Hybrid Differential Evolution Optimization for No-Wait Flow-Shop Scheduling with Sequence-Dependent Setup Times and Release Dates

Bin Qian, Hua-Bin Zhou, Rong Hu, and Feng-Hong Xiang

Department of Automation, Kunming University of Science and Technology, Kunming 650051, China
bin.qian@vip.163.com

Abstract. In this paper, a hybrid algorithm based on differential evolution (DE), namely HDE, is proposed to minimize the total completion time criterion of the no-wait flow-shop scheduling problem (NFSSP) with sequence-dependent setup times (SDSTs) and release dates (RDs), which is a typical NP-hard combinatorial optimization problem with strong engineering background. Firstly, to make DE suitable for solving flow-shop scheduling problem, a largest-order-value (LOV) rule is used to convert the continuous values of individuals in DE to job permutations. Secondly, a speed-up evaluation method is developed according to the property of the NFSSP with SDSTs and RDs. Thirdly, after the DE-based exploration, a problem-dependent local search is developed to emphasize exploitation. Due to the reasonable balance between DE-based global search and problem-dependent local search as well as the utilization of the speed-up evaluation, the NFSSP with SDSTs and RDs can be solved effectively and efficiently. Simulation results and comparisons demonstrate the superiority of HDE in terms of searching quality, robustness, and efficiency.

Keywords: differential evolution, no-wait flow-shop scheduling, sequence-dependent setup times, release dates, local search, speed-up evaluation.

1 Introduction

Flow-shop scheduling problems (FSSPs) have attracted much attention and wide research in both computer science and operation research fields. In many FSSPs, each job is usually required to be processed continuously from start to end without waiting either on or between machines. This type of scheduling problem is known as no-wait flow-shop scheduling problem (NFSSP) [1]. In NFSSP, usually it assumes that the setup time is part of the job processing time and the release date of each job is zero. However, in some practical applications the setup times need to be explicitly treated and the release dates are nonzero. Typical situations are encountered in metal, chemical and pharmaceutical industries. The scheduling problems with sequence-dependent setup times (SDSTs) and release dates (RDs) have gained increasing attention in recent years [2]−[4]. For the total completion time criterion, the NFSSP with SDSTs

D.-S. Huang et al. (Eds.): ICIC 2011, LNCS 6838, pp. 600–611, 2011.

and RDs can be classified as $Fm/no-wait, ST_{sd}, r_j / \sum C_j$, which is NP-hard because it is more complex than the NFSSP with SDSTs [2][5].

Differential evolution (DE) is a novel population-based evolutionary mechanism proposed for global optimization over continuous spaces [6]. DE searches for the global optima by utilizing differences between contemporary population members, which allows the search behaviour of each individual to self-tune. Due to its simple structure, easy implementation, quick convergence, and robustness, DE has attracted much attention and wide applications in a variety of fields. Nevertheless, due to its continuous nature, the work on DE for production scheduling problems is obviously sparser than that for continuous optimization problems. Recently, Tasgetiren et al. [7] proposed a smallest position value (SPV) rule to convert the continuous values of individuals in DE to job permutations and incorporated Interchange-based local searcher into DE to solve flow-shop scheduling problems (FSSPs) with makespan criterion. Onwubolu and Davendra [8] presented a DE-based method to minimize the objectives of makespan, mean flowtime, and total tardiness of FSSPs. Qian et al. [9] developed a hybrid DE approach for FSSPs with makespan criterion. They utilized DE to find the promising solutions over the solution space and applied a simple problem dependent local searcher to exploit the solution space from those solutions. Pan et al. [10] presented a novel discrete differential evolution algorithm with a problem-specific referenced local searcher for FSSPs, which could find several new best known makespans for Taillard benchmark instances. Wang et al. [11] developed an effective discrete differential evolution algorithm for solving blocking FSSPs. To the best of our knowledge, there is no any published paper on DE for NFSSPs with SDSTs and RDs.

This paper develops a hybrid DE (HDE) by combining DE with local search to minimize the total completion time criterion of the NFSSPs with SDSTs and RDs. In the proposed HDE, DE is used to find the promising solutions or regions over the solution space, and then a local search based on the landscape of FSSP and the speed-up evaluation is conceived to exploit the solution space from those regions.

The remaining contents are organized as follows. In Section 2, the NFSSP with SDSTs and RDs is introduced. In Section 3, a brief review of DE is provided. In Section 4, HDE is proposed after presenting solution representation, speed-up evaluation method, DE-based global search, and problem-dependent local search. In Section 5, experimental results and comparisons are presented and analyzed. Finally, in Section 6, we end the paper with some conclusions and future work.

2 Formulation of NFSSP with SDSTs and RDs

The NFSSP with SDSTs and RDs can be described as follows. There are n jobs and m machines. Each of n jobs will be sequentially processed on machine $1,2,...,m$. The processing time of each job on each machine is deterministic. At any time, preemption

is forbidden and each machine can process at most one job. To satisfy the no-wait restriction, each job must be processed without interruptions between consecutive machines. Thus, all jobs are processed in the same sequence on all machines. In a flow-shop with SDSTs, setup must be performed between the completion time of one job and the start time of another job on each machine, and setup time depends on both the current and the immediately preceding jobs at each machine. In a flow-shop with RDs, if a machine is ready to process a job but the job has not been released yet, it stays idle until the release date of the job.

2.1 NFSSP with SDSTs

Let $\boldsymbol{\pi}=[j_1, j_2,..., j_n]$ denote the schedule or permutation of jobs to be processed, $p_{j_i,l}$ the processing time of job j_i on machine l, sp_{j_i} the total processing time of job j_i on all machines, $ML_{j_i,l}$ the minimum delay on the machine l between the completion of job j_{i-1} and j_i, L_{j_{i-1},j_i} the minimum delay on the first machine between the start of job j_{i-1} and j_i, $s_{j_{i-1},j_i,l}$ the sequence-dependent setup time between job j_{i-1} and j_i on machine l. Let $p_{j_0,l}=0$ for $l=1,...,m$. Then $ML_{j_i,l}$ can be calculated as follows:

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

Thus, the total completion time $C_T(\boldsymbol{\pi})$ (i.e., $C_T(\boldsymbol{\pi})=\sum C_j$) is as follows:

$$C_T(\boldsymbol{\pi})=\sum_{i=1}^{n}(n+1-i)ML_{j_i,m}. \tag{2}$$

Fig. 1 shows a small example of a NFSSP with SDSTs when n=3 and m=3.

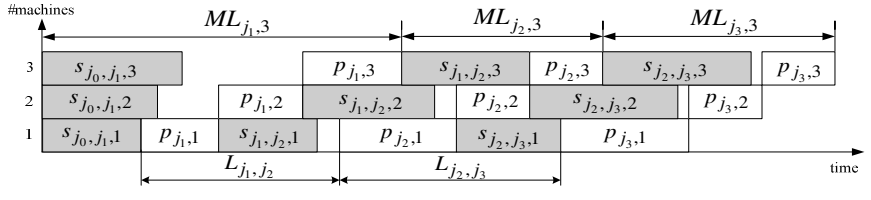

Fig. 1. NFSSP with SDSTs example when n=3 and m=3

Obviously, L_{j_{i-1},j_i} can be calculated by using the following formula:

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

2.2 NFSSP with SDSTs and RDs

Let r_{j_i} denote the arrival time of job j_i, St_{j_i} the process start time of job j_i on machine 1, and C_{j_i} the completion time of job j_i on machine m. Then St_{j_i} can be written as follows:

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

Thus, C_{j_i} and $C_T(\boldsymbol{\pi})$ can be calculated as follows:

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

$$C_T(\boldsymbol{\pi}) = \sum_{i=1}^{n} C_{j_i}. \tag{6}$$

The aim of this paper is to find a schedule $\boldsymbol{\pi}^*$ in the set of all schedules Π such that

$$\boldsymbol{\pi}^* = \arg\{C_T(\boldsymbol{\pi})\} \to \min \quad \forall \boldsymbol{\pi} \in \Pi. \tag{7}$$

3 Brief Review of DE

DE is a branch of metaheuristic proposed by Storn and Price [6] for optimization problems over continuous domains. The basic DE algorithm aims at finding the global optima by utilizing the distance and direction information according to the differentiations among population. The theoretical framework of DE is very simple and DE is easy to be coded and implemented with computer. Thus, nowadays DE has attracted much attention and wide applications in various fields.

In DE, it starts with the random initialization of a population of individuals in the search space and works on the cooperative behaviors of the individuals in the population. The searching behavior of each individual is adjusted by dynamically altering the differentiation's direction and step length in which this differentiation performs. At each generation, the *mutation* and *crossover* operators are applied on the individuals, and a new population arises. Then, *selection* takes place, and the corresponding individuals from both populations compete to comprise the next generation. Currently, there are several variants of DE algorithm can be found in [12]. The details of DE can be found in subsection 4.3.

4 HDE for NFSSP with SDSTs and RDs

In this section, we will present HDE for NFSSP with SDSTs and RDs after explaining the solution representation, speed-up computing method, DE-based global search, problem-dependent local search.

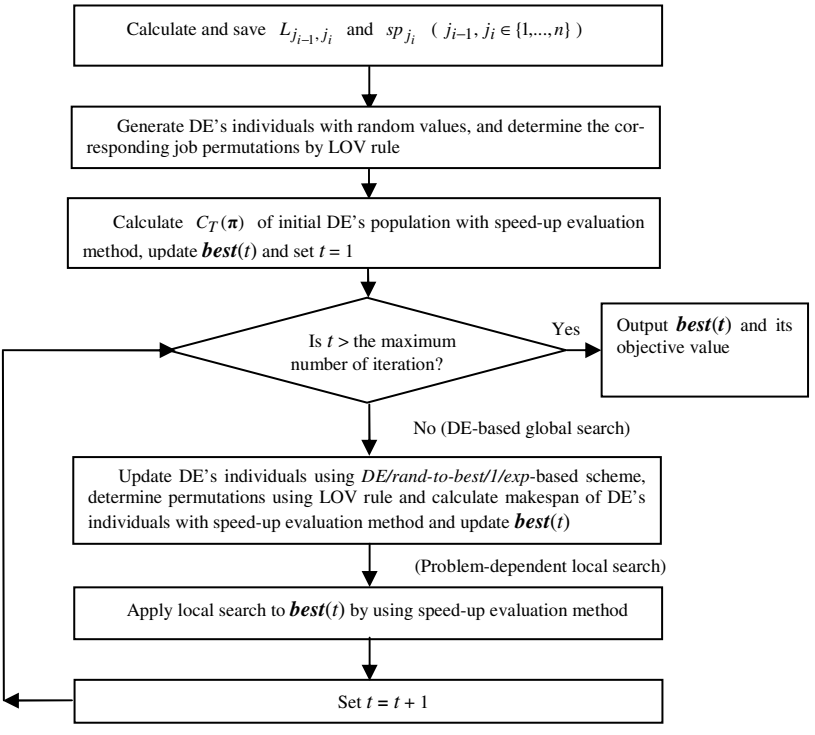

Fig. 2. The procedure framework of HDE

4.1 Solution Representation

The important problem in applying DE to NFSSP is to find a suitable mapping between job sequence and individuals (continuous vectors) in DE. For the *n*-job and *m*-machine problem, each vector contains n number of dimensions corresponding to n operations. In this paper, we adopt a largest-order-value (LOV) rule in [9] to convert DE's individual $X_i = [\, x_{i,1}, x_{i,2}, \cdots, x_{i,n} \,]$ to the job solution/permutation vector $\pi_i = \{\, j_{i,1}, j_{i,2}, \cdots, j_{i,n} \,\}$. According to LOV rule, $X_i = [\, x_{i,1}, x_{i,2}, \cdots, x_{i,n} \,]$ are firstly ranked by descending order to get the sequence $\varphi_i = [\, \varphi_{i,1}, \varphi_{i,2}, \cdots, \varphi_{i,n} \,]$. Then the job permutation π_i is calculated by the following formula:

$$j_{i,\varphi_{i,k}} = k \,. \tag{8}$$

To better understand the LOV rule, a simple example is provided in Table 1. In this instance ($n = 6$), when $k = 1$, then $\varphi_{i,1} = 4$ and $j_{i,\varphi_{i,1}} = j_{i,4} = 1$; when $k = 2$, then $\varphi_{i,2} = 1$ and $j_{i,\varphi_{i,2}} = j_{i,1} = 2$, and so on. This representation is unique and simple in terms of finding new permutations. Moreover, according to our previous tests, this representation is more effective then the famous random key representation [13].

Table 1. Solution Representation

Dimension k	1	2	3	4	5	6
$x_{i,k}$	1.17	3.95	0.23	2.61	2.72	0.77
$\varphi_{i,k}$	4	1	6	3	2	5
$j_{i,k}$	2	5	4	1	6	3

4.2 Speed-Up Evaluation Method

In NFSSPs with SDSTs and RDs, L_{j_{i-1},j_i} is only decided by the job j_{i-1} and j_i. According to this property, one method can be adopted to reduce the computing complexity (CC) of $C_T(\boldsymbol{\pi})$. That is, L_{j_{i-1},j_i}, sp_{j_i} and $\sum_{i=1}^{n} sp_{j_i}$ can be calculated and saved in the initial phase of HDE and then can be used as constant values in the evolution phase of HDE, which can reduce the CC of $C_T(\boldsymbol{\pi})$ from $O(nm)$ to $O(n)$. The procedure of the speed-up evaluation method is given as follows:

// Suppose that L_{j_{i-1},j_i}, sp_{j_i} and $\sum_{i=1}^{n} sp_{j_i}$ have already been calculated and //saved.

Step 1: Set *CT*=0;
Step 2: For i = 1 to n
Calculate St_{j_i} by using (4); // L_{j_{i-1},j_i} is a constant value in (4).
$CT = CT + St_{j_i}$; // $CT = \sum_{ll=1}^{i} St_{j_{ll}}$
End For i;
Step 3: $C_T(\boldsymbol{\pi}) = CT + \sum_{i=1}^{n} sp_{j_i}$; // $\sum_{i=1}^{n} sp_{j_i}$ is a constant value.

Remark. In Step 2, *CT* is utilized to save the current value of $\sum_{ll=1}^{i} St_{j_{ll}}$ in each loop.

4.3 DE-Based Global Search

In HDE, DE-based global search is designed based on *DE/rand-to-best/1/exp* scheme [9][12] to perform parallel exploration, in which base vector is the best individual of the current population. So, those individuals performing DE-based operation will share the information of the best individual of the population. In the *mutation* and *crossover* phase, each individual can transform probabilistically to any other individual in the solution space when the evolution generation $t \to \infty$. In the *selection* phase, only the better individual can be accepted. Therefore, the DE-based search can reach enough promising regions over the solution space.

Let ***best***(*t*) denote the best-so-far individual found by HDE until generation *t*. The procedure of DE-based global search is given as follows:

Step 1: Generate DE's individuals with random values, and determine the corresponding job permutations by LOV rule.
Step 2: Calculate $C_T(\boldsymbol{\pi})$ of initial DE's population, update ***best***(*t*) and set *t* = 1.
Step 3: Update DE's individuals using *DE/rand-to-best/1/exp*-based scheme and determine permutations using LOV rule.
Step 4: Calculate makespan of DE's individuals and update ***best***(*t*).
Step 5: Set *t*=*t*+1. If $t \le t_{\max}$ (the maximum number of iteration), then go to Step 3.
Step 6: Output ***best***(*t*) and its objective value.

Because of the parallel framework of DE, local search is easy to incorporate into DE to develop effective hybrid algorithms. Next, we will present a problem-dependent local search to perform exploitation.

4.4 Problem-Dependent Local Search

For the FSSPs, two effective neighborhoods are often used in the literature. (*i*) remove the job at the u th dimension and insert it in the v th dimension of the job solution $\boldsymbol{\pi}$ ($insert(\boldsymbol{\pi},u,v)$), (*ii*) interchange the job at the u th dimension and the job at the v th dimension of the job solution $\boldsymbol{\pi}$ ($interchange(\boldsymbol{\pi},u,v)$). Thus, we employ *insert* and *interchange* here as the neighborhood for local search.

Denote $X_i(t)=[x_{i,1},x_{i,2},\cdots,x_{i,n}]$ ($i=1,2,\cdots,popsize$) the *i*th individual in the *n*-dimensional search space at generation *t*. The procedure of the local search is given as follows:

Step 1: Convert individual $X_i(t)$ to a job permutation $\boldsymbol{\pi}_{i\_0}$ according to the LOV rule.
Step 2: Perturbation phase.
Randomly select u and v, where $u \ne v$; $\boldsymbol{\pi}_i = interchange(\boldsymbol{\pi}_{i\_0},u,v)$.
Step 3: Exploitation phase.
Set *loop*=1;
Do
Randomly select u and v, where $u \ne v$;
$\boldsymbol{\pi}_{i\_1} = insert(\boldsymbol{\pi}_i,u,v)$;
if $f(\boldsymbol{\pi}_{i\_1}) < f(\boldsymbol{\pi}_i)$, then $\boldsymbol{\pi}_i = \boldsymbol{\pi}_{i\_1}$;
loop++;
While *loop*<(*n*×(*n*-1)).
Step 4: If $f(\boldsymbol{\pi}_i) < f(\boldsymbol{\pi}_{i\_0})$, then $\boldsymbol{\pi}_{i\_0} = \boldsymbol{\pi}_i$.
Step 5: Convert $\boldsymbol{\pi}_{i\_0}$ back to $X_i(t)$.

As can be seen from the above procedure, step 2 is the perturbation phase, which is used to escape local optima and guide the search to a different region, and step 3 executes a thorough exploitation from the region obtained by step 2. Since the speed up computation method is adopted here, HDE has the ability to search more regions in the solution space under the same running time. So, instead of using the same neighborhood for perturbation and exploitation, the *interchange* neighborhood and the *insert* neighborhood are employed in step 2 and step 3, respectively, which can guide the search to a more different region and then improve the search ability of HDE to some extent. The effectiveness of adopting the *interchange* neighborhood as perturbation has already been validated in [14].

4.5 HDE

Based on the above solution representation, speed-up computing method, DE-based search, problem-dependent local search, the procedure framework of HDE is proposed in Fig. 2. It can be seen from Fig.2 that not only does HDE apply DE-based searching mechanism to execute exploration for all individuals to find promising regions or solutions within continuous region, but it also applies problem-dependent local search to perform exploitation for the best individual to improve the solution's quality in the permutation space. Because both exploration and exploitation are well stressed, it is expected to achieve good results for NFSSP with SDSTs and RDs. In the next section, we will investigate the performances of the proposed HDE.

5 Test and Comparisons

5.1 Experimental Setup

To test the performance of the proposed HDE for NFSSPs with SDSTs and RDs, computational simulation is carried out with 12 random instances. That is, the $n \times m$ combinations are: $\{30, 50, 70\} \times \{5, 10, 20\}$ and $\{100\} \times \{10, 20, 40\}$. The processing time $p_{j_i,l}$ and the setup time $s_{j_{i-1},j_i,l}$ are generated from a uniform distribution [1, 100] and a uniform distribution [0, 100], respectively. The job arrival time r_{j_i} is an integer that is uniformly generated in $[0, 150n\alpha]$, where the parameter α is used to control the jobs' arrival speeds. The values of α are set to 0, 0.2, 0.4, 0.6, 0.8, 1 and 1.5, respectively. Each instance at each α is independently run 30 times for every algorithm for comparison. Thus, we have a total of 84 different instances.

To evaluate the effectiveness of HDE, we carry out simulations to compare our HDE with several other scheduling algorithms, which are a modified simulated annealing algorithm with first move strategy (SAFM) [15], a hybrid DE for FSSP (HDE_FSSP) [9], HDE_FSSP with the speed-up evaluation method (HDE_FSSP_SP), and an iterated greedy heuristic (IG) [16]. According to [15], SAFM is superior to a famous simulated annealing algorithm proposed by Osman [17]. Based on our tests, SAFM is also more effective than a hybrid genetic algorithm [18]. Moreover, HDE_FSSP is an effective algorithm to address FSSPs [9] and IG is one of the state-of-the-art algorithms for solving FSSPs with SDSTs [16]. In our tests,

HDE, HDE_FSSP, and HDE_FSSP_SP use the same parameters as follows: the population size $popsize = 30$, the scaling factor F =0.7, and the crossover parameter CR =0.1. Thus, the difference between HDE and HDE_FSSP_SP only lies in the neighborhood used in step 2 (i.e., the perturbation phase) of local search. That is, the *interchange* neighborhood and the *insert* neighborhood are employed in the perturbation phase of HDE's and HDE_FSSP_SP's local search, respectively. We re-implement SAFM and IG according to [15] and [16], respectively. All algorithms used in the comparisons are re-implemented by ourselves and are coded in Delphi 7.0. Experiments are executed on an Intel Q8200 2.33GHz PC with 3 GB RAM.

5.2 Test Results and Comparisons

To investigate the performance of HDE, we set SAFM's maximum generation to $40 \times n$ and let all other compared algorithms run at the same time as SAFM. Denote $\boldsymbol{\pi}_{ini}(\alpha)$ the permutation in which jobs are ranked by descending value of job's release date at α, $C_T(\boldsymbol{\pi}(\alpha))$ the total completion time of permutation $\boldsymbol{\pi}(\alpha)$ at α, $avg\_C_T(\boldsymbol{\pi}(\alpha))$ the average value of $C_T(\boldsymbol{\pi}(\alpha))$, $best\_C_T(\boldsymbol{\pi}(\alpha))$ the best value of $C_T(\boldsymbol{\pi}(\alpha))$, $ARI(\alpha) = \left(C_T(\boldsymbol{\pi}_{ini}(\alpha)) - avg\_C_T(\boldsymbol{\pi}(\alpha))\right) / C_T(\boldsymbol{\pi}_{ini}(\alpha)) \times 100\%$ the average percentage improvement over $C_T(\boldsymbol{\pi}_{ini}(\alpha))$, $BEI(\alpha) = \left(C_T(\boldsymbol{\pi}_{ini}(\alpha)) - best\_C_T(\boldsymbol{\pi}(\alpha))\right) / C_T(\boldsymbol{\pi}_{ini}(\alpha)) \times 100\%$ the best percentage improvement over $C_T(\boldsymbol{\pi}_{ini}(\alpha))$, $ET(\alpha)$ the solution evaluation times at α (i.e., the times of calculating $C_T(\boldsymbol{\pi}(\alpha))$), $SD(\alpha)$ the standard deviation of $C_T(\boldsymbol{\pi}(\alpha))$ at α, S_α the set of all values of α, and $|S_\alpha|$ the number of different values in S_α. Then, we define four measures to evaluate the performances of the compared algorithms, i.e., $ARI = \frac{1}{|S_\alpha|} \sum_{\alpha \in S\alpha} ARI(\alpha)$, $BEI = \frac{1}{|S_\alpha|} \sum_{\alpha \in S\alpha} BEI(\alpha)$, $ET = \frac{1}{|S_\alpha|} \sum_{\alpha \in S\alpha} ET(\alpha)$, and $SD = \frac{1}{|S_\alpha|} \sum_{\alpha \in S\alpha} SD(\alpha)$. The statistical results of *ARI*, *SD*, *BEI*, and *ET* produced by the compared algorithms are reported in Table 2 and Table 3, respectively. Furthermore, in order to show the details of the compared algorithms for solving the instance with different α, we give the test results of the instance 70×20 in Table4. The test results of other instances with different α are similar to those in Table 4.

It can be seen from Table 2-4 that the *ARI* and *BEI* values obtained by HDE and HDE_FSSP_SP are obviously better than those obtained by SAFM, IG, and HDE_FSSP for almost all instances, which not only shows the superiority of HDE but also validates the effectiveness of utilizing speed-up evaluation method in DE. Meanwhile, the *ARI* and *BEI* values obtained by HDE are better than or very close to those obtained by HDE_FSSP_SP for all instances, which demonstrates the necessity of adopting the *interchange* neighborhood in the perturbation phase of local search. Moreover, the *SD* values of HDE are comparatively small, from which it is concluded that HDE is a robust algorithm. Besides, from Table 3, it is shown that the *ET* values of HDE and HDE_FSSP_SP increase quickly with the scale of problem, which is

helpful to search more regions/solutions and achieve better performance under the same amount of computer time. However, the local search of HDE is expensive with bigger problems, which means more effective local search method with low computing complexity needs further study.

Table 2. Comparison of *ARI* and *SD* of SAFM, IG, HDE_FSSP, HDE_FSSP_SP, HDE

Instance	SAFM		IG		HDE_FSSP		HDE_FSSP_SP		HDE	
n, m	*ARI*	*SD*	*ARI*	*SD*	*ARI*	*SD*	*ARI*	*SD*	*ARI*	*SD*
30, 5	17.084	1.351	20.207	0.601	**20.994**	0.732	20.877	0.723	20.775	0.796
30, 10	17.020	1.110	20.369	0.341	20.925	0.505	20.858	0.504	**20.928**	0.431
30, 20	12.903	0.804	14.851	0.433	15.412	0.412	15.452	0.396	**15.534**	0.378
50, 5	26.780	0.983	28.638	0.517	28.961	0.692	**29.042**	0.791	28.991	0.699
50, 10	22.042	0.825	23.846	0.592	24.395	0.612	24.506	0.697	**24.583**	0.565
50, 20	20.054	0.642	21.493	0.382	22.033	0.469	22.270	0.474	**22.295**	0.410
70, 5	27.512	0.741	28.036	0.509	28.920	0.646	29.412	0.605	**29.492**	0.564
70, 10	27.129	0.536	27.453	0.421	28.119	0.520	28.611	0.473	**28.621**	0.449
70, 20	22.229	0.548	22.960	0.346	23.471	0.486	24.024	0.403	**24.106**	0.367
100, 10	29.863	0.441	29.989	0.327	30.417	0.428	30.842	0.423	**30.854**	0.436
100, 20	27.196	0.423	27.328	0.350	27.828	0.451	28.147	0.371	**28.225**	0.326
100, 40	29.617	0.371	30.022	0.256	30.572	0.440	31.028	0.361	**31.058**	0.331
average	23.286	0.731	24.599	**0.423**	25.171	0.533	25.422	0.518	**25.455**	0.479

Table 3. Comparison of *BEI* and *ET* of SAFM, IG, HDE_FSSP, HDE_FSSP_SP, HDE

Instance	SAFM		IG		HDE_FSSP		HDE_FSSP_SP		HDE	
n, m	*BEI*	*ET*	*BEI*	*ET*	*BEI*	*ET*	*BEI*	*ET*	*BEI*	*ET*
30, 5	19.556	27743	21.225	24586	22.129	20210	22.095	19376	**22.188**	19316
30, 10	19.131	27624	21.316	24613	21.710	20541	**21.767**	21129	21.651	21146
30, 20	14.525	27444	15.753	24949	16.178	21542	16.142	25644	**16.190**	25708
50, 5	28.581	76496	29.608	70245	30.189	64977	30.420	69527	**30.506**	69693
50, 10	23.493	76239	25.045	70951	25.572	66892	**25.772**	80495	25.757	80424
50, 20	21.215	75975	22.126	72399	22.887	68605	23.057	106556	**23.126**	106615
70, 5	28.850	149504	29.071	150088	30.195	150082	30.432	312079	**30.695**	312148
70, 10	28.299	148997	28.386	149898	29.109	149873	29.529	371025	**29.530**	371279
70, 20	23.397	147924	23.599	148741	24.330	149364	24.708	499844	**24.778**	499659
100, 10	30.523	302329	30.577	310620	31.068	307923	31.458	763282	**31.528**	763471
100, 20	27.863	301525	27.810	311657	28.559	307781	28.731	1041850	**28.741**	1042891
100, 40	30.216	299744	30.442	305562	31.214	307356	**31.589**	1576192	31.567	1576145
average	24.637	138462	25.413	138692	26.095	136262	26.308	407250	**26.355**	**407375**

Table 4. Comparison of SAFM, IG, HDE_FSSP, HDE_FSSP_SP, HDE on instance 70×20

	SAFM		IG		HDE_FSSP		HDE_FSSP_SP		HDE	
α	*ARI* (α)	*SD* (α)	*ARI* (α)	*SD* (α)	*ARI* (α)	*SD* (α)	*ARI* (α)	*SD* (α)	*ARI* (α)	*SD* (α)
0	25.759	0.566	26.957	0.275	26.876	0.413	27.476	0.419	**27.618**	0.395
0.2	24.024	0.393	24.403	0.242	25.247	0.434	25.716	0.277	**25.856**	0.309
0.4	23.522	0.527	24.440	0.146	24.499	0.465	25.333	0.456	**25.389**	0.306
0.6	22.702	0.611	23.671	0.329	23.702	0.693	24.205	0.431	**24.258**	0.400
0.8	20.416	0.628	20.209	0.563	21.845	0.476	22.368	0.445	**22.468**	0.351
1.0	20.622	0.560	21.684	0.231	22.105	0.514	22.619	0.462	**22.669**	0.475
1.5	18.560	0.549	19.358	0.637	20.020	0.405	20.449	0.334	**20.481**	0.335
average	22.229	0.548	22.960	**0.346**	23.471	0.486	24.024	0.403	**24.106**	0.367

In conclusion, the above comparisons show that HDE is an effective approach with excellent quality and robustness for NFSSPs with SDSTs and RDs.

6 Conclusions and Future Research

This paper presented a hybrid differential evolution (HDE) for the no-wait flow-shop scheduling problem (NFSSP) with sequence-dependent setup times (SDSTs) and release dates (RDs). To the best of the current authors' knowledge, this is the first report on the application of differential evolution (DE) approach to solve the problem considered. The developed algorithm not only applied wide scatter search guided by the evolutionary mechanism of DE, but it also applied thorough local search guided by *interchange*-based perturbation and *insert*-based exploitation. Moreover, a speed-up evaluation method was designed to reduce the computing complexity of evaluation solutions. Thus, both the effectiveness of searching solutions and the efficiency of evaluating solutions were considered. Simulation results and comparisons with some prevail algorithms demonstrated the effectiveness and robustness of our proposed algorithm. For future research, we intend to study DE for other kinds of scheduling problems, such as dynamic job shop scheduling problem.

Acknowledgements. This research is partially supported by National Science Foundation of China (Grant No. 60904081), Applied Basic Research Foundation of Yunnan Province (Grant No. 2009ZC015X), Talent Introduction Foundation of Kunming University of Science and Technology (Grant No. KKZ3200903021), and 863 High Tech Development Plan (Grant No. 2007AA04Z193).

References

1. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Annals of Discrete Mathematics 5, 287–326 (1979)
2. Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y.: A Survey of Scheduling Problems with Setup Times or Costs. European Journal of Operational Research 187(3), 985–1032 (2008)
3. Oguz, C., Salman, F.S., Yalçin, Z.B.: Order Acceptance and Scheduling Decisions in Make-to-order Systems. International Journal of Production Economics 125(1), 200–211 (2010)
4. Urlings, T., Ruiz, R., Stützle, T.: Shifting Representation Search for Hybrid Flexible Flow-line Problems. European Journal of Operational Research 207(2), 1086–1095 (2010)
5. Ruiz, R., Allahverdi, A.: Some Effective Heuristics for no-wait Flowshops with Setup Times to Minimize Total Completion Time. Annals of Operations Research 156(1), 143–171 (2007)
6. Storn, R., Price, K.: Differential Evolution–a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11(4), 341–359 (1997)
7. Tasgetiren, M.F., Liang, Y.C., Sevkli, M., Gencyilmaz, G.: Differential Evolution Algorithm for Permutation Flowshop Sequencing Problem with Makespan Criterion. In: Proceedings of 4th International Symposium on Intelligent Manufacturing Systems, Sakarya, Turkey, pp. 442–452 (2004)
8. Onwubolu, G., Davendra, D.: Scheduling Flow Shops using Differential Evolution Algorithm. European Journal of Operational Research 171(2), 674–692 (2006)

9. Qian, B., Wang, L., Hu, R., Wang, W.L., Huang, D.X., Wang, X.: A Hybrid Differential Evolution for Permutation Flow-shop Scheduling. International Journal of Advanced Manufacturing Technology 38(7-8), 757–777 (2008)
10. Pan, Q.K., Tasgetiren, M.F., Liang, Y.C.: A Discrete Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem. Computers & Industrial Engineering 55(4), 795–816 (2008)
11. Wang, L., Pan, Q.K., Suganthan, P.N., Wang, W.H., Wang, Y.M.: A Novel Hybrid Discrete Differential Evolution Algorithm for Blocking Flow Shop Scheduling Problems. Computers & Operations Research 37(3), 509–520 (2010)
12. Price, K., Storn, R.: Differential Evolution (DE) for Continuous Function Optimization (2011), http://www.icsi.berkeley.edu/%7Estorn/code.html
13. Bean, J.C.: Genetic Algorithm and Random Keys for Sequencing and Optimization. ORSA Journal on Computing 6(2), 154–160 (1994)
14. Qian, B., Wang, L., Hu, R., Huang, D.X., Wang, X.: A DE-based Approach to no-wait Flow-shop Scheduling. Computers & Industrial Engineering 57(3), 787–805 (2009)
15. Ishibuchi, H., Misaki, S., Tanaka, H.: Modified Simulated Annealing Algorithms for the Flow Shop Sequencing Problem. European Journal of Operational Research 81(2), 388–398 (1995)
16. Ruiz, R., Stützle, T.: An Iterated Greedy Heuristic for the Sequence Dependent Setup Times Flowshop Problem with Makespan and Weighted Tardiness Objectives. European Journal of Operational Research 187(3), 1143–1159 (2008)
17. Osman, I.H., Potts, C.N.: Simulated Annealing for Permutation Flow-shop Scheduling. Omega-Int. J. Manage. S. 17(6), 551–557 (1989)
18. Wang, L., Zheng, D.Z.: An effective hybrid heuristic for flow shop scheduling. International Journal of Advanced Manufacturing Technology 21(1), 38–44 (2003)