributed scheduling problems by investigating the problem-specific information.

References

[1] Cheng T C E, Sin C C S. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 1990, **47**(3): 271−292

[2] Chen J, Pan Q K, Wang L, Li J Q. A hybrid dynamic harmony search algorithm for identical parallel machines scheduling. *Engineering Optimization*, 2012, **44**(2): 209−224

[3] Xu Y, Wang L, Wang S Y, Liu M. An effective shuffled frog-leaping algorithm for solving the hybrid flow-shop scheduling problem with identical parallel machines. *Engineering Optimization*, 2013, **45**(12): 1409−1430

[4] Lin S W, Ying K C. A multi-point simulated annealing heuristic for solving multiple objective unrelated parallel machine scheduling problems. *International Journal of Production Research*, 2015, **53**(4): 1065−1076

[5] Mellouli R, Sadfi C, Chu C B, Kacem I. Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. *European Journal of Operational Research*, 2009, **197**(3): 1150−1165

[6] Xu X Q, Lin J, Cui W T. Hedge against total flow time uncertainty of the uniform parallel machine scheduling problem with interval data. *International Journal of Production Research*, 2014, **52**(19): 5611−5625

[7] Fanjul-Peyro L, Ruiz R. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 2010, **207**(1): 55−69

[8] Garey M R, Johnson D S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco, USA.: W.H. Freeman & Co., 1979.

[9] Allahverdi A, Ng C T, Cheng T C E, Kovalyov M Y. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 2008, **187**(3): 985−1032

[10] Vallada E, Ruiz R. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 2011, **211**(3): 612−622

[11] Behnamian J, Zandieh M, Fatemi Ghomi S M T. Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm. *Expert Systems with Applications*, 2009, **36**(6): 9637−9644

[12] Chang P C, Chen S H. Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. *Applied Soft Computing*, 2011, **11**(1): 1263−1274

[13] Fleszar K, Charalambous C, Hindi K S. A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 2012, **23**(5): 1949−1958

[14] Lin S W, Ying K C. ABC-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times. *Computers & Operations Research*, 2014, **51**: 172−181

[15] Yilmaz E D, Ozmutlu H C, Ozmutlu S. Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *International Journal of Production Research*, 2014, **52**(19): 5841−5856

[16] Arnaout J P, Rabadi G, Musa R. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 2010, **21**(6): 693−701

[17] Avalos-Rosales O, Angel-Bello F, Alvarez A. Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 2015, **76**(9−12): 1705−1718

[18] Chen J F. Scheduling on unrelated parallel machines with sequence- and machine-dependent setup times and due-date constraints. *The International Journal of Advanced Manufacturing Technology*, 2009, **44**(11−12): 1204−1212

[19] Lin S W, Lu C C, Ying K C. Minimization of total tardiness on unrelated parallel machines with sequence- and machine-dependent setup times under due date constraints. *The International Journal of Advanced Manufacturing Technology*, 2011, **53**(1−4): 353−361

[20] Hsu C J, Kuo W H, Yang D L. Unrelated parallel machine scheduling with past-sequence-dependent setup time and learning effects. *Applied Mathematical Modelling*, 2011, **35**(3): 1492−1496

[21] Kuo W H, Hsu C J, Yang D L. Some unrelated parallel machine scheduling problems with past-sequence-dependent setup time and learning effects. *Computers & Industrial Engineering*, 2011, **61**(1): 179−183

[22] Joo C M, Kim B S. Hybrid genetic algorithms with dispatching rules for unrelated parallel machine scheduling with setup time and production availability. *Computers & Industrial Engineering*, 2015, **85**: 102−109

[23] Chen C L, Chen C L. Hybrid metaheuristics for unrelated parallel machine scheduling with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 2009, **43**(1−2): 161−169

[24] Larrañaga P, Lozano J A. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. US: Springer, 2002.

[25] Ceberio J, Irurozki E, Mendiburu A, Lozano J A. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 2012, **1**(1): 103−117

[26] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 2007, **177**(3): 2033−2049

[27] Guinet A. Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *International Journal of Production Research*, 1993, **31**(7): 1579−1594

[28] Montgomery D C. *Design and Analysis of Experiments (Seventh Edition)* . New York, U. S. A.: John Wiley & Sons, 2008.

Ling Wang received the B. Sc. and Ph. D. degrees from Department of Automation, Tsinghua University, China in 1995 and 1999, respectively. Now he is a full professor at the Department of Automation, Tsinghua University. His research interests include theories and algorithms for intelligent optimization and scheduling. Corresponding author of this paper.

Shengyao Wang received the B. Sc. and Ph. D. degrees from Department of Automation, Tsinghua University, China in 2010 and 2015, respectively. Now he is a research fellow at CETC. His research interests include shop scheduling and intelligent optimization.

Xiaolong Zheng received the B. Sc. degree from Tsinghua University, China in 2011. Now he is a Ph. D. candidate at the Department of Automation, Tsinghua University. His research interests include resource constrained scheduling, intelligent optimization.