

A Novel Differential Evolution Algorithm for TWET-NFSSP with SDSTs and RDs

Bin Qian^{1, a}, Hua-Bing Zhou^{1, b}, Rong Hu^{1, c} and Li-Ping Wu^{1, d}

¹ Department of Automation, Kunming University of Science and Technology, Kunming 650500, China

^abin.qian@vip.163.com, ^bzhouhuabing2007@yahoo.cn, ^cronghu@vip.163.com, ^dlipingwu1981@163.com

Keywords: differential evolution, no-wait flow-shop scheduling problem, sequence-dependent setup times and release dates, total weighted earliness/tardiness, speed-up method, special local search

Abstract. A novel differential evolution (DE) algorithm, namely DE_TWET, is presented to deal with the no-wait flow-shop scheduling problem (NFSSP) with sequence-dependent setup times (SDSTs) and release dates (RDs). The criterion is to minimize a total weighted earliness/tardiness (TWET) cost function. The presented algorithm is a hybrid of DE, problem's properties, and a special designed local search. In DE_TWET, DE is adopted to execute global search in the solution space, and the problem's properties are utilized to give a speed-up evaluation method and construct the local search, and the special local search is designed to enhance the local search ability of DE. Experimental results and comparisons demonstrate the effectiveness and robustness of the presented algorithm.

Introduction

With the development of *just-in-time* (JIT) manufacturing systems, the study on the scheduling problems with both earliness and tardiness (E/T) costs is of greater significance. In this paper, a typical production scheduling problem with strong engineering background [1,2], the no-wait flow-shop scheduling problem (NFSSP) with sequence-dependent setup times (SDSTs) and release dates (RDs), is considered, whose criterion is to minimize a total weighted earliness/tardiness (TWET) cost function. In such a case, each job j must be processed through all machines without any interruption, and both the setup times and the release dates need to be explicitly treated, and an optimal schedule is the one that all jobs finish exactly on their due dates. This type of model is classified as $Fm/no-wait, ST_{sd}, r_j / \sum (w'_j E_j + w''_j T_j)$, which can also be identified as TWET-NFSSP with SDSTs and RDs. Because $1/\sum T_j$ is NP-hard and it reduces to $Fm/no-wait, ST_{sd}, r_j / \sum (w'_j E_j + w''_j T_j)$ (i.e., $1/\sum T_j \propto Fm/no-wait, ST_{sd}, r_j / \sum (w'_j E_j + w''_j T_j)$), it can be concluded that $Fm/no-wait, ST_{sd}, r_j / \sum (w'_j E_j + w''_j T_j)$ is NP-hard [3]. Moreover, literature review show that the researches on the scheduling problems with both sequence-dependent setup times and release dates are very limited [4]. Thus, it is meaningful and practical to develop an effective algorithm for the considered problem.

Differential evolution (DE) algorithm, which was first designed for optimizing complex continuous problems [5], is one of the latest population-based evolutionary methods. Owing to its quick convergence and easy implementation, nowadays, the DE algorithm has gained many successful applications in different fields. However, due to DE's continuous nature, the applications of the DE-based algorithms to scheduling problems are still limited. Tasgetiren *et al.* [6] devised a DE-based algorithm for flow-shop scheduling problems (FSSPs) to minimize makespan. Onwubolu and Davendra [7] developed a DE-based approach for FSSPs, where makespan, mean flowtime, and total tardiness were considered. Qian *et al.* [8] designed an very efficient DE-based algorithm for

NFSSPs with the makespan criterion. Wang *et al.* [9] proposed an efficient discrete differential evolution algorithm for FSSPs with blocking. Recently, Hu *et al.* [10] presented a hybrid DE algorithm (DE_NTJ) for $Fm/no - wait, ST_{sd}, r_j / \sum U_j$, which is the current best approach for the problem considered. To the best of our knowledge, there has no promising results on $Fm/no - wait, ST_{sd}, r_j / \sum (w'_j E_j + w''_j T_j)$, and there has no published work addressing it by using DE-based algorithm.

In the current paper, a novel DE algorithm (DE_TWET) is proposed to deal with TWET-NFSSP with SDSTs and RDs. In our DE_TWET, firstly, a largest-order-value (LOV) in [11] is utilized to map the real-valued vectors or individuals in DE to job permutations so as to make DE suitable for solving NFSSP; secondly, a speed-up evaluation method based on the property of the considered scheduling problem is given to calculate the cost function efficiently; thirdly, the DE-based search is adopted to perform global exploration in the solution space and guide the whole search to the promising regions/solutions, while a special local search based on problem's properties is developed to emphasize exploitation from those regions. Test results and comparisons demonstrate the efficiency and robustness of the proposed DE_TWET.

The remainder of this paper are partitioned into four sections. Section 2 introduces the mathematical model of TWET-NFSSP with SDSTs and RDs. Section 3 presents DE_TWET in details. Section 4 provides and discusses test results and comparisons. Finally, Section 5 gives some concluding remarks and suggestions of future research.

TWET-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 $1, 2, \dots, m$. 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.

NFSSP with SDSTs. Let $\pi = [j_1, j_2, \dots, j_n]$ 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 $l = 1, \dots, m$. Then $ML_{j_i, l}$ can be calculated as follows:

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

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

$$L_{j_{i-1}, j_i} = ML_{j_i, m} + sp_{j_{i-1}} - sp_{j_i} \quad (2)$$

2.2 TWET-NFSSP with SDSTs. Denote r_{j_i} the arrival time of job j_i , St_{j_i} the process start time of job j_i on machine 1, C_{j_i} the completion time of job j_i on machine m , d_{j_i} the due date of job j_i , E_{j_i} the earliness of job j_i on machine m , T_{j_i} the tardiness of job j_i on machine m , and $TWET(\pi)$ the total weighted earliness/tardiness cost function. Then St_{j_i} can be written as follows:

$$St_{j_i} = \begin{cases} \max\{ML_{j_i,m} - sp_{j_i}, r_{j_i}\}, & i = 1 \\ St_{j_{i-1}} + \max\{L_{j_{i-1},j_i}, r_{j_i} - St_{j_{i-1}}\}, & i = 2, \dots, n \end{cases} \quad (3)$$

So, C_{j_i} , E_{j_i} , T_{j_i} , and $TWET(\pi)$ can be calculated as follows:

$$C_{j_i} = St_{j_i} + sp_{j_i}, i = 1, \dots, n, \quad (4)$$

$$E_{j_i} = \max(d_{j_i} - C_{j_i}, 0), i = 1, \dots, n, \quad (5)$$

$$T_{j_i} = \max(C_{j_i} - d_{j_i}, 0), i = 1, \dots, n, \quad (6)$$

$$TWET(\pi) = \sum_{i=1}^n (w'_{j_i} E_{j_i} + w''_{j_i} T_{j_i}), w'_{j_i} > 0, w''_{j_i} > 0. \quad (7)$$

The aim of this paper is to find a permutation π^* in the set of all permutations Π such that

$$TWET(\pi^*) = \min_{\pi \in \Pi} TWET(\pi). \quad (8)$$

DE_TWET for TWET-NFSSP with SDSTs and RDs

In this section, we will propose DE_TWET for TWET-NFSSP with SDSTs and RDs after explaining the solution representation, speed-up evaluation method, DE-based global search, and special designed local search.

Solution Representation. Owing to the continuous nature of the individuals in DE, the standard encoding scheme of DE cannot be directly applied to NTJ-NFSSP with SDSTs and RDs. In this paper, we adopt a largest-order-value (LOV) rule in [11] to map DE's i th individual $X_i = [x_{i,1}, x_{i,2}, \dots, x_{i,n}]$ to the job permutation vector $\pi_i = [j_{i,1}, j_{i,2}, \dots, j_{i,n}]$. According to LOV rule, $X_i = [x_{i,1}, x_{i,2}, \dots, x_{i,n}]$ are firstly ranked by descending order to get the sequence $\varphi_i = [\varphi_{i,1}, \varphi_{i,2}, \dots, \varphi_{i,n}]$. Then the job permutation π_i is calculated by the following formula:

$$j_{i,\varphi_{i,k}} = k. \quad (9)$$

According to our tests, LOV rule can achieve better results than the well-known random key representation.

Speed-up Evaluation Method. Based on the mathematical model of TWET-NFSSPs with SDSTs and RDs in Section 2, L_{j_{i-1},j_i} is only decided by the job j_{i-1} and j_i . By utilizing this property, one method can be adopted to reduce the computing complexity (CC) of $T_T(\pi)$. 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 DE_TWET and then can be used as constant values in the global and local search phase of DE_TWET, which can reduce the CC of $TWET(\pi)$ from $O(nm)$ to $O(n)$.

DE-based Global Search. Since many published works show that DE-based search has the strong ability of obtaining enough promising regions over the solution space, we use it to execute global search in the solution space. DE algorithm introduced by Storn and Price [5] is a branch of

population-based evolutionary algorithms for solving optimization problems over continuous domain. In the basic DE algorithm, it uses simple differential operator to create new candidate solutions and one-to-one competition strategy to select each new candidate. During its evolutionary phase, the searching behavior of each DE's individual is dynamically adjusted according to the differentiations among population. In our DE_TWET, DE/rand-to-best/1/exp scheme [12] is adopted to perform DE-based global search, in which base vector is the best individual of the present population. So, the information of the best individual can be utilized by all individuals in the population.

Special Designed Local Search.

Interchange-based Neighborhoods. Because *interchange* is an effective neighborhood in the published papers, we select it as the fundamental neighborhood for local search. Denote $interchange(\pi, u, v)$ the interchange of the job at the u th dimension (i.e., j_u) and the job at the v th dimension (i.e., j_v). The *interchange*-based neighborhood of π can be expressed as

$$N_{interchange}(\pi) = \{ \pi^{n,u,v} = interchange(\pi, u, v) \mid u = 1, \dots, n-1 \text{ and } v = u+1, \dots, n \}. \quad (10)$$

As for a fixed u , the subset of $N_{interchange}(\pi)$ can be written as

$$N_{interchange}(\pi, u) = \{ \pi^{n,u,v} = interchange(\pi, u, v) \mid v = u+1, \dots, n \}. \quad (11)$$

Thus, it has

$$N_{interchange}(\pi) = N_{interchange}(\pi, 1) \cup N_{interchange}(\pi, 2) \cup \dots \cup N_{interchange}(\pi, n-1). \quad (12)$$

Obviously, the sizes of $N_{interchange}(\pi)$ and $N_{interchange}(\pi, u)$ are $n(n-1)/2$ and $n-u$, respectively.

Speed-up Search Method for $N_{interchange}(\pi)$. Let $FindBestN_{interchange}(\pi)$ denote the search procedure of finding the best neighbor π_{best} in $N_{interchange}(\pi)$, and $\pi^{n,u,v} = [j'_1, j'_2, \dots, j'_u, \dots, j'_v, \dots, j'_n]$. When $u = 2, \dots, n-1$ and $i = 1, \dots, u-1$, it has $j_i = j'_i$, $St_{j_{u-1}} = St_{j'_{u-1}}$, and $\sum_{i=1}^{u-1} (w'_{j'_i} E_{j'_i} + w''_{j'_i} T_{j'_i}) = \sum_{i=1}^{u-1} (w'_{j_i} E_{j_i} + w''_{j_i} T_{j_i})$. Then, based on Eq. 4-Eq. 7, when $u > 1$, it has

$$\begin{aligned} TWET(\pi^{n,u,v}) &= \sum_{i=1}^{u-1} (w'_{j'_i} E_{j'_i} + w''_{j'_i} T_{j'_i}) + \sum_{i=u}^n (w'_{j'_i} E_{j'_i} + w''_{j'_i} T_{j'_i}) \\ &= \sum_{i=1}^{u-1} (w'_{j_i} E_{j_i} + w''_{j_i} T_{j_i}) + \sum_{i=u}^n (w'_{j'_i} E_{j'_i} + w''_{j'_i} T_{j'_i}). \end{aligned} \quad (13)$$

Therefore, in $FindBestN_{interchange}(\pi)$, St_{j_i} and $\sum_{l=1}^i (w'_{j_{ll}} E_{j_{ll}} + w''_{j_{ll}} T_{j_{ll}})$ ($i = 1, \dots, n-2$) can be calculated and saved before scanning or evaluating the neighbors in $N_{interchange}(\pi)$, and they can be used as constant values for evaluating $TWET(\pi^{n,u,v})$, which can reduce the CC of $FindBestN_{interchange}(\pi)$ to some extent. In other words, if $u > 1$, $St_{j'_{u-1}}$ and $\sum_{i=1}^{u-1} (w'_{j'_i} E_{j'_i} + w''_{j'_i} T_{j'_i})$ need not be computed and can be replaced with $St_{j_{u-1}}$ and $\sum_{i=1}^{u-1} (w'_{j_i} E_{j_i} + w''_{j_i} T_{j_i})$, respectively, and $St_{j'_u}$ can be computed directly from $St_{j_{u-1}}$. In DE_TWET's local search, the above speed-up search method is utilized in $FindBestN_{interchange}(\pi)$, which is denoted as $SP_FindBestN_{interchange}(\pi)$.

Two Strategies for $SP_FindBestN_{interchange}(\pi)$. Denote $SP_FindBestN_{interchange}(\pi, u)$ the search procedure of finding the best neighbor $\pi_{local_best}^u$ in $N_{interchange}(\pi, u)$ by using speed-up search method in subsection 3.4.2. The procedure of $SP_FindBestN_{interchange}(\pi)$ is given as follows:

Step 1: Set $u = 1$ and $\pi_{best} = \pi$;

Step 2: $\pi_{local_best}^u = SP_FindBestN_{interchange}(\pi, u)$;

Step 3: If $TWET(\pi_{local_best}^u) < TWET(\pi_{best})$, then $\pi_{best} = \pi_{local_best}^u$;

Step 4: If $u < n - 1$, then $u = u + 1$ and go to Step 2;

Step 5: Output π_{best} .

In order to reach quite different regions, two strategies are adopted in $SP_FindBestN_{interchange}(\pi)$.

The first one is a first move strategy. That is, if the first neighbor π_{first}^u in $N_{interchange}(\pi, u)$ that can improve π_{best} is obtained, $SP_FindBestN_{interchange}(\pi, u)$ terminates and outputs π_{first}^u . The second one is a promising region search strategy. That is, the permutation π in $SP_FindBestN_{interchange}(\pi, u)$ is replaced by π_{best} , which is helpful in executing the search immediately from the current promising regions found by $SP_FindBestN_{interchange}(\pi, u)$. Denote $FirstS_SP_FBN_{interchange}(\pi, u)$ the procedure of $SP_FindBestN_{interchange}(\pi, u)$ with the first move strategy. Then, a changed search procedure with the above two strategies, namely $TwoS_SP_FBN_{interchange}(\pi, KL)$, can be given as follows:

Step 1: Set $u = KL$ and $\pi_{best} = \pi$;

Step 2: $\pi_{local_best}^u = FirstS_SP_FBN_{interchange}(\pi_{best}, u)$;

Step 3: If $TWET(\pi_{local_best}^u) < TWET(\pi_{best})$, then $\pi_{best} = \pi_{local_best}^u$;

Step 4: If $u < n - 1$, then $u = u + 1$ and go to Step 2;

Step 5: Output π_{best} .

$TwoS_SP_FBN_{interchange}(\pi, KL)$ is the key element of DE_TWET's local search. KL is a variable in $TwoS_SP_FBN_{interchange}(\pi, KL)$, which is used to control the actual search range in $N_{interchange}(\pi)$ and can be set to any value in $\{1, \dots, n - 1\}$. Obviously, the larger value of KL is set to, the smaller search range in $N_{interchange}(\pi)$ is.

Procedure of Local Search. Let $insert(\pi, u, v)$ denote the insertion of j_u in the v th dimension of π . The procedure of DE_TWET's local search is given as follows:

Step 1: Convert DE's individual $X_i(t)$ to a job permutation π_{i_0} according to the LOV rule.

Step 2: Perturbation phase.

Set $\pi_{i_t} = \pi_{i_0}$.

For $kk = 1$ to KM

Randomly select ii and ll , where $|ii - ll| > n/3$;

$\pi_i = insert(\pi_{i_t}, ii, ll)$;

$\pi_{i_t} = \pi_i$;

End.

Step 3: Exploitation phase.

Set $loop = 0$;

repeat

$\pi_{i_1} = TwoS_SP_FBN_{interchange}(\pi_i, KL)$;

If $TWET(\pi_{i_1}) < TWET(\pi_i)$ then

$\pi_i = \pi_{i_1}$;

else

$loop++$;

end;

until $loop = 1$.

Step 4: If $TWET(\pi_i) \leq TWET(\pi_{i_0})$, then $\pi_{i_0} = \pi_i$.

Step 5: Convert π_{i_0} back to $X_i(t)$.

In the above procedure, Step 2 is the perturbation phase, which can avoid cycling and overcome local optima, and Step 3 executes exploitation from the region obtained by Step 2.

DE_TWET. According to the above solution representation, speed-up evaluation method, speed-up evaluation method, and special designed local search, the procedure of DE_TWET is presented as follows:

Step 0: Let t denote a generation, $Pop(t)$ a population with size N_p in generation t , $X_i(t)$ the i th individual with dimension N ($N = n$) in $Pop(t)$, $x_{i,l}(t)$ the l th variable of individual $X_i(t)$, tmp_l the l th variable of tmp , CR the crossover probability, and $random(0,1)$ the random value in the interval $[0,1]$. The objective value of each individual is calculated by using speed-up evaluation method.

Step 1: Input N , $N_p \geq 3$, $CR \in [0,1]$, let bounds be $lower(x_{i,l}) = 0$ and $upper(x_{i,l}) = 4$, $l = 1, \dots, N$.

Step 2: Calculate and save sp_{j_i} and L_{j_{i-1}, j_i} ($j_{i-1}, j_i \in 1, \dots, n$). //prepare for using speed-up
// evaluation method

Step 3: Population initialization.

$x_{i,l}(0) = lower(x_{i,l}) + random(0,1) * (upper(x_{i,l}) - lower(x_{i,l}))$, $l = 1, \dots, N$ for $i = 1, \dots, N_p$.

Step 4: Set $t=1$ and select an individual $X_{best}(0)$ from $Pop(0)$ as $best$ with the minimum objective value.

Step 5: Evolution phase (Step 5 through Step 11). Set $i=1$.

Step 6: Set the trial vector $tmp = X_i(t-1)$ and $L' = 0$. Randomly select $r1, r2 \in (1, \dots, N_p)$, where $r1 \neq r2 \neq i$, and randomly select $l \in (1, \dots, N)$.

Step 7: Perform DE's *Mutation* and *Crossover*.

Step 7.1: Let $tmp_l = tmp_l + F * (best_l - tmp_l) + F * (x_{r1,l}(t-1) - x_{r2,l}(t-1))$.

If $tmp_l < lower(x_{i,l})$, then let $tmp_l = 2 * lower(x_{i,l}) - tmp_l$.

If $tmp_l > upper(x_{i,l})$, then let $tmp_l = 2 * upper(x_{i,l}) - tmp_l$.

Step 7.2: Set $l = (l \bmod N) + 1$ and $L' = L' + 1$.

Step 7.3: If $(random(0,1) < CR)$ and $(L' < N)$, go to Step 7.1.

Step 8: Perform DE's *Selection*.

If $(f(tmp) \leq f(X_i(t-1)))$, then set $X_i(t) = tmp$;

else, set $X_i(t) = X_i(t-1)$.

Step 9: If $f(tmp) < f(best)$, then $best = tmp$.

Step 10: Set $i=i+1$. If $i \leq N_p$, then go to Step 6.

Step 11: Apply *special designed local search* to $best$.

Step 12: Set $t=t+1$. If $t \leq t_max$ (the maximum number of iteration), then go to Step 5.

Step 13: Output $best$ and its objective value.

It can be seen that DE_TWET not only applies the DE-based algorithm to obtain promising regions within the entire solution space, but also applies a special designed local search to enhance the search quality. Because both global and local search are well balanced, DE_TWET is expected to achieve good results.

Test Results and Comparisons

Experiment Setup. Some random generated instances with different scales are used to test the performance of DE_TWET. That is, the $n \times m$ combinations are: $\{20, 30, 50, 70, 100\} \times \{10, 20\}$. 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]. The job arrival time r_{j_i} is an integer that is randomly 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. Moreover, the due date of each job is specified as follows:

Step 1: For each problem p , randomly generate a permutation of the jobs.

Step 2: Calculate the completion time of each job in the permutation specified in Step 1.

Step 3: Specify the due date of each job by

$$d_{p,j_i} = C_{p,j_i} + \text{random}[-C_{p,j_i}, 0], \quad (14)$$

where d_{p,j_i} is the due date of job j_i to problem p , C_{p,j_i} is the completion time of job j_i to problem p , and $\text{random}[-C_{p,j_i}, 0]$ is a random value in the interval $[-C_{p,j_i}, 0]$. For each instance at each α , every algorithm independently run 20 replications for comparison. Thus, it has a total of 70 different instances.

For the purpose of evaluating the effectiveness of DE_TWET, we carry out simulations to compare our DE_TWET with an iterated greedy heuristic (IG) [13] and a hybrid DE (DE_NTJ) [10]. IG is the new state-of-the-art approach for solving FSSPs with SDSTs [13], and DE_NTJ is one of the current best algorithms for NFSSPs with SDSTs and RDs [10]. Moreover, we also compare DE_TWET with its five variants, whose abbreviations are as follows:

(1) DE_TWET_S1: In DE_TWET's local search, only speed-up evaluation method is used.

(2) DE_TWET_S12: In DE_TWET's local search, only speed-up evaluation method and speed-up search method are used.

(3) DE_TWET_noST2: In DE_TWET's local search, the promising region search strategy (i.e., strategy two) is not used.

(4) DE_TWET_0.3n: In Step 3 of DE_TWET's local search, KL is set to $\text{round}(0.3 * n)$, where $\text{round}(a)$ is a round function rounds a real-type value a to an integer-type value.

(5) DE_TWET_0.6n: In Step 3 of DE_TWET's local search, KL is set to $\text{round}(0.6 * n)$.

DE_TWET's parameters are set as follows: the population size $\text{popsize} = 30$, the scaling factor $F = 0.7$, the crossover parameter $CR = 0.1$, the local search variable $KM = 3$, and the local search variable $KL = 1$. Moreover, all values of w'_{j_i} and w''_{j_i} in $TWET(\pi)$ are set to 0.1. To make a fair comparison, all the compared algorithms use the same runtime limit of $2nm$ milliseconds as a termination criterion. We code all procedures in Delphi 7.0 and run all tests on an Intel Q8200 2.33GHz PC with 3 GB memory.

Performance Metrics. Denote $\pi_{ini}(\alpha)$ the permutation in which jobs are ranked by ascending value of job's release date at α , $TWET(\pi(\alpha))$ the total weighted earliness/tardiness of the permutation $\pi(\alpha)$ at α , $\text{avg_TWET}(\pi(\alpha))$ the average value of $TWET(\pi(\alpha))$, $\text{best_TWET}(\pi(\alpha))$ the best value of $TWET(\pi(\alpha))$, $\text{worst_TWET}(\pi(\alpha))$ the worst value of $TWET(\pi(\alpha))$, $ARI(\alpha) = (TWET(\pi_{ini}(\alpha)) - \text{avg_TWET}(\pi(\alpha))) / TWET(\pi_{ini}(\alpha)) \times 100\%$ the average percentage improvement over $TWET(\pi_{ini}(\alpha))$, $BEI(\alpha) = (TWET(\pi_{ini}(\alpha)) - \text{best_TWET}(\pi(\alpha))) / TWET(\pi_{ini}(\alpha)) \times 100\%$ the best percentage improvement over $TWET(\pi_{ini}(\alpha))$, $WRI(\alpha) = (TWET(\pi_{ini}(\alpha)) - \text{worst_TWET}(\pi(\alpha))) / TWET(\pi_{ini}(\alpha)) \times 100\%$ the worst percentage improvement over $TWET(\pi_{ini}(\alpha))$, $SD(\alpha)$ the standard deviation of $TWET(\pi(\alpha))$ 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 metrics to evaluate the performances of the compared algorithms, i.e., $ARI = \sum_{\alpha \in S_\alpha} ARI(\alpha) / |S_\alpha|$, $BEI = \sum_{\alpha \in S_\alpha} BEI(\alpha) / |S_\alpha|$, $WRI = \sum_{\alpha \in S_\alpha} WRI(\alpha) / |S_\alpha|$, and $SD = \sum_{\alpha \in S_\alpha} SD(\alpha) / |S_\alpha|$.

Comparisons of DE_TWET with Its Five Variants. In order to investigate DE_TWET's local search ability, we compare DE_TWET with its five variants. Statistical results can be found in Table 1. In Table 1, it can be seen from *ARI* metric that DE_TWET can obtain better results than its five variants for most instances, which manifest the importance of simultaneously adopting two speed-up methods and two strategies in local search and show the necessity of setting *KL* to the smallest value. From *SD* metric, it can be seen that the *SD* values of DE_TWET are smaller than those of five variants for almost all instances. This means DE_TWET is more robust than its five variants.

Table 1 Statistical results of testing DE_TWET and its variants

| Instance <i>n, m</i> | DE_TWET_S1 | | DE_TWET_S12 | | DE_TWET_noST2 | | DE_TWET_0.3n | | DE_TWET_0.6n | | DE_TWET | |
|-------------------------|------------|-----------|---------------|--------------|---------------|-----------|--------------|-----------|--------------|-----------|---------------|--------------|
| | <i>ARI</i> | <i>SD</i> | <i>ARI</i> | <i>SD</i> | <i>ARI</i> | <i>SD</i> | <i>ARI</i> | <i>SD</i> | <i>ARI</i> | <i>SD</i> | <i>ARI</i> | <i>SD</i> |
| 20, 10 | 45.827 | 0.728 | 46.662 | 0.247 | 46.569 | 0.323 | 43.119 | 1.749 | 39.501 | 1.888 | 46.655 | 0.207 |
| 20, 20 | 43.094 | 0.384 | 43.527 | 0.116 | 43.503 | 0.149 | 42.218 | 1.084 | 41.012 | 1.046 | 43.479 | 0.123 |
| 30, 10 | 51.787 | 1.015 | 53.547 | 0.420 | 53.522 | 0.345 | 44.417 | 3.743 | 41.091 | 2.365 | 53.570 | 0.246 |
| 30, 20 | 54.753 | 0.726 | 56.054 | 0.253 | 56.024 | 0.265 | 51.304 | 1.918 | 49.800 | 1.505 | 56.027 | 0.247 |
| 50, 10 | 58.414 | 1.644 | 61.389 | 0.442 | 61.266 | 0.420 | 42.194 | 5.772 | 39.124 | 2.778 | 61.581 | 0.334 |
| 50, 20 | 55.472 | 1.699 | 58.618 | 0.385 | 58.383 | 0.508 | 43.949 | 3.678 | 44.459 | 1.998 | 58.693 | 0.379 |
| 70, 10 | 57.447 | 2.245 | 62.067 | 0.526 | 61.164 | 0.971 | 28.760 | 6.178 | 28.003 | 3.873 | 62.414 | 0.406 |
| 70, 20 | 58.895 | 2.337 | 63.425 | 0.386 | 63.120 | 0.479 | 40.245 | 5.328 | 42.068 | 2.492 | 63.552 | 0.325 |
| 100, 10 | 62.876 | 1.555 | 65.642 | 0.680 | 64.044 | 1.547 | 23.096 | 5.885 | 24.262 | 4.065 | 66.508 | 0.405 |
| 100, 20 | 63.296 | 1.402 | 66.545 | 0.422 | 65.799 | 0.652 | 32.481 | 6.390 | 34.898 | 3.006 | 66.979 | 0.346 |
| average | 55.186 | 1.374 | 57.748 | 0.388 | 57.339 | 0.566 | 39.178 | 4.172 | 38.422 | 2.502 | 57.946 | 0.302 |

Comparisons of DE_TWET, IG, and DE_NTJ. To show the effectiveness of DE_TWET, we carry out some comparisons with IG [13] and DE_NTJ [10]. The test results of the three algorithms are shown in Table 2. From Table 2, it can be concluded that the searching quality of DE_TWET is better than that of IG for almost all instances and is superior or comparable to that of DE_NTJ. Therefore, it is concluded that DE_TWET is an effective and robust algorithm for dealing with TWET-NFSSP with SDSTs and RDs.

Table 2 Statistical results of testing DE_TWET, IG, and DE_NTJ

| Instance <i>n, m</i> | IG | | | | DE_NTJ | | | | DE_TWET | | | |
|-------------------------|---------------|---------------|---------------|--------------|---------------|---------------|---------------|--------------|---------------|---------------|---------------|--------------|
| | <i>BEI</i> | <i>ARI</i> | <i>WRI</i> | <i>SD</i> | <i>BEI</i> | <i>ARI</i> | <i>WRI</i> | <i>SD</i> | <i>BEI</i> | <i>ARI</i> | <i>WRI</i> | <i>SD</i> |
| 20, 10 | 46.846 | 46.014 | 45.213 | 0.497 | 46.875 | 46.641 | 45.905 | 0.263 | 46.875 | 46.655 | 46.201 | 0.207 |
| 20, 20 | 43.222 | 42.219 | 41.084 | 0.649 | 43.615 | 43.550 | 43.262 | 0.097 | 43.615 | 43.479 | 43.165 | 0.123 |
| 30, 10 | 53.838 | 52.655 | 51.019 | 0.857 | 54.035 | 53.614 | 53.127 | 0.254 | 54.005 | 53.570 | 53.095 | 0.246 |
| 30, 20 | 55.960 | 54.837 | 53.378 | 0.789 | 56.352 | 56.101 | 55.503 | 0.246 | 56.360 | 56.027 | 55.496 | 0.247 |
| 50, 10 | 61.668 | 60.824 | 59.848 | 0.497 | 61.900 | 61.351 | 60.613 | 0.346 | 62.153 | 61.581 | 60.917 | 0.334 |
| 50, 20 | 58.847 | 57.944 | 56.801 | 0.589 | 59.246 | 58.337 | 57.497 | 0.504 | 59.347 | 58.693 | 57.827 | 0.379 |
| 70, 10 | 62.987 | 61.984 | 60.873 | 0.566 | 62.311 | 60.904 | 59.288 | 0.737 | 63.240 | 62.414 | 61.654 | 0.406 |
| 70, 20 | 63.633 | 62.942 | 61.919 | 0.434 | 63.815 | 62.975 | 62.211 | 0.477 | 64.124 | 63.552 | 62.907 | 0.325 |
| 100, 10 | 67.143 | 66.527 | 65.953 | 0.389 | 66.667 | 66.011 | 65.397 | 0.410 | 67.138 | 66.508 | 65.743 | 0.405 |
| 100, 20 | 67.386 | 66.730 | 65.954 | 0.437 | 66.836 | 66.153 | 65.525 | 0.372 | 67.465 | 66.979 | 66.380 | 0.346 |
| average | 58.153 | 57.268 | 56.204 | 0.570 | 58.165 | 57.564 | 56.833 | 0.371 | 58.432 | 57.946 | 57.338 | 0.302 |

Conclusion and Future Research

To the best of the current authors' knowledge, this is the first report on the application of differential evolution (DE) algorithm for NFSSP with SDSTs and RDs with the criterion to minimize the total weighted earliness/tardiness. In our presented algorithm, DE-based global search was used to perform exploration for promising regions within the entire solution space, while a special local search based on problem's properties was developed to stress exploitation in these regions. Due to the hybridization of DE and local search, DE_TWET's search behavior can be enriched and its search ability can be enhanced. Simulations and comparisons showed the effectiveness and robustness of DE_TWET. The future work is to develop some effective DE-based algorithms for dynamical scheduling and reentrant scheduling.

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), and 2012 Academic and Technical Leader Candidate Project for Young and Middle-Aged Persons of Yunnan Province.

References

- [1] W.H.M. Raaymakers, J.A. Hoogeveen, Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing, *Eur. J. Oper. Res.* 126(1) (2000) 131–151.
- [2] F. Jin, S.J. Song, C. Wu, A simulated annealing algorithm for single machine scheduling problems with family setups, *Comput. Oper. Res.* 36(7) (2009) 2133–2138.
- [3] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, fourth ed., Springer, 2012.
- [4] A. Allahverdi, C.T. Ng, T.C.E. Cheng, M.Y. Kovalyov, A survey of scheduling problems with setup times or costs, *Eur. J. Oper. Res.* 187(3) (2008) 985–1032.
- [5] R. Storn, K. Price, Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces, *J. Glob. Optim.* 11(4) (1997) 341–359.
- [6] M.F. Tasgetiren, Y.C. Liang, M. Sevkli, G. Gencyilmaz, Differential evolution algorithm for permutation flowshop sequencing problem with makespan criterion, In *Proceedings of 4th International Symposium on Intelligent Manufacturing Systems*, Sakarya, Turkey, (2004) 442–452.
- [7] G. Onwubolu, D. Davendra, Scheduling flow shops using differential evolution algorithm, *Eur. J. Oper. Res.* 171(2) (2006) 674–692.
- [8] B. Qian, L. Wang, R. Hu, D.X. Huang, X. Wang, A DE-based approach to no-wait flow-shop scheduling, *Comput. Ind. Eng.* 57(3) (2009) 787–805.
- [9] L. Wang, Q.K. Pan, P.N. Suganthan, W.H. Wang, Y.M. Wang, A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems, *Comput. Oper. Res.* 37(3) (2010) 509–520.
- [10] R. Hu, X.H. Meng, B. Qian, K. Li, A differential evolution approach for NTJ-NFSSP with SDSTs and RDs, *Lecture Notes in Artificial Intelligence*, 7390 (2012) 288–299.
- [11] B. Qian, L. Wang, R. Hu, W.L. Wang, D.X. Huang, X. Wang, A hybrid differential evolution for permutation flow-shop scheduling, *Int. J. Adv. Manuf. Tech.*, 38(7-8) (2008) 757–777.
- [12] Information on <http://www.icsi.berkeley.edu/%7Estorn/code.html>
- [13] R. Ruiz, T. Stützle, An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, *Eur. J. Oper. Res.* 187(3) (2008) 1143–1159.