Computers & Operations Research 37 (2010) 509 – 520

Contents lists available at ScienceDirect

Computers & Operations Research

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

A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems

Ling Wang[a], Quan-Ke Pan[b,*], P.N. Suganthan[c], Wen-Hong Wang[b], Ya-Min Wang[b]

[a]Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Automation, Tsinghua University, Beijing 100084, China
[b]College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c]School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Ave., 639798 Singapore, Singapore

ARTICLE INFO

Available online 24 December 2008

Keywords:
Blocking flow shop
Differential evolution
Discrete differential evolution
Hybrid algorithm
Local search
Speed-up

ABSTRACT

This paper proposes a novel hybrid discrete differential evolution (HDDE) algorithm for solving blocking flow shop scheduling problems to minimize the maximum completion time (i.e. makespan). Firstly, in the algorithm, the individuals are represented as discrete job permutations, and new mutation and crossover operators are developed for this representation, so that the algorithm can directly work in the discrete domain. Secondly, a local search algorithm based on insert neighborhood structure is embedded in the algorithm to balance the exploration and exploitation by enhancing the local searching ability. In addition, a speed-up method to evaluate insert neighborhood is developed to improve the efficiency of the whole algorithm. Computational simulations and comparisons based on the well-known benchmark instances of Taillard [Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:278–285], by treating them as blocking flow shop problem instances with makespan criterion, are provided. It is shown that the proposed HDDE algorithm not only generates better results than the existing tabu search (TS) and TS with multi-moves (TS + M) approaches proposed by Grabowski and Pempera [The permutation flow shop problem with blocking. A tabu search approach 2007;35:302–311], but also outperforms the hybrid differential evolution (HDE) algorithm developed by Qian et al. [An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. Computers and operations research 2009;36(1):209–233] in terms of solution quality, robustness and search efficiency. Ultimately, 112 out of 120 best known solutions provided by Grabowski and Pempera [The permutation flow shop problem with blocking. A tabu search approach 2007;35:302–311] and Ronconi [A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking. Annals of Operations Research 2005;138(1):53–65] are further improved by the proposed HDDE algorithm.

1. Introduction

Flow shop scheduling problem is one of the most popular machine scheduling problems with extensive engineering relevance, representing nearly a quarter of manufacturing systems, assembly lines, and information service facilities in use nowadays [1–8]. This paper considers flow shop scheduling problems with blocking constraint. In blocking flow shops, there are no buffers between machines and hence intermediate queues of jobs waiting in the production system for their next operations are not allowed. Therefore, when needed, a job having completed processing on a machine has to remain on this machine and block itself until next machine is available for processing. The blocking flow shop scheduling problems have important applications in the production environment where processed jobs are sometimes kept in the machines because of the lack of intermediate storage [9], or stock is not allowed in some stages of the manufacturing process because of technological requirements [10]. On the other hand, it has been proved that the blocking flow shop scheduling problem to minimize makespan with three machines is NP-hard in the strong sense [11]. Therefore, it is of significance both in theory and in engineering applications to develop effective and efficient novel solution procedures to solve such problems.

However, blocking flow shop scheduling problems have not captured enough research so far [2,12]. Among the research, McCormich et al. [13] developed a constructive heuristic, known as profile fitting (PF), for solving sequencing problems in an assembly line with blocking to minimize cycle time. In their heuristic, the authors created a partial sequence by adding an unscheduled job to obtain the minimum sum of idle times and blocking times on machines. A more comprehensive approach is presented by Leisten [14] for dealing

* Corresponding author. Tel.: +86 635 8238360; fax: +86 635 8238809.
E-mail address: qkpan@lcu.edu.cn (Q.-K. Pan).

0305-0548/$ - see front matter © 2009 Published by Elsevier Ltd.
doi:10.1016/j.cor.2008.12.004

with permutation and non-permutation flow shops with finite and unlimited buffers to maximize the use of buffers and to minimize the machine blocking. However, the author concluded that the heuristic did not produce better solutions than the Nawaz–Enscore–Ham (NEH) [16] heuristic. Ronconi [9] proposed three constructive heuristics, called minmax (MM), combination of MM and NEH (MME), and combination of PF and NEH (PFE), respectively, for blocking flow shop problems with makespan criterion. It was demonstrated by the authors that the MME and PFE heuristics outperformed the NEH algorithm in problems with up to 500 jobs and 20 machines. Based on the connection between no-wait flow shop scheduling problems and flow shop scheduling problems with blocking, Abadi et al. [17] proposed a heuristic for minimizing cycle time in blocking flow shops. Recently, Ronconi and Henriques [12] studied the minimization of the total tardiness in flow shops with blocking scheduling and presented some constructive heuristics. By computational tests, the authors showed that their new approaches were promising for the problems considered.

With the development of computer technology, a few metaheuristics have been used to solve blocking flow shop scheduling problems. Caraffa [18] developed a genetic algorithmic approach for solving large size restricted slowdown flow shop problems in which blocking flow shop problems were special cases. Ronconi [4] proposed a heuristic algorithm based on branch-and-bound method by using the new lower bounds which exploited the blocking nature and were better than those presented in their earlier paper [15]. By applying some properties of the problems associated with the blocks of jobs as well as using multi-moves to accelerate the convergence to more promising areas of the solutions space, recently, Grabowski and Pempera [2] developed two tabu search (TS) and TS with multi-move (TS+M) approaches. Computational results demonstrated that the TS and TS + M algorithms outperformed both the genetic algorithm [18] and Ronconi's method [4].

The differential evolution (DE) algorithm was first introduced by Storn and Price [19] to optimize complex continuous nonlinear functions. As a population-based evolutionary algorithm, the DE uses simple mutation and crossover operators to generate new candidate solutions, and applies one-to-one competition scheme to greedily decide 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 gained much attention and a wide range of successful applications such as digital PID controller design [20], feed-forward neural networks training [21,35], digital filter design [22] and earthquake hypocenter location [23]. However, because of its continuous nature, applications of the DE algorithm on scheduling problems are still considerably limited [24–31]. Therefore, in this paper, we propose a hybrid discrete DE (HDDE) algorithm for solving blocking flow shop scheduling problems with makespan criterion. In the proposed DDE algorithm, individuals are represented as discrete job permutations, and novel job-permutation-based mutation and crossover operators are employed to generate new candidate solutions, and an effective insert-neighborhood-based local search is embedded to enhance exploitation. Furthermore, a speed-up method for insert neighborhood structure is developed to reduce computational time requirements. Simulation results and comparisons demonstrate the effectiveness of the proposed HDDE algorithm in solving blocking flow shop scheduling problems with makespan criterion.

This paper is organized as follows. In Section 2, the blocking flow shop scheduling problem is stated and formulated. In Section 3, the speed-up method for the insert neighborhood structure is proposed. In Section 4, the discrete DE (DDE) algorithm is proposed in detail. Section 5 presents a HDDE algorithm after explaining an effective local search. The computational results and comparisons are provided in Section 6. Finally, we conclude the paper with some concluding remarks in Section 7.

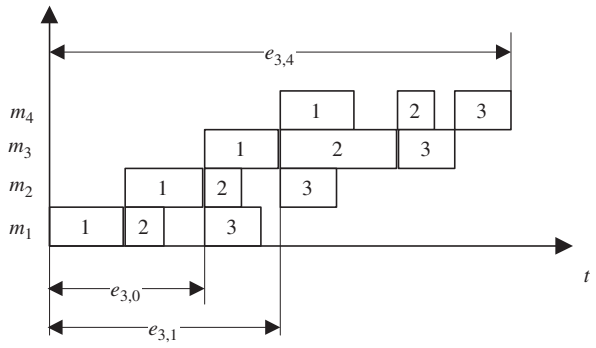

Fig. 1. Computation of $e_{j,k}$.

2. The blocking flow shop scheduling problem

The blocking flow shop scheduling problem can be defined as follows. There are n jobs from the set $J=\{1,2,\ldots,n\}$ and m machines from the set $M=\{1,2,\ldots,m\}$. Each job $j\in J$ will be sequentially processed on machine $1,2,\ldots,m$. Operation $o_{j,k}$ corresponds to the processing of job $j\in J$ on machine k $(k=1,2,\ldots,m)$ during an uninterrupted processing time $p_{j,k}$, where its setup time is included into the processing time. At any time, each machine can process at most one job and each job can be processed on at most one machine. The sequence in which the jobs are to be processed is the same for each machine. Since the flow shop has no intermediate buffers, a job cannot leave a machine until its next machine downstream is free. In other words, the job has to be blocked on its machine if its next machine is not free. The aim is then to find a sequence for processing all jobs on all machines so that its maximum completion time (i.e. makespan) is minimized.

2.1. Mathematical model

Let a job permutation $\pi=\{\pi_1,\pi_2,\ldots,\pi_n\}$ represent the schedule of jobs to be processed, and $e_{j,k}$ be the departure time of operation $o_{j,k}$ (illustrated in Fig. 1). According to the literature [9], $e_{j,k}$ can be calculated as follows

$$e_{1,0}=0, \tag{1}$$

$$e_{1,k}=e_{1,k-1}+p_{1,k},\quad k=1,\ldots,m-1, \tag{2}$$

$$e_{j,0}=e_{j-1,1},\quad j=2,\ldots,n, \tag{3}$$

$$e_{j,k}=\max\{e_{j,k-1}+p_{j,k},\ e_{j-1,k+1}\},\quad j=2,\ldots,n,\ k=1,\ldots,m-1, \tag{4}$$

$$e_{j,m}=e_{j,m-1}+p_{j,m},\quad j=1,\ldots,n, \tag{5}$$

where $e_{j,0}$, $j=1,\ldots,n$, denotes the starting time of job π_j on the first machine. In the above recursion, the departure times of the first job on every machine are calculated first, then the second job, and so on until the last job. Then the makespan of the job permutation $\pi=\{\pi_1,\pi_2,\ldots,\pi_n\}$ is given by $C_{\max}(\pi)=e_{n,m}$, and its computational complexity is $O(mn)$.

Let $f_{j,k}$ be the tail, i.e., duration between the latest loading time of operation $o_{j,k}$ $(k=m,\ldots,1)$ and the end of the operations, and $f_{j,m+1}$ be the duration between the latest completion time of operation $o_{j,m}$ and the end of the operations (illustrated in Fig. 2). To follow the blocking constraints, we can obtain the recursion below

$$f_{n,m+1}=0, \tag{6}$$

$$f_{n,k} = f_{n,k+1} + p_{n,k}, \quad k = m, \ldots, 2, \tag{7}$$

$$f_{j,m+1} = f_{j+1,m}, \quad j = n-1, \ldots, 1, \tag{8}$$

$$f_{j,k} = \max(f_{j,k+1} + p_{j,k},\ f_{j+1,k-1}), \quad j = n-1, \ldots, 1,\ \ k = m, \ldots, 2, \tag{9}$$

$$f_{j,1} = f_{j,2} + p_{j,1}, \quad j = n, \ldots, 1. \tag{10}$$

In the above recursion, the tails of the last job on every machine are calculated first, then the second last job, and so on until the first job. Then, an alternative approach to calculate the makespan of job permutation $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ can be given by $C_{\max}(\pi) = f_{1,1}$ in time $O(mn)$.

Therefore, the objective of the blocking flow shop scheduling problem with makespan criterion is to find a permutation π^* in the set of all permutations Π such that

$$C_{\max}(\pi^*) \leqslant C_{\max}(\pi), \quad \forall \pi \in \Pi. \tag{11}$$

2.2. Graph representation

A graph representation model is used by Grabowski and Pempera [2] to present the blocking flow shop scheduling problem (see Fig. 3). For a permutation π, a graph $G(\pi) = (Z, A)$ is created with a set of nodes $Z = \bigcup_{j=1}^{n} \bigcup_{k=1}^{m} (j,k)$ and a set of arcs $A = A^+ \cup A^0 \cup A^-$, where the node (j,k) with weight $p_{\pi_j,k}$ represents the processing of operation $O_{\pi_j,k}$, and $A^+ = \bigcup_{j=1}^{n} \bigcup_{k=1}^{m-1} \{((j,k),\ (j,k+1))\}$ is a subset of the arcs connecting consecutive operations of the same job, and $A^0 = \bigcup_{j=1}^{n-1} \bigcup_{k=1}^{m} \{((j,k),\ (j+1,k))\}$ is a subset of the arcs connecting operations on the same machine in the processing order given by π, and $A^- = \bigcup_{j=1}^{n-1} \bigcup_{k=1}^{m-1} \{((j,k+1),\ (j+1,k))\}$ is a subset of arcs associated with the blocking constraints, and each arc $((j,k+1),\ (j+1,k))$ from A^- has weight $-p_{\pi_j,k+1}$. The makespan $C_{\max}(\pi)$ is equal to the longest (critical) path from node $(1,1)$ to node (n,m) in the graph $G(\pi)$, which can be decomposed into several specific sub-paths and each of them contains the nodes linked with the same type of arcs. The first type of sub-paths is determined by the arcs in A^0 which correspond to sequences of operations processed on the same machine without inserted idle time. The second type of sub-paths is defined by the arcs in A^- which correspond to blocked jobs. For simplicity, denote the first type of sub-paths and the second type of sub-paths as blocks and anti-blocks, respectively. Note that the critical path can contain many different blocks and anti-blocks, and all the blocks and anti-blocks are connected in series. A simple graph model is shown in Fig. 3 for the blocking flow shop scheduling problem with $n = 8$ and $m = 6$, where arcs from A^+ and A^0 are represented by solid lines, and arcs from A^- by dashed lines, and nodes in the critical path by bold circles.

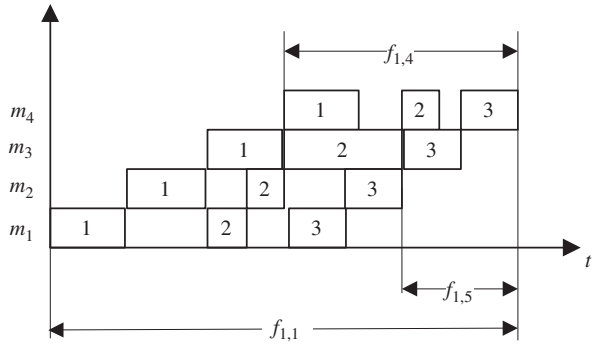

Fig. 2. Computation of $f_{j,k}$.

3. Insert-neighborhood-based speed-up method

Insert neighborhood of a job permutation $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is widely used for flow shop scheduling problems in the literature [32], and is defined by considering all possible insert moves in the set $V = \{v(i,q) | i, q \in \{1,2,\ldots,n\} \wedge q \notin \{i, i-1\}\}$. The insert move $v(i,q)$ generates a permutation π' from the permutation π by removing a job π_i from its original position i and inserting it into another position q $(q \notin \{i, i-1\})$:

$$\pi' = \{\pi_1, \ldots, \pi_{i-1},\ \pi_{i+1}, \ldots, \pi_i,\ \pi_q,\ \pi_{q+1}, \ldots, \pi_n\} \quad \text{if } (i < q), \tag{12}$$

$$\pi' = \{\pi_1, \ldots, \pi_{q-1},\ \pi_i,\ \pi_{q+1}, \ldots, \pi_{i-1},\ \pi_{i+1}, \ldots, \pi_n\} \quad \text{if } (i > q). \tag{13}$$

The number of neighbors in the insert neighborhood of a permutation π is $(n-1)^2$, so the computational complexity to evaluate the whole insert neighborhood is $O(mn^3)$ by using each approach proposed in Section 2.1. However, making use of the similarity of the permutation π and its neighbor π', and inspired by reduction of

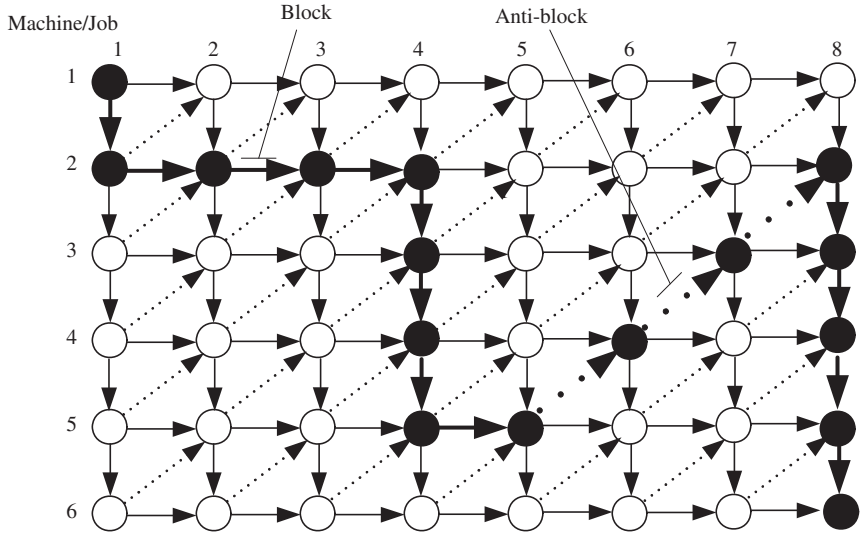

Fig. 3. A graph model blocking flow shop scheduling problem [2].

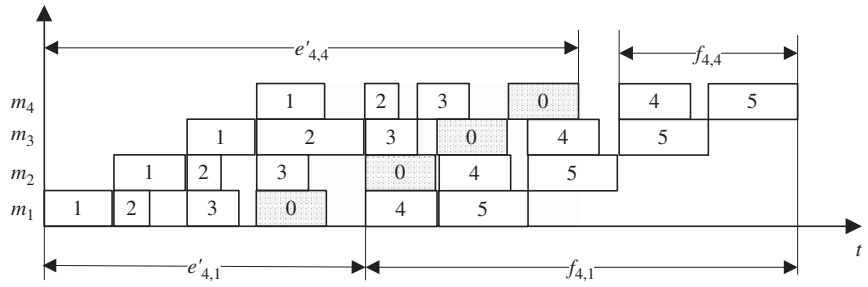

Fig. 4. Computation of makespan after inserting job 0 into position 4.

computational complexity presented by Grabowki [10], Taillard [33] and Smutnicki [34], we develop a speed-up to evaluate the insert neighborhood for the problems considered here. The procedure is given below:

Step 1: Calculate the departure times $e_{j,k}, j=1,2,\ldots,n, k=1,2,\ldots,m$ using Eqs. (1)–(5).

Step 2: Calculate the tails $f_{j,k}$, $j=n,n-1,\ldots,1$, $k=m,m-1,\ldots,1$ using Eqs. (6)–(10).

Step 3: Let $i=1$.

Step 4: Let $\pi''=\{\pi''_1,\pi''_2,\ldots,\pi''_{n-1}\}$ be a partial permutation generated by removing job π_i from the permutation π.

Step 5: Get the departure times $e''_{j,k}, j=1,2,\ldots,n-1, k=1,2,\ldots,m$, for the partial permutation π'' as follows. If $j<i$, then let $e''_{j,k}=e_{j,k}$; else calculate $e''_{j,k}$ using Eqs. (1)–(5).

Step 6: Get the tails $f''_{j,k}$, $j=n-1,n-2,\ldots,1$, $k=m,m-1,\ldots,1$, for the permutation π'' as follows. If $j>i$, then let $f''_{j,k}=f_{j+1,k}$; else calculate $f''_{j,k}$ using Eqs. (6)–(10).

Step 7: Repeat the following steps until all possible positions q, $q\in\{1,2,\ldots,n\}\wedge q\notin\{i,i-1\}$, of the permutation π'' are considered.

Step 7.1: Insert job π_i into position q and generates a permutation π'.

Step 7.2: Calculate the departure times $e'_{q,k}$ by using the obtained departure time $e''_{q-1,k}$, where $k=1,2,\ldots,m$.

Step 7.3: The makespan of the permutation π' is given as follows (illustrated in Fig. 4):

$$C_{\max}(\pi')=\max_{k=1}^{m}(e'_{q,k}+f''_{q,k}).$$

Step 8: Let $i=i+1$. If $i>n$ then stop; otherwise go back to Step 4.

There are n iterations for Steps 5–7, and each of these steps can be executed in time $O(mn)$. So, the computational complexity of this speed-up method is $O(mn^2)$ to evaluate the whole insert neighborhood of the permutation π.

In addition, according to the block properties presented by Grabowski [2,10,34] based on the graph model, we can further reduce the computational time to evaluate an insert neighborhood. The theorem is given as follows:

Theorem 1 (*Grabowski [2]*). *Let $\pi\in\Pi$ be any permutation with blocks $B_{g_i,h_i}=(\pi_{g_i},\pi_{g_i+1},\ldots,\pi_{h_i})$, $(g_i,h_i)\in GH=\{(g_i,h_i)|i=1,\ldots,l_B\}$ and anti-blocks $A_{s_i,t_i}=(\pi_{s_i},\pi_{s_i+1},\ldots,\pi_{t_i})$ $(s_i,t_i)\in ST=\{(s_i,t_i)|i=1,\ldots,l_A\}$. If the permutation π' has been obtained by an interchange of jobs that $C_{\max}(\pi')<C_{\max}(\pi)$, then in π'*

(1) *at least one job $\pi_j\in B_{g_i,h_i}$ precedes job π_{g_i}, for some $(g_i,h_i)\in GH$, or*
(2) *at least one job $\pi_j\in B_{g_i,h_i}$ succeeds job π_{h_i}, for some $(g_i,h_i)\in GH$, or*
(3) *at least one job $\pi_j\in A_{s_i,t_i}$ precedes job π_{s_i}, for some $(s_i,t_i)\in ST$, or*
(4) *at least one job $\pi_j\in A_{s_i,t_i}$ succeeds job π_{t_i}, for some $(s_i,t_i)\in ST$.*

It is concluded from the above theorem that some insert moves cannot produce the permutations with better makespan. Since in our algorithm, we aim to find a better permutation than π in its neighborhood, we will remove these non-improving moves from the insert move set V to further save computational time. The procedure is given as follows:

Step 1: Set $j=1$ and $k=m$.

Step 2: If $e_{j,k}+f_{j+1,k}=C_{\max}(\pi)$, then the following two cases may arise

Case 1: $e_{j,k}=e_{j,k-1}+p_{j,k}$. In this case, let

$$B_k=\begin{cases}\{o_{j,k},o_{j+1,k}\} & \text{if } B_k=\{\},\\ B_k\cup\{o_{j+1,k}\} & \text{otherwise,}\end{cases}$$

where B_k is a block.

Case 2: $e_{j,k}>e_{j,k-1}+p_{j,k}$. In this case, let

$$AB_k=\begin{cases}\{o_{j,k+1},o_{j+1,k}\} & \text{if } AB_{k+1}=\{\},\\ AB_{k+1}\cup\{o_{j+1,k}\} & \text{otherwise,}\end{cases}$$

and

$$AB_{k+1}=\{\},$$

where AB_k is an anti-block. If $k=1$, then remove the non-improving moves from V according to anti-block AB_k and Theorem 1, and let $AB_k=\{\}$.

Step 3: If $e_{j,k}+f_{j+1,k}<C_{\max}(\pi)$, then remove the invalid moves from V according to theorem 1, B_k and AB_{k+1}, then let $B_k=\{\}$ and $AB_{k+1}=\{\}$.

Step 4: If $k>0$, let $k=k-1$ and go to Step 2.

Step 5: If $j<n$, let $j=j+1$ and $k=m$, then go to Step 2; otherwise stop the procedure.

The above method can effectively extract some non-improving insert moves from the set V, so the neighbors of the permutation π to be evaluated are tailed off and the computational time requirement to find the better neighbors is further decreased. It is easy to verify that in some special cases, such as all $p_{j,k}$ are identical and $n\geqslant 3$, each machine has single block containing all jobs. After executing the procedure, the remaining moves in the set V is decreased to $4n-8$, whereas the initial number of moves is no less than $(n-1)^2$. Note that when the non-improving moves are extracted, the indices of the neighbors will be changed.

4. DDE algorithm

4.1. Brief introduction to DE algorithm

The basic DE algorithm [19] is one of the latest population-based and stochastic global optimizers. It adopts a floating-point encoding scheme and takes advantage of the differentiation information among the population to find the global optimum in the continuous

search space. Starting from a random initialization of *PS* target individuals, it creates *PS* new candidate solutions by *mutation* and *crossover* operators, and then applies *selection* operator to determine the new target individuals in the next generation. Let $X_i^t = \lfloor x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t \rfloor$, $V_i^t = \lfloor v_{i1}^t, v_{i2}^t, \ldots, v_{in}^t \rfloor$ and $U_i^t = \lfloor u_{i1}^t, u_{i2}^t, \ldots, u_{in}^t \rfloor$ denote the ith target individual, the ith mutant individual, and the ith trial individual at iteration t, respectively. Following the *DE/rand/1/bin* scheme [19], the procedure of the basic DE algorithm can be described as follows:

Step 1: *Initialization* phase. Set the parameters and generate a population of *PS* individuals randomly. Let *bestsofar* be the best individual found so far, and $t = 0$.

Step 2: *Mutation* phase. Set $t = t + 1$, and generate mutant individual V_i^t, $i = 1, 2, \ldots, PS$, as follows:

$$v_{ij}^t = x_{aj}^{t-1} + Z \times (x_{bj}^{t-1} - x_{cj}^{t-1}), \tag{14}$$

where a, b and c are three random integers in the range [1, *PS*] such that a, b, c and i are pairwise different, and $j = 1, 2, \ldots, n$, and $Z \in (0, 2)$ is a scale factor to control the amplification of the differential variation between two target individuals.

Step 3: *Crossover* phase. Generate trial individual U_i^t, $i = 1, 2, \ldots, PS$, as follows:

$$u_{ij}^t = \begin{cases} v_{ij}^t & \text{if } rand(\,) < CR \text{ or } j = D_j, \\ x_{ij}^{t-1} & \text{otherwise,} \end{cases} \tag{15}$$

where $CR \in [0, 1]$ is a crossover parameter to control the diversity of the population, $rand(\,)$ denotes a uniform random number between 0 and 1, and D_j refers to an index randomly chosen from the set $\{1, 2, \ldots, n\}$ to ensure that at least one dimension of U_i^t is different from its counterpart X_i^{t-1}.

Step 4: *Selection* phase. For a minimization problem, each new target individual can be generated as follows:

$$X_i^t = \begin{cases} U_i^t & \text{if } f(U_i^t) \leqslant f(X_i^{t-1}), \\ X_i^{t-1} & \text{otherwise,} \end{cases} \tag{16}$$

where $f(\cdot)$ is the objective function.

Step 5: Update *bestsofar*.

Step 6: If a stopping criterion is satisfied, then output *bestsofar*; otherwise go back to Step 2.

4.2. DDE algorithm

The basic DE algorithm cannot be used to directly generate discrete job permutations since it is originally designed to solve continuous optimization problems where the individuals are represented by floating-point numbers. Therefore, in this section, we will present a novel DDE algorithm to solve the blocking flow shop scheduling problem after explaining solution representation, new mutation and crossover operators, and initialization.

4.2.1. Solution representation

The job-permutation-based representation has been widely used in the literature for flow shop scheduling problems and it is easy to decode to reduce the cost of the algorithm [6]. In this section, we adopt this representation and an example is given in Table 1.

4.2.2. Job-permutation-based mutation operator

According to the basic DE algorithm, a mutant individual is generated by adding the weighted difference between two target individuals randomly selected from previous population to another individual, so a job-permutation-based mutation operator can be presented as follows:

$$V_i^t = X_a^{t-1} \oplus Z \otimes (X_b^{t-1} - X_c^{t-1}), \tag{17}$$

where a, b and c are three random integers in the range [1, *PS*] such that a, b, c and i are pairwise different, and $i = 1, 2, \ldots, PS$, and $Z \in [0, 1]$ is a mutant scale factor.

The above formula consists of two components. The first component refers to the weighted difference between two target individuals X_b^{t-1} and X_c^{t-1}, that is,

$$\begin{aligned} &\Delta_i^t = Z \otimes (X_b^{t-1} - X_c^{t-1}) \\ &\Leftrightarrow \delta_{i,j}^t = \begin{cases} x_{bj}^{t-1} - x_{cj}^{t-1} & \text{if } rand(\,) < Z, \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{18}$$

where $\Delta_i^t = (\delta_{i,0}^t, \delta_{i,1}^t, \ldots, \delta_{i,n}^t)$ is a temporary vector.

The second component is to produce a mutant individual V_i^t by adding Δ_i^t to another target individual X_a^{t-1}, that is,

$$\begin{aligned} &V_i^t = X_a^{t-1} \oplus \Delta_i^t \\ &\Leftrightarrow v_{ij}^t = x_{aj}^{t-1} \oplus \delta_{ij}^t = \mathrm{mod}((x_{aj}^{t-1} + \delta_{ij}^t + n), n), \end{aligned} \tag{19}$$

where "mod" denotes the modulus operator whose result is the remainder when the first operand is divided by the second. This operator ensures that each dimension of V_i^t can represent a job. In order to better understand the proposed mutation operator, an example is given in Fig. 5.

From Fig. 5, it follows that each dimension of V_i^t represents a job, but V_i^t itself cannot always represent a complete sequence since some jobs may repeat many times whereas other jobs may be lost. However, from the crossover operator of the basic DE algorithm, it can be seen that a mutant individual is used to enhance perturbation to a target individual to avoid premature convergence to a non-global local optimum. Therefore, it is not necessary to ensure a feasible solution by mutation operator, and a legal target individual can be obtained by the following crossover operator.

4.2.3. Job-permutation-based crossover operator

The crossover operator generates a trial individual U_i^t by combining a mutant individual V_i^t and a target individual X_i^{t-1}, and we expect the U_i^t to be better than X_i^{t-1}. To well inherit good structures from the target individual X_i^{t-1} and to yield a better U_i^t, the job-permutation-based crossover operator is given as follows (illustrated by an example in Fig. 6):

Step 1: From $j = 1$ to n, remove job $v_{i,j}^t \in V_i^t$ if $rand(\,) \geqslant CR$ or the job has already been in V_i.

Step 2: Let $U_i^t = X_i^{t-1}$, and eliminate the jobs included in V_i from U_i^t.

Step 3: The first job from V_i is taken and inserted in the best position of U_i^t by evaluating all the possible slots of U_i^t. This step is repeated until V_i is empty. Now a complete schedule is obtained.

4.2.4. Population initialization

Since NEH heuristic [16] is one of the well-know heuristics in the literature, and it performs better than PF approach developed by Mccormick et al. [13] and the heuristics by Leistein [14] for the considered problem, to guarantee an initial population with a certain quality and diversity, we take advantage of NEH to produce an individual, while the other individuals are randomly generated in the solution space. The NEH heuristic has two phases. In phase I, jobs are ordered in descending sums of their processing times. In phase II, a job sequence is established by evaluating the partial schedules based on the initial order of phase I. In this paper, the speed-up method proposed in Section 3 is used to evaluate the partial schedules, so the NEH heuristic can be executed in time $O(mn^2)$ for the blocking flow shop scheduling problem.

Table 1
An example of the job-permutation-based representation.

Dimension j	1	2	3	4	5	6	7
$x_{i,j}$	3	4	6	2	5	0	1
Job permutation	3	4	6	2	5	0	1

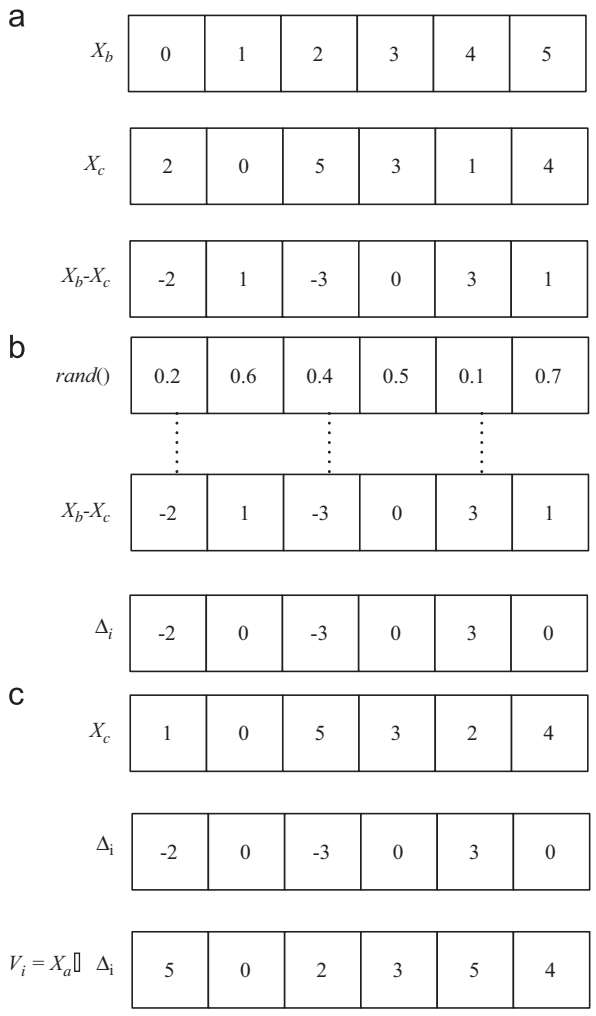

Fig. 5. Mutation operator: (a) $X_b - X_c$; (b) $\Delta_i = Z \otimes (X_a - X_b)$ with $Z = 0.5$ and (c) $V_i^t = X_a^{t-1} \oplus \Delta_i^t$.

4.2.5. Procedure of the DDE algorithm

Based on the above design, the procedure of the DDE algorithm for solving the blocking flow shop scheduling problem is presented as follows:

Step 1: *Initialization* phase. Set the parameters *PS*, *CR*, and *Z*. Initialize population. Let *bestsofar* be the best individual found so far, and $t = 0$.

Step 2: *Mutation* phase. Set $t = t + 1$, and generate *PS* mutant individuals by using the job-permutation-based mutation operator.

Step 3: *Crossover* phase. Generate *PS* trial individuals by using the job-permutation-based crossover operator.

Step 4: *Selection* phase. Decide *PS* target individuals for next generation.

Step 5: Update *bestsofar*.

Step 6: If a stopping criterion is satisfied, then output *bestsofar*; otherwise go back to Step 2.

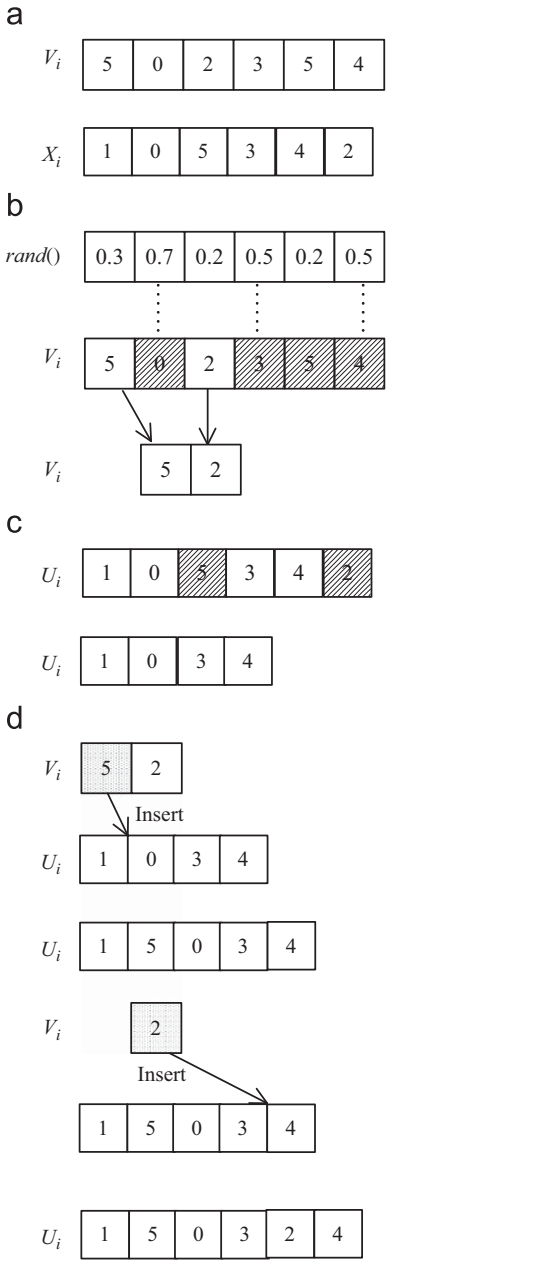

Fig. 6. Crossover operator: (a) V_i and X_i; (b) remove jobs from V_i ($CR = 0.5$); (c) let $U_i = X_i$ and remove the jobs included in V_i from U_i and (d) obtain U_i by inserting jobs of V_i to the best position of U_i.

Unlike the basic DE algorithm, the DDE algorithm adopts the job-permutation-based encoding scheme, employs the job-permutation-based mutation and crossover operators, and takes advantage of the initialization based on NEH heuristic, so it can be easily applied to the blocking flow shop scheduling problem.

5. HDDE algorithm

5.1. A problem-dependent local search

To balance the exploration and exploitation, a problem-dependent local search is presented and applied to each trial individual U_i^t with a probability P_l to enhance the local searching ability. That is, a uniform random number between 0 and 1 is firstly generated, and the individual will undergo the local search if the random number is less than P_l. For the job-permutation-based optimization problems, *insert*, *swap*, and *inverse* operators are commonly used to produce a neighboring solution [2]. Among them, *insert* operator is the most suitable for performing a fine local search [28]. Thus, we employ *insert* operator to be embedded in the DDE algorithm to refine local search for the problem considered. The procedure of the local search is described as follows:

Step 1: Let $i=0$, $j=0$, and produce a reference sequence $\pi^r=\{\pi_1^r,\pi_2^r,\ldots,\pi_n^r\}$ randomly.

Step 2: Let $i=i+1$, and set $i=i-n$ if $i>n$.

Step 3: Extract job π_i^r from the trial individual sequence U_i^t, and insert it into all the other possible slots of U_i^t, respectively. Denote the best obtained sequence as π^b.

Step 4: If $f(\pi^b)<f(U_i^t)$, then let $U_i^t=\pi^b$ and $j=0$; otherwise, $j=j+1$.

Step 5: If $j\leqslant n$, then go back to step 2; otherwise stop the procedure.

The above local search algorithm is effective and efficient due to the following reasons. Firstly, the local search is performed to the trial individual U_i^t but not the target individual X_i^t, so the algorithm can avoid both cycling search and getting trapped in a non-global local optimum. Secondly, in the local search, the incumbent solution is updated only when a better solution is found, so the algorithm can rapidly reach a local solution. Lastly, by using the speed-up method proposed in Section 3, the insert neighbors can be evaluated efficiently. Thus, the local search algorithm tends to enhance the local exploitation of the DDE algorithm in a relatively short time.

5.2. The HDDE algorithm

Based on the DDE algorithm and the local search presented in Section 5.1, the basic procedure of the HDDE algorithm is given as follows:

Step 1: *Initialization* phase. Set the parameters *PS*, *CR*, Z, and P_l. Initialize population. Let *bestsofar* be the best individual found so far, and $t=0$.

Step 2: *Mutation* phase. Set $t=t+1$, and generate *PS* mutant individuals by using the job-permutation-based mutation operator.

Step 3: *Crossover* phase. Generate *PS* trial individuals by using the job-permutation-based crossover operator.

Step 4: *Local search* phase. Perform the local search to U_i^t, $i=1,2,\ldots,PS$, with probability P_l.

Step 5: *Selection* phase. Select *PS* target individuals by one-to-one selection operator for next generation.

Step 6: Update *bestsofar*.

Step 7: If a stopping criterion is satisfied, then stop the procedure and output *bestsofar*; otherwise go back to Step 2.

It can be seen that the HDDE algorithm not only applies the DDE-based evolutionary searching mechanism to effectively perform exploration for promising solutions within the whole solution space, but it also employs the problem-dependent local search to perform exploitation for solution improvement in the local neighborhoods. Since both the balance of exploration and exploitation and efficiency of solution evaluation are stressed, it is expected to achieve good results for the blocking flow shop scheduling problems with makespan criterion.

6. Computational results

6.1. Experimental setup

To test the performance of the proposed DDE/HDDE algorithm, a comprehensive experimental evaluation and comparison with other powerful methods are presented based on the well-known flow shop benchmark set of Taillard [1]. The benchmark set is composed of 12 subsets of given problems with the size ranging from 20 jobs and five machines to 500 jobs and 20 machines, and each subset consists of ten instances. We treat them as the blocking flow shop scheduling problems with makespan criterion. And, in our test, the proposed DDE/HDDE algorithm is coded in C++ and the experiments are executed on a Pentium P-IV 3.0 GHz PC with 512 MB memory. The parameters are set as follows: $F=0.2$, $CR=0.2$, $PS=20$, $P_l=0.2$, and the maximum computational time is set as $T=5mn$ ms. Each instance is run 10 independent replications and in each replication we compute the percentage relative difference (*PRD*) as follows:

$$PRD=\frac{100\times(C^{Ron}-C^{A})}{C^{Ron}}, \tag{20}$$

where C^{Ron} is the referenced makespan provided by Ronconi [4], and C^{A} is the makespan found by the DDE/HDDE algorithm. Furthermore, average percentage relative difference (*APRD*), maximum percentage relative difference (*MaxPRD*), minimum percentage relative difference (*MinPRD*), and the standard deviation (SD) of *PRD* are calculated as the performance statistics.

6.2. Comparison of the DDE/HDDE and DE/HDE algorithms

Recently, an effective HDE-based algorithm was developed by Qian et al. [3] for solving multi-objective flow shop scheduling problems with limited buffers between consecutive machines. In the HDE algorithm, a largest-order-value rule was employed to convert the continuous values of individuals to discrete job permutations, and a very efficient local search algorithm was incorporated to emphasize exploitation. By the experimental simulations based on the well-known benchmark instances collected from OR library, the authors demonstrated the effectiveness and efficiency of their HDE algorithm. In this paper, we modify the HDE algorithm for the blocking flow shop scheduling problem and code it in C++. The parameters of the HDE algorithm is consistent with those in [3], and the maximum computation time is also set as $T=5mn$ ms. Computational simulations are carried out on the same PC as mentioned before on the same benchmarks. For each instance, 10 independent replications are also conducted to obtain statistics. The computational results produced by the DE (the HDE algorithm without local search) and HDE algorithms are reported in Tables 2 and 3, respectively.

It can be seen from Table 2 for the same computational time, the overall mean APRD and SD values yielded by the DDE algorithm are equal to 2.79% and 0.16%, respectively, which are much better than those (−3.84% and 0.33%) generated by the DE algorithm. More importantly, the DDE algorithm produced substantially better PRD values with smaller SD values than the DE algorithm for all the problem sizes and all instances as well. From these observations, it is concluded that the DDE algorithm is more robust, effective and efficient than the DE algorithm for blocking flow shop scheduling problems.

The results reported in Table 3 indicate that the HDDE algorithm generates significantly better APRD values with smaller SD values than the HDE algorithm for each problem size using the same computational time. As the problem size increases, the superiority of the HDDE algorithm over the HDE algorithm increases. Thus, it is concluded that the HDDE algorithm can reach optimal

Table 2
Computation results of the DDE and DE algorithms.

$n \times m$	DDE					DE				
	APRD	MinPRD	MaxPRD	SD	T (s)	APRD	MinPRD	MaxPRD	SD	T (s)
20 × 5	0.34	0.18	0.45	0.10	0.50	−0.78	−1.32	−0.05	0.40	0.50
20 × 10	2.34	2.16	2.39	0.08	1.00	1.73	1.29	2.14	0.30	1.00
20 × 20	3.25	3.15	3.30	0.06	2.00	2.86	2.49	3.18	0.22	2.00
50 × 5	4.15	3.73	4.61	0.28	1.25	−4.32	−5.20	−3.33	0.61	1.25
50 × 10	5.36	4.94	5.83	0.28	2.50	−0.30	−0.89	0.43	0.44	2.50
50 × 20	5.55	5.25	5.87	0.20	5.00	1.99	1.47	2.51	0.33	5.00
100 × 5	0.37	0.03	0.79	0.24	2.50	−13.99	−14.62	−13.06	0.48	2.50
100 × 10	3.90	3.59	4.25	0.21	5.00	−5.70	−6.25	−5.11	0.35	5.00
100 × 20	3.62	3.36	3.88	0.16	10.00	−2.42	−2.80	−1.93	0.29	10.00
200 × 10	1.29	1.04	1.58	0.17	10.00	−11.12	−11.46	−10.63	0.26	10.00
200 × 20	2.17	1.99	2.35	0.11	20.00	−5.82	−6.10	−5.46	0.20	20.00
500 × 20	1.19	1.08	1.34	0.08	50.00	−8.20	−8.33	−8.02	0.10	50.00
Mean	2.79	2.54	3.05	0.16	9.15	−3.84	−4.31	−3.28	0.33	9.15

Table 3
Computation results of the HDDE and HDE algorithms.

$n \times m$	HDDE					HDE				
	APRD	MinPRD	MaxPRD	SD	T (s)	APRD	MinPRD	MaxPRD	SD	T (s)
20 × 5	0.43	0.33	0.46	0.05	0.50	0.26	0.00	0.44	0.16	0.50
20 × 10	2.38	2.36	2.40	0.02	1.00	2.30	2.13	2.39	0.10	1.00
20 × 20	3.29	3.24	3.30	0.02	2.00	3.25	3.14	3.30	0.06	2.00
50 × 5	4.24	3.88	4.67	0.25	1.25	3.09	2.59	3.62	0.36	1.25
50 × 10	5.75	5.43	6.12	0.23	2.50	4.57	4.10	5.06	0.31	2.50
50 × 20	6.03	5.74	6.34	0.20	5.00	5.00	4.58	5.51	0.30	5.00
100 × 5	1.42	1.04	1.86	0.26	2.50	−0.32	−0.84	0.31	0.36	2.50
100 × 10	5.17	4.92	5.56	0.21	5.00	3.40	2.88	3.99	0.35	5.00
100 × 20	4.68	4.39	5.01	0.19	10.00	3.05	2.69	3.47	0.26	10.00
200 × 10	3.09	2.80	3.47	0.20	10.00	0.33	−0.14	0.88	0.34	10.00
200 × 20	3.57	3.31	3.86	0.17	20.00	1.36	1.00	1.82	0.26	20.00
500 × 20	2.47	2.16	2.78	0.20	50.00	0.25	−0.06	0.59	0.20	50.00
Mean	3.54	3.30	3.82	0.17	9.15	2.21	1.84	2.62	0.26	9.15

Table 4
Computation results of the [a]GA and DDE algorithms.

$n \times m$	GA		DDE				
	PRD	T (s)	APRD	MinPRD	MaxPRD	SD	T (s)
20 × 5	−6.36	0.1	0.34	0.18	0.45	0.10	0.50
20 × 10	−4.35	0.2	2.34	2.16	2.39	0.08	1.00
20 × 20	−1.26	0.4	3.25	3.15	3.30	0.06	2.00
50 × 5	−8.53	0.3	4.15	3.73	4.61	0.28	1.25
50 × 10	−5.97	0.5	5.36	4.94	5.83	0.28	2.50
50 × 20	−4.33	1.1	5.55	5.25	5.87	0.20	5.00
100 × 5	−14.40	0.5	0.37	0.03	0.79	0.24	2.50
100 × 10	−7.89	1.1	3.90	3.59	4.25	0.21	5.00
100 × 20	−5.64	2.1	3.62	3.36	3.88	0.16	10.00
200 × 10	−11.04	2.2	1.29	1.04	1.58	0.17	10.00
200 × 20	−7.00	4.3	2.17	1.99	2.35	0.11	20.00
500 × 20	−8.08	10.8	1.19	1.08	1.34	0.08	50.00
Mean	−7.07	1.97	2.79	2.54	3.05	0.16	9.15

[a]CPU times of GA on Pentium IV 1000 MHZ.

or near-optimal solutions for blocking flow shop scheduling problems more robustly, effectively and efficiently than the HDE algorithm.

6.3. Comparison of the DDE and GA algorithms

To further show the effectiveness of the proposed DDE algorithm, we compare the DDE algorithm with the existing GA developed by Caraffa et al. [18] for blocking flow shop scheduling problems with makespan criterion. The computational results of these algorithms are shown in Table 4.

From Table 4, it follows that the DDE algorithm produces significantly better APRD values than the GA algorithm for all groups. Though the maximum running time of the DDE algorithm is larger than that of the GA algorithm, it is still acceptable since the overall mean maximum running time of the DDE algorithm is less than 10 s, and for the group with the extreme case of 500 jobs and 20 machines, the maximum computational time is not greater than 50 s. It

Table 5
Computation results[a] of the TS, TS + M and HDDE algorithms.

$n \times m$	TS		TS + M		HDDE				
	PRD	T(s)	PRD	T (s)	APRD	MinPRD	MaxPRD	SD	T (s)
20 × 5	−1.64	2.4	−0.34	2.7	0.43	0.33	0.46	0.05	0.50
20 × 10	1.45	4.1	1.76	4.6	2.38	2.36	2.40	0.02	1.00
20 × 20	2.88	7.1	2.94	7.6	3.29	3.24	3.30	0.02	2.00
50 × 5	−0.55	6.0	0.55	6.2	4.24	3.88	4.67	0.25	1.25
50 × 10	1.98	10.6	3.52	10.8	5.75	5.43	6.12	0.23	2.50
50 × 20	3.68	19.0	4.26	19.3	6.03	5.74	6.34	0.20	5.00
100 × 5	−3.03	12.2	−2.62	12.4	1.42	1.04	1.86	0.26	2.50
100 × 10	1.71	21.9	2.66	22.1	5.17	4.92	5.56	0.21	5.00
100 × 20	2.01	39.2	3.03	39.4	4.68	4.39	5.01	0.19	10.00
200 × 10	−0.60	44.1	0.58	44.3	3.09	2.80	3.47	0.20	10.00
200 × 20	1.24	79.2	2.31	79.4	3.57	3.31	3.86	0.17	20.00
500 × 20	0.63	207	1.47	209	2.47	2.16	2.78	0.20	50.00
Mean	0.81	37.73	1.68	38.15	3.54	3.30	3.82	0.17	9.15

[a]CPU times of TS and TS + M on Pentium IV 1000 MHZ.

Table 6
Upper bounds produced by the HDDE algorithm.

HDDE 20 × 5	TS + M	RON	HDDE 50 × 5	TS + M	RON	HDDE 100 × 5	TS + M	RON	HDDE 200 × 10	TS + M	RON
1374	1387	1384	**3033**	3163	3151	**6291**	6639	6455	**13,756**	14,220	14,113
1408	1424	1411	**3226**	3348	3395	**6136**	6481	6214	**13,621**	14,089	14,127
1280	1293	1294	**3039**	3173	3184	**6063**	6299	6124	**13,741**	14,149	14,416
1448	1451	1448	**3147**	3277	3303	**5839**	6120	5976	**13,718**	14,156	14,435
1341	1348	1366	**3192**	3338	3272	**6065**	6340	6173	**13,721**	14,130	14,119
1363	1366	1363	**3183**	3330	3400	**5971**	6244	6094	**13,474**	13,963	13,909
1381	1387	1381	**3054**	3168	3228	**6095**	6346	6262	**13,925**	14,386	14,563
1379	1388	1384	**3081**	3228	3260	**5985**	6289	6061	**13,863**	14,256	14,329
1373	1392	1378	**2929**	3068	3104	**6234**	6559	6474	**13,659**	13,954	13,923
1283	1302	1283	**3146**	3285	3264	**6273**	6509	6366	**13,744**	14,224	14,435
20 × 10			50 × 10			100 × 10			200 × 20		
1698	1698	1736	**3667**	3776	3913	**7131**	7320	7496	**15,057**	15,334	15,579
1833	1836	1897	**3523**	3641	3798	**6816**	7108	7281	**15,284**	15,522	15,728
1659	1674	1677	**3515**	3588	3723	**6956**	7233	7400	**15,360**	15,713	15,915
1535	1555	1622	**3685**	3786	3885	**7261**	7413	7670	**15,276**	15,687	16,039
1617	1631	1658	**3650**	3745	3934	**6913**	7168	7317	**15,183**	15,443	15,938
1590	1603	1640	**3622**	3747	3831	**6739**	6993	7301	**15,223**	15,472	15,911
1622	1629	1634	**3704**	3778	3957	**6874**	7092	7247	**15,296**	15,522	15,898
1731	1754	1741	**3590**	3708	3774	**6940**	7143	7315	**15,310**	15,540	16,022
1747	1759	1777	**3556**	3668	3784	**7133**	7327	7631	**15,243**	15,394	15,817
1782	1782	1847	**3642**	3729	3928	**7065**	7299	7411	**15,284**	15,523	15,969
20 × 20			50 × 20			100 × 20			500 × 20		
2436	2449	2530	**4516**	4627	4886	**7891**	8101	8347	**37,172**	37,860	38,334
2234	2242	2297	**4296**	4411	4668	**7931**	8105	8372	**37,485**	38,044	38,642
2479	2483	2560	**4290**	4388	4666	**7935**	8071	8265	**37,209**	37,732	38,163
2348	2348	2399	**4393**	4479	4650	**7930**	8081	8365	**37,291**	38,062	38,625
2435	2450	2538	**4284**	4359	4475	**7944**	8074	8304	**37,232**	37,991	38,492
2383	2398	2467	**4308**	4372	4521	**7971**	8151	8450	**37,513**	38,132	38,551
2390	2397	2502	**4325**	4402	4576	**8051**	8273	8507	**37,121**	37,561	38,179
2328	2345	2411	**4337**	4444	4688	**8102**	8248	8584	**37,202**	37,750	38,664
2363	2363	2421	**4332**	4423	4532	**8007**	8116	8341	**37,116**	37,730	38,339
2323	2334	2407	**4439**	4609	4846	**8050**	8261	8489	**37,492**	38,014	38,540

should be noted that when the maximum running time of the GA algorithm is increased to 475.0 s for the extreme case, it still produced the much worse APRD value (−2.57) [2] than the DDE algorithm. Therefore, we can conclude that the DDE algorithm outperforms the GA algorithm on the considered problems.

6.4. Comparison of the HDDE and TS/TS + M algorithms

In this section, the HDDE algorithm is compared with two TS algorithms (TS and TS + M) [2] proposed by Grabowski and Pempera who concluded that the TS and TS + M algorithm were the best performing algorithms when compare to GA developed by Caraffa et al. [18] and the heuristic presented by Ronconi [4]. The computational results of these algorithms are reported in Table 5.

From Table 5, it follows that the HDDE algorithm performs significantly better than both the TS and TS + M algorithms, since the overall mean ARPD value yielded by the HDDE algorithm is equal to 3.54%, which is 4.37 and 2.11 times higher than those generated by the TS (0.81%) and TS + M (1.68%) algorithms, respectively. More importantly, the HDDE algorithm produces much better APRD value

Table 7
Comparison of the HDDE, DDE and MLS algorithms.

$n \times m$	HDDE					MLS					DDE				
	APRD	MinPRD	MaxPRD	SD	T (s)	APRD	MinPRD	MaxPRD	SD	T (s)	APRD	MinPRD	MaxPRD	SD	T (s)
20×5	0.43	0.33	0.46	0.05	0.50	−0.86	−1.60	−0.09	0.49	0.50	0.34	0.18	0.45	0.10	0.50
20×10	2.38	2.36	2.40	0.02	1.00	1.23	0.60	1.93	0.41	1.00	2.34	2.16	2.39	0.08	1.00
20×20	3.29	3.24	3.30	0.02	2.00	2.47	1.95	3.04	0.36	2.00	3.25	3.15	3.30	0.06	2.00
50×5	4.24	3.88	4.67	0.25	1.25	1.62	0.97	2.25	0.41	1.25	4.15	3.73	4.61	0.28	1.25
50×10	5.75	5.43	6.12	0.23	2.50	3.10	2.34	3.87	0.48	2.50	5.36	4.94	5.83	0.28	2.50
50×20	6.03	5.74	6.34	0.20	5.00	3.86	3.23	4.65	0.44	5.00	5.55	5.25	5.87	0.20	5.00
100×5	1.42	1.04	1.86	0.26	2.50	−0.70	−1.28	−0.12	0.38	2.50	0.37	0.03	0.79	0.24	2.50
100×10	5.17	4.92	5.56	0.21	5.00	3.06	2.49	3.69	0.38	5.00	3.90	3.59	4.25	0.21	5.00
100×20	4.68	4.39	5.01	0.19	10.00	2.77	2.39	3.17	0.25	10.00	3.62	3.36	3.88	0.16	10.00
200×10	3.09	2.80	3.47	0.20	10.00	1.64	1.04	2.28	0.39	10.00	1.29	1.04	1.58	0.17	10.00
200×20	3.57	3.31	3.86	0.17	20.00	2.29	1.88	2.79	0.28	20.00	2.17	1.99	2.35	0.11	20.00
500×20	2.47	2.16	2.78	0.20	50.00	2.25	1.87	2.58	0.24	50.00	1.19	1.08	1.34	0.08	50.00
Mean	3.54	3.30	3.82	0.17	9.15	1.89	1.32	2.50	0.38	9.15	2.79	2.54	3.05	0.16	9.15

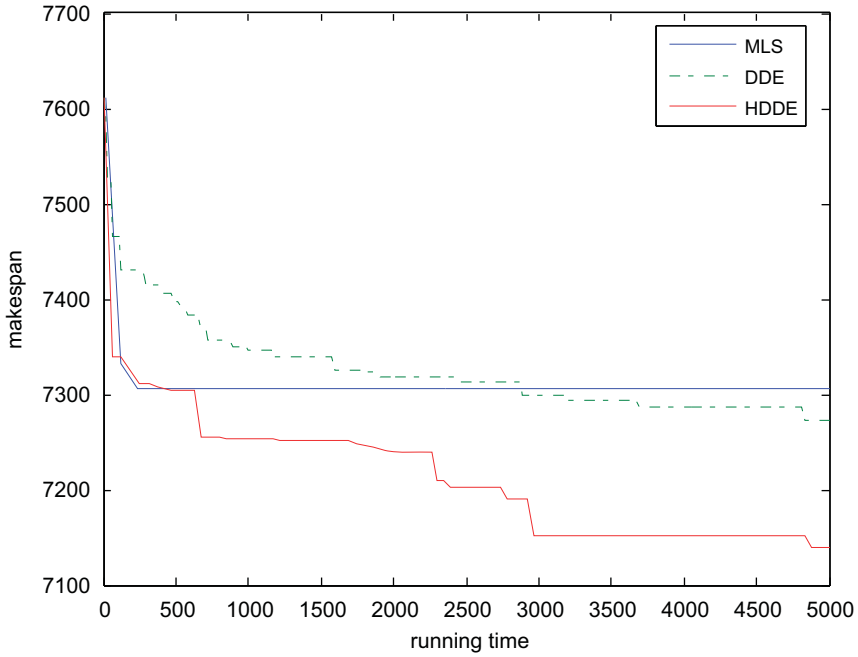

Fig. 7. Convergence rate cure of the MLS, DDE and HDDE algorithms for problem *ta* 81.

than both the TS and TS + M algorithms for all problem size and all instances as well. Even the MinPRD value produced by the HDDE algorithm is much better than the PRD value by the TS and TS + M algorithms. Especially, the HDDE algorithm is far superior to the TS and TS + M algorithms for problem sizes of 50×5 and 100×5. In terms of the computational time requirements, although the computer for the HDDE algorithm is about three times faster than the one used by Grabowski and Pempera [2], the average computation time of the HDDE algorithm is shorter than one third of that of both the TS and TS+M algorithms for each instances. It highlights the fact that the HDDE algorithm performed significantly better than the TS and TS+M algorithms. In addition, the mean SD value resulting from the HDDE algorithm is very small, demonstrating the robustness of the HDDE algorithm to the initialization.

Table 6 reports the makespan found by the HDDE algorithm. It should be noted that for 112 out of 120 instances, the HDDE algorithm has found better makespan values (highlighted in bold) than both the TS + M [2] and Ron's algorithms (here denoted as Ron) [4]. All these results confirm the favorable performance of the proposed HDDE algorithm over the TS and TS + M algorithms in terms of average percent relative deviation value and computational time as well. Hence, we can concluded that the HDDE algorithm is more effective and efficient than both the TS and TS + M algorithms for solving blocking flow shop scheduling problems with makespan criterion.

6.5. Effectiveness of combining the DDE and local search algorithms

The computational experiments and comparisons are conducted between the HDDE algorithm with the DDE and multi-start random local search (denoted as MLS) algorithms to show the effectiveness of combining the DDE-based global search and problem-dependent local search. The MLS algorithm is presented by removing the DDE-based search from the HDDE algorithm, and its parameters are set as follows: $PS = 20$, $P_l = 0.2$. The computational results are shown in Table 7.

It is easily observed from Table 7 that the HDDE algorithm is the winner since the results generated by the HDDE algorithm are significantly better than those by the DDE and MLS algorithms. To better

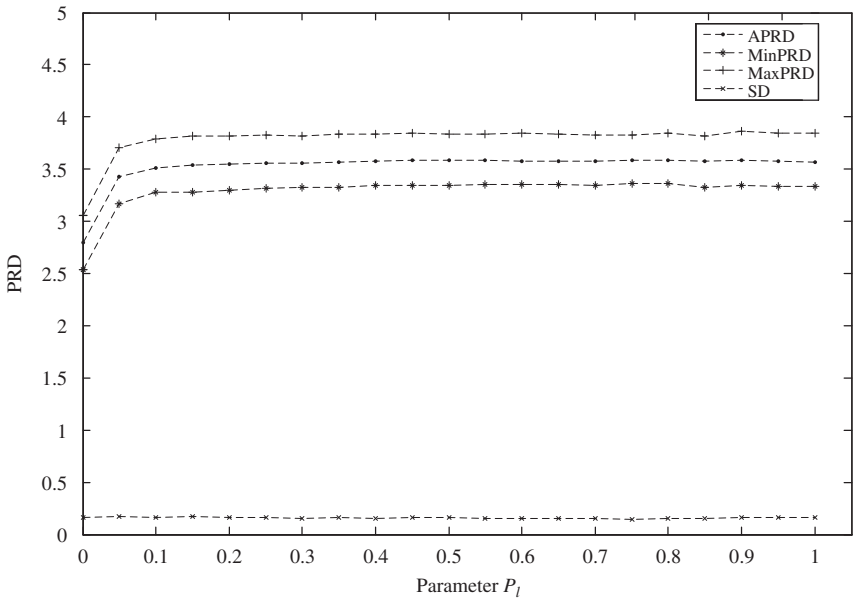

Fig. 8. Effect of the parameter P_l.

understand the performance of the HDDE algorithm, the typical convergence rate curves for these three algorithms based on benchmark instance *ta* 81 are shown in Fig. 7. It can be easily seen from Fig. 7 that the HDDE algorithm converges much faster to reach lower levels than both the DDE and MLS algorithms. The conclusion is similar for other benchmark instances.

Based on the above comparisons, it is concluded that the HDDE algorithm is superior to the DDE and MLS algorithms in terms of solution quality, robustness and convergence rate for the blocking flow shop scheduling problems with makespan criterion. This could be explained by the fact that in the HDDE algorithm, the DDE algorithm generates good start points for the local search algorithm by performing global exploration, while the local search algorithm further refines the obtained solutions by performing local exploitation, and guides the DDE algorithm to more promising search area. That is to say, the superiority in terms of searching quality and robustness of the HDDE algorithm should be attributed to the combination of global search and local search, i.e., the balance of exploration and exploitation.

6.6. Effect of the parameter P_l

Clearly, P_l is a key parameter for the HDDE algorithm. Hence, we further investigate the effect of P_l on solution quality. We set P_l from 0.05 to 1.0 with a step equal to 0.05 and fix other parameters. The similar computational experiments are conducted. The statistical results are presented in Fig. 8.

From Fig. 8, it follows that as P_l increases, the *APRD* value produced by the HDDE algorithm varies within a very small range. This suggested that P_l does not affect the searching quality of the HDDE algorithm too much. In other words, the HDDE algorithm is robust in regard to the parameter P_l.

7. Conclusions

By applying a job-permutation-based representation, a novel job-permutation-based mutation and crossover operators, and a problem-dependent local search, we first propose a novel hybrid discrete differential evolution (HDDE) algorithm for solving blocking flow shop scheduling problems with makespan criterion. Due to the effective hybridization of the differential-evolution-based global search and insert-neighborhood-based local search, the global exploration and local exploitation of the HDDE algorithm are well balanced. Furthermore, the efficiency of the HDDE algorithm is stressed by using a speed-up method to evaluate whole insertion neighborhood. Simulation results and comparisons demonstrated the superiority of the proposed HDDE algorithm in terms of solution quality, robustness and effectiveness. The future work is to apply the HDDE algorithm to other kinds of combinatorial optimization problems and develop multi-objective HDDE algorithms for multi-objective scheduling problems.

Acknowledgments

This research is partially supported by National Science Foundation of China under Grants 60874075, 70871065, 60834004, 60774082, National 863 Hi-Tech RandD Plan under Grant 2007AA04Z155, and Open Research Foundation from State Key Laboratory of Digital Manufacturing Equipment and Technology (Huazhong University of Science and Technology), the Project-sponsored by SRF for ROCS, SEM, and Postdoctoral Science Foundation of China under Grants 20070410791. QKP and PNS acknowledge the financial support offered by the A*Star (Agency for Science, Technology and Research, Singapore) under Grant #052 101 0020.

References

[1] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:278–85.
[2] Grabowski J, Pempera J. The permutation flow shop problem with blocking. A tabu search approach 2007;35:302–11.
[3] Qian B, Wang L, Huang DX, Wang WL, Wang X. An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. Computers and operations research 2009;36(1):209–33.
[4] Ronconi DP. A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking. Annals of Operations Research 2005;138(1):53–65.
[5] Pinedo M. Scheduling: theory, algorithms and systems. NJ: Prentice-Hall; 2002.
[6] Wang L, Zheng DZ. An effective hybrid heuristic for flow shop scheduling. International Journal of Advanced Manufacturing Technology 2003;21:38–44.
[7] Pan Q-K, Wang L. No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. International Journal of Advanced Manufacturing Technology 2008;39:796–807.

[8] Pan Q-K, Tasgetiren MF, Liang Y-C. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Computers and Operations Research 2008;35(9):2807–39.
[9] Ronconi DP. A note on constructive heuristics for the flowshop problem with blocking. International Journal of Production Economics 2004;87:39–48.
[10] Grabowski J, Pempera J. Sequencing of jobs in some production system. European Journal of Operational Research 2000;125:535–50.
[11] Hall NG, Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research 1996;44:510–25.
[12] Tonconi DP, Henriques LRS. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. OMEGA—International Journal of Management Science 2009;37(2):272–81.
[13] McCormich ST, Pinedo ML, Shenker S, Wolf B. Sequencing in an assembly line with blocking to minimize cycle time. Operations Research 1989;37:925–36.
[14] Leisten R. Flowshop sequencing problems with limited buffer storage. International Journal of Production Research 1990;28:2085–100.
[15] Ronconi DP, Armentano VA. Lower bounding schemes for flowshops with blocking in-process. Journal of the Operational Research Society 2001;52: 1289–97.
[16] Nawaz M, Enscore EEJ, Ham I. A heuristic algorithm for the m-machine, *n*-job flow shop sequencing problem. OMEGA—International Journal of Management Science 1983;11:91–5.
[17] Abadi INK, Hall NG, Sriskandarajh C. Minimizing cycle time in a blocking flowshop. Operations Research 2000;48:177–80.
[18] Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C. Minimizing makespan in a blocking flowshop using genetic algorithms. International Journal of Production Economics 2001;70:101–15.
[19] Storn R, Price K. Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Journal of Global Optimization 1997;11:341–59.
[20] Chang FP, Hwang C. Design of digital PID controllers for continuous-time plants with integral performance criteria. Journal of the Chinese Institute of Chemical Engineers 2004;35:683–96.
[21] Ilonen J, Kamarainen JK, Lampinen J. Differential evolution training algorithm for feed-forward neural networks. Neural Process Letters 2003;17:93–105.
[22] Storn R. Designing digital filters with differential evolution. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. London, UK: McGraw-Hill; 1999.
[23] Ruzek B, Kvasnicka M. Differential evolution in the earthquake hypocenter location. Pure and Applied Geophysics 2001;158:667–93.
[24] Qian B, Wang L, Hu R. et al. A hybrid differential evolution for permutation flow-shop scheduling. International Journal of Advanced Manufacturing Technology 2008;38(7–8):757–77.
[25] Onwubolu GC, Davendra D. Scheduling flow shops using differential evolution algorithm. European Journal of Operational Research 2006;171(2): 674–92.
[26] Tasgetiren MF, Pan QK, Suganthan PN, Liang YC. A discrete differential evolution algorithm for the no-wait flowshop problem with total flowtime criterion. In: Proceedings of the 2007 IEEE symposium on computational intelligence in scheduling. p. 251–8.
[27] Tasgetiren MF, Sevkli M, Liang YC. et al. A particle swarm optimization and differential algorithm for job shop scheduling problem. International Journal of Operations Research 2006;3:120–35.
[28] Qian B, Wang L, Huang DX, Wang X. An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. International Journal of Production Research 2009;36(1):209–33.
[29] Pan Q-K, Wang L. A novel differential evolution algorithm for the no-idle permutation flow shop scheduling problems. European Journal of Industrial Engineering 2008;2(3):279–97.
[30] Pan Q-K, Tasgetiren MF, Liang Y-C. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. In: Proceedings of the Ninth annual conference on genetic and evolutionary computation, London, England; 2007. p. 126–33.
[31] Tasgetiren MF, Pan Q-K, Liang Y-C. A discrete particle swarm optimization algorithm for the generalized traveling salesman problem. In: Proceedings of the ninth annual conference on genetic and evolutionary computation, London, England; 2007. p. 158–67.
[32] Grabowski J, Pempera J. Some local search algorithms for no-wait flow-shop problem with makespan criterion. Computers and Operations Research 2005;32:2197–212.
[33] Taillard E. Some efficient heuristic methods for the flow shop sequencing problems. European Journal of Operational Research 1990;47:65–74.
[34] Smutncki C. A two-machine permutation flow shop scheduling problem with buffers. OR Spectrum; 1998.
[35] Zhu Q-Y, Qin AK, Suganthan PN, Huang G-B. Evolutionary extreme learning machine. Pattern Recognition 2005;38(10):1759–63.