Computers & Industrial Engineering 65 (2013) 246–260

Contents lists available at SciVerse ScienceDirect

Computers & Industrial Engineering

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

Flexible job shop scheduling using hybrid differential evolution algorithms ☆

Yuan Yuan, Hua Xu *

State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, PR China

ARTICLE INFO

Article history:
Received 4 May 2012
Received in revised form 22 September 2012
Accepted 25 February 2013
Available online 14 March 2013

Keywords:
Scheduling
Flexible job shop
Differential evolution
Local search
Neighborhood structure
Makespan

ABSTRACT

This paper proposes hybrid differential evolution (HDE) algorithms for solving the flexible job shop scheduling problem (FJSP) with the criterion to minimize the makespan. Firstly, a novel conversion mechanism is developed to make the differential evolution (DE) algorithm that works on the continuous domain adaptive to explore the problem space of the discrete FJSP. Secondly, a local search algorithm based on the critical path is embedded in the DE framework to balance the exploration and exploitation by enhancing the local searching ability. In addition, in the local search phase, the speed-up method to find an acceptable schedule within the neighborhood structure is presented to improve the efficiency of whole algorithms. Extensive computational results and comparisons show that the proposed algorithms are very competitive with the state of the art, some new best known solutions for well known benchmark instances have even been found.

1. Introduction

The job shop scheduling problem (JSP) is one of most important and difficult problems in the field of production scheduling. The flexible job shop scheduling problem (FJSP) is an extension of the classical JSP, in which operations are allowed to be processed by any machine from a given set, rather than one specified machine. Generally, the FJSP is much closer to a real production environment and has more practical applicability. However, the FJSP is more complex than the JSP because of its additional decision to assign each operation to the appropriate machine (routing) besides sequencing operations on machines. It has been proved that the FJSP is strongly NP-hard even if each job has at most three operations and there are two machines (Garey et al., 1976).

Although exact algorithms based on a disjunctive graph representation of the problem have been developed, but they are not applicable for instances with more than 20 jobs and 10 machines (Pinedo, 2002). So, metaheuristics for the FJSP, which aim to find the near-optimal schedule with acceptable computational time, have gained increasing attention in the past decades. Among them, tabu search (TS), genetic algorithm (GA), particle swarm optimization (PSO) were most frequently adopted to solve the FJSP.

As for TS, Brandimarte (1993) proposed a hybrid TS heuristic with some known dispatching rules to solve the FJSP. Hurink et al. (1994) presented a TS procedure in which routing and sequencing are regarded as two different types of moves. Dauzère-Pérès and Paulli (1997) also developed a TS algorithm based on a new neighborhood structure that makes no distinction between routing and sequencing. Mastrolilli and Gambardella (2000) further improved their TS techniques and proposed two neighborhood functions for the FJSP. Recently, Bożejko et al. (2010) dealt with the FJSP by using a parallel TS based meta^{2}heuristics which treats routing and sequencing separately. Li et al. (2011) developed a hybrid TS with an efficient neighborhood structure for the FJSP.

As for GA, Chen et al. (1999) pronounced an effective GA to solve the FJSP. In their method, the chromosome representation is divided into two parts, the first one denotes a concrete allocation of operations to each machine and the second one describes the sequence of operations on each machine. Kacem et al. (2002a) developed a GA for the FJSP controlled by the assigned model which is generated by the approach of localization. Jia et al. (2003) proposed a modified GA which is able to solve distribute scheduling problems and the FJSP. Pezzella et al. (2008) presented a GA in which a mix of different strategies for generating the initial population, selecting the individuals for reproduction, and reproducing new

☆ This manuscript was processed by Area Editor T.C. Edwin Cheng.
* Corresponding author.
E-mail address: xuhua@tsinghua.edu.cn (H. Xu).

0360-8352/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cie.2013.02.022

individuals are integrated. Gao et al. (2008) combined GA with variable neighborhood descent (VND) procedure for solving the FJSP with three objectives.

As for PSO, Xia and Wu (2005) made use of PSO to assign operations on machines and simulated annealing (SA) algorithm to sequence operations on each machine. Zhang et al. (2009) combined a PSO algorithm with a TS procedure for solving the FJSP. Moslehi and Mahnam (2011) studied the FJSP with an integrated multi-objective approach based on hybridization of PSO and local search. All the three works mentioned above mainly considered the multi-objective FJSP. Their experimental results demonstrate the effectiveness of the PSO.

In addition, a growing number of studies for the FJSP have concerned other metaheuristics during most recent years. Bagheri et al. (2010) presented an artificial immune algorithm (AIA) using some resultful rules. Yazdani et al. (2010) proposed a parallel variable neighborhood search (PVNS) algorithm based on the application of multiple independent searches. Xing et al. (2010) developed a knowledge-based ant colony optimization (KBACO) algorithm that provides an effective integration between the ant colony optimization (ACO) model and the knowledge model. Wang et al. (2011) and Wang et al. (2012) applied the artificial bee colony (ABC) algorithm and estimation of distribution algorithm (EDA) to the FJSP, both of which stress the balance between global exploration and local exploitation. It is also worth noting that metaheuristics based on constraint programming (CP) techniques have gradually shown great potential in solving the FJSP. The discrepancy search (DS), large neighborhood search (LNS) and iterative flattening search (IFS) have been well tested on the FJSP and achieved the excellent performance on some standard benchmarks (Ben Hmida et al., 2010; Oddi et al., 2011; Pacino and Hentenryck, 2011).

The differential evolution (DE) algorithm proposed by Storn and Price (1997) is one of the latest population-based evolutionary metaheuristics, which was originally devised for solving continuous optimization problems. As a stochastic real-parameter global optimizer, the DE employs simple mutation and crossover operators to generate new candidate solutions, and applies one-to-one competition scheme to greedily determine whether the new candidate or its parent will survive in the next generation. Due to its simplicity, ease of implementation, fast convergence, and robustness, the DE algorithm has captured much attention and gained a wide range of successful applications such as digital filter design (Storn, 1999), feed-forward neural networks training (Ilonen et al., 2003; Zhu et al., 2005), economic load dispatch in power systems (Noman and Iba, 2008), and traveling salesman problem (Fatih Tasgetiren et al., 2010). However, because of its continuous nature, the study on DE for scheduling problems is still considerably limited (Damak et al., 2009; Kazemipoor et al., 2012; Onwubolu and Davendra, 2006; Qian et al., 2008; Qian et al., 2009). And, as far as we are aware, there is no published research work that describes the using of DE to deal with the FJSP. In this paper, we will propose hybrid differential evolution (HDE) algorithms for solving the FJSP with the criterion to minimize the makespan. In particular, a novel conversion mechanism is developed to make the continuous DE applicable for solving the discrete FJSP. To achieve a balance between the global exploration and local exploitation of the search space, a local search procedure based on the critical path is embedded into the global search based DE. Besides, in the local search phase, two neighborhood structures are presented, and the speed-up method is also developed to find an acceptable schedule in the neighborhood more quickly. Based on the two neighborhood structures, two variants of HDE algorithms are formed, HDE-N_1 and HDE-N_2. Experimental studies demonstrate the effectiveness and efficiency of the proposed algorithms in comparison with the state of the art.

The remainder of this paper is organized as follows. In Section 2, the FJSP is formulated and an illustrative problem instance is given. In Section 3, the basic DE algorithm is introduced. In Section 4, the proposed HDE algorithms for the FJSP are illustrated in detail. The extensive computational results and comparisons are provided in Section 5. Finally, we end the paper with some conclusions in Section 6.

2. Problem formulation

The FJSP is formally formulated as the following. There are a set of n independent jobs $J = \{J_1, J_2, \ldots, J_n\}$ and a set of m machines $M = \{M_1, M_2, \ldots, M_m\}$. A job J_i is formed by a sequence of n_i operations $\{O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}\}$ to be performed one after another according to the given sequence. Each operation $O_{i,j}$, i.e the jth operation of job J_i, must be executed on one machine chosen from a given subset $M_{i,j} \subseteq M$. The processing time of the operation is machine dependent. $p_{i,j,k}$ is denoted to be the processing time of $O_{i,j}$ on machine M_k. The scheduling consists of two subproblems: the routing subproblem that assigns each operation to an appropriate machine and the sequencing subproblem that determines a sequence of operations on all the machines. The objective is to find a schedule that minimize the makespan. The makespan means the time needed to complete all the jobs and can be defined as $C_{\max} = \max_{1 \leqslant i \leqslant n}(C_i)$, where C_i is the completion time of job J_i.

Moreover, the following assumptions are made in this study: all the machines are available at time 0; all the jobs are released at time 0; each machine can process only one operation at a time; each operation must be completed without interruption once it starts; the order of operations for each job is predefined and cannot be modified; the setting up time of machines and transfer time of operations are negligible.

For illustrating explicitly, a sample instance of FJSP is shown in Table 1, where rows correspond to operations and columns correspond to machines. Each entry of the input table denotes the processing time of that operation on the corresponding machine. In this table, the tag "–" means that a machine cannot execute the corresponding operation.

3. Basic DE algorithm

The DE algorithm (Storn and Price, 1997) is a population-based evolutionary algorithm, utilizing NP real-valued parameter vectors as a population for each generation G. Each vector, also known as *chromosome*, forms a candidate solution to the optimization problem, which is generally defined as minimizing (or maximizing) $f(\overrightarrow{X})$, such that $x_j \in [x_{j,\min}, x_{j,\max}]$, where $f(\overrightarrow{X})$ is the objective function (or fitness function), $\overrightarrow{X} = [x_1, x_2, \ldots, x_D]^T$ is a vector consisting of D dimensional decision variables, and $x_{j,\min}$ and $x_{j,\max}$ are the lower and upper bounds for each decision variable, respectively. The basic DE works through a simple cycle of stages to do the task of optimization until the termination criterion (i.e. maximum number of generations, or maximum computation time) is

Table 1
Processing time table of an instance of FJSP.

Job	Operation	M_1	M_2	M_3
J_1	$O_{1,1}$	2	–	3
	$O_{1,2}$	4	1	3
J_2	$O_{2,1}$	–	5	3
	$O_{2,2}$	6	2	4
	$O_{2,3}$	3	–	–
J_3	$O_{3,1}$	4	5	2
	$O_{3,2}$	3	–	2

satisfied, just as presented in Fig. 1. The details of each stage are described as follows.

3.1. Initialization

The initialization of the basic DE aims to set the control parameters and the initial population of vectors. The parameters include the size of the population (*NP*), the mutation scale factor (*F*), and the crossover probability (*Cr*). It is obvious that a good set of parameters can enhance the ability of the algorithm to search for the global optimum or near optimum region with a high convergence rate.

Let us denote the subsequent generations in DE by $G = 0, 1, \ldots, G_{\max}$. The *i*th vector of the population at the current generation is represented as the following:

$$\overrightarrow{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, \ldots, x_{D,i,G}]^T \tag{1}$$

The initial population ($G = 0$) of vectors are usually generated randomly which could cover the constrained search space as much as possible. Thus, the *j*th component of the *i*th vector may be initialized as

$$x_{j,i,0} = x_{j,\min} + (x_{j,\max} - x_{j,\min}) \times rand(0,1) \tag{2}$$

where $rand(0,1)$ is a random function returning a real number between 0 and 1 with uniform distribution.

3.2. Mutation

The mutation can be seen as a perturbation to the individual during the evolutionary process. In the basic DE, a parent vector from the current generation is called *target vector*, the mutation operator is to construct a mutant vector $V_{i,G} = [v_{1,i,G}, v_{2,i,G}, \ldots, v_{D,i,G}]^T$, which is known as *donor vector*, corresponding to the *i*th target vector $X_{i,G}$. As known, there exist several different mutation schemes of the DE algorithm. In this paper, the scheme *DE/best/1* is followed, which is illustrated as

$$\overrightarrow{V}_{i,G} = \overrightarrow{X}_{best,G} + F \cdot \left(\overrightarrow{X}_{r_1^i,G} - \overrightarrow{X}_{r_2^i,G}\right) \tag{3}$$

$X_{best,G}$ is the vector with the best fitness (i.e. lowest objective function value for a minimization problem) in the population at generation *G*. The indices r_1^i and r_2^i are two distinct integers randomly chosen in the range $[1, NP]$, and both are different from the base index *i*. The scale factor *F* is a constant real number within the range [0, 2], usually less than 1.

From the above description, the mutation in DE is quite different from its counterpart in traditional GA. Instead of small alterations of genes in GA mutation, the DE mutation is performed by means of combinations of individuals (Panduro et al., 2009).

3.3. Crossover

The crossover represents as a typical case of the information exchange between the individuals. In the basic DE, the crossover operator is to generate a vector $U_{i,G} = [u_{1,i,G}, u_{2,i,G}, \ldots, u_{D,i,G}]^T$, called *trial vector*, for the *i*th target vector $\overrightarrow{X}_{i,G}$ through combing components from $\overrightarrow{X}_{i,G}$ and its corresponding donor vector $\overrightarrow{V}_{i,G}$. The generation of a trial vector is outlined as the following equation:

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } rand(0,1) \leqslant Cr \text{ or } j = q \\ x_{j,i,G}, & \text{otherwise} \end{cases} \tag{4}$$

q is a randomly chosen integer within the range $[1, D]$, which guarantees that the $U_{i,G}$ gets at least one component from $V_{i,G}$. The crossover probability *Cr* is a constant real number taken from the interval [0, 1].

3.4. Selection

The selection operator is to decide whether or not the trial vector $U_{i,G}$ is the member of the population for the next generation, which can be described as

$$\overrightarrow{X}_{i,G+1} = \begin{cases} \overrightarrow{U}_{i,G}, & \text{if } f(\overrightarrow{U}_{i,G}) \leqslant f(\overrightarrow{X}_{i,G}) \\ \overrightarrow{X}_{i,G}, & \text{otherwise} \end{cases} \tag{5}$$

where $f(\overrightarrow{X})$ is the objective function to be minimized. Therefore, for the minimization problem, the new trial vector will replace the corresponding target vector in the next generation if its objective function value is not greater than that of the target vector; otherwise, the target is retained in the population.

4. Proposed HDE for the FJSP

4.1. Overview of the HDE

The framework of the proposed HDE is based on the basic DE and its algorithmic flow is depicted as follows:

Step 1: Set the size of the population (*NP*), scale factor (*F*), crossover probability (*Cr*), maximum number of generations ($G_{\max}$), probability of carrying out local search (P_l), and maximum local iterations ($iter_{\max}$).
Step 2: Set $G = 0$ and initialize the population.
Step 3: Evaluate each chromosome and label the $\overrightarrow{X}_{best,G}$.
Step 4: Mutation phase. Generate *NP* donor vectors $\overrightarrow{V}_{i,G}$, $i = 1, 2, \ldots, NP$, by using the mutation operator described in Eq. (3).
Step 5: Crossover phase. Generate *NP* trial vectors $\overrightarrow{U}_{i,G}$, $i = 1, 2, \ldots, NP$, according to the crossover operator illustrated in Eq. (4).

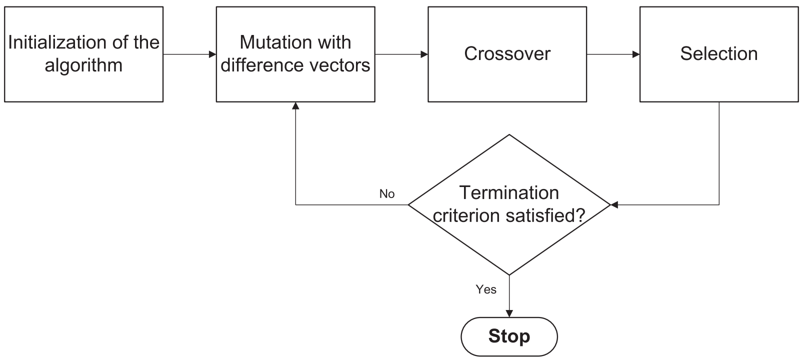
Initialization of the algorithm
Mutation with difference vectors
Crossover
Selection
Termination criterion satisfied?
No
Yes
Stop

Fig. 1. Main stages of the basic DE algorithm.

Step 6: For each trial vector $\overrightarrow{U}_{i,G}$, sample $rn \in U(0,1)$, if $rn < P_l$, perform the local search to the $\overrightarrow{U}_{i,G}$; otherwise evaluate it directly.

Step 7: Selection phase. Determine NP target vectors $\overrightarrow{X}_{i,G+1},\ i = 1, 2, \ldots, NP$, by one-to-one selection operator described in Eq. (5) for the next generation. Set $G = G + 1$.

Step 8: Update the $\overrightarrow{X}_{best,G}$.

Step 9: If $G < G_{\max}$, then go to Step 4; otherwise stop the procedure and return the $\overrightarrow{X}_{best,G}$.

As can be seen, the maximum number of generations is adopted as the termination criterion. Unlike the basic DE, the HDE not only employs the DE based evolutionary searching mechanism to effectively perform the exploration for promising solutions within the entire region, but it also employs the well developed local search algorithm to perform the exploitation for local improvement of solutions. In the HDE, the local search is applied to the trial vector $\overrightarrow{U}_{i,G}$ but not the target vector $\overrightarrow{X}_{i,G}$, which is beneficial to avoid both cycling search and getting trapped in a local optimum. Moreover, the frequency and intensity of the local search are controlled by the parameters P_l and $iter_{\max}$, respectively.

The implementation of the proposed HDE for the FJSP concerns two crucial issues. One is the evaluation of a chromosome, the other is how to apply the local search to a chromosome.

To evaluate a chromosome (represented by real-parameter vector, see Section 4.2), it is firstly converted to a kind of discrete *two-vector code* (see Section 4.3), called *forward conversion* (see Section 4.4.1) in this paper, then the two-vector code is decoded to an *active schedule* (Pinedo, 2002). The fitness of the chromosome is given as the value of makespan for this schedule, just as depicted in Fig. 2. The optimal schedule is always within the set of active schedules for the problems to minimize the makespan, so only the active schedule is considered here to correspond to a chromosome, which could largely reduce the search space.

As for the local search to a chromosome, its computational flow is depicted in Fig. 3. In fact, the local search algorithm is not directly applied to a chromosome, but to the schedule corresponding to the chromosome, which is helpful for introducing the problem-specific knowledge. So the operator of evaluation is firstly adopted to obtain the schedule, then the schedule is further improved by the local search (see Section 4.5). After that, the improved schedule is encoded to a two-vector code, which is then converted to a chromosome by using *backward conversion* (see Section 4.4.2). The obtained improved chromosome enters into the evolutionary process in replace of the original one.

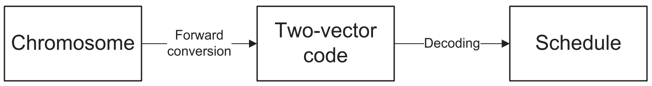

Fig. 2. The computational flow of the evaluation.

In the following subsections, we will detail the implementation of the proposed HDE for the FJSP. Section 4.2 introduces the representation of the chromosome and the initialization of the population. In Section 4.3, the two-vector code will be illustrated including its encoding and decoding method. In Section 4.4, the conversion techniques, forward conversion and backward conversion, are presented. The proposed local search is described in Section 4.5.

4.2. Representation and initialization

In the proposed HDE algorithms, a chromosome $\overrightarrow{X} = [x_1, x_2, \ldots, x_D]^T$, is still represented as an D-dimensional real-parameter vector. But the dimension D should satisfy the constraint $D = 2d$, where d is the total number of operations in the FJSP to solve. To denote the two-level decision making (routing and sequencing) of the FJSP respectively, the chromosome $\overrightarrow{X}$ is divided into two separate parts. The first half part, $\overrightarrow{X}^{(1)} = [x_1, x_2, \ldots, x_d]^T$, describes the information of machine assignment for each operation, while the last half part, $\overrightarrow{X}^{(2)} = [x_{d+1}, x_{d+2}, \ldots, x_{2d}]^T$, presents the information of operations sequencing on all the machines. What's more, to deal with the problem conveniently, the ranges $[x_{j,\min}, x_{j,\max}]$, $j = 1, 2, \ldots, D$ for decision variables are all set as $[-\delta, \delta]$, $\delta > 0$, where δ is referred as bound factor in our proposed algorithms.

The population of the proposed HDE is also initialized randomly and uniformly just as the basic DE. A chromosome in the HDE can be produced randomly according to Eq. (2), where $x_{j,\min} = -\delta$, $x_{j,\max} = \delta$, $j = 1, 2, \ldots, D$.

4.3. Two-vector code

The two-vector code consists of two vectors: machine assignment vector and operation sequence vector, corresponding well to two subproblems in the FJSP.

For explaining the two vectors, a fixed ID for each operation is first given in accordance with the job number and operation order within the job. This numbering scheme is illustrated in Table 2 for the instance shown in Table 1. After numbered, the operation can also be referred to by the fixed ID, for example, operation 5 has the same reference with the operation $O_{2,3}$ as shown in Table 2.

4.3.1. Machine assignment vector

The machine assignment vector, represented by $\overrightarrow{R} = [r_1, r_2, \ldots, r_d]^T$, is an array of d integer values. In the vector, r_j, $j = 1, 2, \ldots, d$, denotes the operation j chooses the r_jth machine in its alternative machine set. In Fig. 4, a possible machine assignment vector for the problem in Table 1 is shown and its meaning is

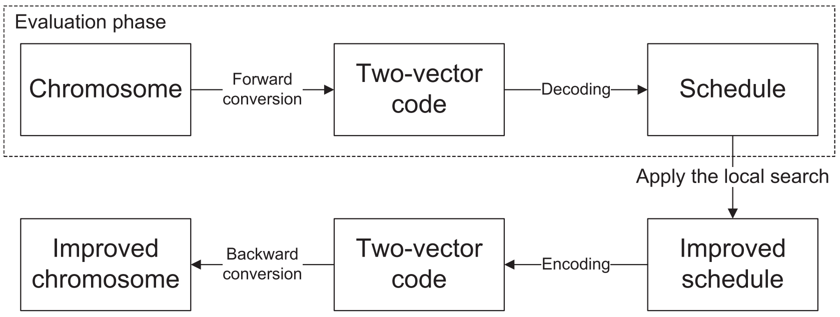

Fig. 3. The computational flow of the local search to a chromosome.

Table 2
Illustration of numbering scheme for operations.

Operation indicated	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{3,1}$	$O_{3,2}$
Fixed ID	1	2	3	4	5	6	7

also revealed. For example, $r_7 = 2$ indicates that the operation $O_{3,2}$ selects the 2nd machine in its alternative machine set, that is machine M_3 where the corresponding square is shadowed.

4.3.2. Operation sequence vector

The operation sequence vector, expressed as $\overrightarrow{S} = [s_1, s_2, \ldots, s_d]^T$, is the ID permutation of all the operations. The order of occurrence for each operation in the $\overrightarrow{S}$ indicates its scheduling priority. Take the instance shown in Table 1 for example, a possible operation sequence vector is represented as $\overrightarrow{S} = [3, 6, 4, 7, 1, 5, 2]^T$. The vector $\overrightarrow{S}$ can be directly translated into a unique list of ordered operations: $O_{2,1} \succ O_{3,1} \succ O_{2,2} \succ O_{3,2} \succ O_{1,1} \succ O_{2,3} \succ O_{1,2}$. Operation $O_{2,1}$ has the highest priority and is scheduled first, then the operation $O_{3,1}$, and so on. It must be noted that not all the ID permutations are feasible for the operation sequence vector because of the designated priority of operations lying in a job. That is to say, the operations within a job should keep the relative priority order in the $\overrightarrow{S}$.

4.3.3. Encoding and decoding

To encode a schedule of the FJSP to the two-vector code is simple and direct, the vector $\overrightarrow{R}$ is obtained just by the machine assignment in the schedule, while the vector $\overrightarrow{S}$ is got through sorting all the operations in the non-decreasing order of the earliest start time.

The decoding of the two-vector code is divided into two steps. The first step is to assign each operation to the selected machine according to the $\overrightarrow{R}$. Then the second is to treat all the operations one by one according to their order in the $\overrightarrow{S}$, each operation under treatment is allocated in the best available processing time for the corresponding machine. A schedule generated by this way can be ensured to be an active schedule (Cheng et al., 1996).

4.4. Conversion techniques

4.4.1. Forward conversion

The forward conversion is to convert a chromosome represented by the real-parameter vector $\overrightarrow{X} = [x_1, x_2, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, x_{2d}]^T$ to a two-vector code consisting of two integer-parameter vectors $\overrightarrow{R} = [r_1, r_2, \ldots, r_d]^T$ and $\overrightarrow{S} = [s_1, s_2, \ldots, s_d]^T$. This kind of conversion is divided into two separate parts.

For the conversion in the first part, the vector $\overrightarrow{X}^{(1)} = [x_1, x_2, \ldots, x_d]^T$ is converted to the machine assignment vector $\overrightarrow{R} = [r_1, r_2, \ldots, r_d]^T$. Let $\overrightarrow{L} = [l_1, l_2, \ldots, l_d]^T$, where $l_j j = 1,2,\ldots,d$, denotes the size of alternative machine set for operation j. What we need is to map the real number $x_j \in [-\delta, \delta]$ to the integer $r_j \in [1, l_j]$. The concrete procedure is: firstly convert x_j to a real number within the range $[1, l_j]$ by linear transformation, then r_j is given the nearest integer value for the converted real number, which is described as Eq. (6).

$$r_j = round\left(\frac{1}{2\delta}(l_j - 1)(x_j + \delta) + 1\right), \quad j = 1, 2, \ldots, d \tag{6}$$

where $round(x)$ is the function that rounds the number x to the nearest integer. In the particular case $l_j = 1$, r_j always equals to 1 no matter what value of x_j is.

In the second part, $\overrightarrow{X}^{(2)} = [x_{d+1}, x_{d+2}, \ldots, x_{2d}]^T$ is converted to the operation sequence vector $\overrightarrow{S} = [s_1, s_2, \ldots, s_d]^T$. To realize this conversion, the largest position value (LPV) rule (Wang et al., 2010) is first employed to construct an ID permutation of operations by ordering the operations in their non-increasing position value. However, as mentioned in Section 4.3.2, the obtained permutation may be not feasible for the $\overrightarrow{S}$. So, the repair procedure illustrated in Algorithm 1 is further carried out to adjust the relative order of operations within a job in the permutation.

Algorithm 1. RepairPermutation ($\overrightarrow{S}$)

```
1: Set $\overrightarrow{Q} = [q_1, q_2, \ldots, q_n]^T$
2: $[q_1, q_2, \ldots, q_n]^T \leftarrow [0, 0, \ldots, 0]^T$
3: for $i = 1$ to $d$ do
4:     Get the job $J_k$ that the operation $s_i$ belongs to
5:     $q_k \leftarrow q_k + 1$
6:     Get the fixed ID $op$ for the operation $O_{k,q_k}$
7:     $s_i \leftarrow op$
8: end for
```

Suppose that we have a vector $\overrightarrow{X}^{(2)} = [0.6, -0.5, 0.4, -0.1, 0.8, 0.2, -0.3]^T$ for the instance shown in Table 1, then an example of conversion is illustrated in Fig. 5.

4.4.2. Backward conversion

The backward conversion is to convert a two-vector code containing the vectors $\overrightarrow{R} = [r_1, r_2, \ldots, r_d]^T$ and $\overrightarrow{S} = [s_1, s_2, \ldots, s_d]^T$ to a chromosome $\overrightarrow{X} = [x_1, x_2, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, x_{2d}]^T$, which occurs after the local improvement to the schedule as described in Fig. 3. This type of conversion also consists of two separated parts just like the forward conversion.

In the first part related to machine assignment, the vector $\overrightarrow{R} = [r_1, r_2, \ldots, r_d]^T$ is converted to the vector $\overrightarrow{X}^{(1)} = [x_1, x_2, \ldots, x_d]^T$, the conversion is in fact an inverse linear transformation of Eq. (6). But the case of $l_j = 1$ should be considered alone, l_j chooses a random

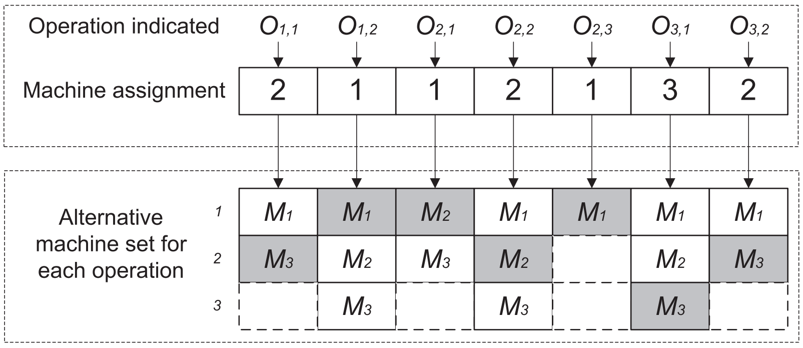

Fig. 4. Illustration of the machine assignment vector.

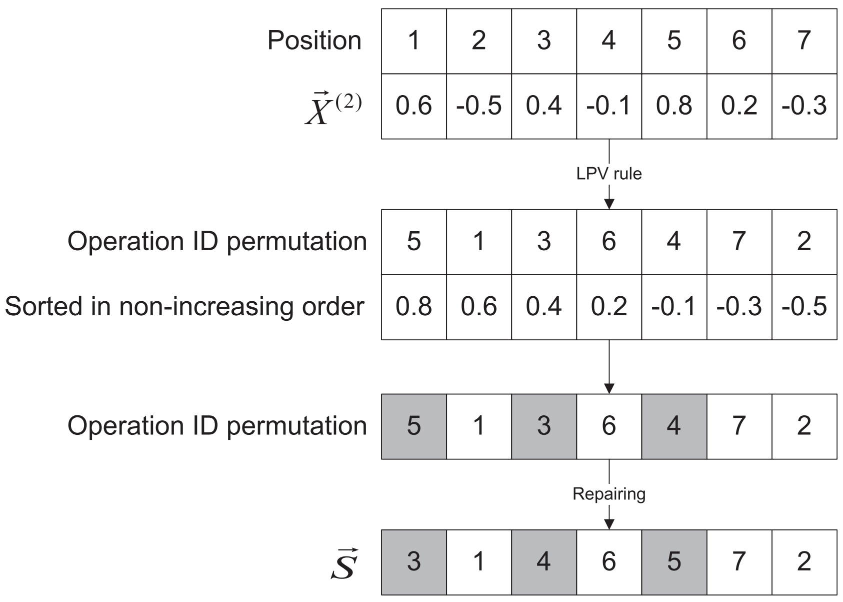

Fig. 5. The conversion from $\vec{X}^{(2)}$ to the operation sequence vector $\vec{S}$.

value in the range $[-\delta, \delta]$ when $l_j = 1$. This transformation can be performed as follows

$$x_j = \begin{cases} \frac{2\delta}{l_j-1}(r_j - 1) - \delta, & l_j \neq 1 \\ x_j \in [-\delta, \delta], & l_j = 1 \end{cases} \tag{7}$$

where $j = 1, 2, \ldots, d$.

For the second part, the vector $\vec{X}^{(2)} = [x_{d+1}, x_{d+2}, \ldots, x_{2d}]^T$, is obtained by rearranging elements in the original $\vec{X}^{(2)}$ before local improved. The rearrangement makes the yielded new $\vec{X}^{(2)}$ correspond to the $\vec{S}$ of the improved schedule according to the LPV rule. Take the problem shown in Table 1 for instance, a possible conversion is depicted in Fig. 6.

4.5. Local search algorithm

In this subsection, how to apply the local search to improve a schedule will be illustrated in detail. Firstly, the *disjunctive graph*, a kind of representation for the schedule, is introduced. Then, two neighborhood structures N_1 and N_2 based on the critical path in the disjunctive graph are presented. Finally, we summarize the procedure of local search.

4.5.1. Disjunctive graph

A schedule of the FJSP can be represented by the disjunctive graph $G = (V, C \bigcup D)$. In the graph, V denotes a set of all the nodes, each node represents an operation in the FJSP (including dummy starting and terminating operations); C is the set of all the conjunctive arcs, these arcs connect two adjacent operations within one job and the directions of them represent the processing order between two connected operations; D means a set of all the disjunctive arcs, these arcs connect two adjacent operations performed on the same machine and their directions also show the processing order. The processing time for each operation is generally labeled above the corresponding node and regarded as the weight of the node. For example, a possible schedule represented by the disjunctive graph for the problem shown in Table 1 is illustrated in Fig. 7, in which $O_{3,1}$, $O_{1,1}$, $O_{2,3}$ are processed in succession on the machine M_1, $O_{2,1}$, $O_{1,2}$ are executed successively on the machine M_2, and $O_{3,2}$, $O_{2,2}$ are performed in turn on the machine M_3.

A schedule of the FJSP is feasible, if and only if there exist no cyclic paths in its corresponding disjunctive graph. If a disjunctive graph is acyclic, then the longest path from the starting node S to the ending node E is called critical path, whose length defines the makespan for the schedule. Operations on the critical path are known as critical operations. For example, the disjunctive graph illustrated in Fig. 7 is acyclic, so it is a feasible schedule;

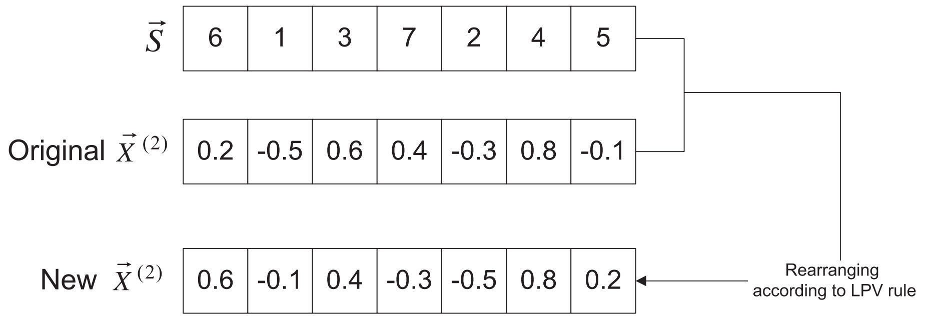

Fig. 6. The conversion from the operation sequence vector $\vec{S}$ to the $\vec{X}^{(2)}$.

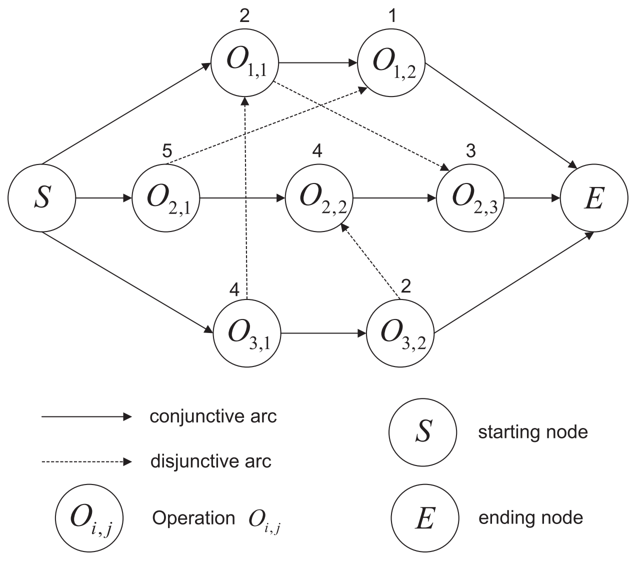

Fig. 7. Illustration of the disjunctive graph.

its critical path is $S \to O_{3,1} \to O_{3,\,2} \to O_{2,2} \to O_{2,3} \to E$ and the makespan equals to 13; the operations $O_{3,1}$, $O_{3,2}$, $O_{2,2}$ and $O_{2,3}$ are all critical operations.

4.5.2. Neighborhood structures

Since the makespan is no shorter than any other path in the disjunctive graph, the makespan may be only improved by moving critical operations. In the disjunctive graph, to move an operation $O_{i,j}$ is performed in two steps consisting of deletion and insertion (Mastrolilli and Gambardella, 2000):

Step 1: Delete the node v representing $O_{i,j}$ from its current machine sequence by removing all its disjunctive arcs in the disjunctive graph. Set the weight of node v equal to 0.
Step 2: Assign $O_{i,j}$ to an machine M_k and choose the position of v in the processing order of M_k, insert the node v by adding its disjunctive arcs and setting the weight of node v equal to $p_{i,j,k}$.

Let G be the current schedule, a critical path in G is represented by $S \to co_1 \to co_2 \cdots \to co_w \to E$. The neighborhood structure $N_1(G)$ is defined as the set of schedules (including infeasible schedules) obtained by moving one critical operation co_x, $x = 1,2,\ldots,w$. Denote u_k is the number of operations processed on M_k in the schedule G, then the size of $N_1(G)$ can be calculated as

$$U_{total} = w \cdot \left(\sum_{k=1}^{m}(u_k + 1) - 1\right) = w \cdot \left(\sum_{k=1}^{m}u_k + m - 1\right) = w \cdot (d + m - 1) \tag{8}$$

Our local search algorithm is to continually choose an acceptable schedule G' from the set of $N_1(G)$ and set G' as the new current schedule. A schedule $G' \in N_1(G)$ is acceptable in the proposed local search if it satisfies G' is acyclic and $C_{\max}(G') \leqslant C_{\max}(G)$. Obviously, we can form schedules in the neighborhood structure orderly until an acceptable one is found. But it is very time consuming because we must check whether it is cyclic and recalculate its makespan for each formed schedule. Here, the speed-up method will be developed to find an acceptable schedule in $N_1(G)$ more quickly.

Denote $ES^G(O_{i,j})$ as the earliest start time of operation $O_{i,j}$ in the schedule G and $LS^G(O_{i,j})$ as the latest start time without delaying the makespan. Hence, the earliest completion time of $O_{i,j}$ is $EC^G(O_{i,j}) = ES^G(O_{i,j}) + p_{i,j,k}$, and the latest completion time $LC^G(O_{i,j}) = LS^G(O_{i,j}) + p_{i,j,k}$, where $O_{i,j}$ is processed on machine M_k. Let $PM(O_{i,j})$ be the operation processed on the same machine right before $O_{i,j}$ and $SM(O_{i,j})$ be the operation processed on the same machine right after $O_{i,j}$. Let $PJ(O_{i,j}) = O_{i,j-1}$ be the operation that precedes $O_{i,j}$ within the job J_i and $SJ(O_{i,j}) = O_{i,j+1}$ be the operation of job J_i that follows $O_{i,j}$. Let co_x is the critical operation to be moved, G^- is the schedule obtained after deleting the operation co_x in G. We take $C_{\max}(G)$ as the "required" makespan and calculate the latest start time $LS^{G^-}(O_{i,j})$ for each operation $O_{i,j}$ in G^- according to this makespan.

If co_x is inserted before $O_{i,j}$ on machine M_k in G^- to get a schedule G' that satisfies $C_{\max}(G') \leqslant C_{\max}(G)$, it should be started as early as $EC^{G^-}(PM(O_{i,j}))$, and can be completed as late as $LS^{G^-}(O_{i,j})$ without delaying $C_{\max}(G)$. Besides, co_x must follow the precedence constraints within the same job. So, if the position before $O_{i,j}$ is said to be available for co_x to insert into, the following equation should be met

$$\max\{EC^{G^-}(PM(O_{i,j})), EC^{G^-}(PJ(co_x))\} + p_{co_x,k} < \min\{LS^{G^-}(O_{i,j}), LS^{G^-}(SJ(co_x))\} \tag{9}$$

In Eq. (9), the "<" not the "⩽" is used because this can guarantee co_x is not the critical operation in G' and avoid cyclic search as much as possible.

Unfortunately, to insert c_x before $O_{i,j}$ under Eq. (9) is satisfied cannot ensure the yielded G' is acyclic. Let Θ_k be the set of operations processed by machine M_k in G^- and ordered by increasing earliest start time (note that $co_x \notin \Theta_k$). Let Φ_k and Ψ_k denote two subsequences of Θ_k defined as follows:

$$\Phi_k = \{v \in \Theta_k | ES^G(v) + p_{v,k} > ES^{G^-}(co_x)\} \tag{10}$$

$$\Psi_k = \{v \in \Theta_k | LS^G(v) < LS^{G^-}(co_x)\} \tag{11}$$

Denote Υ_k is the set of positions before all the operations of $\Phi_k \backslash \Psi_k$ and after all the operations of $\Psi_k \backslash \Phi_k$. Then the following theorem is established

Theory 1. *The schedule obtained by inserting the operation co_x into a position $\gamma \in \Upsilon_k$ is always feasible, and there exists a position in the set Υ_k that is the optimal one for co_x to insert into on machine M_k.*

The detail proof of this theory can be referred in Mastrolilli and Gambardella (2000). According to Theory 1, we have the direct corollary as follows.

Corollary 1. *If an acceptable schedule can be obtained by inserting co_x into a position on machine M_k, then there always exists an acceptable schedule that is yielded by inserting co_x into a position in the set Υ_k.*

So, when we want to find a position for co_x to insert into on machine M_k, only the positions in Υ_k are considered, and if the position met Eq. (9), an acceptable schedule is got immediately by inserting co_x into it. The detail procedure of getting an acceptable schedule from the neighborhood structure N_1 is described in Algorithm 2.

Algorithm 2. GetAcceptableSchedule–I (G)

```
1: Get a critical path $S \to co_1 \to co_2 \cdots \to co_w \to E$ in $G$
2: for $x = 1$ to $w$ do
3:   Delete the operation $co_x$ from $G$ to get $G^-$
4:   for $k = 1$ to $m$ do
5:     Get the set of positions $\Upsilon_k$ on the machine $M_k$
6:     for each position $\gamma \in \Upsilon_k$ do
7:       if $\gamma$ satisfies Eq. (9) then
8:         Insert $co_x$ into the position $\gamma$ to get the schedule $G'$
```

```
9:             return $G'$
10:          end if
11:      end for
12:    end for
13: end for
14: return a null schedule
```

To make a more intensive search, the neighborhood structure $N_2(G)$ is also defined. It not only includes the schedules obtained by moving one critical operation in a critical path in G, but also includes the schedules through moving two operations, at least one of which is critical. Obviously, $N_2(G)$ is much larger than $N_1(G)$, and $N_1(G) \subset N_2(G)$. Our method to get an acceptable schedule from the neighborhood structure N_2 is depicted in Algorithm 3.

Algorithm 3. GetAcceptableSchedule-II (G)

```
1. $G' \leftarrow$ GetAcceptableSchedule-I (G)
2. if $G'$ is not a null schedule then
3.   return $G'$
4. end if
5: for each critical operation $co_x$ on a critical path in $G$ then
6.   Delete the operation $co_x$ from $G$ to get $G^{-}$
7.   for each operation $o$ in $G^{-}$ do
8.     Delete the operation $o$ from $G^{-}$ to get $G^{-'}$
9.     if a suitable position $\gamma$ in $G^{-'}$ is found for $co_x$ to insert
   into then
10.      Insert $co_x$ into $\gamma$ to get $G^{-''}$
11.      if a suitable position $\gamma'$ in $G^{-''}$ is found for $o$ to insert
   into then
12.        Insert $o$ into $\gamma'$ to get $G'$
13.        return $G'$
14.      end if
15.    end if
16.  end for
17. end for
18. return a null schedule
```

From Algorithm 3, it can be seen that moving two operations is only executed when failing to move one operation, because it is much more time consuming. In other word, we prefer to choose an acceptable schedule from N_1, the schedules in $N_2 \backslash N_1$ are considered only when there is no acceptable one in N_1. The "suitable position" in step 9 and 11 means that to insert the operation into this position cannot delay $C_{\max}(G)$ which could be judged like Eq. (9), and the schedule obtained after insertion should be feasible. Because Theorem 1 does not hold when two operations are moved, it is necessary to check whether the obtained graph is cyclic after inserting an operation. In addition, the set $N_2 \backslash N_1$ is very large, so we do not consider all the possible insertions when moving two operations. In fact, as illustrated in Algorithm 3, for the two operations co_x and o to be moved, once co_x has been inserted into a suitable position, the other suitable positions for co_x will not be tried no matter whether a suitable position is found for o to insert into. This is a compromise between the computation cost and optimization level.

4.5.3. Procedure of local search

The procedure of local search is given in Algorithm 4. In step 3, we can also call Algorithm 3 to generate an acceptable schedule instead of calling Algorithm 2. If the embedded local search adopts Algorithm 2, the corresponding proposed HDE is denoted by HDE-N_1, otherwise the proposed HDE is named as HDE-N_2 when Algorithm 3 is used.

Algorithm 4. LocalSearch $(G, iter_{\max})$

```
1. $i \leftarrow 0$
2. while G is not a null schedule and $i < iter_{\max}$
3.   $G \leftarrow$ GetAcceptableSchedule-I (G)
4.   $i \leftarrow i + 1$
5. end while
6. return G
```

5. Experimental studies

5.1. Experimental setup

The proposed HDE algorithms were implemented in Java language on an Intel 2.83 GHz Xeon processor with 15.9 Gb of RAM. To evaluate the performance of HDE algorithms (HDE-N_1 and HDE-N_2), the following four sets of well known benchmark instances in the FJSP literature are considered:

(1) *Kacem data*: The data set consists of five instances from Kacem et al. (2002b) with number of jobs ranging from 4 to 15, number of machine ranging from 5 to 10, number of operations for each job ranging from 2 to 4, and number of operations for all jobs ranges from 12 to 56.
(2) *BRdata*: The data set consists of 10 instances from Brandimarte (1993), which were generated randomly generated using a uniform distribution between given limits. The number of jobs ranges from 10 to 20, number of machines ranges from 4 to 15, number of operations for each job ranging from 5 to 15, and number of operations for all jobs ranges from 55 to 240.
(3) *BCdata*: The data set consists of 21 instances from Barnes and Chambers (1996), which were obtained from three ever challenging classical JSP instances (mt10, la24, la40) (Fisher and Thompson, 1963; Lawrence, 1984). The number of jobs ranges from 10 to 15, number of machines ranges from 11 to 18, number of operations for each job ranging from 10 to 15, and number of operations for all jobs ranges from 100 to 225.
(4) *HUdata*: The data set consists of 129 instances from Hurink et al. (1994), which were constructed from three instances (mt06, mt10, mt20) by Fisher and Thompson (1963) and 40 instances (la01–la40) by Lawrence (1984). Depending on the average number of alternative machines for each operation, HUdata was divided into three subsets: Edata, Rdata, and Vdata. The number of jobs ranges from 6 to 30, number of machines ranges from 5 to 15, number of operations for each job ranges from 5 to 15, and number of operations for all jobs ranges from 36 to 300.

The proposed algorithms run 50 independent times for each instance from Kacem data, BRdata, and BCdata, and only run 10 independent times for each instance from HUdata due to the large number of instances in this data set. The results will involve four metrics including the best makespan (Best), the average makespan (AVG), the standard deviation of makespan (SD), and the average computational time in seconds (CPU_{av}) obtained by the related algorithms.

To show the superiority of our HDE algorithms, we compare our computational results with several most competitive algorithms in the literature for each data set. The mean relative error (MRE) is also introduced to analyze the quality of the solutions. For a given instance, the relative error is defined as $RE = (MK - LB)/LB \times 100\%$, where MK is the makespan obtained by the reported algorithm and LB is the best known lower bound. For our analysis, the LB of BRdata and BCdata instances are taken from Mastrolilli and Gambardella (2000), while the LB of HUdata instances are computed by Jurisch (1992). The LB of Kacem instances are not available.

An issue usually involved in comparative assessment of the algorithms for the FJSP is to quantify the computational effort. However, the different computing hardware, programming platforms and coding skills used in each algorithm make this comparisons notoriously problematic, and the real computer-independent CPU time is hard to get. Hence, when concerning this issue in this paper, we enclose the original name of the CPU, the programming language, and the original computational time for the corresponding algorithm, which is enough for us to have a roughly understanding of the efficiency of referred algorithms. This practice has been often adopted in the existing research of the JSP (Nasiri and Kianfar, 2012; Sha and Hsu, 2006; Zhang et al., 2008).

The proposed algorithms HDE-N_1 and HDE-N_2 adopt the same parameters in our experiments. The parameter settings for each data set are summarized in Table 3, which are set in such a way that a relatively good trade-off between solution quality and computational time can be obtained.

5.2. Results of Kacem instances

The first data set under study is Kacem data. We compare our HDE-N_1 and HDE-N_2 with two recently proposed algorithms including TSPCB of Li et al. (2011) and BEDA of Wang et al. (2012). The detail results are listed in Table 4. The first column symbolizes the name for each instance; the second column shows the size of the instance, in which n stands for the number of jobs and m represents the number of machines; the third column lists the best known solution (BKS) ever reported in the literature for each instance; the remaining columns describe the computational results of TSPCB, BEDA, HDE-N_1 and HDE-N_2 respectively, where the SD values of TSPCB are not available. The bold values in the table mean the best results.

It can be seen from Table 4 that BEDA and HDE-N_2 are the most effective among the four algorithms, both of which can consistently obtain the best known solutions for all five instances. As for the computational effort, HDE-N_2 is also safely comparable with BEDA, considering that the efficiency of Java is much lower than C++. HDE-N_2 shows superiority to solve the instance 15×10, which spends about 2 s and is much more efficient than TSPCB and BEDA. The proposed HDE-N_1 just cannot obtain the best consistently for the instance 15×10, but it seems to be the most efficient one among the four related algorithms.

5.3. Results of BRdata instances

The second data set investigated is BRdata. Our algorithms are still compared with TSPCB and BEDA. The detail results are shown in Table 5 that bears the same quantities of Table 4.

From Table 5, it can be seen that HDE-N_1 is more effective, efficiency and robust than TSPCB and BEDA in solving BRdata instances. In particular, as for the best makespan obtained, HDE-N_1 outperforms TSPCB in 5 out of 10 instances, and outperforms BEDA in 2 out of 10 instances; HDE-N_1 outperforms both TSPCB and BEDA in all 10 instances for the average makespan obtained; the SD values of HDE-N_1 is relative smaller than BEDA in general, so it shows more robustness; as for the efficiency, the overall average computational time of HDE-N_1 is less than TSPCB and BEDA for most instances. The results of the MRE also reveal the effectiveness of HDE-N_1. HDE-N_1 obtains the MRE of the best makespan which is equal to 15.58%, while TSPCB and BEDA is 18.66% and 16.07% respectively. For the MRE of the average makespan obtained, HDE-N_1 generates 16.52%, faced to 18.95% for TSPCB, and 19.24% for BEDA. Compared with HDE-N_1, the proposed HDE-N_2 further improves the best results of three instances (MK05, MK06, MK10). On the whole, HDE-N_2 matches eight best known solutions and even finds a new best solution for the instance MK06 (improved from 58 to 57). However, it is much more time consuming than HDE-N_1, and also appears to be less efficient than TSPCB and BEDA.

5.4. Results of BCdata instances

The BCdata is one of the largest data set for the FJSP in the literature. Recent important work on this data set can be referred in Bożejko et al. (2010) and Oddi et al. (2011). Bożejko et al. (2010) proposed a parallel TS algorithm using a high performance GPU with 128 processors, which is possible to obtain 6 new best

Table 3
Parameter settings of HDE algorithms.

Parameter	Description	Kacem data	BRdata	BCdata	HUdata		
					Edata	Rdata	Vdata
NP	Population size	20	30	50	50	50	50
F	Mutation scale factor	0.1	0.1	0.5	0.5	0.1	0.1
Cr	Crossover probability	0.3	0.3	0.1	0.1	0.3	0.3
$G_{\max}$	Maximum number of generations	100	200	700	700	700	700
P_l	Probability of carrying out local search	0.7	0.7	0.8	0.8	0.8	0.8
$iter_{\max}$	Maximum local iterations	80	80	90	90	120	150
δ	Bound factor	1.0	1.0	1.0	1.0	1.0	1.0

Table 4
Results of five Kacem instances.

Instance	$n \times m$	BKS	TSPCB[a]			BEDA[b]				HDE-N_1[c]				HDE-N_2[c]			
			Best	AVG	CPU$_{av}$	Best	AVG	SD	CPU$_{av}$	Best	AVG	SD	CPU$_{av}$	Best	AVG	SD	CPU$_{av}$
Case 1	4×5	11	**11**	11.00	0.05	**11**	11.00	0.00	0.01	**11**	11.00	0.00	0.06	**11**	11.00	0.00	0.09
Case 2	8×8	14	**14**	14.20	4.68	**14**	14.00	0.00	0.23	**14**	14.00	0.00	0.14	**14**	14.00	0.00	0.31
Case 3	10×7	11	**11**	11.00	5.21	**11**	11.00	0.00	0.30	**11**	11.00	0.00	0.19	**11**	11.00	0.00	0.46
Case 4	10×10	7	**7**	7.10	1.72	**7**	7.00	0.00	0.42	**7**	7.00	0.00	0.22	**7**	7.00	0.00	0.37
Case 5	15×10	11	**11**	11.70	9.82	**11**	11.00	0.00	14.88	**11**	11.86	0.35	0.66	**11**	11.00	0.00	2.19

[a] The CPU time on a Pentium IV 1.6 GHz processor in C++.
[b] The CPU time on an Intel Core i5 3.2 GHz processor in C++.
[c] The CPU time on an Intel 2.83 GHz Xeon processor in Java.

Table 5
Results of 10 BRdata instances.

Instance	$n \times m$	BKS	TSPCB[a]			BEDA[b]				HDE-N_1[c]				HDE-N_2[c]			
			Best	AVG	CPU$_{av}$	Best	AVG	SD	CPU$_{av}$	Best	AVG	SD	CPU$_{av}$	Best	AVG	SD	CPU$_{av}$
MK01	10 × 6	40	**40**	40.30	2.80	**40**	41.02	0.83	1.09	**40**	40.00	0.00	1.16	**40**	40.00	0.00	4.01
MK02	10 × 6	26	**26**	26.50	19.31	**26**	27.25	0.67	2.16	**26**	26.52	0.50	1.48	**26**	26.00	0.00	6.09
MK03	15 × 8	204	**204**	204.00	0.98	**204**	204.00	0.00	2.18	**204**	204.00	0.00	9.18	**204**	204.00	0.00	30.70
MK04	15 × 8	60	62	64.88	40.82	**60**	63.69	1.99	9.02	**60**	60.20	0.53	2.35	**60**	60.00	0.00	12.58
MK05	15 × 4	172	**172**	172.90	20.23	**172**	173.38	0.56	7.10	173	173.02	0.14	3.70	**172**	172.82	0.39	37.89
MK06	10 × 15	58	65	67.38	27.18	60	62.83	1.06	30.21	59	60.20	0.97	10.70	**57**	58.64	0.66	98.32
MK07	20 × 5	139	140	142.21	35.29	**139**	141.55	1.07	17.07	**139**	140.12	1.08	3.26	**139**	139.42	0.50	26.38
MK08	20 × 10	523	**523**	523.00	4.65	**523**	523.00	0.00	4.30	**523**	523.00	0.00	11.52	**523**	523.00	0.00	189.41
MK09	20 × 10	307	310	311.29	70.38	**307**	310.35	0.96	91.99	**307**	307.00	0.00	28.94	**307**	307.00	0.00	122.87
MK10	20 × 15	197	214	219.15	89.83	206	211.92	2.59	190.11	202	205.84	1.79	33.44	**198**	201.52	1.33	265.80
MRE (%)			18.66	18.95		16.07	19.24			15.58	16.52			14.67	15.46		

[a] The CPU time on a Pentium IV 1.6 GHz processor in C++.
[b] The CPU time on an Intel Core i5 3.2 GHz processor in C++.
[c] The CPU time on an Intel 2.83 GHz Xeon processor in Java.

Table 6
Computational results of HDE algorithms on BCdata.

Instance	$n \times m$	HDE-N_1				HDE-N_2			
		Best	AVG	SD	CPU$_{av}$	Best	AVG	SD	CPU$_{av}$
mt10x	10 × 11	**918**	922.86	6.11	21.43	**918**	918.58	2.20	179.22
mt10xx	10 × 12	**918**	922.04	6.31	21.70	**918**	918.38	1.90	179.84
mt10xxx	10 × 13	**918**	919.94	3.96	23.05	**918**	918.00	0.00	179.39
mt10xy	10 × 12	**905**	906.52	1.09	22.51	**905**	905.56	0.79	169.77
mt10xyz	10 × 13	**847**	856.80	3.99	21.79	**847**	851.14	4.65	160.24
mt10c1	10 × 11	**927**	928.92	1.96	21.07	**927**	927.72	0.45	174.19
mt10cc	10 × 12	910	913.92	3.40	21.00	**908**	910.60	2.40	165.61
setb4x	15 × 11	**925**	931.50	2.48	33.04	**925**	925.82	2.11	338.30
setb4xx	15 × 12	**925**	930.38	3.29	29.76	**925**	925.64	1.98	336.24
setb4xxx	15 × 13	**925**	931.42	3.59	29.89	**925**	925.48	1.68	353.55
setb4xy	15 × 12	**910**	921.38	4.44	31.13	**910**	914.00	3.50	330.18
setb4xyz	15 × 13	905	913.40	4.21	30.39	**903**	905.28	1.16	314.64
setb4c9	15 × 11	**914**	919.32	2.87	32.19	**914**	917.12	2.52	313.02
setb4cc	15 × 12	909	912.58	3.81	32.00	**907**	909.58	1.89	316.89
seti5x	15 × 16	1204	1215.48	5.36	73.20	**1200**	1205.64	3.43	1112.77
seti5xx	15 × 17	1202	1205.66	2.56	72.52	**1197**	1202.68	2.02	1078.60
seti5xxx	15 × 18	1202	1206.10	3.18	72.07	**1197**	1202.26	2.37	1087.12
seti5xy	15 × 17	1138	1146.86	5.04	78.98	**1136**	1137.98	2.82	1250.62
seti5xyz	15 × 18	1130	1137.44	3.42	80.85	**1125**	1129.76	2.44	1244.22
sei5c12	15 × 16	1175	1182.54	7.62	69.06	**1171**	1175.42	1.63	1141.43
seti5cc	15 × 17	1137	1145.62	5.58	78.83	**1136**	1137.76	2.48	1222.53
MRE (%)		22.55	23.27			22.39	22.67		

known solutions for the BCdata instances. Oddi et al. (2011) developed an IFS procedure that can achieve the state-of-the-art performance on BCdata.

Our proposed HDE algorithms are also evaluated on this data set, and the detail computational results are reported in Table 6. As can be seen from Table 6, HDE-N_2 is more effective than HDE-N_1. In detail, HDE-N_2 improves 10 best results and all the average results obtained by HDE-N_1 for all 21 instances. Given the SD values, HDE-N_2 is more robust than HDE-N_1, too. Nevertheless, the computational time of HDE-N_2 is much longer than that of HDE-N_1.

To demonstrate the effectiveness and efficiency of HDE algorithms, we compare the best makespan and the average computational time obtained by HDE-N_1 and HDE-N_2 with two aforesaid state-of-the-art algorithms, which are TSBM^{2}h of Bożejko et al. (2010) and IFS of Oddi et al. (2011). The comparison results are shown in Table 7. For the IFS algorithm, its performance depends on the relaxing factor γ, the table lists the results obtained by running IFS with $\gamma = 0.2$ to $\gamma = 0.7$ respectively, and a maximum CPU time limit of 3200 s is set for each run. From Table 7, we note that HDE-N_2 outperforms all the other three algorithms in terms of quality of solutions. Indeed, HDE-N_2 confirms 19 best known solutions among 21 instances and even finds a new best known solution for the instance seti5c12 (improved from 1174 to 1171). HDE-N_2 is only dominated by TSBM^{2}h on only one instance namely seti5x, and outperforms TSBM^{2}h in 4 instances out of 21 instances. When all the 21 instances are considered, the MRE of HDE-N_2 is 22.39%, faced to 22.45% for TSBM^{2}h, 22.55% for HDE-N_1, 23.09% for IFS ($\gamma = 0.7$). As for the efficiency, HDE-N_2 seems to be more efficient than IFS and is also compared favorably with TSBM^{2}h considering the much more advanced computing hardware used by TSBM^{2}h. HDE-N_1 spends much less computational time than the other three algorithms, but it is still very effective. In fact, HDE-N_1 matches 11 best known solutions among all 21 instances, while IFS totally matches 9 best known ones under different settings of γ.

5.5. Results of HUdata instances

The HUdata is another well known data set for the FJSP. In the literature, TS of Mastrolilli and Gambardella (2000) and hGA of Gao et al. (2008) are two algorithms that exhibit the state-of-the-art performance on this data set. In Table 8, we give the MRE

Table 7
Comparison between the proposed HDE algorithms with TSBM^{2}h and IFS on BCdata.

Instance	$n \times m$	BKS	TSBM^{2}h[a]		IFS[b]						HDE-N_1[c]		HDE-N_2[c]	
			Best	CPU$_{av}$	0.2	0.3	0.4	0.5	0.6	0.7	Best	CPU$_{av}$	Best	CPU$_{av}$
mt10x	10×11	918	922	55.11	980	936	936	934	**918**	**918**	**918**	21.43	**918**	179.22
mt10xx	10×12	918	**918**	50.18	936	929	936	933	**918**	926	**918**	21.70	**918**	179.84
mt10xxx	10×13	918	**918**	47.57	936	929	936	926	926	926	**918**	21.43	**918**	179.39
mt10xy	10×12	905	**905**	76.26	922	923	923	915	**905**	909	**905**	22.51	**905**	169.77
mt10xyz	10×13	847	849	110.13	878	858	851	862	**847**	851	**847**	21.79	**847**	160.24
mt10c1	10×11	927	**927**	44.50	943	937	986	934	934	**927**	**927**	21.07	**927**	174.19
mt10cc	10×12	908	**908**	65.74	926	923	919	919	910	911	910	21.00	**908**	165.61
setb4x	15×11	925	**925**	93.76	967	945	930	**925**	937	937	**925**	33.04	**925**	338.30
setb4xx	15×12	925	**925**	92.28	966	931	933	**925**	937	929	**925**	29.76	**925**	336.24
setb4xxx	15×13	925	**925**	89.405	941	930	950	950	942	935	**925**	29.89	**925**	353.55
setb4xy	15×12	910	**910**	150.83	**910**	941	936	936	916	914	**910**	31.13	**910**	330.18
setb4xyz	15×13	903	**903**	152.67	928	909	905	905	905	905	905	30.39	**903**	314.64
setb4c9	15×11	914	**914**	111.40	926	937	926	926	920	920	**914**	32.19	**914**	313.02
setb4cc	15×12	907	**907**	151.19	929	917	**907**	914	**907**	909	909	32.00	**907**	316.89
seti5x	15×16	1198	**1198**	257.75	1210	1199	1199	1205	1207	1209	1204	73.20	1200	1112.77
seti5xx	15×17	1197	**1197**	264.58	1216	1199	1205	1211	1207	1206	1202	72.52	**1197**	1078.60
seti5xxx	15×18	1197	**1197**	226.29	1205	1206	1206	1199	1206	1206	1202	72.07	**1197**	1087.12
seti5xy	15×17	1136	**1136**	675.40	1175	1171	1175	1166	1156	1148	1138	78.98	**1136**	1250.62
seti5xyz	15×18	1125	1128	717.60	1165	1149	1130	1134	1144	1131	1130	80.85	**1125**	1244.22
sei5c12	15×16	1174	1174	351.32	1196	1209	1200	1198	1198	1175	1175	69.06	**1171**	1141.43
seti5cc	15×17	1136	**1136**	670.35	1177	1155	1162	1166	1138	1150	1137	78.83	**1136**	1222.53
MRE (%)			22.45		25.48	24.25	24.44	23.96	23.28	23.09	22.55		22.39	

[a] The CPU time on the nVidia Tesla C870 GPU (512 GFLOPS) with 128 streaming processors cores in C.
[b] The CPU time on an AMD Phenom II X4 Quad 3.5 GHz processor in Java.
[c] The CPU time on an Intel 2.83 GHz Xeon processor in Java.

of the best makespan and average makespan obtained by HDE-N_1 and HDE-N_2 on HUdata, and compare them with the results of TS and hGA. Globally, it can be seen that HDE-N_2 outperforms TS and hGA on all the three subsets of HUdata in terms of the best makespan obtained. As for the average makespan obtained, hGA seems to be the best one among the four algorithms. Besides, TS, hGA and HDE-N_2 all work better than HDE-N_1 in terms of quality of solutions.

Also, it is encouraging that our proposed HDE-N_2 identifies 22 new best known solutions for HUdata instances (16 instances from Rdata, 6 instances from Vdata), among which five solutions are optimal ones. In Table 9, we record both the previous and our newly obtained best known solutions for these instances.

In Table 10, we summarize the MRE of the best makespan obtained by the proposed HDE algorithms and other known algorithms in the literature, and all these algorithms are ranked on each data set (Edata, Rdata, Vdata) according to this metric. From Table 10, it can be concluded that the proposed HDE-N_1 and HDE-N_2 are both extremely effective on HUdata. Indeed, among all the 11 referred algorithms, HDE-N_1 is the fifth rank both on Edata and Vdata and is the fourth rank on Rdata; HDE-N_2 ranks first on all the three subsets of HUdata.

Table 8
Comparison of the proposed HDE algorithms with TS and hGA in the MRE on HUdata.

Instance	$n \times m$	Edata				Rdata				Vdata			
		TS	hGA	HDE-N_1	HDE-N_2	TS	hGA	HDE-N_1	HDE-N_2	TS	hGA	HDE-N_1	HDE-N_2
mt06/10/20	6×6 10×10 20×5	**0.00** (0.10)	**0.00** (0.10)	0.05 (0.13)	**0.00** (0.07)	**0.34** (0.36)	**0.34** (0.34)	**0.34** (0.45)	**0.34** (0.34)	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.01)	**0.00** (0.00)
la01-la05	10×5	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.00)	0.11 (0.24)	0.07 (0.07)	0.11 (0.31)	**0.04** (0.10)	**0.00** (0.11)	**0.00** (0.00)	0.04 (0.19)	**0.00** (0.01)
la06-la10	15×5	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.10)	**0.00** (0.00)	0.03 (0.08)	**0.00** (0.00)	0.05 (0.10)	**0.00** (0.01)	**0.00** (0.03)	**0.00** (0.00)	0.03 (0.10)	**0.00** (0.00)
la11-la15	20×5	**0.29** (0.29)	**0.29** (0.29)	**0.29** (0.29)	**0.29** (0.29)	0.02 (0.02)	**0.00** (0.00)	**0.00** (0.02)	**0.00** (0.00)	**0.00** (0.01)	**0.00** (0.00)	**0.00** (0.01)	**0.00** (0.00)
la16-la20	10×10	**0.00** (0.00)	0.02 (0.02)	0.02 (0.48)	0.02 (0.02)	**1.64** (1.68)	**1.64** (1.64)	**1.64** (1.69)	**1.64** (1.69)	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.00)
la21-la25	15×10	5.62 (5.93)	5.60 (5.66)	5.82 (6.41)	**5.46** (5.91)	3.82 (4.38)	3.57 (3.69)	3.73 (4.57)	**3.13** (3.66)	0.70 (0.85)	0.60 (0.68)	1.63 (2.15)	**0.57** (0.96)
la26-la30	20×10	3.47 (3.76)	3.28 (3.32)	3.89 (4.71)	**3.11** (3.64)	**0.59** (0.76)	0.64 (0.72)	1.04 (1.41)	0.60 (0.81)	0.11 (0.18)	0.11 (0.13)	0.42 (0.63)	**0.10** (0.19)
la31-la35	30×10	**0.30** (0.32)	0.32 (0.32)	0.50 (0.59)	0.32 (0.39)	0.09 (0.14)	0.09 (0.12)	0.22 (0.33)	**0.08** (0.13)	0.01 (0.03)	**0.00** (0.00)	0.12 (0.18)	0.03 (0.03)
la36-la40	15×15	8.99 (9.13)	**8.82** (8.95)	9.63 (10.43)	8.89 (9.36)	3.97 (4.47)	3.86 (3.92)	3.98 (4.92)	**3.38** (4.19)	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.01)	**0.00** (0.00)
MRE (%)		2.17 (2.27)	2.13 (2.17)	2.35 (2.68)	**2.11** (2.29)	1.24 (1.36)	1.19 (1.21)	1.28 (1.59)	**1.05** (1.26)	0.095 (0.13)	0.082 (0.09)	0.26 (0.38)	**0.080** (0.14)

Table 9
The makespan of new best known solutions identified by the proposed HDE-N_2 on HUdata.

Instance	Data set	LB	Prev. best known	New best known
la01	Rdata	570	571	570
la03	Rdata	477	478	477
la07	Rdata	749	750	749
la15	Rdata	1089	1090	1089
la21	Rdata	808	835	833
la22	Rdata	737	760	758
la23	Rdata	816	842	832
la24	Rdata	775	808	801
la25	Rdata	752	791	785
la27	Rdata	1085	1091	1090
la29	Rdata	993	998	997
la33	Rdata	1497	1499	1498
la36	Rdata	1016	1030	1028
la37	Rdata	989	1077	1066
la38	Rdata	943	962	960
la40	Rdata	955	970	956
la21	Vdata	800	806	805
la22	Vdata	733	739	735
la23	Vdata	809	815	813
la26	Vdata	1052	1054	1053
la27	Vdata	1084	1085	1084
la30	Vdata	1068	1070	1069

5.6. Further performance analysis of HDE

5.6.1. Significance tests between HDE algorithms

In order to show whether the two proposed HDE algorithms have significant performance differences for different problems, statistical analysis is further carried out. Since the obtained makespan values may present neither normal distribution nor homogeneity of variance, the non-parametric tests are considered to be used according to the recommendations made in Demšar (2006). Specifically, the Wilcoxon signed-rank test, a pairwise non-parametric statistical test, is adopted to check whether there are significant differences in the optimization effects of both algorithms on each problem instance. The results are summarized in Table 11. The first column reports the data set; the second column shows the number of instances for each class; in the third column, we list instances on which HDE-N_2 is statically better than HDE-N_1, with a significance level of 0.05. From the significance tests, it seems that HDE-N_1 is enough for solving some small-scale or relative easy problems, for example, there are no statistical significant differences in obtained makespan values between HDE-N_1 and HDE-N_2 on instances mt06 and mt10 in Hurink Edata. HDE-N_2 appears to be more adaptive to handle large-scale or hard problems, for instance, it is more possible for HDE-N_2 to find higher quality solutions to all the problems in BCdata. However, as mentioned before, HDE-N_2 requires much more computation effort because of a larger neighborhood structure used in its embedded local search.

5.6.2. Influence of parameters on HDE

To obtain a better performance of our HDE, some experiments are conducted to investigate the influence of parameters.

The impact of parameters NP and $G_{\max}$ is firstly considered. NP is increased from 10 to 40 in steps of 10, while $G_{\max}$ is increased from 100 to 250 in steps of 50. The other parameters are fixed according to Table 3 and the algorithm is run 50 times under each parameter setting. The results got on the instance MK06 are displayed in Table 12. As can be seen from Table 12, the increase of NP or $G_{\max}$ is beneficial to the quality of solutions at the beginning. However, there seems to be little effect to increase NP or $G_{\max}$ further as they reach certain values, sometimes the obtained results even get worse. For example, when $NP = 30$, the performance of HDE-N_2 is not improved with the increasing of $G_{\max}$ from 200 to 250. Moreover, no matter the increase of NP or $G_{\max}$ will increase the computational effort.

In Tables 13 and 14, the influences of parameters P_l and $iter_{\max}$, F and Cr are reported for the instance MK06, respectively. From Table 13, P_l and $iter_{\max}$ have the impact on HDE similar to NP and $G_{\max}$. As for F and $C_{\max}$, many settings of them could keep the good performance of HDE, but there also exist some settings resulting in relatively poor performance. Take MK06 for example, the setting of $F = 0.1, Cr = 0.3$ seems to be an ideal choice for HDE, while $F = 0.7$, $Cr = 0.7$ is not recommended.

We also carry out the experiments on some other instances to observe the influence of parameters. Due to the space limitations, the results will no longer be listed here. Overall, NP and $G_{\max}$ (P_l and $iter_{\max}$) should be set suitably to balance the optimization

Table 10
Summary results of the MRE of the best makespan obtained by the proposed HDE algorithms and other known algorithms in the literature.

Algorithm	Reference	Edata		Rdata		Vdata	
		MRE (%)	Rank	MRE (%)	Rank	MRE (%)	Rank
HDE-N_1	This study	2.35	5	1.28	4	0.26	5
HDE-N_2	This study	2.11	1	1.05	1	0.080	1
TS	Hurink et al. (1994)	4.50	7	2.30	7	0.40	6
GA	Chen et al. (1999)	5.59	8	4.41	9	2.59	10
TS	Mastrolilli and Gambardella (2000)	2.17	3	1.24	3	0.095	3
GA	Jia et al. (2003)	9.01	11	8.34	11	3.24	11
hGA	Gao et al. (2008)	2.13	2	1.19	2	0.082	2
GA	Pezzella et al. (2008)	6.00	9	4.42	10	2.04	9
AIA	Bagheri et al. (2010)	6.83	10	3.98	8	1.29	8
PVNS	Yazdani et al. (2010)	3.86	6	1.88	6	0.42	7
CDDS	Ben Hmida et al. (2010)	2.32	4	1.34	5	0.12	4

Table 11
Summary results of significance tests between HDE-N_1 and HDE-N_2 on each data set. The level of significance α is set to 0.05.

Data set	Num	Instances on which HDE-N_2 is significantly better than HDE-N_1
Kacem data	5	Case 5
BRdata	10	MK02, MK05, MK06, MK07, MK10
BCdata	21	All the instances
Hurink Edata	43	mt20, la22, la24-la31, la34, la36-la40
Hurink Rdata	43	mt10, mt20, la01-la03, la06-la08, la10, la15, la21-la35, la37, la39, la40
Hurink Vdata	43	la01-la03, la06-la10, la15, la21-la35

Table 12
Influence of parameters NP and $G_{\max}$ on HDE.

NP	$G_{\max}$ = 100						$G_{\max}$ = 150						$G_{\max}$ = 200						$G_{\max}$ = 250					
	HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2		
	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}
10	61.68	1.11	1.87	60.02	1.10	16.53	61.38	1.01	2.68	59.62	0.88	24.35	61.14	1.07	3.38	59.30	0.79	31.66	60.68	0.87	4.14	59.58	0.81	38.64
20	61.02	0.87	3.90	59.36	0.72	32.93	60.70	0.81	5.48	59.20	0.76	50.16	60.76	0.96	7.18	59.14	0.78	65.53	60.32	1.00	8.64	58.78	0.74	80.55
30	60.98	0.98	5.91	59.04	0.67	49.33	60.38	0.83	8.36	58.78	0.71	75.61	60.20	0.95	10.56	58.64	0.66	98.32	60.20	1.01	12.80	58.70	0.65	123.35
40	60.64	0.88	7.67	59.08	0.80	66.82	60.22	0.79	11.40	58.80	0.61	97.78	60.02	0.65	13.95	58.58	0.57	130.62	59.62	0.85	17.44	58.54	0.61	164.79

Table 13
Influence of parameters P_l and $iter_{\max}$ on HDE.

P_l	$iter_{\max}$ = 10						$iter_{\max}$ = 40						$iter_{\max}$ = 70						$iter_{\max}$ = 100					
	HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2		
	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}
0.1	66.30	2.13	2.26	64.18	1.87	3.09	62.44	1.31	3.29	60.68	1.04	11.18	62.08	0.78	3.45	60.28	0.76	15.37	62.18	1.16	3.49	60.24	0.85	16.24
0.3	64.38	1.92	3.05	62.52	1.37	7.20	61.12	1.10	5.51	59.46	0.93	33.36	60.76	0.94	5.99	59.24	0.82	43.50	61.00	0.86	6.26	59.18	0.72	45.31
0.5	63.30	1.66	4.03	61.92	1.51	12.61	60.72	0.93	7.26	59.34	0.77	56.67	60.44	0.73	8.11	58.88	0.85	70.86	60.46	0.76	8.50	58.92	0.70	71.99
0.7	63.00	1.40	4.89	61.80	1.51	17.19	60.38	0.92	9.48	59.12	0.90	80.25	59.92	0.80	10.48	58.76	0.66	97.60	59.94	0.77	10.80	58.70	0.58	100.39
0.9	62.60	1.60	5.70	61.78	1.49	22.00	60.24	0.80	11.32	58.98	0.89	104.24	60.04	0.73	12.67	58.64	0.69	127.30	59.94	0.84	12.83	58.54	0.58	128.02

Table 14
Influence of parameters F and Cr on HDE.

F	Cr = 0.1						Cr = 0.3						Cr = 0.5						Cr = 0.7					
	HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2			HDE-N_1			HDE-N_2		
	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}	AVG	SD	CPU_{av}
0.1	61.04	0.57	9.53	59.34	0.48	92.60	60.20	0.97	10.70	58.64	0.66	98.32	60.48	1.09	7.73	59.38	0.88	82.04	62.74	1.32	5.30	60.76	1.27	70.96
0.3	60.68	0.62	12.24	59.26	0.69	99.96	60.68	0.77	17.79	58.72	0.73	110.41	60.58	0.88	19.98	59.42	0.84	105.02	60.88	1.12	19.64	59.50	1.02	100.77
0.5	60.94	0.65	12.66	59.16	0.55	101.86	61.18	0.77	19.67	59.38	0.60	108.57	61.74	0.72	22.82	59.56	0.73	96.37	62.08	1.19	24.21	60.12	0.87	84.88
0.7	60.94	0.62	13.92	59.06	0.51	104.67	61.90	0.68	21.91	59.94	0.47	102.15	63.04	0.83	25.22	60.90	0.79	75.54	64.18	1.24	25.87	62.20	0.99	49.57

Table 15
Comparison of DE, MRLS, and HDE algorithms.

Data set	Num	DE		MRLS-N_1		MRLS-N_2		HDE-N_1		HDE-N_2	
		MRE_b (%)	MRE_{av} (%)	MRE_b (%)	MRE_{av} (%)	MRE_b (%)	MRE_{av} (%)	MRE_b (%)	MRE_{av} (%)	MRE_b (%)	MRE_{av}
BRdata	10	34.47	42.98	36.63	41.44	34.24	39.87	15.58	16.52	14.67	15.46
BCdata	21	24.76	28.53	25.52	27.78	23.78	25.37	22.55	23.27	22.39	22.67
Hurink Edata	43	5.26	6.79	4.70	5.42	3.70	4.28	2.35	2.68	2.11	2.29
Hurink Rdata	43	7.51	9.92	5.28	5.94	4.55	5.18	1.28	1.59	1.05	1.26
Hurink Vdata	43	5.50	7.86	4.18	5.08	3.77	4.79	0.26	0.38	0.080	0.14

effect and computation effort, and larger problems usually deserve larger values of them. But for certain problem instance, they are not simply the larger the better. The performance of HDE is not very sensitive to F and Cr. But we find that $F = 0.5$, $Cr = 0.1$ seems to be more effective for the problems with a lower degree of flexibility (usually less than 1.5), while the setting of $F = 0.1$, $Cr = 0.3$ appears to be better for others. The flexibility means the average number of alternative machines for each operation in the problem.

5.6.3. Effect of hybridizing DE and local search algorithms

To investigate the effectiveness of hybridizing DE-based global search and local search algorithms, the experiments and comparisons are carried out between the HDE algorithms with DE and multi-start random local search (MRLS) algorithms.

The DE algorithm is formed by removing the local search procedure directly from the HDE. The MRLS is designed by replacing the DE operators in the HDE with random generating method to produce new solutions. Specifically, the MRLS works as follows: a solution is randomly generated every time, then the local search is applied to it with certain probability; this procedure is repeated until maximum replications ($R_{\max}$) are reached. Corresponding to the HDE, the MRLS also has two variants: MRLS-N_1 and MRLS-N_2.

In Table 15, we report the performance of DE, MRLS and HDE algorithms in terms of MRE of the best makespan (MRE_b) and average makespan (MRE_{av}) obtained. The parameters of DE and MRLS are consistent with those of the HDE. For the MRLS, its unique parameter $R_{\max}$ is set as $NP \times G_{\max}$ in order to make a fair comparing with the HDE. From Table 15, it is easily observed that the results generated by HDE-N_1 (or HDE-N_2) is obviously better than those by DE and MRLS-N_1 (or MRLS-N_2) on each data set. To better show the superiority of hybridization, the typical convergence rate curves for these algorithms based on the instance la40 in Hurink Edata are depicted in Fig. 8. It can be seen from Fig. 8 that the HDE algorithms could converge fast to lower makespan values

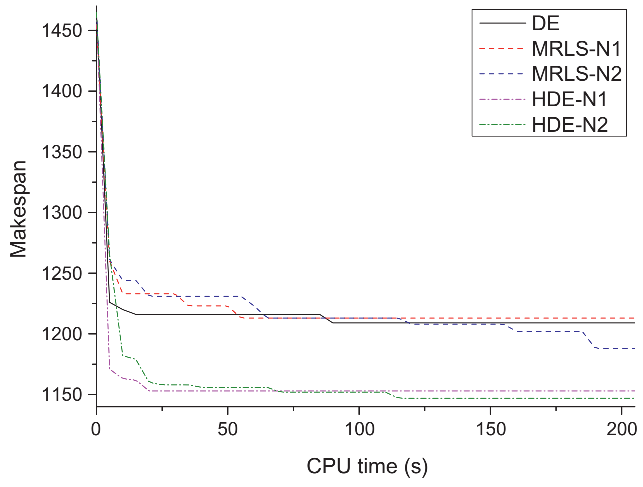

Fig. 8. Convergence rate curve of the DE, MRLS, and HDE algorithms on the instance la40 in Hurink Edata.

Table 16
Results of P_{DE} and P_{LS} on each data set.

Data set	Num	HDE-N_1		HDE-N_2	
		P_{DE} (%)	P_{LS} (%)	P_{DE} (%)	P_{LS} (%)
Kacem data	5	24.01	75.99	8.90	91.10
BRdata	10	18.08	81.82	2.39	97.61
BCdata	21	15.11	84.89	1.16	98.84
Hurink Edata	43	11.44	88.56	1.27	98.73
Hurink Rdata	43	6.49	93.51	1.25	98.75
Hurink Vdata	43	4.12	95.88	1.32	98.68

Table 17
Comparison of pure DE, GA of Chen, GA of Jia and GA of Pezzella on BRdata.

Instance	$n \times m$	DE	GA_Chen	GA_Jia	GA_Pezzella
MK01	10×6	**40**	**40**	**40**	**40**
MK02	10×6	27	29	28	**26**
MK03	15×8	**204**	**204**	**204**	**204**
MK04	15×8	**60**	63	61	**60**
MK05	15×4	**173**	181	176	**173**
MK06	10×15	63	**60**	62	63
MK07	20×5	144	148	145	**139**
MK08	20×10	**523**	**523**	**523**	**523**
MK09	20×10	311	**308**	310	311
MK10	20×15	223	212	216	212
MRE (%)		18.99	19.55	19.11	17.53

than DE and MRLS algorithms. The conclusion holds similarly for other benchmark instances.

Based on the above results and comparisons, it is concluded that the HDE yields superior performance to its individual components (DE and local search). Its success could be attributed that the DE provides good start points for local search algorithm by performing the global exploration, while the local search further improves these obtained solutions through conducting local exploitation, and guides the DE to more promising search space. That is, the HDE integrates the advantages of DE for diversification and local search for intensification, which well achieves the balance between exploration and exploitation.

In Table 16, the relation of computational efforts between DE and local search in the HDE are recorded, where P_{DE} and P_{LS} denote as the percentage of DE and local search occupy respectively. As can be seen from Table 16, the local search needs most of the computational cost, especially in the HDE-N_2. It is not surprising because each iteration of local search is much more computational expensive than the evaluation of the vectors.

5.6.4. Performance potential of pure DE algorithm

The emphasis of this paper is on the hybrid algorithms, but here we would like to roughly show the performance potential of pure DE algorithm. In Table 15, some results of DE have ever been reported, where the parameters of DE there are set according to the HDE that is only to demonstrate the effectiveness of hybridization. However, the parameters of HDE are not so suitable for the pure DE. To investigate how well the performance of pure DE can achieve, NP and G_{max} are reset as 500 and 10,000; the resulting number of $NP \times G_{max} = 5 \times 10^6$ objective function evaluations equals to the one adopted in Pezzella et al. (2008), where the population size and number of generations are set as 5000 and 1000 respectively. In Table 17, we compare the best makespan values obtained on BRdata by our DE with those by three pure GA algorithms, which are GA of Chen et al. (1999), GA of Jia et al. (2003) and GA of Pezzella et al. (2008). From Table 17, the overall results of DE are just a little worse than those got by GA of Pezzella et al. (2008). But it should be noted that different strategies such as problem-dependent initialization and intelligent mutation are integrated in GA of Pezzella et al. (2008), while our DE only employs the basic DE operators to explore the problem space.

6. Conclusions

This paper presented hybrid differential evolution (HDE) algorithms for solving the FJSP with the makespan criterion, which has a significant application value in modern manufacturing environments. With a novel conversion mechanism, the DE algorithm that works on the continuous domain is adapted to explore the problem space of the discrete FJSP. It is worth noting that this conversion mechanism is appropriate not only for the DE but also for other continuous evolutionary algorithms to deal with the FJSP, such as harmony search (HS) (Geem and Kim, 2001) and artificial bee colony (ABC) (Karaboga and Basturk, 2007) algorithms. To enhance the intensification search and to balance the exploration and exploitation, a well developed local search algorithm based on the critical path is incorporated in the framework of DE. Furthermore, two neighborhood structures are presented in the local search, and the efficiency is stressed by using a speed-up method to find an acceptable schedule more quickly within the neighborhood. Computational results and comparisons demonstrate that, the proposed HDE-N_1 is especially effective and efficient for the FJSP and outperforms several recently proposed algorithms; HDE-N_2 could obtain higher quality solutions than HDE-N_1 and is compared favorably with the state of the art, some best known solutions for well known benchmark instances have even been further improved by HDE-N_2. The future work is to develop multi-objective HDE algorithms for the multi-objective FJSP and apply the DE algorithm to other kinds of combinational optimization problems.

Acknowledgements

This research is supported by National Natural Science Foundation of China (Grant No. 61175110), National S&T Major Projects of China (Grant No. 2011ZX02101-004) and National Basic Research Program of China (973 Program) (Grant No. 2012CB316300).

Appendix A

The details of all the new best known solutions obtained by HDE-N_2 for benchmark instances (1 BRdata instance, 1 BCdata instance, 22 HUdata instances) are available online:
http://www.166.111.4.17:8080/2012310563/NBKS.rar.

References

Bagheri, A., Zandieh, M., Mahdavi, I., & Yazdani, M. (2010). An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Generation Computer Systems, 26*(4), 533–541.

Barnes, J. W., & Chambers, J. B. (1996). Flexible job shop scheduling by tabu search. Graduate program in operations research and industrial engineering, The University of Texas at Austin, Technical Report Series, ORP96-09.

Ben Hmida, A., Haouari, M., Huguet, M., & Lopez, P. (2010). Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research, 37*(12), 2192–2201.

Bożejko, W., Uchroński, M., & Wodecki, M. (2010). Parallel hybrid metaheuristics for the flexible job shop problem. *Computers & Industrial Engineering, 59*(2), 323–333.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*(3), 157–183.

Chen, H., Ihlow, J., & Lehmann, C. (1999). A genetic algorithm for flexible job-shop scheduling. *IEEE International Conference on Robotics and Automation* (Vol. 2, pp. 1120–1125). IEEE.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. *Computers & Industrial Engineering, 30*(4), 983–997.

Damak, N., Jarboui, B., Siarry, P., & Loukil, T. (2009). Differential evolution for solving multi-mode resource-constrained project scheduling problems. *Computers & Operations Research, 36*(9), 2653–2659.

Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research, 70*(0), 281–306.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research, 7*, 1–30.

Fatih Tasgetiren, M., Suganthan, P., & Pan, Q. (2010). An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem. *Applied Mathematics and Computation, 215*(9), 3356–3368.

Fisher, H., & Thompson, G. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling*, 225–251.

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research, 35*(9), 2892–2907.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*, 117–129.

Geem, Z., Kim, J., et al. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation, 76*(2), 60–68.

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spectrum, 15*(4), 205–215.

Ilonen, J., Kamarainen, J., & Lampinen, J. (2003). Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters, 17*(1), 93–105.

Jia, H., Nee, A., Fuh, J., & Zhang, Y. (2003). A modified genetic algorithm for distributed scheduling problems. *Journal of Intelligent Manufacturing, 14*(3), 351–362.

Jurisch, B. (1992). Scheduling jobs in shops with multi-purpose machines. Thesis PhD. Fachbereich Mathematik/Informatik, Universitat Osnabruck

Kacem, I., Hammadi, S., & Borne, P. (2002a). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 32*(1), 1–13.

Kacem, I., Hammadi, S., & Borne, P. (2002b). Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation, 60*(3-5), 245–276.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm. *Journal of Global Optimization, 39*(3), 459–471.

Kazemipoor, H., Tavakkoli-Moghaddam, R., Shahnazari-Shahrezaei, P., & Azaron, A. (2012). A differential evolution algorithm to solve multi-skilled project portfolio scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 1–13.

Lawrence, S. (1984). *Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*. Pittsburg, PA: GSIA, Carnagie Mellon University.

Li, J., Pan, Q., Suganthan, P., & Chua, T. (2011). A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 52*(5), 683–697.

Mastrolilli, M., & Gambardella, L. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling, 3*(1), 3–20.

Moslehi, G., & Mahnam, M. (2011). A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. *International Journal of Production Economics, 129*(1), 14–22.

Nasiri, M., & Kianfar, F. (2012). A GES/TS algorithm for the job shop scheduling. *Computers & Industrial Engineering, 62*(4), 946–952.

Noman, N., & Iba, H. (2008). Differential evolution for economic load dispatch problems. *Electric Power Systems Research, 78*(8), 1322–1331.

Oddi, A., Rasconi, R., Cesta, A., & Smith, S. (2011). Iterative flattening search for the flexible job shop scheduling problem. In *Proceedings of the 22th international joint conference on artificial intelligence*, Barcelona.

Onwubolu, G., & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research, 171*(2), 674–692.

Pacino, D., & Van Hentenryck, P. (2011). Large neighborhood search and adaptive randomized decompositions for flexible jobshop scheduling. In *Proceedings of the 22th international joint conference on artificial intelligence*, Barcelona.

Panduro, M., Brizuela, C., Balderas, L., & Acosta, D. (2009). A comparison of genetic algorithms, particle swarm optimization and the differential evolution method for the design of scannable circular antenna arrays. *Progress in Electromagnetics Research B, 13*, 171–186.

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research, 35*(10), 3202–3212.

Pinedo, M. (2002). *Scheduling: Theory, algorithms, and systems*. Englewood Cliffs, NJ: Prentice-Hall.

Qian, B., Wang, L., Hu, R., Huang, D., & Wang, X. (2009). A de-based approach to no-wait flow-shop scheduling. *Computers & Industrial Engineering, 57*(3), 787–805.

Qian, B., Wang, L., Hu, R., Wang, W., Huang, D., & Wang, X. (2008). A hybrid differential evolution method for permutation flow-shop scheduling. *The International Journal of Advanced Manufacturing Technology, 38*(7), 757–777.

Sha, D., & Hsu, C. (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering, 51*(4), 791–808.

Storn, R. (1999). Designing digital filters with differential evolution. In *New ideas in optimization* (pp. 109–126). UK: McGraw-Hill Ltd..

Storn, R., & Price, K. (1997). Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.

Wang, L., Pan, Q., & Fatih Tasgetiren, M. (2010). Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. *Expert Systems with Applications, 37*(12), 7929–7936.

Wang, L., Wang, S., Xu, Y., Zhou, G., & Liu, M. (2012). A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. *Computers & Industrial Engineering, 62*(4), 917–926.

Wang, L., Zhou, G., Xu, Y., Wang, S., & Liu, M. (2011). An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 1–13.

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering, 48*(2), 409–425.

Xing, L., Chen, Y., Wang, P., Zhao, Q., & Xiong, J. (2010). A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing, 10*(3), 888–896.

Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications, 37*(1), 678–687.

Zhang, C., Li, P., Rao, Y., & Guan, Z. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research, 35*(1), 282–294.

Zhang, G., Shao, X., Li, P., & Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering, 56*(4), 1309–1318.

Zhu, Q., Qin, A., Suganthan, P., & Huang, G. (2005). Evolutionary extreme learning machine. *Pattern Recognition, 38*(10), 1759–1763.