Neural Comput & Applic
DOI 10.1007/s00521-016-2692-y

ORIGINAL ARTICLE

A hybrid differential evolution algorithm with estimation of distribution algorithm for reentrant hybrid flow shop scheduling problem

Bing-hai Zhou[1] · Li-man Hu[1] · Zhen-yi Zhong[1]

Received: 2 April 2015 / Accepted: 1 November 2016

Abstract This paper proposes a reentrant hybrid flow shop scheduling problem where inspection and repair operations are carried out as soon as a layer has completed fabrication. Firstly, a scheduling problem domain of reentrant hybrid flow shop is described, and then, a mathematical programming model is constructed with an objective of minimizing total weighted completion time. Then, a hybrid differential evolution (DE) algorithm with estimation of distribution algorithm using an ensemble model (eEDA), named DE–eEDA, is proposed to solve the problem. DE–eEDA incorporates the global statistical information collected from an ensemble probability model into DE. Finally, simulation experiments of different problem scales are carried out to analyze the proposed algorithm. Results indicate that the proposed algorithm can obtain satisfactory solutions within a short time.

Keywords Reentrant hybrid flow shop · Scheduling · Differential evolution algorithm · Estimation of distribution algorithm

1 Introduction

A new kind of manufacturing workshop called reentrant workshop has caused more and more concern recently. The characteristic of a reentrant shop is that a job visits certain stages more than once, which is found in many industries, especially in semiconductor manufacturing enterprises. In semiconductor manufacturing systems, a wafer needs to be processed in successive stages layer-by-layer for several times. This manufacturing process can be regarded as a reentrant hybrid flow shop (RHFS) problem. In a RHFS, all jobs have the same routing over the machines of the shop and the same sequence is traversed several times to complete the jobs and there is more than one machine in at least one station [1].

Interest about the RHFS scheduling problem has been increasing in the academic society during recent years. Some researchers considered the fabrication stage only. Choi et al. [2] designed heuristic algorithms for the hybrid flow shop problem with reentrant flows with the objective of minimizing total tardiness of a given set of orders where an order was divided into multiple lots. Jiang and Tang [3] used Lagrangian relaxation algorithms for the RHFS scheduling problem with the objective of minimizing the sum of weighted completion time of jobs. Cho et al. [4] proposed a local-search-based Pareto genetic algorithm with Minkowski distance-based crossover operator to approximate the Pareto optimal solutions for the minimization of makespan and total tardiness in a RHFS. Choi et al. [5] suggested a real-time scheduling mechanism with a decision tree when selecting appropriate dispatching rules to solve reentrant hybrid flow shops. The decision tree was adopted to eliminate the computational burden required to carry out simulation runs to select dispatching rules. Others considered the inspection and repair stage. Dugardin et al. [6] proposed Lorenz Non-dominated Sorting Genetic Algorithm (L-NSGA) and the Strength Pareto Evolutionary Algorithm version 2 (SPEA2) to minimize the maximum completion time and the sum of the tardiness for a two-stage RHFS. Dugardin et al. [7] focused on the multi-objective resolution of a RHFS scheduling problem and proposed a genetic algorithm called L-NSGA. In their case,

✉ Bing-hai Zhou
bhzhou@tongji.edu.cn

1 School of Mechanical Engineering, Tongji University, Shanghai 201804, People's Republic of China

two objectives are the maximization of the utilization rate of the bottleneck and the minimization of the maximum completion time. Yalaoui et al. [8] solved a hybrid flow shop scheduling problem where each order was composed of several batches and some ones were reentrant. A particle swarm optimization method under fuzzy logic controller method was developed to solve the problem with the objective of minimizing the total tardiness. One researcher, Hekmatfar et al. [1], proposed a new hybrid genetic algorithm to solve a two-stage RHFS in which the separate second stage inspects the products after all the layers of all the jobs had been completed in the first stage, i.e., the fabrication stage. However, this separate second stage can only be used as a final test stage which can not inspect defects generated at each layer.

As we can see, the previous literatures have considered the fabrication stage and the inspection and repair stage, respectively, or separately. However, in fact, as the wafers are manufactured layer-by-layer, defects generated at the fabrication stage and covered by a new layer cannot be corrected or even measured [9]. As a result, inspection and repair should be carried out as soon as a layer has completed the fabrication stage. In other words, fabrication and inspection and repair should be considered together. Therefore, in this paper, different from all the previous literature, wafers must successively undergo inspection after one layer has been fabricated. This step is carried out at the inspection stage. After being inspected, the wafer is then directed toward a repair stage where failed components are replaced. The wafer repeats the inspection and the repair stage for a given number of cycles until all the defects have been corrected or repaired and only then can the wafer be fabricated at the next layer. In addition, in this paper, different products have different layers and different number of cycles.

On the other hand, the meta-heuristics used in the above literature are mainly developed from genetic algorithm (GA). As some new meta-heuristics have been developed during the last decades,this suggests the exploration of the potentials for the RHFS using the new meta-heuristics. This paper proposes DE–eEDA, a hybrid differential evolution (DE) algorithm with estimation of distribution algorithm (EDA) using an ensemble model, i.e., simultaneously using the univariate and the bivariate probability model to solve the RHFS scheduling problem.

DE was first introduced in 1997 by Storn and Price [10] for complex continuous nonlinear problems. Because of its simplicity, easy implementation and fast convergence, DE has received much attention and a variety of successful applications have been implemented in optimization problems over continuous space [11–17].

DE has also been applied to optimization problems over discrete space. Chiou et al. [18] presented an ant direction hybrid differential evolution with integer programming for solving large capacitor placement problems in distribution systems. Onwubolu et al. [19] described a novel optimization method based on DE to solve nonlinear programming problems containing integer and discrete variables. Qian et al. [20] proposed a memetic algorithm based on DE for multi-objective job shop scheduling problems. Damak et al. [21] proposed a DE to solve the resource-constrained project scheduling problem with multiple execution modes for each activity and minimize the makespan. Pan et al. [22] proposed an effective hybrid discrete differential evolution to minimize the makespan for a flow shop scheduling problem with intermediate buffers located between two consecutive machines.

DE algorithm employs mutation and crossover operators to generate new candidates. Generally, DE generates new parameter vectors by adding the weighted difference between two population vectors to a third vector during the mutation operation. However, by this way, the new parameter vectors can only evolve from the best vector or a few randomly chosen vectors, and therefore, each vector can not learn from the global information. To improve the performance of DE, Omran et al. [23, 24] proposed a self-adaptive DE (SaDE) algorithm, in which the trial vector generation strategies and their associated control parameter values were gradually self-adapted by learning from their previous experiences in generating promising solutions. Ling et al. [25, 26] introduced two hybrid differential evolutions, in which a local search algorithm and Taguchi's method were embedded in the algorithm to improve the searching abilities of DE, respectively.

In this paper, different from all the previous studies, we will propose a DE–eEDA which incorporates the global statistical information collected from an ensemble probability model into DE. As an ensemble probability model characterizes the accurate distribution of promising solutions in the global search space at each generation, each vector can have comprehensive search ability, thus leading to better performance of DE.

The ensemble probability model adopted in this paper is quite novel. In general, most EDAs obtained statistical information from only one model [27–31]. However, the ensemble probability model which simultaneously uses the univariate and the bivariate probability model can generate more accurate parental distributions for EDA. The more accurate the probability model is, the more effective the algorithm will be [32]. As far as we know, only [33–35] used the ensemble model and they have used eEDA to solve the single-machine scheduling problem and the flow shop scheduling problem. This paper will use the ensemble model to solve the RHFS scheduling problem and hybrid it with the DE to improve the performance of the proposed DE.

Considering the difficulty of inspecting defects after the wafers are covered by the next layer, this paper proposes the problem of considering immediate inspection and repair after one layer being fabricated and with the objective of minimizing the total weighted completion time. In addition, a novel algorithm which hybrids DE and eEDA is proposed to solve the RHFS problem. The performance of combining the two algorithms and using the ensemble model will be validated.

The rest of this paper is organized as follows: Sect. 2 is devoted to describe the problem formulations of the RHFS problem. Section 3 briefly introduces DE and EDA and then proposes the DE–eEDA. In Sect. 4, the DE–eEDA is applied to the RHFS. The computational results of the proposed algorithms compared with the lower bound (LB) of the problem which we propose by using the Lagrangian relaxation algorithm are shown in Sect. 5. Section 6 draws some conclusions. "Appendix" gives out the details for constructing the lower bound.

2 Problem formulation

The RHFS considered in this paper is to schedule N jobs, and different jobs have different layers L_i, i.e., number of times of reentrance. There are two main stages. The first stage is a fabrication stage in which there are J_f workstations and the workstation j consists of M_j parallel identical machines. In the second stage, there is a small reentrant system that consists of an inspection workstation and a repair workstation and each workstation j consists of M_j parallel identical machines. Every layer l of job i has to reenter the second stage for different times L'_i. In the proposed RHFS, the job i consists of $(J_f + 2L'_i)L_i$ processes. The RHFS system is depicted in Fig. 1.

2.1 Model assumptions

1. Machines are available at all times, with no breakdowns or maintenance delays.
2. Job processing cannot be interrupted, i.e., no preemption is allowed and jobs have no priority values.

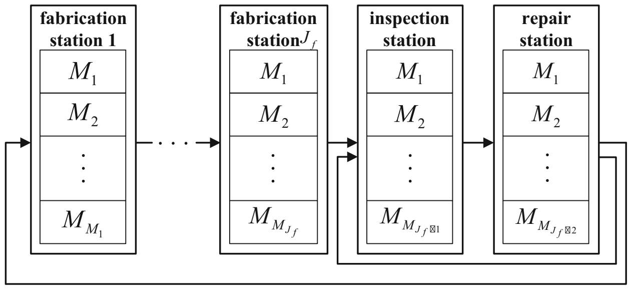

Fig. 1 The RHFS system

3. No limited buffer exists between stages.
4. There is no travel time between workstations.
5. Machines in parallel are identical in capability and processing rate.
6. Each machine can process at most one job at a time.

2.2 The model

Parameters:

N	Number of jobs
L_i	Number of layers of job i
L'_i	Number of cycles of each layer of job i performed by the inspection and the repair station
J_f	Number of workstations at the fabrication stage
J_i	Number of workstations that each layer l of job i has to go through $J_i = J_f + 2 * L'_i$
M	Set of total machines
K	Total time horizon
w_i	Weight of job i
$p_{i,l,j}$	Processing time of job i at layer l at station j
M_j	Set of parallel machines at station j

$$M_1 \cup M_2 \cup \ldots \cup M_{J_f+2} = M$$
$$M_j = M_{j-2} \quad \forall J_f + 2 < j \le J_i$$
$$M_{j1} \cap M_{j2} = \phi, \quad \forall 1 \le j1, j2 \le J_f + 2$$

Decision variables:

$C_{i,l,j}$	Completion time of job i at layer l at station j
$m_{i,l,j}$	Machine assigned to job i at layer l at station j

The deterministic reentrant hybrid flow shop problem **P** can be formulated as follows:

$$\min \sum_{i=1}^{N} w_i C_{i,L_i,J_i} \tag{1}$$

$$\text{s.t.} \quad C_{i,l,j} \ge p_{i,l,j}, \quad i = 1,\ldots,N, \quad l = 1,\ldots,L_i, \quad j = 1,\ldots,J_i, \tag{2}$$

$$C_{i,l,j-1} \le C_{i,l,j} - p_{i,l,j}, \quad i = 1,\ldots,N, \quad l = 1,\ldots,L_i, \quad j = 2,\ldots,J_i, \tag{3}$$

$$C_{i,l-1,J_i} + p_{i,l,1} \le C_{i,l,1}, \quad i = 1,\ldots,N, \quad l = 2,\ldots,L_i, \tag{4}$$

$$\sum_{(i,l,j)\in O_u} \{\phi(k - C_{i,l,j} + p_{i,l,j} - 1) - \phi(k - C_{i,l,j} - 1)\} \le 1 \quad k = 1,\ldots,K, \quad u = 1,\ldots,|M| \tag{5}$$

The objective function of (1) is to minimize the total weighted completion time. Constraint (2) ensures the starting time constraint from the processing time requirement. Constraint (3) denotes the precedence constraints that an operation of a certain layer of a certain job cannot be started until its preceding operation is finished. Constraint (4) denotes the precedence constraints that the first operation of a certain layer of a certain job cannot be started until the last operation of the previous layer of this job is finished. Constraint (5) states machine capacity constraints which say that each machine can process at most one operation at a time. In the constraint (5), $\phi(k)$ is an indicator function where $\phi(k) = \begin{cases} 1 & \text{if } k \geq 0 \\ 0 & \text{otherwise} \end{cases}$ and $O_u = \{(i, l, j) | m_{i,l,j} = u\}$, the set of operations to be performed on machine u. Thus, if the job i at layer l at station j is being processed on machine u at time point k, $\phi(k - C_{i,l,j} + p_{i,l,j} - 1) - \phi(k - C_{i,l,j} - 1) = 1$; otherwise, $\phi(k - C_{i,l,j} + p_{i,l,j} - 1) - \phi(k - C_{i,l,j} - 1) = 0$.

3 DE–eEDA

In the following section, the DE and the EDA are briefly reviewed, and then, the DE–eEDA is introduced.

3.1 De

The DE algorithm proposed by Storn and Price [10] is a novel evolutionary optimization method, which utilizes NP parameter vectors as a population for each generation G. The initial vector populations are generated randomly, and new parameter vectors are generated by mutation and crossover operators.

For each target vector $x_{i,G}$, $i = 1, 2, \ldots, NP$, a mutant vector is generated according to:

$$v_{i,G+1} = x_{r1,G} + F \times (x_{r2,G} - x_{r3,G}) \tag{6}$$

with random indexes $r1 \neq r2 \neq r3 \in 1, 2, \ldots, NP$ and $F > 0$ is a mutation scale factor $\in (0, 1]$ affecting the differential variation $(x_{r2,G} - x_{r3,G})$.

Following the mutation operator, a crossover operator is introduced to form a trial vector:

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Ni,G+1}) \tag{7}$$

where

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} \ \text{if}(\text{rand}_1(j) \leq \text{CR}) & \text{or} \quad j = \text{rand}_2(i) \\ x_{ji,G} \ \text{if}(\text{rand}_1(j) > \text{CR}) & \text{and} \quad j \neq \text{rand}_2(i) \end{cases} \\ j = 1, 2, \ldots, N \tag{8}$$

In (8), $\text{rand}_1(j)$ is the jth evaluation of a uniform random number generator with outcome $\in [0, 1]$. CR is a crossover constant $\in [0, 1]$. $\text{rand}_2(i)$ is a randomly chosen index $\in 1, 2, \ldots, N$ to ensure that at least one parameter of $u_{i,G+1}$ differs from $x_{i,G}$.

To decide whether or not the trial vector $u_{i,G+1}$ should survive at generation $G + 1$, it is compared with the target vector based on the greedy criterion. If vector $u_{i,G+1}$ generates a smaller objective function value than $x_{i,G}$, then $u_{i,G+1}$ is assigned to $x_{i,G+1}$; otherwise, $x_{i,G}$ is assigned to $x_{i,G+1}$.

3.2 EDA

As a relatively new paradigm in the field of evolutionary computation, EDA employs explicit probability distributions in optimization. Different from GA that produces offspring through crossover and mutation operators, EDA does it by sampling according to a probability model. So, the probability model has a great effect on the performance of EDA.

The model generally used in EDA is the univariate probability model. The detailed information about the model is as follows.

The element $p_{ij}(G)$ of the univariate probability model P represents the probability that job j appears before or in position i of the solution sequence at generation G.

The population with popsize individuals determines the superior subpopulation that consists of the best SP solutions. Then, the univariate probability model P is formulated according to the following equation:

$$p_{ij}(G) = \frac{1}{i \times SP} \cdot \eta_{ij}(G) = \frac{1}{i \times SP} \cdot \sum_{s=1}^{SP} I_{ij}^s(G), \quad \forall i, j \tag{9}$$

where $I_{ij}^s(G)$ is the following indicator function of the sth individual in the superior subpopulation.

$$I_{ij}^s(G) = \begin{cases} 1 & \text{if job } j \text{ appears before or in position } i \\ 0 & \text{else} \end{cases} \tag{10}$$

$\eta_{ij}(G) = \sum_{s=1}^{SP} I_{ij}^s(G)$ denotes the number of times that job j appears before or in position i, which provides the information of the importance of a job when deciding the scheduling order.

The probability model P is updated according to the following equation:

$$p_{ij}(G+1) = (1 - \alpha)p_{ij}(G) + \frac{\alpha}{i \times SP} \cdot \sum_{s=1}^{SP} I_{ij}^s(G), \quad \forall i, j \tag{11}$$

where $\alpha \in (0, 1)$ is the learning rate of P.

The updating process can be regarded as a kind of incremental learning, where the second term on the right-hand side of the equation represents learning information from the superior subpopulation.

The framework of a canonical EDA can be described as:

Step 1: Initialize the population and the probability model.
Step 2: Sample the probability model to generate new population and select superior subpopulation.
Step 3: Update the probability model.
Step 4: If the termination criterion is not met, go to Step 2; otherwise, stop.

3.3 DE–eEDA

In DE, most versions of the mutation equation consist of two parts. Generally, the first part is an existing vector and the second part is the difference vector of two randomly chosen population vectors. To evolve from the collective information in global scope, a novel mutation operator which combines the idea of EDA is proposed.

The mutant operator of the DE–eEDA is described as follows:

$$\begin{aligned} v_{i,G+1} = x_{best,G} + F_1 \times (x_{r1,G} - x_{r2,G}) \oplus F_2 \\ \times (x_{EDA,G} - x_{best,G}) \end{aligned} \tag{12}$$

where $x_{\text{best},G}$ is the best individual in the current target population, $x_{r1,G}$ and $x_{r2,G}$ are two randomly chosen target individuals with $r1 \neq r2 \in 1, 2, \ldots, NP$ and F_1, F_2 are two mutant scale factors $\in (0, 1]$.

In the above formula, $\oplus F_2 \times (x_{\text{EDA},G} - x_{\text{best},G})$ represents how EDA is incorporated. That is,

$$\begin{aligned} v_{i,G+1} &= x_{\text{best},G} + F_1 \times (x_{r1,G} - x_{r2,G}) \oplus F_2 \times (x_{\text{EDA},G} - x_{\text{best},G}) \\ &\Leftrightarrow \begin{cases} x_{\text{best},G} + F_1 \times (x_{r1,G} - x_{r2,G}) + F_2 \times (x_{\text{EDA},G} - x_{\text{best},G}) & \text{if rand}() > L \\ x_{\text{best},G} + F_1 \times (x_{r1,G} - x_{r2,G}) & \text{otherwise} \end{cases} \end{aligned} \tag{13}$$

where rand() is a uniform random number $\in [0, 1]$, L is a constant $\in [0, 1]$ which has to be determined by the user and $x_{\text{EDA},G}$ is a vector that employs the information sampled from an EDA probability model.

In the third term of Eq. (12), the idea of EDA is incorporated, and therefore, the mutant vector can learn the difference from the global statistical information derived from the probability model of EDA, not just a few individual vectors. The following gives out the details about the probability model adopted in this EDA operator.

The probability model introduced here is an ensemble probability model that uses the univariate model described in Sect. 3.2 and the bivariate probability model at the same time. To describe the bivariate probability model, $I'^{s}_{jj'}(i)(G)$, a new indicator function of the sth individual in the superior subpopulation is defined.

$$I'^{s}_{jj'}(i)(G) = \begin{cases} 1 & \text{if job } j \text{ appears immediately after job } j' \\ & \text{when job } j' \text{ is in postion } i-1 \\ 0 & \text{else} \end{cases} \tag{14}$$

$\eta'_{ij}(G) = \sum_{s=1}^{SP} J'^{s}_{jj'}(i)(G)$ refers to the number of times that job j appears immediately after job j' when job j' is in position $i-1$. $\eta'_{ij}(G)$ provides the information of the similar blocks of jobs in the selected sequences, which gives more accurate distribution information, thus leading to the performance improvement of the algorithm.

Then, the ensemble probability model is determined by:

$$P_{ij}(G) = \frac{\eta^{s}_{ij}(G) \times \eta'^{s}_{jj'}(i)(G)}{\sum_{l \in \Omega(i)} \eta^{s}_{il}(G) \times \eta'^{s}_{lj'}(i)(G)} \tag{15}$$

where $\Omega(i)$ is the set of jobs not scheduled until position i.

Because there is no prior job at the first position, the univariate model should be used at the first position. As a result, the ensemble probability model adopted in this paper is formulated as follows:

$$P_{ij}(G) = \begin{cases} p_{ij}(G), & i = 1 \\ \dfrac{\eta^{s}_{ij}(G) \times \eta'^{s}_{jj'}(i)(G)}{\sum_{l \in \Omega(i)} \eta^{s}_{il}(G) \times \eta'^{s}_{lj'}(i)(G)}, & i = 2, 3, \ldots, N \end{cases} \tag{16}$$

At each generation, the ensemble model is updated by incremental learning as described in Sect. 3.2 with α_1 and α_2 defined as the learning rates for the univariate model and the bivariate model, respectively.

4 DE–eEDA for RHFS

In this section, the DE–eEDA is applied to the RHFS. Firstly, the encoding and decoding schemes are given. Then, the initialization method and forward and backward transformation techniques are also described. Finally, the pseudo-code of the proposed algorithm is presented.

4.1 Encoding schemes

Every individual of the population denotes a solution, which is represented by a target vector $x_{i,G} = \{x_i(1), x_i(2), \ldots, x_i(N)\}$, $i = 1, 2, 3, \ldots, NP$ to determine the schedule order of all the jobs. For example, $x_{i,G} = \{1, 2, 5, 4, 3, 0\}$ implies that job 1 is scheduled first, and next are job 2, job 5, job 4 and job 3 in sequence. Job 0 is the last job to be scheduled.

4.2 Decoding schemes

Firstly, order the jobs according to $x_{i,G} = \{x_i(1), x_i(2), \ldots, x_i(N)\}$ at the first station of the first layer. Assign the jobs to the first machine that becomes available successively. Then, for the following stations, update the ready times in each station to be the completion times of the previous station. Arrange the jobs in increasing order of ready times and assign the jobs to the first available machine successively.

4.3 Initialization

In order to guarantee the diversity of the initial population, we use an initial random population of *NP* individuals, uniformly distributed.

4.4 Forward transformation and backward transformation technique

As solving the RHFS scheduling problem requires discrete variables and DE operators requires continuous variables, two strategies known as forward and backward transformation techniques, respectively, proposed in [19] are adopted in this paper.

4.4.1 Forward transformation technique

The forward transformation method is used for transforming integer variables into continuous variables. Let a set of integer number be presented as $z_i' \in z'$. Let the real number equivalence of this integer number be z_i. The length of the real number depends on the required precision. In this paper, we choose two places after the decimal point.

The formulation for transforming an integer number into a real number z_i for the given range is given as follows:

$$z_i = -1 + \frac{z_i' \times N \times 100}{10^3 - 1} \quad (17)$$

4.4.2 Backward transformation technique

Integer variables are used to evaluate the objective function. The backward transformation method is used for transforming continuous variables into integer variables. The scheme is given as follows:

$$z_i' = \frac{(1 + z_i) \times (10^3 - 1)}{N \times 100} \quad (18)$$

To ensure that each number is discrete and unique, some modifications are required as follows:

$$\alpha = \mathrm{int}(z_i' + 0.5) \quad (19)$$

$$\beta = \alpha - z_i' \quad (20)$$

$$z_i^* = \begin{cases} (\alpha - 1), & \text{if } \beta > 0.5 \\ \alpha, & \text{if } \beta < 0.5 \end{cases} \quad (21)$$

Equation (21) gives z_i^*, which is the transformed value used to compute the objective function.

The trial vector derived from the backward transformation is continuously checked until feasible solution is found.

4.5 Pseudo-code of the proposed DE–eEDA

The pseudo-code of the DE–eEDA algorithm proposed in this paper is shown in Fig. 2.

5 Experimentations

This section contains the method of generating datasets for the proposed RHFS and setting parameters for the tested algorithms and the experimental results of the tested algorithms. All the algorithms are run on a 1.86 GHz portable computer with 896 MB of RAM running Windows XP professional. The codes are written in C++ language.

5.1 Datasets

Since there are no standard test data in the open literature, the test problems are randomly generated on the basis of the following factors:

1. Number of jobs (N),
2. Number of layers of job i (L_i),
3. Number of workstations at the fabrication stage (J_f),
4. Set of parallel machines at station j (M_j).

For our set of test instances, the level settings for each factor are: five levels for N, two for L_i, three for J_f and three for M_j. For each parameter combination, 10 instances will be generated randomly according to the parameter settings in Table 1. This results in a total of 900 test problems.

5.2 Parameters setting

The DE contains several key parameters: *NP*, F and *CR*. F is usually between 0.5 and 1.0. CR usually should be 0.3, 0.7, 0.9 or 1.0. To investigate the influence of these parameters on the performance of the DE, we implement experiment by using a medium-sized problem with $N = 30, L_i = [1, 2]$, $J_i = [5, 7]$ and $M_j = 8$. The experiment

Fig. 2 Pseudo-code of the DE–eEDA

```
Initialize $F_1, F_2, CR, L, NP, SP, G, \alpha_1, \alpha_2$
Set $generation = 0$
Generate Initial Population
Calculate Objective Function
Generate Initial Probability Model
For ($generation$=1:$G$)
    For ($i$=1:$NP$)
        $r_1 = rand[1, NP]$   $r_2 = rand[1, NP]$   $r_1 \neq r_2$
        For ($j$=1:$N$)
            $z_i' \rightarrow z_i$ Forward transformation
            $mutant\ vector = target\ vector_{best} + F_1 \times (target\ vector_{r_1} - target\ vector_{r_2})$
        Increment $j$ by 1
        Endfor
        $r_3 = rand[0,1]$
        If $r_3 > L$ Then
            Sample from the ensemble probability model
            $mutant\ vector = mutant\ vector + F_2 \times (target\ vector_{EDA} - target\ vector_{best})$
        Endif
    Increment $i$ by 1
    Endfor
    For ($i$=1:$NP$)
        $index = rand[1, N]$
        For ($j$=1:$N$)
            $r_4 = rand[0,1]$
            If $r_4 \leq CR$ or $j == index$ Then
                $trial\ vector = mutant\ vector$
            Elseif $r_4 > CR$ and $j != index$ Then
                $trial\ vector = target\ vector$
            Endifelseif
        Increment $j$ by 1
        Endfor
    Increment $i$ by 1
    Endfor
    For ($i$=1:$NP$)
        For ($j$=1:$N$)
            $z_i \rightarrow z_i'$ Backward transformation
        Increment $j$ by 1
        Endfor
        Calculate Objective Function
    Increment $i$ by 1
    Endfor
    For ($i$=1:$NP$)
        If $objective_{trial\ vector} < objective_{target\ vector}$ Then
            $target\ vector = trial\ vector$
        Endif
    Increment $i$ by 1
    Endfor
    Update Probability Model
Increment $generation$ by 1
Endfor
```

Table 1 Parameters for the test data

N	10, 20, 30, 40, 50
L_i	$U[1,2]$,$U[2,3]$
J_f	2, 3, 4
M_j	$[N/5]+1,\ [N/5]+2,\ [N/5]+3$
L'_i	$U[1,2]$
$J_i\left(J_f+2\times L'_i\right)$	$U[4,6],\ U[5,7],\ U[6,8]$
Processing time of job i at station j, $p_{i,j}$	$U[10,20]$
Weight of job i, w_i	$U[1,10]$

trials are carried out consisting of three levels of NP(50, 80 and 100), three levels of F(0.5, 0.7 and 0.9) and three levels of CR(0.7, 0.9 and 1.0). A good choice of parameter combination for the DE is suggested as $NP = 80$, $F = 0.5$ and $CR = 0.9$. The number of iterations is set as 200. The parameters for the DE–EDA and the DE–eEDA are set in the same way. The parameters for the DE–EDA and the DE–eEDA are both set as $NP = 80, F_1 = 0.5, F_2 = 0.6, L = 0.4, CR = 0.9$.

5.3 Comparative study

The performance measure employed in our numerical study is the percentage deviation of the total weighted completion time (TWCT) of the algorithm from the lower bound proposed in "Appendix". It can be expressed as follows:

$$\delta = \frac{\text{TWCT} - \text{LB}}{\text{LB}} \times 100\%$$

The average δ value and CPU time of 10 instances under 90 problems obtained by the EDA, the eEDA, the DE–EDA and the DE–eEDA after 200 generations are listed in Table 2, and those obtained by the DE, the DE–eEDA, the GA and the particle swarm optimization (PSO) are listed in Table 3.

5.3.1 Comparing the EDA with the eEDA

The EDA and the eEDA are both of canonical procedure with the encoding and decoding schemes the same as the ones proposed in this paper. The parameters are also set by using a medium-sized problem. The parameters for the algorithms are set as follows: (1) For the EDA, popsize = 40, superior subpopulation SP = 10, learning rate $\alpha = 0.4$. (2) For the eEDA, popsize = 40, superior subpopulation SP = 15, learning rate $\alpha_1 = 0.3$ and learning rate $\alpha_2 = 0.4$.

As shown in Table 2, the EDA and the eEDA are nearly the same in terms of the CPU time. However, the eEDA performs better than the EDA in most of the problem sizes. Similarly, the DE–eEDA can further improve the performance of the DE–EDA by using the ensemble model without sacrificing much CPU time. The results indicate that the ensemble model that combines the univariate probability model and the bivariate probability model works better than the univariate probability model. It follows from what has been said that the order of jobs and the similar block of jobs can provide important information for evolution, which has been verified not only in single-machine and flow shop scheduling problem [35] but in RHFS scheduling problem as well.

5.3.2 Comparing the DE–eEDA with other algorithms

The GA and the PSO are of canonical procedure with the encoding and decoding schemes the same as the ones proposed in this paper. For the GA, the main procedures are selection, crossover and mutation. In this paper, we use roulette wheel selection, partial mapped crossover and swap mutation. The parameters are also set by using a medium-sized problem with $N = 30$, $L_i = [1,2]$, $J_i = [5,7]$ and $M_j = 8$. The experiment trials are carried out consisting of three levels of popsize (50, 80 and 110), three levels of crossover probability p_c (0.5, 0.7 and 0.9) and three levels of mutation probability p_m (0.15, 0.2 and 0.25). A good choice of parameter combination for the GA is suggested as popsize = 80, $p_c = 0.5$ and $p_m = 0.15$. As for the PSO, each particle flies in the multi-dimensional search space of the optimization problem looking for optimal solutions, while its position and velocity are updated by learning from its own experience and the performance of its peers. The parameters are also set by using a medium-sized problem the same as the GA. The experiment trials are carried out consisting of three levels of popsize (40, 70 and 100). A good choice of parameter is popsize = 70. Other parameters are set as follows: learning factors $c_1 = c_2 = 0.2$, and inertia weight w is initially set as 0.9 and then linearly decreased to 0.4 according to the number of iterations, where the parameters are set according to general conditions. The number of iterations for the GA and the PSO is set as 200.

Table 3 depicts that the DE performs the worst among the four algorithms with the least CPU time. The DE–eEDA greatly improves the performance of the DE, while its CPU time is still satisfactory. It validates that the incorporation of the idea of the EDA with the ensemble model into the DE is very effective for improving the performance of the DE.

Besides, compared with the GA and the PSO, the DE–eEDA achieves the best solutions among the three algorithms. This is because DE–eEDA embeds the probabilistic model in the mutation operators to exploit the solution

Table 2 Comparing δ average in different algorithms

$N \times L_i \times J_f \times M_j$	EDA		eEDA		DE–EDA		DE–eEDA	
	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)
10 × [1, 2] × [4, 6] × 3	7.38	1.18	7.02	1.06	7.48	1.22	6.78	1.39
10 × [1, 2] × [4, 6] × 4	3.65	1.18	3.32	1.27	3.51	1.51	3.32	1.52
10 × [1, 2] × [4, 6] × 5	2.73	1.06	2.64	1.30	2.89	1.45	2.65	1.39
10 × [1, 2] × [5, 7] × 3	7.23	0.85	7.23	1.13	7.45	1.41	7.62	1.45
10 × [1, 2] × [5, 7] × 4	5.92	1.01	5.67	1.14	6.13	1.30	5.81	1.32
10 × [1, 2] × [5, 7] × 5	2.58	1.07	2.54	1.11	2.58	1.46	2.58	1.43
10 × [1, 2] × [6, 8] × 3	6.93	1.22	6.71	1.27	7.24	1.48	6.70	1.46
10 × [1, 2] × [6, 8] × 4	4.09	1.12	3.94	1.24	4.37	1.60	4.07	1.57
10 × [1, 2] × [6, 8] × 5	2.46	1.24	2.42	1.23	2.35	1.38	2.33	1.36
10 × [2, 3] × [4, 6] × 3	13.81	1.08	13.86	1.14	14.41	1.70	13.64	1.66
10 × [2, 3] × [4, 6] × 4	7.93	1.17	7.98	1.35	8.08	1.78	7.95	1.89
10 × [2, 3] × [4, 6] × 5	4.80	1.31	4.76	1.34	4.79	1.87	4.77	1.85
10 × [2, 3] × [5, 7] × 3	12.98	1.28	12.57	1.43	12.92	1.92	12.82	1.68
10 × [2, 3] × [5, 7] × 4	3.79	1.32	3.87	1.38	4.21	1.85	3.96	1.82
10 × [2, 3] × [5, 7] × 5	2.80	1.40	2.80	1.45	2.70	1.84	2.80	1.86
10 × [2, 3] × [6, 8] × 3	6.09	1.36	5.91	1.39	5.71	1.86	6.10	1.86
10 × [2, 3] × [6, 8] × 4	3.48	1.34	3.26	1.57	3.44	2.20	3.26	2.02
10 × [2, 3] × [6, 8] × 5	2.08	1.47	1.93	1.63	2.04	2.16	2.22	2.17
20 × [1, 2] × [4, 6] × 5	13.44	2.55	13.80	2.67	13.47	3.33	12.63	3.38
20 × [1, 2] × [4, 6] × 6	9.59	2.75	8.75	2.99	9.07	3.71	8.38	3.74
20 × [1, 2] × [4, 6] × 7	6.75	2.69	6.60	3.04	6.02	3.77	5.69	3.80
20 × [1, 2] × [5, 7] × 5	10.87	2.89	11.08	3.10	11.27	3.89	9.91	3.89
20 × [1, 2] × [5, 7] × 6	9.64	2.95	9.26	3.14	9.83	3.93	8.91	3.98
20 × [1, 2] × [5, 7] × 7	6.00	2.91	6.08	3.16	6.20	3.98	5.59	4.01
20 × [1, 2] × [6, 8] × 5	7.73	2.87	7.37	3.22	8.20	4.10	7.32	4.13
20 × [1, 2] × [6, 8] × 6	5.95	2.99	5.95	3.07	5.86	3.79	5.39	3.88
20 × [1, 2] × [6, 8] × 7	6.11	3.02	5.80	3.33	6.11	4.20	5.59	4.23
20 × [2, 3] × [4, 6] × 5	22.10	2.89	22.05	3.18	22.16	4.34	22.03	4.36
20 × [2, 3] × [4, 6] × 6	14.44	3.22	13.91	3.41	13.97	4.37	13.34	4.40
20 × [2, 3] × [4, 6] × 7	11.13	2.71	10.34	2.90	10.51	3.79	10.16	3.93
20 × [2, 3] × [5, 7] × 5	16.52	3.04	17.05	3.26	15.95	3.90	16.35	3.73
20 × [2, 3] × [5, 7] × 6	10.08	3.24	10.00	3.34	9.89	4.19	9.25	4.25
20 × [2, 3] × [5, 7] × 7	6.70	3.44	6.57	3.68	6.66	4.45	6.11	4.37
20 × [2, 3] × [6, 8] × 5	11.67	2.87	11.41	3.33	11.23	4.39	11.16	4.41
20 × [2, 3] × [6, 8] × 6	6.59	3.25	5.95	3.69	6.02	4.99	5.51	5.05
20 × [2, 3] × [6, 8] × 7	4.84	3.72	4.86	3.89	5.14	5.06	4.73	4.99
30 × [1, 2] × [4, 6] × 7	17.47	5.41	16.75	5.65	17.20	6.66	16.39	6.80
30 × [1, 2] × [4, 6] × 8	14.01	5.44	13.52	5.68	13.23	6.91	13.19	6.90
30 × [1, 2] × [4, 6] × 9	11.12	5.17	9.80	5.71	10.03	6.91	9.91	7.04
30 × [1, 2] × [5, 7] × 7	13.03	5.18	13.05	5.84	12.37	7.25	12.23	7.29
30 × [1, 2] × [5, 7] × 8	11.69	5.78	11.38	5.95	11.70	7.34	11.41	7.39
30 × [1, 2] × [5, 7] × 9	8.77	5.74	9.41	5.96	8.56	7.34	7.90	7.45
30 × [1, 2] × [6, 8] × 7	11.08	5.88	10.94	6.15	10.84	7.59	10.81	7.69
30 × [1, 2] × [6, 8] × 8	8.20	5.97	7.77	6.20	7.19	7.75	7.76	7.70
30 × [1, 2] × [6, 8] × 9	9.29	5.87	8.18	6.26	8.70	7.77	7.88	7.91
30 × [2, 3] × [4, 6] × 7	24.71	6.11	24.81	6.43	24.31	8.02	24.19	8.06
30 × [2, 3] × [4, 6] × 8	18.64	5.75	18.85	6.50	18.99	8.17	18.70	8.24

Table 2 continued

$N \times L_i \times J_f \times M_j$	EDA		eEDA		DE–EDA		DE–eEDA	
	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)
30 × [2, 3] × [4, 6] × 9	16.96	5.87	16.78	6.51	16.10	8.17	16.28	8.32
30 × [2, 3] × [5, 7] × 7	19.73	6.42	19.74	6.80	19.57	8.61	19.45	8.73
30 × [2, 3] × [5, 7] × 8	13.77	6.57	13.64	6.77	12.81	8.81	13.32	8.87
30 × [2, 3] × [5, 7] × 9	9.78	6.64	8.79	6.95	9.56	8.76	9.01	8.92
30 × [2, 3] × [6, 8] × 7	13.45	6.34	13.04	6.80	12.78	8.75	12.44	8.78
30 × [2, 3] × [6, 8] × 8	10.47	6.98	10.57	7.29	8.73	8.87	9.04	8.85
30 × [2, 3] × [6, 8] × 9	7.43	6.42	6.57	6.61	7.02	8.49	6.66	8.68
40 × [1, 2] × [4, 6] × 9	19.67	9.15	18.64	9.56	18.03	11.32	18.05	10.84
40 × [1, 2] × [4, 6] × 10	16.13	9.41	15.83	9.45	15.66	11.12	15.12	11.57
40 × [1, 2] × [4, 6] × 11	14.87	9.42	13.80	9.45	13.82	11.50	13.65	11.67
40 × [1, 2] × [5, 7] × 9	13.18	9.62	13.52	9.79	12.73	11.85	12.04	12.20
40 × [1, 2] × [5, 7] × 10	14.49	9.83	12.82	9.92	13.07	12.07	12.69	12.21
40 × [1, 2] × [5, 7] × 11	10.11	9.78	9.68	10.01	9.26	11.95	8.93	11.57
40 × [1, 2] × [6, 8] × 9	14.36	10.22	13.48	10.35	13.34	12.54	13.07	12.80
40 × [1, 2] × [6, 8] × 10	8.54	10.03	7.64	10.37	7.97	12.32	7.00	12.66
40 × [1, 2] × [6, 8] × 11	8.91	8.64	8.32	9.27	7.92	12.34	7.95	12.94
40 × [2, 3] × [4, 6] × 9	28.35	10.06	28.36	10.87	28.08	11.85	28.12	13.17
40 × [2, 3] × [4, 6] × 10	21.64	9.76	21.48	10.57	21.47	12.70	21.35	13.05
40 × [2, 3] × [4, 6] × 11	19.35	10.75	18.93	10.41	18.57	12.56	18.88	12.35
40 × [2, 3] × [5, 7] × 9	22.61	11.24	21.66	11.31	21.11	13.76	21.50	14.33
40 × [2, 3] × [5, 7] × 10	19.62	10.61	18.86	11.20	18.57	12.49	18.61	14.21
40 × [2, 3] × [5, 7] × 11	13.10	11.39	11.87	11.57	11.86	14.07	11.59	14.27
40 × [2, 3] × [6, 8] × 9	16.21	11.56	14.57	11.87	14.50	15.21	14.34	15.32
40 × [2, 3] × [6, 8] × 10	14.81	11.64	14.36	11.46	13.23	15.19	13.20	14.95
40 × [2, 3] × [6, 8] × 11	10.72	9.78	9.92	9.62	10.10	14.30	9.77	13.46
50 × [1, 2] × [4, 6] × 11	20.77	14.81	19.87	14.73	19.96	17.26	19.56	16.46
50 × [1, 2] × [4, 6] × 12	17.51	15.02	16.92	14.41	17.02	16.58	16.13	17.45
50 × [1, 2] × [4, 6] × 13	14.01	13.62	13.88	13.65	12.71	15.73	12.07	15.82
50 × [1, 2] × [5, 7] × 11	18.07	12.92	16.34	13.95	16.73	16.80	16.00	16.23
50 × [1, 2] × [5, 7] × 12	14.86	14.44	14.30	14.33	13.08	16.12	12.87	17.31
50 × [1, 2] × [5, 7] × 13	13.06	14.39	11.85	14.23	11.76	16.73	11.46	17.11
50 × [1, 2] × [6, 8] × 11	12.54	14.63	11.91	14.60	11.56	17.41	10.82	17.30
50 × [1, 2] × [6, 8] × 12	13.21	16.13	11.69	15.72	11.98	17.93	11.50	17.27
50 × [1, 2] × [6, 8] × 13	10.10	16.36	9.66	15.17	9.24	18.64	8.92	18.48
50 × [2, 3] × [4, 6] × 11	30.26	14.62	30.10	14.97	29.93	17.92	30.01	18.45
50 × [2, 3] × [4, 6] × 12	25.53	16.49	25.01	16.42	24.82	19.93	24.65	20.55
50 × [2, 3] × [4, 6] × 13	22.36	15.70	21.89	16.00	21.76	19.80	21.77	20.19
50 × [2, 3] × [5, 7] × 11	23.34	15.47	22.32	15.67	22.19	18.10	21.87	18.79
50 × [2, 3] × [5, 7] × 12	19.66	17.36	18.99	17.24	18.71	20.57	18.91	19.88
50 × [2, 3] × [5, 7] × 13	15.92	17.49	15.08	17.68	14.51	21.33	14.56	21.96
50 × [2, 3] × [6, 8] × 11	19.22	17.47	17.64	18.47	17.62	21.87	17.43	22.18
50 × [2, 3] × [6, 8] × 12	14.23	17.63	13.29	18.38	12.56	22.43	12.75	21.47
50 × [2, 3] × [6, 8] × 13	11.77	16.92	10.29	16.61	10.36	20.63	10.03	20.69
Average	12.48	7.20	12.04	7.39	11.95	9.01	11.68	9.10

Table 3 Comparing δ average in different algorithms

$N \times L_i \times J_f \times M_j$	DE		DE–eEDA		GA		PSO	
	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)
10 × [1, 2] × [4, 6] × 3	6.90	1.25	6.78	1.39	6.98	1.51	7.11	1.29
10 × [1, 2] × [4, 6] × 4	4.29	1.36	3.32	1.52	3.63	1.60	3.65	1.32
10 × [1, 2] × [4, 6] × 5	2.67	1.39	2.65	1.39	2.67	1.33	2.64	1.25
10 × [1, 2] × [5, 7] × 3	7.74	1.31	7.62	1.45	7.97	1.67	7.18	1.35
10 × [1, 2] × [5, 7] × 4	5.83	1.25	5.81	1.32	5.75	1.42	5.53	1.23
10 × [1, 2] × [5, 7] × 5	2.54	1.32	2.58	1.43	2.75	1.59	2.54	1.53
10 × [1, 2] × [6, 8] × 3	6.75	1.44	6.70	1.46	6.91	1.58	6.70	1.46
10 × [1, 2] × [6, 8] × 4	3.95	1.48	4.07	1.57	4.07	1.72	3.95	1.60
10 × [1, 2] × [6, 8] × 5	2.50	1.38	2.33	1.36	2.49	1.59	2.55	1.55
10 × [2, 3] × [4, 6] × 3	14.27	1.43	13.64	1.66	13.65	1.65	13.61	1.62
10 × [2, 3] × [4, 6] × 4	8.54	1.53	7.95	1.89	7.88	1.98	7.99	1.82
10 × [2, 3] × [4, 6] × 5	4.89	1.68	4.77	1.85	4.76	1.99	4.76	1.93
10 × [2, 3] × [5, 7] × 3	12.53	1.76	12.82	1.68	12.85	2.04	12.44	1.85
10 × [2, 3] × [5, 7] × 4	4.42	1.64	3.96	1.82	4.05	1.97	3.80	1.95
10 × [2, 3] × [5, 7] × 5	2.70	1.65	2.80	1.86	2.80	2.06	2.70	2.01
10 × [2, 3] × [6, 8] × 3	5.75	1.66	6.10	1.86	6.05	2.09	5.75	2.05
10 × [2, 3] × [6, 8] × 4	3.25	1.98	3.26	2.02	3.29	2.08	3.26	1.97
10 × [2, 3] × [6, 8] × 5	2.06	1.96	2.22	2.17	2.18	2.41	1.93	2.44
20 × [1, 2] × [4, 6] × 5	14.01	2.59	12.63	3.38	14.72	3.14	14.36	2.90
20 × [1, 2] × [4, 6] × 6	9.62	2.89	8.38	3.74	10.16	3.51	9.41	3.26
20 × [1, 2] × [4, 6] × 7	7.23	2.94	5.69	3.80	6.86	3.56	6.82	3.26
20 × [1, 2] × [5, 7] × 5	13.14	3.05	9.91	3.89	11.32	3.69	12.05	3.40
20 × [1, 2] × [5, 7] × 6	9.59	3.10	8.91	3.98	10.17	3.81	11.17	3.39
20 × [1, 2] × [5, 7] × 7	7.39	3.16	5.59	4.01	7.13	3.88	6.04	3.39
20 × [1, 2] × [6, 8] × 5	10.48	3.21	7.32	4.13	8.46	3.99	7.76	3.76
20 × [1, 2] × [6, 8] × 6	6.98	3.02	5.39	3.88	6.74	3.65	5.48	3.57
20 × [1, 2] × [6, 8] × 7	6.02	3.15	5.59	4.23	5.95	4.14	5.69	3.81
20 × [2, 3] × [4, 6] × 5	22.29	3.49	22.03	4.36	22.67	4.31	22.38	4.26
20 × [2, 3] × [4, 6] × 6	14.24	3.52	13.34	4.40	14.67	4.34	13.86	4.35
20 × [2, 3] × [4, 6] × 7	10.92	3.34	10.16	3.93	10.92	4.00	11.26	4.05
20 × [2, 3] × [5, 7] × 5	16.35	3.45	16.35	3.73	16.70	3.88	16.49	4.42
20 × [2, 3] × [5, 7] × 6	9.83	3.50	9.25	4.25	10.47	4.30	9.70	4.42
20 × [2, 3] × [5, 7] × 7	6.88	3.56	6.11	4.37	7.08	4.76	6.74	5.06
20 × [2, 3] × [6, 8] × 5	12.17	3.67	11.16	4.41	11.76	4.59	11.35	4.73
20 × [2, 3] × [6, 8] × 6	6.17	4.17	5.51	5.05	6.65	5.26	5.69	5.18
20 × [2, 3] × [6, 8] × 7	4.96	4.20	4.73	4.99	5.10	5.29	4.67	5.37
30 × [1, 2] × [4, 6] × 7	17.97	4.65	16.39	6.80	18.76	5.71	17.02	5.54
30 × [1, 2] × [4, 6] × 8	13.92	4.71	13.19	6.90	14.96	5.69	14.33	5.61
30 × [1, 2] × [4, 6] × 9	11.43	4.73	9.91	7.04	11.42	5.88	10.41	5.66
30 × [1, 2] × [5, 7] × 7	14.52	5.05	12.23	7.29	13.09	6.26	13.40	6.13
30 × [1, 2] × [5, 7] × 8	12.26	5.15	11.41	7.39	12.85	6.34	11.66	6.30
30 × [1, 2] × [5, 7] × 9	8.80	5.15	7.90	7.45	9.14	6.40	8.84	6.22
30 × [1, 2] × [6, 8] × 7	12.20	5.41	10.81	7.69	11.56	6.67	10.96	6.58
30 × [1, 2] × [6, 8] × 8	10.18	5.46	7.76	7.70	9.25	6.83	8.33	6.92
30 × [1, 2] × [6, 8] × 9	9.21	5.61	7.88	7.91	8.81	6.97	8.39	6.86
30 × [2, 3] × [4, 6] × 7	24.81	5.84	24.19	8.06	24.93	7.31	24.49	7.50
30 × [2, 3] × [4, 6] × 8	19.13	5.77	18.70	8.24	19.53	7.42	18.97	7.34

Table 3 continued

$N \times L_i \times J_f \times M_j$	DE		DE–eEDA		GA		PSO	
	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)	δ (%)	CPU (s)
30 × [2, 3] × [4, 6] × 9	16.87	5.92	16.28	8.32	17.39	7.52	16.47	7.75
30 × [2, 3] × [5, 7] × 7	20.35	6.43	19.45	8.73	19.95	8.12	19.91	8.15
30 × [2, 3] × [5, 7] × 8	14.29	6.59	13.32	8.87	13.71	8.28	13.42	8.41
30 × [2, 3] × [5, 7] × 9	9.72	6.56	9.01	8.92	10.15	8.36	9.25	8.80
30 × [2, 3] × [6, 8] × 7	13.54	6.61	12.44	8.78	13.40	8.25	12.49	8.80
30 × [2, 3] × [6, 8] × 8	10.94	6.61	9.04	8.85	9.96	8.08	9.68	8.53
30 × [2, 3] × [6, 8] × 9	7.03	6.51	6.66	8.68	7.12	8.76	7.47	8.78
40 × [1, 2] × [4, 6] × 9	19.59	6.76	18.05	10.84	20.19	7.92	19.36	7.90
40 × [1, 2] × [4, 6] × 10	17.45	6.74	15.12	11.57	16.48	8.41	15.92	8.43
40 × [1, 2] × [4, 6] × 11	14.24	6.91	13.65	11.67	14.48	8.56	14.39	7.72
40 × [1, 2] × [5, 7] × 9	16.31	7.49	12.04	12.20	14.00	9.31	12.24	9.45
40 × [1, 2] × [5, 7] × 10	14.17	7.44	12.69	12.21	13.64	9.34	13.03	9.04
40 × [1, 2] × [5, 7] × 11	11.23	7.34	8.93	11.57	10.31	9.42	9.66	8.80
40 × [1, 2] × [6, 8] × 9	15.09	8.13	13.07	12.80	14.39	9.88	13.04	10.46
40 × [1, 2] × [6, 8] × 10	10.82	8.19	7.00	12.66	9.04	9.35	7.90	9.78
40 × [1, 2] × [6, 8] × 11	8.69	7.71	7.95	12.94	8.68	10.25	9.00	9.98
40 × [2, 3] × [4, 6] × 9	28.90	8.57	28.12	13.17	28.83	10.97	28.30	11.48
40 × [2, 3] × [4, 6] × 10	21.73	8.70	21.35	13.05	21.83	10.61	21.67	9.92
40 × [2, 3] × [4, 6] × 11	19.00	8.89	18.88	12.35	19.35	10.43	19.58	10.75
40 × [2, 3] × [5, 7] × 9	21.88	9.57	21.50	14.33	22.37	12.25	21.71	13.11
40 × [2, 3] × [5, 7] × 10	19.83	9.60	18.61	14.21	19.46	12.28	19.35	12.92
40 × [2, 3] × [5, 7] × 11	12.85	9.49	11.59	14.27	12.76	12.35	12.15	13.53
40 × [2, 3] × [6, 8] × 9	17.31	10.48	14.34	15.32	15.84	13.18	14.74	13.39
40 × [2, 3] × [6, 8] × 10	14.01	10.77	13.20	14.95	14.32	13.54	12.76	13.38
40 × [2, 3] × [6, 8] × 11	10.34	9.64	9.77	13.46	11.22	13.13	10.37	13.47
50 × [1, 2] × [4, 6] × 11	21.34	9.17	19.56	16.46	20.90	11.16	19.99	10.92
50 × [1, 2] × [4, 6] × 12	17.75	8.97	16.13	17.45	17.90	10.19	16.26	10.30
50 × [1, 2] × [4, 6] × 13	14.57	8.33	12.07	15.82	13.53	10.73	12.13	10.94
50 × [1, 2] × [5, 7] × 11	18.58	9.19	16.00	16.23	17.70	11.33	16.33	11.43
50 × [1, 2] × [5, 7] × 12	14.14	9.24	12.87	17.31	15.02	11.81	13.19	12.44
50 × [1, 2] × [5, 7] × 13	12.27	9.23	11.46	17.11	12.83	11.81	11.47	12.22
50 × [1, 2] × [6, 8] × 11	12.05	10.00	10.82	17.30	12.59	12.41	11.05	13.56
50 × [1, 2] × [6, 8] × 12	12.79	10.96	11.50	17.27	12.87	12.65	11.93	13.37
50 × [1, 2] × [6, 8] × 13	10.72	11.07	8.92	18.48	10.13	12.65	9.16	13.68
50 × [2, 3] × [4, 6] × 11	30.10	10.74	30.01	18.45	30.40	13.55	30.22	14.13
50 × [2, 3] × [4, 6] × 12	25.43	11.52	24.65	20.55	25.23	14.91	24.90	15.50
50 × [2, 3] × [4, 6] × 13	22.61	12.06	21.77	20.19	22.05	15.22	21.67	14.94
50 × [2, 3] × [5, 7] × 11	23.63	11.19	21.87	18.79	22.71	14.53	21.97	18.34
50 × [2, 3] × [5, 7] × 12	19.73	12.70	18.91	19.88	19.63	16.77	18.98	18.58
50 × [2, 3] × [5, 7] × 13	15.53	13.42	14.56	21.96	15.67	16.59	15.31	17.50
50 × [2, 3] × [6, 8] × 11	18.69	13.82	17.43	22.18	18.73	18.74	17.67	20.64
50 × [2, 3] × [6, 8] × 12	13.59	14.64	12.75	21.47	13.83	18.69	12.69	20.30
50 × [2, 3] × [6, 8] × 13	10.80	13.26	10.03	20.69	10.91	16.52	11.81	19.07
Average	12.72	6.02	11.68	9.10	12.60	7.52	12.07	7.71

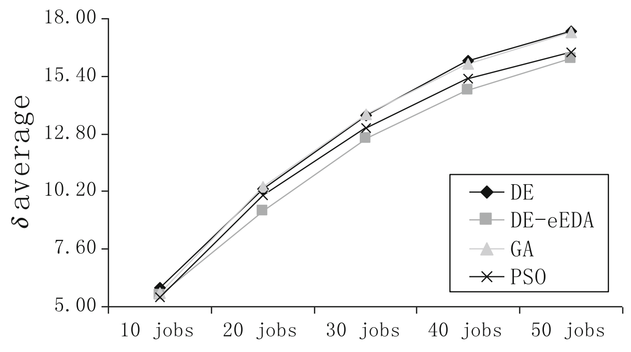

Fig. 3 δ average in different number of jobs

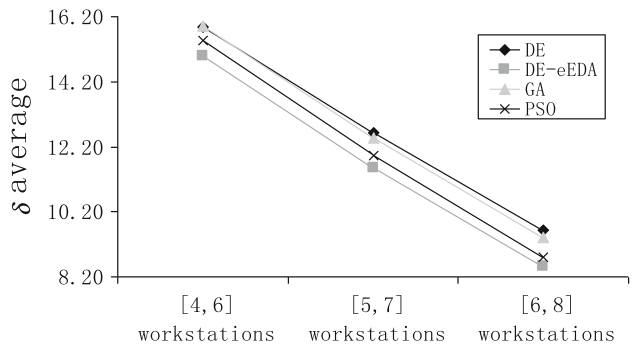

Fig. 5 δ average in different number of workstations

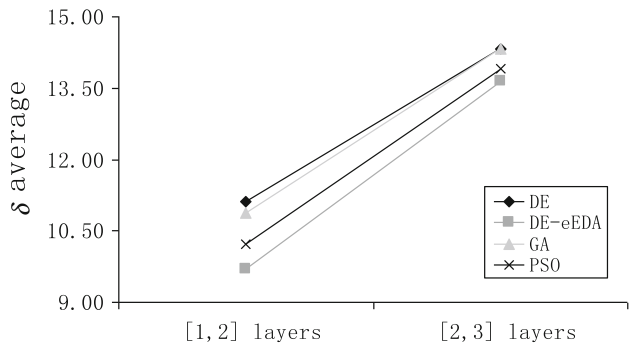

Fig. 4 δ average in different number of layers

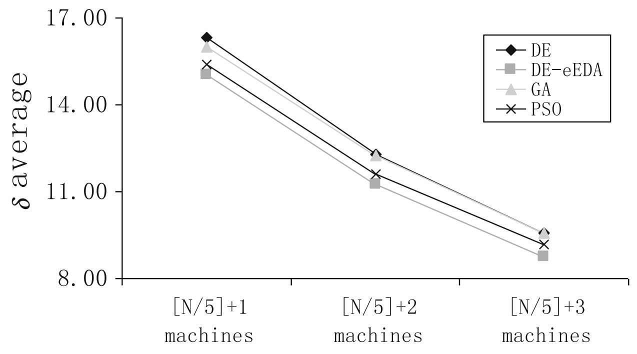

Fig. 6 δ average in different number of machines

space with collective information in global scope,especially the information of the similar blocks of jobs in the selected sequences, which has been verified to provide important information for evolution [33–35], while GA and PSO are updated using no such information. It can also be known that in order to collect and process the global information, the CPU time of the DE–eEDA is a little more than that of the GA and the PSO, which is still within a few seconds though. Thus, DE–eEDA can be applied to solve the RHFS problem and is usually true for its outperformance.

To further analyze the data, we tested the effect of number of jobs, number of layers, number of workstations and number of machines on the performance of the four algorithms, respectively. The results are shown in Figs. 3, 4, 5 and 6. The vertical axis represents the δ average.

Figure 3 shows that the DE–eEDA works the best in most of the job sizes except for the problem of 10 jobs. Figures 4, 5 and 6 show that the DE–eEDA outperforms the others in all different problem sizes.

In Fig. 3, for all the algorithms, when the other factors remain unchanged, δ average becomes large with the increase in the number of jobs for the possible reason that the competition for the machines between different jobs becomes more intense when the number of jobs increases. As what we describe in "Appendix", the lower bound of our problem is got by using LR algorithm relaxing the machine capacity, so when the competition for the machines between jobs becomes tight, the lower bound becomes loose, thus leading to the sharp increase in δ average. The same reason goes for the trends shown in Figs. 4, 5 and 6. In Fig. 4, δ average becomes large with the increase in the number of layers for the possible reason that the competition for the machines between different layers becomes more intense when the number of layers increases. In Fig. 5, δ average becomes small with the increase in the number of workstations for the possible reason that the competition for the machines between different layers becomes less intense when the number of workstations increases. In Fig. 6, δ average becomes small with the increase in the number of machines for the possible reason that the competition for the machines between different jobs becomes less intense when the number of machines increases.

6 Conclusions

This paper studies a RHFS scheduling problem with inspection and repair operations involved in fabrication of each layer. A mathematical programming model is constructed with an objective of minimizing the total weighted completion time. Seven meta-heuristics have been examined, and the percentage deviation of the total weighted completion time of each algorithm from the lower bound proposed in "Appendix" is used as a performance measure. In the proposed DE–eEDA, an ensemble model is introduced into the DE. Results have shown that the ensemble model improves the performance of the univariate model for the EDA and the DE by 3.53 and 2.26%, respectively, while the CPU times are nearly the same. We can also find that the DE with the eEDA involved improves the DE by 8.18%, and it works better than both the GA and the PSO.

Acknowledgements This study was supported by the National Natural Science Foundation of China under Grant Nos. 61273035 and 71471135.

Appendix

This section proposes how to construct the lower bound of our problem. The lower bound is got by the Lagrangian relaxation algorithm. The details are as follows.

Relaxing machine capacity constraints

The machine capacity constraint (5) can be relaxed by using nonnegative Lagrangian multipliers $\lambda_{k,u}(k = 1, \ldots, K,\ u = 1, \ldots, |M|)$. Then, the relaxed problem **RP** is as follows:

$$\min\left\{\sum_{i=1}^{N} w_i C_{i,L_i,J_i} + \sum_{u=1}^{M}\sum_{k=1}^{K}\lambda_{k,u}\times\left[\sum_{(i,j,l)\in O_u}(\phi(k - C_{i,l,j} + p_{i,l,j} - 1) - \phi(k - C_{i,l,j} - 1)) - 1\right]\right\} \tag{22}$$

$$\lambda_{k,u} \geq 0, \quad k = 1, \ldots, K \quad u = 1, \ldots, |M| \tag{23}$$

s.t. (2), (3), (4) and (23).

The objective function of problem **RP** can be written as:

$$\sum_{i=1}^{N}\min\left\{w_i C_{i,L_i,J_i} + \sum_{l=1}^{L_i}\sum_{j=1}^{J_i}\sum_{k=C_{i,l,j}-p_{i,l,j}+1}^{C_{i,l,j}}\lambda_{k,m_{i,l,j}}\right\} - \sum_{u=1}^{M}\sum_{k=1}^{K}\lambda_{k,u} \tag{24}$$

Then, the problem **RP** can be decomposed into independent job-level subproblems. The job-level subproblem denoted as SP_i can be presented as follows:

$$\min\left\{w_i C_{i,L_i,J_i} + \sum_{l=1}^{L_i}\sum_{j=1}^{J_i}\sum_{k=C_{i,l,j}-p_{i,l,j}+1}^{C_{i,l,j}}\lambda_{k,u}\right\} \tag{25}$$

s.t. (2), (3), (4) and (23).

Solving job-level subproblems

The subproblems are solved by dynamic programming with precedence constraints embedded in the dynamic programming recursion.

$h_{i,l,j}(u,t)$ represents the cost for completing the operation of job i at layer l at station j in time t on machine u. It is defined as follows:

$$h_{i,l,j}(u,t) = \begin{cases} \sum_{x=t-p_{i,l,j}+1}^{t}\lambda_{x,u} + w_i t & \text{if } l = L_i, j = J_i \\ \sum_{x=t-p_{i,l,j}+1}^{t}\lambda_{x,u} & \text{otherwise} \end{cases}$$

$$i = 1, \ldots, N, \quad l = 1, \ldots, L_i, \quad j = 1, \ldots, J_i, \quad t = T_{i,l,j}, \ldots, K, \quad u \in M_j \tag{26}$$

$T_{i,l,j}$ represents the earliest completion time of the operation of job i at layer l at station j. Let $f_{i,l,j}(u,t)$ be the optimal criterion value of state (u,t) for the operation of job i at layer l at station j. Then, the dynamic programming recursion for solving each job-level subproblem is expressed as:

$$f_{i,l,j}(u,t) = \begin{cases} h_{i,l,j}(u,t) \text{ if } l = 1, j = 1 \\ h_{i,l,j}(u,t) + \min\limits_{v\in M_{i,l-1,J_i}, T_{i,l-1,J_i}\leq x\leq t-p_{i,l-1,J_i}} f_{i,l-1,J_i}(v,x) & \text{if } \quad l \neq 1, j = 1 \\ h_{i,l,j}(u,t) + \min\limits_{v\in M_{i,l,j-1}, T_{i,l,j-1}\leq x\leq t-p_{i,l,j-1}} f_{i,l,j-1}(v,x) & \text{otherwise} \end{cases} \tag{27}$$

$$i = 1, \ldots, N, l = 1, \ldots, L_i, \quad j = 1, \ldots, J_i, \quad t = T_{i,l,j}, \ldots, K, \quad u \in M_j$$

The optimal machine selection and completion time for the operation of job i at layer L_i at station J_i can be obtained recursively by

$$(u^*_{i,L_i,J_i}, t^*_{i,L_i,J_i}) = \arg \min_{u \in M_{i,L_i,J_i}, T_{i,L_i,J_i} \le t \le K} f_{i,L_i,J_i}(u,t)$$

The optimal machine selection and completion time for the operation of job i at layer $L_i - 1, L_i - 2, \ldots, 1$ at station $J_i - 1, J_i - 2, \ldots, 1$ can be derived recursively by:

$$(u^*_{i,l,j}, t^*_{i,l,j}) = \begin{cases} \arg \min\limits_{u \in M_{i,l,j}, T_{i,l,j} \le t \le t^*_{i,l+1,0} - p_{i,l+1,0}} f_{i,l,j}(u,t) & \text{if } l \ne L_i, j = J_i \\ \arg \min\limits_{u \in M_{i,l,j}, T_{i,l,j} \le t \le t^*_{i,l,j+1} - p_{i,l,j+1}} f_{i,l,j}(u,t) & \text{otherwise} \end{cases} \tag{28}$$

Construction of a feasible solution

The priority list of jobs are created by sorting the job number according to the ascending order of the completion time of the operation for the job at the first station of the first layer from the solution of the relaxed problem. Assign the jobs to the first machine that becomes available successively. Then, for the following stations, update the ready times in each station to be the completion times of the previous station. Arrange the jobs in increasing order of ready times and assign the jobs to the first available machine successively.

Subgradient algorithm

The Lagrange multipliers are updated by:

$$\lambda_{k,u} = \max\left\{0, \lambda_{k,u} + \frac{UB - LB}{\sum_{u=1}^{M}\sum_{k=1}^{K} h^2_{k,u}} \times h_{k,u}\right\} \tag{29}$$

$$h_{k,u} = \sum_{(i,l,j) \in O_u} \left\{\phi(k - C_{i,l,j} + p_{i,l,j} - 1) - \phi(k - C_{i,l,j} - 1)\right\} - 1$$
$$k = 1, \ldots, K, \quad u = 1, \ldots, |M| \tag{30}$$

where UB and LB represents the upper bound and the lower bound, respectively.

Lagrangian relaxation algorithm

Step 1: Set the number of iterations n = 0, set the Lagrange multipliers $\lambda_{k,u} = 0$.
Step 2: Each job-level subproblem (SP_i) is solved by the dynamic programming recursion. The lower bound LB is calculated.
Step 3: Construct a feasible solution. The upper bound UB is calculated.
Step 4: If $n < 300$, go to Step 5. Otherwise, stop.
Step 5: Update the Lagrange multipliers and $n = n + 1$ and then return to Step 2.

The value of lower bound

See Table 4.

Table 4 The value of lower bound

Problem	$N \times L_i \times J_f \times M_j$	Lower bound
1	10 × [1, 2] × [4, 6] × 3	6161.73
2	10 × [1, 2] × [4, 6] × 4	5435.46
3	10 × [1, 2] × [4, 6] × 5	7432.97
4	10 × [1, 2] × [5, 7] × 3	8561.44
5	10 × [1, 2] × [5, 7] × 4	7932.97
6	10 × [1, 2] × [5, 7] × 5	7711.56
7	10 × [1, 2] × [6, 8] × 3	9306.18
8	10 × [1, 2] × [6, 8] × 4	9388.63
9	10 × [1, 2] × [6, 8] × 5	11,067.71
10	10 × [2, 3] × [4, 6] × 3	10,207.04
11	10 × [2, 3] × [4, 6] × 4	9280.70
12	10 × [2, 3] × [4, 6] × 5	8907.10
13	10 × [2, 3] × [5, 7] × 3	11,999.90
14	10 × [2, 3] × [5, 7] × 4	14,333.05
15	10 × [2, 3] × [5, 7] × 5	12,278.40
16	10 × [2, 3] × [6, 8] × 3	13,408.40
17	10 × [2, 3] × [6, 8] × 4	14,471.53
18	10 × [2, 3] × [6, 8] × 5	13,094.30
19	20 × [1, 2] × [4, 6] × 5	15,445.80
20	20 × [1, 2] × [4, 6] × 6	13,378.90
21	20 × [1, 2] × [4, 6] × 7	13,484.78
22	20 × [1, 2] × [5, 7] × 5	17,557.43
23	20 × [1, 2] × [5, 7] × 6	14,473.98
24	20 × [1, 2] × [5, 7] × 7	16,682.78
25	20 × [1, 2] × [6, 8] × 5	18,036.98
26	20 × [1, 2] × [6, 8] × 6	18,424.58
27	20 × [1, 2] × [6, 8] × 7	16,528.08
28	20 × [2, 3] × [4, 6] × 5	24,259.93
29	20 × [2, 3] × [4, 6] × 6	20,917.70
30	20 × [2, 3] × [4, 6] × 7	20,496.35
31	20 × [2, 3] × [5, 7] × 5	25,862.50
32	20 × [2, 3] × [5, 7] × 6	25,830.55
33	20 × [2, 3] × [5, 7] × 7	27,976.03
34	20 × [2, 3] × [6, 8] × 5	30,100.63
35	20 × [2, 3] × [6, 8] × 6	32,195.70
36	20 × [2, 3] × [6, 8] × 7	28,458.03

Table 4 continued

Problem	$N \times L_i \times J_f \times M_j$	Lower bound
37	30 × [1, 2] × [4, 6] × 7	20,004.08
38	30 × [1, 2] × [4, 6] × 8	18,850.28
39	30 × [1, 2] × [4, 6] × 9	18,304.98
40	30 × [1, 2] × [5, 7] × 7	26,110.65
41	30 × [1, 2] × [5, 7] × 8	23,277.90
42	30 × [1, 2] × [5, 7] × 9	21,997.85
43	30 × [1, 2] × [6, 8] × 7	27,748.70
44	30 × [1, 2] × [6, 8] × 8	26,583.10
45	30 × [1, 2] × [6, 8] × 9	31,420.55
46	30 × [2, 3] × [4, 6] × 7	33,717.23
47	30 × [2, 3] × [4, 6] × 8	31,403.30
48	30 × [2, 3] × [4, 6] × 9	34,776.75
49	30 × [2, 3] × [5, 7] × 7	36,496.73
50	30 × [2, 3] × [5, 7] × 8	39,413.20
51	30 × [2, 3] × [5, 7] × 9	38,430.10
52	30 × [2, 3] × [6, 8] × 7	41,394.13
53	30 × [2, 3] × [6, 8] × 8	43,733.63
54	30 × [2, 3] × [6, 8] × 9	42,295.78
55	40 × [1, 2] × [4, 6] × 9	28,890.33
56	40 × [1, 2] × [4, 6] × 10	28,922.95
57	40 × [1, 2] × [4, 6] × 11	27,622.23
58	40 × [1, 2] × [5, 7] × 9	33,323.38
59	40 × [1, 2] × [5, 7] × 10	30,590.48
60	40 × [1, 2] × [5, 7] × 11	32,263.58
61	40 × [1, 2] × [6, 8] × 9	38,511.83
62	40 × [1, 2] × [6, 8] × 10	31,138.85
63	40 × [1, 2] × [6, 8] × 11	39,580.38
64	40 × [2, 3] × [4, 6] × 9	43,547.23
65	40 × [2, 3] × [4, 6] × 10	45,313.23
66	40 × [2, 3] × [4, 6] × 11	47,309.90
67	40 × [2, 3] × [5, 7] × 9	56,573.48
68	40 × [2, 3] × [5, 7] × 10	50,950.70
69	40 × [2, 3] × [5, 7] × 11	53,072.23
70	40 × [2, 3] × [6, 8] × 9	57,084.43
71	40 × [2, 3] × [6, 8] × 10	60,739.25
72	40 × [2, 3] × [6, 8] × 11	57,656.08
73	50 × [1, 2] × [4, 6] × 11	35,556.95
74	50 × [1, 2] × [4, 6] × 12	33,702.48
75	50 × [1, 2] × [4, 6] × 13	35,507.50
76	50 × [1, 2] × [5, 7] × 11	40,492.40
77	50 × [1, 2] × [5, 7] × 12	38,021.98
78	50 × [1, 2] × [5, 7] × 13	39,354.50
79	50 × [1, 2] × [6, 8] × 11	51,147.75
80	50 × [1, 2] × [6, 8] × 12	45,837.88
81	50 × [1, 2] × [6, 8] × 13	46,825.70
82	50 × [2, 3] × [4, 6] × 11	56,248.38
83	50 × [2, 3] × [4, 6] × 12	53,639.18
84	50 × [2, 3] × [4, 6] × 13	56,316.03
85	50 × [2, 3] × [5, 7] × 11	65,342.13
86	50 × [2, 3] × [5, 7] × 12	64,000.60
87	50 × [2, 3] × [5, 7] × 13	65,317.40
88	50 × [2, 3] × [6, 8] × 11	75,627.28
89	50 × [2, 3] × [6, 8] × 12	76,810.45
90	50 × [2, 3] × [6, 8] × 13	75,521.28
Average		31,504.54

References

1. Hekmatfar M, Ghomi SMTF, Karimi B (2011) Two stage reentrant hybrid flow shop with setup times and the criterion of minimizing makespan. Appl Soft Comput 11(8):4530–4539
2. Choi SW, Kim YD, Lee GC (2005) Minimizing total tardiness of orders with reentrant lots in a hybrid flowshop. Int J Prod Res 43(11):2149–2167
3. Jiang S, Tang L (2008) Lagrangian relaxation algorithms for reentrant hybrid flowshop scheduling. In: 2008 IEEE international conference on information management, innovation management and industrial engineering, vol 1. IEEE, Taipei, pp 78–81
4. Cho HM, Bae SJ, Kim J et al (2011) Bi-objective scheduling for reentrant hybrid flow shop using Pareto genetic algorithm. Comput Ind Eng 61(3):529–541
5. Choi HS, Kim JS, Lee DH (2011) Real-time scheduling for reentrant hybrid flow shops: a decision tree based mechanism and its application to a TFT-LCD line. Expert Syst Appl 38(4):3514–3521
6. Dugardin F, Amodeo L, Yalaoui F (2009) Multiobjective scheduling of a reentrant hybrid flowshop. In: 2009 IEEE international conference on computers & industrial engineering. IEEE, Troyes, pp 193–195
7. Dugardin F, Yalaoui F, Amodeo L (2010) New multi-objective method to solve reentrant hybrid flow shop scheduling problem. Eur J Oper Res 203(1):22–31
8. Yalaoui N, Amodeo L, Yalaoui F, et al (2010) Particle swarm optimization under fuzzy logic controller for solving a hybrid Reentrant Flow Shop problem. In: 2010 IEEE international symposium on parallel and distributed processing, workshops and Phd forum. IEEE, Atlanta, pp 1–6
9. Rau H, Cho KH (2009) Genetic algorithm modeling for the inspection allocation in reentrant production systems. Expert Syst Appl 36(8):11287–11295
10. Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359
11. Ilonen J, Kamarainen JK, Lampinen J (2003) Differential evolution training algorithm for feed-forward neural networks. Neural Process Lett 17(1):93–105
12. Liu J, Lampinen J (2005) A fuzzy adaptive differential evolution algorithm. Soft Comput 9(6):448–462
13. Wang H, Wu Z, Rahnamayan S (2011) Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems. Soft Comput 15(11):2127–2140
14. Pan QK, Suganthan PN, Wang L et al (2011) A differential evolution algorithm with self-adapting strategy and control parameters. Comput Oper Res 38(1):394–408

15. Arslan M, Çunkaş M, Sağ T (2012) Determination of induction motor parameters with differential evolution algorithm. Neural Comput Appl 21(8):1995–2004
16. Wang GG, Gandomi AH, Alavi AH et al (2014) Hybrid krill herd algorithm with differential evolution for global numerical optimization. Neural Comput Appl 25(2):297–308
17. Wang L, Zou F, Hei X et al (2014) A hybridization of teaching–learning-based optimization and differential evolution for chaotic time series prediction. Neural Comput Appl 25(6):1407–1422
18. Chiou JP, Chang CF, Su CT (2004) Ant direction hybrid differential evolution for solving large capacitor placement problems. IEEE Trans Power Syst 19(4):1794–1800
19. Onwubolu G, Davendra D (2006) Scheduling flow shops using differential evolution algorithm. Eur J Oper Res 171(2):674–692
20. Qian B, Wang L, Huang DX et al (2008) Scheduling multi-objective job shops using a memetic algorithm based on differential evolution. Int J Adv Manuf Technol 35(9–10):1014–1027
21. Damak N, Jarboui B, Siarry P et al (2009) Differential evolution for solving multi-mode resource-constrained project scheduling problems. Comput Oper Res 36(9):2653–2659
22. Pan QK, Wang L, Gao L et al (2011) An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers. Inf Sci 181(3):668–685
23. Omran MG, Salman A, Engelbrecht AP (2005) Self-adaptive differential evolution. In: International conference on computational and information science. Springer, Berlin, pp 192–199
24. Qin AK, Huang VL, Suganthan PN (2009) Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evol Comput 13(2):398–417
25. Wang L, Pan QK, Suganthan PN, Wang WH, Wang YM (2010) A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Comput Oper Res 37(3):509–520
26. Yildiz AR (2013) Hybrid Taguchi-differential evolution algorithm for optimization of multi-pass turning operations. Appl Soft Comput 13(3):1433–1439
27. Zhang Y, Li X (2011) Estimation of distribution algorithm for permutation flow shops with total flowtime minimization. Comput Ind Eng 60(4):706–718
28. Wang L, Wang S, Xu Y et al (2012) A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. Comput Ind Eng 62(4):917–926
29. Wang S, Wang L, Liu M et al (2013) An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. Int J Prod Econ 145(1):387–396
30. Donate JP, Li X, Sánchez GG et al (2013) Time series forecasting by evolving artificial neural networks with genetic algorithms, differential evolution and estimation of distribution algorithm. Neural Comput Appl 22(1):11–20
31. Cheng S, Lu X, Zhou X (2014) Globally optimal selection of web composite services based on univariate marginal distribution algorithm. Neural Comput Appl 24(1):27–36
32. Lozano JA, Larranaga P, Inza I (2006) Towards a new evolutionary computation: advances on estimation of distribution algorithms. Springer, Berlin
33. Jarboui B, Eddaly M, Siarry P (2009) An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. Comput Oper Res 36(9):2638–2646
34. Pan QK, Ruiz R (2012) An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. Omega-Int J Manage S 40(2):166–180
35. Chen SH, Chen MC (2013) Addressing the advantages of using ensemble probabilistic models in estimation of distribution algorithms for scheduling problems. Int J Prod Econ 141(1):24–33