Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2011, Article ID 390593, 20 pages
doi:10.1155/2011/390593

Research Article

A Hybrid Differential Evolution and Tree Search Algorithm for the Job Shop Scheduling Problem

Rui Zhang[1] and Cheng Wu[2]

[1] *School of Economics and Management, Nanchang University, Nanchang 330031, China*
[2] *Department of Automation, Tsinghua University, Beijing 100084, China*

Correspondence should be addressed to Rui Zhang, r.zhang@ymail.com

Received 15 July 2011; Accepted 26 August 2011

Academic Editor: Furong Gao

The job shop scheduling problem (JSSP) is a notoriously difficult problem in combinatorial optimization. In terms of the objective function, most existing research has been focused on the makespan criterion. However, in contemporary manufacturing systems, due-date-related performances are more important because they are essential for maintaining a high service reputation. Therefore, in this study we aim at minimizing the total weighted tardiness in JSSP. Considering the high complexity, a hybrid differential evolution (DE) algorithm is proposed for the problem. To enhance the overall search efficiency, a neighborhood property of the problem is discovered, and then a tree search procedure is designed and embedded into the DE framework. According to the extensive computational experiments, the proposed approach is efficient in solving the job shop scheduling problem with total weighted tardiness objective.

1. Introduction

The job shop scheduling problem (JSSP) has been known as a very stubborn combinatorial optimization problem since the 1950s. In terms of computational complexity, JSSP is $\mathcal{NP}$-hard in the strong sense [1]. Therefore, even for very small JSSP instances, it is by no means easy to guarantee the optimal solution. In recent years, the metaheuristics—such as genetic algorithm (GA) [2, 3], tabu search (TS) [4, 5], particle swarm optimization (PSO) [6, 7], and ant colony optimization (ACO) [8, 9]—have clearly become the research focus in practical optimization methods for solving JSSPs.

Remarkably, most recent research concentrates on the hybridization of two or more heuristics to solve JSSP. For example, ACO is hybridized with TS in [10]; GA is hybridized with VND (variable neighborhood decent) in [11]; a hybrid algorithm based on PSO, VNS

(variable neighborhood search), and SS (scatter search) is proposed in [12] for solving a biobjective JSSP. The underlying motivation for constructing hybrid algorithms is that different search mechanisms and neighborhood structures are usually complementary to each other (i.e., when one fails, the other may be effective). Therefore, a combination of several optimizers is likely to promote the probability of finding optimal solutions. A common strategy is to hybridize an algorithm good at large-scale exploration (diversification) with an algorithm good at fine-scale exploitation (intensification), which provides reliable searching ability for tackling complex solution spaces.

In a general JSSP instance, a set of n jobs $\mathcal{J} = \{J_j\}_{j=1}^{n}$ are to be processed on a set of m machines $\mathcal{M} = \{M_k\}_{k=1}^{m}$ under the following basic assumptions.

(i) There is no machine breakdown.

(ii) No preemption of operations is allowed.

(iii) The transportation time and the setup time can be neglected.

(iv) Each machine can process at most one job at a time.

(v) Each job may be processed by at most one machine at a time.

Each job has a fixed processing route which traverses all the machines in a predetermined order, and the manufacturing process of a job on a machine is called an operation. The duration time of each operation is fixed and known. Besides, a preset due date d_j and a preset weight w_j are given for each job. Due date is the preferred latest finishing time of a job, so completion after this specific time will result in losses such as a worsened reputation in customers. Weights reflect the importance level of the orders from different customers, larger values suggesting higher strategic importance. If we use C_j to denote the completion time of job j, the objective function of JSSP can be makespan ($C_{\max} = \max_{j=1}^{n}\{C_j\}$), maximum lateness ($L_{\max} = \max_{j=1}^{n}\{C_j - d_j\}$), total weighted tardiness (TWT $= \sum_{j=1}^{n} w_j T_j$, where $T_j = \max\{0, C_j - d_j\}$), number of tardy jobs ($U = \sum_{j=1}^{n} U_j$, where $U_j = 1$ if $C_j > d_j$ and $U_j = 0$ otherwise), and so forth.

Up till now, most research on JSSP has been focused on the makespan criterion. However, due-date-related performances are becoming more significant in the make-to-order manufacturing environment nowadays. In this sense, the total weighted tardiness measure better reflects the critical factors that affect the profits of a firm. Meanwhile, from the theoretical perspective, total weighted tardiness is more difficult to optimize than makespan. According to the concepts of computational complexity, minimizing $C_{\max}$ is only a special case of minimizing TWT, which means the complexity of solving JSSP with total weighted tardiness objective is much greater than that of solving JSSP with makespan objective. Below we will use the abbreviation "TWT-JSSP" to denote the job shop scheduling problem with total weighted tardiness objective.

The rest of this paper is organized as follows. Section 2 provides a brief review on existing solution methods for TWT-JSSP and the differential evolution algorithm. Section 3 discusses the mathematical model of TWT-JSSP and a neighborhood property. Section 4 describes the design of a hybrid differential evolution algorithm for solving TWT-JSSP. Section 5 presents the computational results. Finally, Section 6 concludes the paper.

2. Literature Review

2.1. Existing Algorithms for TWT-JSSP

The contributions on TWT-JSSP are relatively rare in the literature. The only exact solution methodology is the branch-and-bound algorithm proposed by Singer and Pinedo [13], while all the following surveyed methods belong to the heuristic category.

Reference [14] presents efficient dispatching rules for sequencing the operations in TWT-JSSP, the most powerful one being the ATC (apparent tardiness cost) rule. In [15, 16], the authors propose modified shifting bottleneck heuristics, in which the subproblems are solved by dispatching rules (the basic ATC rule or the BATCS rule for complex job shops). The large step random walk (LSRW) algorithm is designed in [17] and aimed at the TWT-JSSP with release times ($Jm|r_j|\sum w_jT_j$). References [2, 3] present hybrid genetic algorithms for TWT-JSSP. Reference [18] presents a rolling horizon approach for TWT-JSSP, in which each time window is scheduled with a modified shifting bottleneck heuristic. Reference [19] presents a tabu search algorithm for the generalized TWT-JSSP with release times and precedence constraints. Recently, a new meta-heuristic called electromagnetic algorithm (EM) [20] has been explored and tested on TWT-JSSP.

Although these algorithms are effective, they have not fully utilized the inherent properties of TWT-JSSP. Hence, they will not perform satisfactorily when faced with large-scale instances of the problem. Exploring the structural properties (including neighborhood properties) of TWT-JSSP is an important research direction, which also constitutes the motivation of this study.

2.2. The Differential Evolution Algorithm

The differential evolution (DE) algorithm, which was first proposed by Storn and Price [21] in the mid-1990s, is a relatively new evolutionary optimizer. Characterized by a novel mutation operator, the algorithm has been found to be a powerful tool for continuous function optimization [22]. Due to its easy implementation, quick convergence and robustness, the DE algorithm is becoming increasingly popular in recent years. A wide range of successful applications have been reported, such as the design of fixed-structure robust controllers [23], space trajectory optimization [24], multiarea economic dispatch [25], and exergoeconomic analysis and optimization [26].

Despite the low computational complexity, DE has also been shown to have some weaknesses. In particular, DE is good at exploring the search space and locating the promising region, but it is slow at exploiting the high-quality solutions [27]. Recently, some researchers try to improve the performance of DE by hybridizing it with other local search-based algorithms. In [28], the authors propose the 2-Opt based DE (2-Opt DE) which is inspired by 2-Opt algorithms to accelerate DE. They show that 2-Opt DE can outperform the original DE in terms of solution accuracy and convergence speed. In [29], the authors incorporate the orthogonal design method into DE to accelerate its convergence rate. They show that the proposed approach outperforms the classical DE in terms of the quality, speed, and stability of the final solutions.

Due to its continuous feature, the traditional DE algorithm cannot be directly applied to scheduling problems with inherent discrete nature. Indeed, in canonical DE, each solution is represented by a vector of floating-point numbers. But for scheduling problems, each solution is a permutation of integers. To address this issue, two kinds of approaches can be identified in the literature.

(1) A transformation scheme is established to convert permutations into real numbers and vice versa [30, 31]. In this way, we only need to add a few lines to the encoding and decoding procedures, and it is not necessary to change the implementation of DE itself. The advantage is that the search mechanism of DE is well preserved, while the disadvantage is the redundancy in the mapping from real to permutation spaces.

(2) The mutation and crossover operators in DE are modified to suit the permutation representation [32, 33]. Clearly, the difficulty is how to redesign these operators such that the characteristics of high-quality solutions can be inherited and exploited. The design of operators should be problem dependent and thus requires a specific analysis of the optimization problem. Therefore, the lack of generality is a disadvantage of this approach.

Despite the success on permutation flow shop scheduling problems, the application of DE to JSSP has rarely been reported. The job shop model is more complex because the processing sequences of each machine need to be optimized separately. To tackle such a difficult problem, we design a tree-based local search module which can enhance the exploitation ability of DE. To our knowledge, this is the first attempt that DE is applied to TWT-JSSP.

3. The Mathematical Model and Neighborhood Properties of TWT-JSSP

3.1. The Mathematical Model and Its Duality

We utilize the concept of disjunctive graph for formulating TWT-JSSP [34]. In the graph $G(N, A, E)$, $N = O \cup \{0\} = \{0, 1, 2, \ldots, n \times m\}$ is the set of nodes, where $O = \{1, \ldots, n \times m\}$ corresponds to the operation set of the JSSP instance. Node 0 stands for a dummy operation which starts before all the real operations, while the starting time and the processing time of this dummy operation are both zero. A is the set of conjunctive arcs, and each conjunctive arc indicates the processing order of two operations belonging to the same job. Meanwhile, node 0 is connected with the first operation of each job with a separate conjunctive arc. In other words, if we use $F(O)$ to denote the set of the first operations of all jobs, then for all $f_j \in F(O)$, $(0, f_j) \in A$. $E = \bigcup_{k \in \mathcal{M}} E_k$ is the set of disjunctive arcs, where E_k represents the disjunctive arcs related with machine k. Each disjunctive arc connects two operations that should be processed by the same machine, but the processing order of the two operations is yet to be determined.

Under the disjunctive graph representation, TWT-JSSP can be described as a mixed-integer linear disjunctive programming model:

$$\begin{aligned}
\min \quad & \mathrm{TWT}(\mathbf{S}) = \sum_{u_j \in U(O)} w_{u_j} T_{u_j} \\
\text{s.t.} \quad & \text{(a) } s_{i_1} + p_{i_1} \leqslant s_{i_2} \quad \forall (i_1, i_2) \in A, \\
& \text{(b) } \left(s_{i_1} + p_{i_1} \leqslant s_{i_2}\right) \vee \left(s_{i_2} + p_{i_2} \leqslant s_{i_1}\right) \quad \forall \langle i_1, i_2 \rangle \in E_k,\ k = 1, \ldots, m, \\
& \text{(c) } T_{u_j} \geqslant s_{u_j} + p_{u_j} - d_{u_j} \quad \forall u_j \in U(O), \\
& \text{(d) } T_{u_j} \geqslant 0 \quad \forall u_j \in U(O).
\end{aligned} \tag{3.1}$$

In the above formulation, s_i denotes the starting time of operation i, and the decision variable $\mathbf{S}$ is a vector consisting of the starting times of all the operations. p_i denotes the required processing time of operation i. For the dummy operation 0, we set $s_0 = p_0 = 0$. u_j represents the last operation of job j, and thus $U(O) = \{u_1, u_2, \ldots, u_n\}$ denotes the set of ultimate operations of all the jobs. For any operation i, d_i and w_i, respectively, denote the due date and the weight of the job to which operation i belongs. For the ultimate operation u_j of job j, T_{u_j} is the tardiness of job j.

From the perspective of disjunctive graphs, finding a feasible solution to JSSP is equivalent to determining the directions of all the disjunctive arcs such that, for any $\langle i_1, i_2 \rangle \in E$, only one of the two disjunctive inequalities in constraint (b) is satisfied. Now, let us suppose the directions of all disjunctive arcs have been determined, and we use σ to denote the set of directed disjunctive arcs. In this situation, the problem (3.1) becomes a linear programming model, that is,

$$\begin{aligned}
\min \quad & \mathrm{TWT}_\sigma(\mathbf{S}) = \sum_{u_j \in U(O)} w_{u_j} T_{u_j} \\
\text{s.t.} \quad & \text{(a)}\ s_{i_2} - s_{i_1} \geqslant p_{i_1} \quad \forall (i_1, i_2) \in A, \\
& \text{(b)}\ s_{i_2} - s_{i_1} \geqslant p_{i_1} \quad \forall (i_1, i_2) \in \sigma, \\
& \text{(c)}\ T_{u_j} - s_{u_j} \geqslant p_{u_j} - d_{u_j} \quad \forall u_j \in U(O), \\
& \text{(d)}\ T_{u_j} \geqslant 0 \quad \forall u_j \in U(O).
\end{aligned} \tag{3.2}$$

As suggested by [19], the dual problem of the linear program (3.2) is a maximum cost flow problem:

$$\begin{aligned}
\max \quad & \mathrm{TC}_\sigma(\mathbf{F}) = \sum_{(i_1, i_2) \in A \cup \sigma} p_{i_1} F_{i_1, i_2} + \sum_{u_j \in U(O)} \left(p_{u_j} - d_{u_j} \right) F_{u_j, 0} \\
\text{s.t.} \quad & \text{(e)}\ F_{u_j, 0} \leqslant w_{u_j} \quad \forall u_j \in U(O), \\
& \text{(f)} \sum_{\xi : (\xi, i) \in A \cup \sigma} F_{\xi, i} - \sum_{\zeta : (i, \zeta) \in A \cup \sigma \cup U} F_{i, \zeta} = 0 \quad \forall i \in O, \\
& \text{(g)}\ F_{i_1, i_2} \geqslant 0 \quad \forall (i_1, i_2) \in A \cup \sigma \cup U.
\end{aligned} \tag{3.3}$$

In the above formulation, F_{i_1, i_2} represents the flow over the arc (i_1, i_2); $U = \{(u_1, 0), (u_2, 0), \ldots, (u_n, 0)\} = \{(u_j, 0)\}_{j=1}^{n}$ is a newly defined arc set, each arc pointing from the last operation of job j to node 0; TC denotes the total cost of the flows.

Theorem 3.1. *For the maximum cost flow problem* (3.3)*, there exists an optimal solution* $\mathbf{F}^*$ *which satisfies the following: each node (except node 0) has at most one incoming arc with nonzero flow.*

Proof. See Appendix A. □

3.2. A Neighborhood Property for the Swap of Adjacent Operations

We know from the previous subsection that, given a feasible σ (i.e., the directions of all disjunctive arcs), the schedule is determined, and meanwhile, a network flow graph is associated with the current schedule. The task of neighborhood search is to modify certain

parts of σ to get σ', so that the total weighted tardiness can be reduced, that is, $\mathrm{TWT}_{\sigma'}^{\min} < TWT_{\sigma}^{\min}$ (or equivalently, $\mathrm{TC}_{\sigma'}^{\max} < TC_{\sigma}^{\max}$).

Definition 3.2 (block). A sequence of operations in a critical path is called a block if (3.1) it contains at least two operations and (3.2) the sequence includes a maximum number of operations that are consecutively processed by the same machine.

If we want to improve the schedule under the current σ, we should only consider modifying the disjunctive arcs that satisfy the following two conditions: (1) the arc belongs to a certain block and (2) the arc carries positive flow in the dual network. The latter condition implies that this disjunctive arc is on the critical path of at least one tardy job, so altering this arc can possibly reduce the TWT.

Now we consider whether swapping two adjacent operations in a block can really improve the TWT. Under a given σ, suppose the operations $(1, 2, \ldots, z)$ constitute a block in the corresponding schedule, and the associated network flows are partially shown in Figure 1. The amount of flow on the outgoing disjunctive arc of operation i is denoted by x_i and suppose $x_i > 0$, for all $i = 1, 2, \ldots, z-1$. We assume this dual solution satisfies the condition described in Theorem 3.1, that is, each node can have at most one incoming arc with positive flow. In this case, the input flows of these nodes all come from the input disjunctive arcs, so the input conjunctive arcs of these nodes must carry zero flow and thus they are not marked in the figure. However, each of the nodes can still have an outgoing arc with positive flow. For simplicity, these outgoing arcs with possibly positive flow are drawn in solid arrows in the figure, and the amount of flow is denoted by $F_i \geqslant 0$, respectively.

After executing the SWAP operator, we can construct a new feasible solution to the dual problem by adjusting the amount of flows on each arc. In the new network, the flow on the outgoing disjunctive arc of operation i is denoted by y_i. In order to enable accurate analysis, we keep all the F_i values constant in the process of adjusting the local flows within the block (so that the flow equilibrium outside the considered scope will not be affected).

Theorem 3.3. *Suppose a block with consecutive positive flows contains the following operations:* $(1, 2, \ldots, \alpha, \beta, \ldots, z)$. *If the condition*

$$\frac{F_\alpha}{p_\alpha} \geqslant \frac{F_\beta}{p_\beta} \tag{3.4}$$

is satisfied, then swapping the two operations α *and* β *will not lead to improvement on the objective function (TWT).*

Proof. See Appendix B. □

Therefore, the function of Theorem 3.3 is that it helps to exclude some nonimproving moves in the local search process, so that the optimization efficiency can be improved. However, such a greedy mechanism must be combined with a large-scale randomized search (like DE) in order not to get trapped by local optima.

4. The Hybrid DE Algorithm for TWT-JSSP

4.1. The DE Optimization Framework

Like other evolutionary optimizers, DE is a population-based stochastic global optimizer. In DE, each individual in the population is represented by a D-dimensional real vector

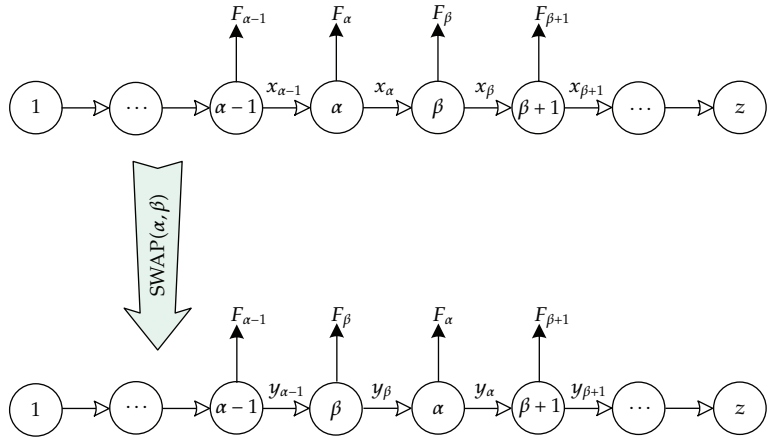

Figure 1: Illustration of the neighborhood operation SWAP(α, β).

$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$, $i = 1, \ldots, \mathrm{SN}$, where SN is the population size. In each iteration, DE employs the mutation and crossover operators to generate new candidate solutions and then applies a one-to-one selection policy to determine whether the offspring or the parent can survive to the next generation. This process is repeated until a preset termination criterion is met.

The standard DE algorithm can be described as follows.

Step 1 (Initialization). Randomly generate a population of SN solutions, $\{\mathbf{x}_1, \ldots, \mathbf{x}_{\mathrm{SN}}\}$.

Step 2 (Mutation). For $i = 1, \ldots, \mathrm{SN}$, generate a mutant solution $\mathbf{v}_i$ as follows:

$$\mathbf{v}_i = \mathbf{x}_{\mathrm{best}} + F \times (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}), \tag{4.1}$$

where $\mathbf{x}_{\mathrm{best}}$ denotes the best solution in the current population; r_1 and r_2 are randomly selected from $\{1, \ldots, \mathrm{SN}\}$ such that $r_1 \neq r_2 \neq i$; $F > 0$ is a weighting factor.

Step 3 (Crossover). For $i = 1, \ldots, \mathrm{SN}$, generate a trial solution $\mathbf{u}_i$ as follows:

$$u_{i,j} = \begin{cases} v_{i,j} & \text{if } \xi_j \leqslant \mathrm{CR} \quad \text{or } j = r_j, \\ x_{i,j} & \text{otherwise,} \end{cases} \qquad (j = 1, \ldots, D) \tag{4.2}$$

where r_j is an index randomly selected from $\{1, \ldots, D\}$ to guarantee that at least one dimension of the trial solution $\mathbf{u}_i$ differs from its parent $\mathbf{x}_i$; ξ_j is a random number generated from the uniform distribution $\mathcal{U}[0, 1]$; $\mathrm{CR} \in [0, 1]$ is the crossover parameter.

Step 4 (Selection). If $\mathbf{u}_i$ is better than $\mathbf{x}_i$, let $\mathbf{x}_i = \mathbf{u}_i$.

Step 5. If the termination condition is not satisfied, go back to Step 2.

According to the algorithm description, DE has three important parameters, that is, SN, F, and CR. In order to ensure a good performance of DE, the setting of these parameters should be reasonably adjusted based on specific optimization problems.

It is worth noting that there exist other variants of the DE algorithm with respect to the mutation and crossover method [35]. In fact, the above procedure is noted as "DE/best/1/bin" in the literature. If we change the mutation policy, we can obtain the "DE/rand/1/$*$" variant: $\mathbf{v}_i = \mathbf{x}_{r_1} + F \times (\mathbf{x}_{r_2} - \mathbf{x}_{r_3})$ where the base solution ($\mathbf{x}_{r_1}$) is also randomly chosen from the population.

4.2. The Encoding and Decoding Schemes

The encoding scheme used here is based on the random key representation and the smallest position value (SPV) rule. Each solution is described by $n \times m$ continuous values, and in the decoding process, this set of values will be transformed to a permutation of operations by the SPV rule.

Formally, let $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n\times m})$ denote the ith solution, where $x_{i,d}$ is the position value of the ith solution with respect to the dth dimension ($d = 1, \ldots, n \times m$). To decode a solution, the SPV rule is applied to sort the position values within a solution and then determine the relative positions of the corresponding operations. This process is exemplified in Table 1 for a problem containing 6 operations (suppose $N = \{1, \ldots, 6\}$). In this example, the smallest position value (–0.99) resides in the second dimension, and thus, the operation "2" comes first in the resulting sequence (the third row of the table). After sorting all these dimension values constituting solution $\mathbf{x}_i$, the operation sequence $\pi_i = (2, 5, 4, 1, 6, 3)$ can be obtained.

Then, the decoded operation sequence π_i can be used to build an active schedule for TWT-JSSP. The detailed schedule building algorithm is described as follows.

Input. An operation sequence π.

Step 1. Let $Q(1) = O = \{1, \ldots, nm\}$ (the set of all operations), $R(1) = F(O) = \{f_1, \ldots, f_n\}$ (the set of first operations of each job). Set $t = 1$.

Step 2. Find the operation $i^* = \arg\min_{i\in R(t)}\{r_i + p_i\}$, and let m^* be the index of the machine on which this operation should be processed. Use $B(t)$ to denote all the operations from $R(t)$ which should be processed on machine m^*.

Step 3. Delete from $B(t)$ the operations that satisfy $r_i \geqslant r_{i^*} + p_{i^*}$.

Step 4. Find the operation $\hat{o}$ which belongs to $B(t)$ and meanwhile ranks first in π. Schedule operation $\hat{o}$ on machine m^* at the earliest possible time.

Step 5. Let $Q(t+1) = Q(t) \setminus \{\hat{o}\}$, $R(t+1) = R(t) \setminus \{\hat{o}\} \cup \{\mathrm{JS}(\hat{o})\}$, where $\mathrm{JS}(\hat{o})$ is the immediate job successor of operation $\hat{o}$.

Step 6. If $Q(t+1) \neq \emptyset$, set $t \leftarrow t + 1$ and go to Step 2. Otherwise, the decoding procedure is terminated.

In the above description, the release time r_i equals the completion time of the immediate job predecessor of operation i. So $(r_i + p_i)$ is the earliest possible completion time of operation i. $Q(t)$ represents the set of operations yet to be scheduled at iteration t, while $R(t)$ represents the set of ready operations (whose job predecessors have all been scheduled) at iteration t.

Table 1: Illustration of the decoding process using SPV.

Dim. d	1	2	3	4	5	6
$x_{i,d}$	1.80	−0.99	3.01	0.72	−0.45	2.22
$\pi_{i,d}$	2	5	4	1	6	3

4.3. The Local Search Module Based on Tree Search

As mentioned in Section 2.2, DE alone does not have satisfactory "exploitation" ability. In order to cope with complex search spaces, it is usually required that a local search module be embedded into the general framework of DE. In this paper, we devise a tree-based local search optimizer by borrowing ideas from the filter-and-fan algorithm in [36]. In each iteration of DE, the local search is carried out for the best e% of solutions in the current population immediately after the selection phase. Thus, e is an important parameter for adjusting the frequency of local search and achieving a balance between exploration and exploitation.

In order to improve a selected solution, the proposed tree search algorithm generates compound neighborhoods for the solution based on the SWAP operator. The algorithm searches in a breadth-first manner, but unlike the beam search heuristic, each tree node represents a complete schedule rather than a partial schedule. The tree is gradually expanded by applying the SWAP operator iteratively. In each trial, the pair of operations to be swapped is randomly chosen from the critical blocks in the current schedule. To avoid repeated search, the reverse of the previous SWAP operator is prohibited. For example, if the current schedule is obtained by swapping (α, β), then a swap on (β, α) (if it is in the critical block again) should be tabooed in the immediate expansion from the current node. In the search process, we can utilize the relevant neighborhood property (Theorem 3.3) to promote the efficiency. In other words, when the theorem predicts that a certain move will not lead to improvement, then the action is canceled and another neighborhood move will be tried.

The proposed tree search algorithm is heuristic in nature, because it is computationally infeasible to enumerate all the possible swaps of the critical operations. Indeed, the algorithm only makes η_2 trails for each solution other than the "root" solution. Meanwhile, we need to apply a pruning strategy in the breadth-first search process in order to control the computational time. In particular, only the η_1 best solutions on each level of the tree will be exploited in the subsequent trials. We implement the tree search algorithm using the queue data structure as follows.

Input. A selected base solution σ.

Step 1. Let $l = 1$. Create an empty queue.

Step 2. Try applying the SWAP operator on η_1 different locations in the critical blocks of σ. Let the produced η_1 solutions enter the queue. Denote the best solution among these by σ^*.

Step 3. Perform the following steps for η_1 times.

(3.1) Take out the first solution in the queue, denoted by σ_f.

(3.2) Try applying the SWAP operator on η_2 different locations in the critical blocks of σ_f. Let the produced η_2 solutions enter the queue.

Step 4. Retain the best η_1 solutions in the queue, and delete the rest $\eta_1(\eta_2 - 1)$ ones. Denote the currently best solution in the queue as σ_l^*. If $\mathrm{TWT}(\sigma_l^*) < \mathrm{TWT}(\sigma^*)$, let $\sigma^* = \sigma_l^*$.

Step 5. Let $l \leftarrow l + 1$. If $l < L$, go back to Step 3.

Step 6. Output σ^*.

According the above description, the tree has η_1 nodes on level 1, each of which will be expanded to η_2 nodes on level 2. Thus, there are totally $\eta_1 \times \eta_2$ nodes on level 2. However, only the best η_1 nodes among these have a chance to be exploited and further expanded to level 3, while the rest will be abandoned. In this way, the number of nodes being considered on each level is controlled at $\eta_1 \times \eta_2$ and will not increase exponentially. After expanding to L levels, the algorithm is terminated.

For example, if we set $\eta_1 = 3$, $\eta_2 = 2$, and $L = 4$, the entire tree structure may look like Figure 2. The starting solution is denoted by σ_0. On each level from $l = 2$, three promising solutions are expanded further while the other three are discarded. The best solution found by this local search endeavor may appear on the last level.

The complexity of the tree search algorithm can be briefly analyzed as follows. There are roughly $\eta_1 \times L$ solutions that need to be expanded in the tree search process. For each expansion, the algorithm should first find the critical paths related with the tardy jobs. According to the Bellman's algorithm [37], this can be done in $O(nm)$ time. Then, the neighborhood operator has to be applied for approximately $\eta_1\eta_2 \times L$ times in the tree search process. Therefore, the computational complexity of the algorithm can be described as $O(\eta_1 Lnm + \eta_1\eta_2 L)$.

Compared with the extensive exploration behavior of DE, the proposed tree search algorithm works in a greedy manner. First, each neighborhood move is performed on the critical blocks of the corresponding schedule, because only such moves are possible to produce improvements. Second, the neighborhood property is utilized to exclude some unpromising moves when there are more than η_2 candidates to choose from. Third, on each level of the tree, only η_1 best solutions from the totally $\eta_1 \times \eta_2$ will be further considered. These features make the tree search algorithm extremely concentrated, which provides a complementary mechanism to DE's search process. Thus, using the tree search as an embedded local search module is beneficial for enhancing the exploitation capability and the overall performance of the hybrid DE.

5. The Computational Results

5.1. The Test Problems and Parameter Setting

In order to test the performance of the proposed hybrid DE (abbreviated as HDE hereinafter), randomly generated TWT-JSSP instances with different sizes are used in the computational experiment. For a specific problem size, the processing route of each job is a random permutation of the m machines, and the required processing time of each operation follows a uniform distribution $\mathcal{U}[1, 99]$ and takes only integer values. The due date of each job is set based on the total processing time of the job as $d_j = \lfloor u \times f \times \sum_{i \in O_j} p_i \rfloor$. In this expression, $u \sim \mathcal{U}[1, 1.1 \times \max\{1, n/m\}]$ is a random number uniformly distributed in the interval $[1, 1.1 \times \max\{1, n/m\}]$, and O_j denotes the set of operations that constitute job j. $f \in \{1.1, 1.3, 1.5\}$ is a coefficient that reflects the tightness level of the due date setting. The weight of each job (integer values) follows a uniform distribution, that is, $\mathcal{U}[1, 10]$.

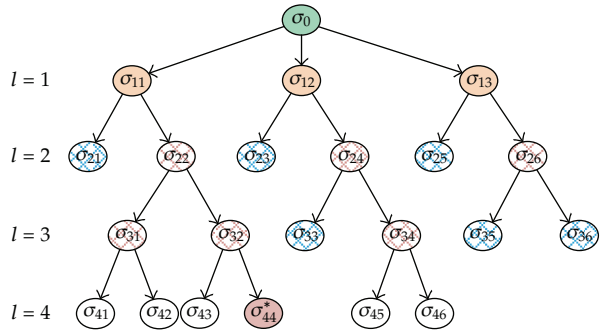

Figure 2: A possible tree structure under $\eta_1 = 3$, $\eta_2 = 2$, and $L = 4$.

In order to reduce random errors in the computation, we randomly generate 5 different instances for each due date tightness and scale. In particular, 10 problem sizes (from 100 operations to 500 operations) and 3 due date levels ($f = 1.5$ for loose due dates, $f = 1.3$, for moderate due dates and $f = 1.1$ for tight due dates) are tested, so the total number of instances is 150. The first two columns of Tables 2, 3 and 4 list the scales of the 10 instance sets. We compare the performance of the proposed HDE with the hybrid genetic algorithm GLS for TWT-JSSP [2]. The algorithms have been implemented using Visual C++ 2010 and tested on a platform of Intel Core i5-750 2.67 GHz, 3 GB RAM, and Windows 7.

In this experiment, the parameters of HDE are set as follows.

(i) The DE parameters: SN = 50, $F \sim \mathcal{U}(0.5, 1.0)$ (the "dither" strategy [38]), CR = 0.9.

(ii) The local search parameters: $e\% = 50\%$, $\eta_1 = 18$, $\eta_2 = 9$, $L = 14$.

(iii) Termination criterion: the best-so-far solution has not been updated for 50 generations (or controlled by the external computational time limit).

5.2. The Results and Discussions

The computational results are processed in the following way before listed in Tables 2, 3 and 4. For each instance i, HDE and GLS are, respectively, run for 10 independent times. The best objective value obtained in the 10 runs by algorithm a ($a \in \{\text{HDE}, \text{GLS}\}$) is denoted by $\text{TWT}^i_b(a)$, the worst denoted by $\text{TWT}^i_w(a)$, and the mean denoted by $\text{TWT}^i_m(a)$.

Next, we can calculate the relative objective values by taking $\text{TWT}^i_b(\text{HDE})$ as reference: $\text{RTWT}^i_b(a) = \text{TWT}^i_b(a)/\text{TWT}^i_b(\text{HDE})$, $\text{RTWT}^i_w(a) = \text{TWT}^i_w(a)/\text{TWT}^i_b(\text{HDE})$, $\text{RTWT}^i_m(a) = \text{TWT}^i_m(a)/\text{TWT}^i_b(\text{HDE})$.

Finally, when the above steps have been performed for each instance in the considered instance set, we calculate the average relative values (over this set) as $\overline{\text{RTWT}_b}(a) = (1/5)\sum_{i=1}^{5}\text{RTWT}^i_b(a)$, $\overline{\text{RTWT}_w}(a) = (1/5)\sum_{i=1}^{5}\text{RTWT}^i_w(a)$, and $\overline{\text{RTWT}_m}(a) = (1/5)\sum_{i=1}^{5}\text{RTWT}^i_m(a)$. In this way, the computational results are summarized in the tables with respect to each instance set.

Meanwhile, with respect to the first instance of each set under $f = 1.3$ (marked with "#-1" where # represents the index of instance sets), we record the average computational time (over 10 independent runs) of the two algorithms and plot the data as bar graphs in Figure 3.

Table 2: The computational results under loose due dates ($f = 1.5$).

Instance set no.	Size ($n \times m$)	HDE			GLS		
		$\overline{\text{RTWT}_b}$	$\overline{\text{RTWT}_w}$	$\overline{\text{RTWT}_m}$	$\overline{\text{RTWT}_b}$	$\overline{\text{RTWT}_w}$	$\overline{\text{RTWT}_m}$
1	10 × 10	1.000	1.150	1.021	1.002	1.187	1.052
2	20 × 5	1.000	1.128	1.085	1.004	1.136	1.073
3	10 × 20	1.000	1.149	1.099	1.008	1.144	1.117
4	20 × 10	1.000	1.129	1.065	1.010	1.113	1.098
5	20 × 15	1.000	1.120	1.054	1.000	1.118	1.075
6	50 × 6	1.000	1.126	1.072	1.017	1.103	1.087
7	20 × 20	1.000	1.115	1.034	1.013	1.119	1.068
8	40 × 10	1.000	1.119	1.062	1.027	1.122	1.076
9	50 × 10	1.000	1.103	1.027	1.015	1.130	1.100
10	100 × 5	1.000	1.121	1.078	1.026	1.126	1.092

Table 3: The computational results under moderate due dates ($f = 1.3$).

Instance set no.	Size ($n \times m$)	HDE			GLS		
		$\overline{\text{RTWT}_b}$	$\overline{\text{RTWT}_w}$	$\overline{\text{RTWT}_m}$	$\overline{\text{RTWT}_b}$	$\overline{\text{RTWT}_w}$	$\overline{\text{RTWT}_m}$
1	10 × 10	1.000	1.181	1.090	1.085	1.248	1.124
2	20 × 5	1.000	1.146	1.099	1.084	1.266	1.151
3	10 × 20	1.000	1.165	1.117	1.010	1.205	1.144
4	20 × 10	1.000	1.200	1.153	1.005	1.306	1.178
5	20 × 15	1.000	1.218	1.101	1.094	1.279	1.154
6	50 × 6	1.000	1.164	1.115	1.095	1.252	1.139
7	20 × 20	1.000	1.184	1.110	1.087	1.324	1.167
8	40 × 10	1.000	1.166	1.086	1.086	1.214	1.118
9	50 × 10	1.000	1.147	1.096	1.010	1.262	1.130
10	100 × 5	1.000	1.195	1.086	1.080	1.329	1.159

Table 4: The computational results under tight due dates ($f = 1.1$).

Instance set no.	Size ($n \times m$)	HDE			GLS		
		$\overline{\text{RTWT}_b}$	$\overline{\text{RTWT}_w}$	$\overline{\text{RTWT}_m}$	$\overline{RTWT_b}$	$\overline{\text{RTWT}_w}$	$\overline{\text{RTWT}_m}$
1	10 × 10	1.000	1.235	1.112	1.092	1.249	1.123
2	20 × 5	1.000	1.207	1.113	1.083	1.253	1.158
3	10 × 20	1.000	1.262	1.115	1.028	1.343	1.132
4	20 × 10	1.000	1.235	1.135	1.091	1.318	1.181
5	20 × 15	1.000	1.252	1.139	1.033	1.274	1.179
6	50 × 6	1.000	1.213	1.119	1.061	1.329	1.180
7	20 × 20	1.000	1.219	1.126	1.099	1.329	1.195
8	40 × 10	1.000	1.180	1.124	1.097	1.367	1.216
9	50 × 10	1.000	1.194	1.137	1.023	1.359	1.213
10	100 × 5	1.000	1.256	1.162	1.010	1.324	1.209

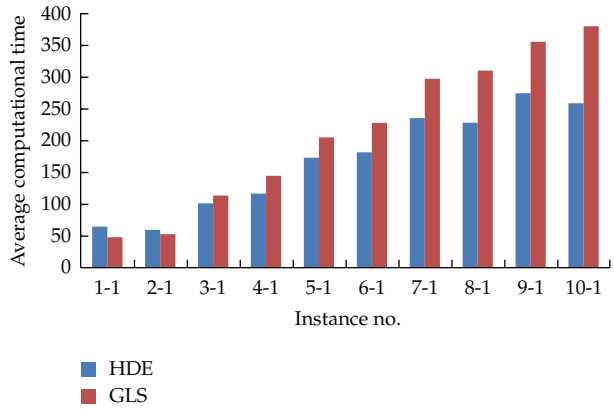

Figure 3: Comparison of computational time.

The following comments can be made according to the results displayed in Tables 2–4 and Figure 3.

(1) For most instances, the gap between the best and the worst solutions obtained by HDE is smaller than the gap obtained by GLS, which implies that HDE performs more robustly in different executions and for different scheduling instances.

(2) When the due dates are tighter ($f = 1.3$ or 1.1), we find that the solution quality of GLS is considerably inferior to that of HDE. This is because under tight due dates, many jobs are prone to be tardy, and the optimization difficulty increases systematically. In this situation, the proposed tree search mechanism exhibits greater advantage. The search is more effective because the neighborhood moves are conducted on the critical blocks and a part of unpromising moves are eliminated with simple calculations. To certain extent, HDE has overcome the blindness of traditional local search, and thus it can access the high-quality solutions with larger probability.

(3) By comparing the computational time, we see that, for most instances, the consumed time of HDE is less than that of GLS. More remarkably, the increasing speed of computational time with instance size is slower on the part of HDE. This is because the mutation and crossover based on random key representation in HDE is more efficient than the crossover and mutation based on operation sequence representation in GLS. Meanwhile, the phenomenon also verifies the fact that the tree-based local search is able to accelerate the convergence of DE (note that the termination criterion of HDE is decided by the convergence status).

5.3. Influence of the Parameter Settings

The following experiments are designed for observing the impact of parameter settings on the final solution quality of HDE. A 10×10 instance with $f = 1.3$ is used in these experiments. The termination criterion adopted here is an exogenously given computational time limit. When one parameter is being tested, the other parameters are all fixed at their recommended values given in Section 5.1.

The population size (SN) is one of the key parameters for the HDE algorithm. If SN is large, many solutions need to be maintained in the population, which increases the computational burden of solution decoding and evaluation. If SN is small, more generations can be implemented within the DE framework, but the limited population diversity will impair the effectiveness of mutation and crossover. Therefore, under a given computational time limit, the selection of SN can influence the overall optimization performance. In this experiment, we fix the available computational time at three different levels and then identify the most suitable value of SN in each scenario. The time limit is set as 20 sec (tight), 40 sec (moderate), and 60 sec (loose), respectively. The computational results are displayed in Figure 4, where the vertical axis gives the average objective value obtained from 20 independent executions of the proposed HDE under each SN.

According to the results, the selection of SN has a noticeable impact on the solution quality. Generally, the solution quality will deteriorate if SN is either too small or too large. But the best SN varies with different time constraints. If the time budget is tight or moderate, the best population size is $40 \sim 50$. If the time resource is abundant, the best population size is $60 \sim 70$. Thus, setting SN = 50 is reasonable for ordinary uses.

Another important parameter for HDE is e%, the percentage of solutions that undergo the local search process. A reasonable selection of e will result in an effective balance between exploration and exploitation. The computational results for this experiment are displayed in Figure 5.

According to the results, the setting of e has a considerable impact on the solution quality, especially when the computational time is scarce (20 sec). A small e means that only a few solutions in each generation can be improved by the local search, which has little effect on the entire population. A large e suggests that too much time is consumed on local search, which may reduce the normal function of DE.

The best setting of e under each constraint level is 70 (for tight time budget), 60 (for moderate time budget) and 40 (for loose time budget). When the exogenous restriction on computational time is tight, DE has to rely on frequent local search to find good solutions. This is because, in the short term, the tree-based local search is more efficient than DE's mechanism (mutation and crossover) in improving a solution. However, the price to pay is possibly a premature convergence of the optimization process due to the greedy nature of the tree search. On the other hand, when the computational time is sufficient, DE will prefer a larger number of generations to conduct a systematic exploration of the solution space. In this case, the local search need not be used very frequently, otherwise the steady searching process may be disturbed. Overall, a recommended value for e is 50.

Finally, we focus on the local search parameters η_1, η_2, and L. Following the suggestion of [36], we fix the relationship between η_1 and η_2 as $\eta_1 = 2\eta_2$ and then test the influence of L and η_1 on the final solution quality. The tested ranges are $L \in \{8, 10, 12, \ldots, 24\}$, $\eta_1 \in \{10, 12, 14, \ldots, 26\}$, with a step size of 2, which leads to 81 combinations. Again, we choose the first instance of each set (#-1) under $f = 1.3$ for this experiment. The computational time limit is set as 0.6 nm sec for an $n \times m$ instance. The results are displayed as relative values in Figure 6.

The following comments can be made according to the results.

(1) The tree search parameters produce a remarkable effect on the final solutions of HDE, which confirms that such a local search optimizer is effective in improving the performance of DE. When L and η_1 both take small values, the tree search process does not have chance to examine a sufficient number of neighborhood solutions

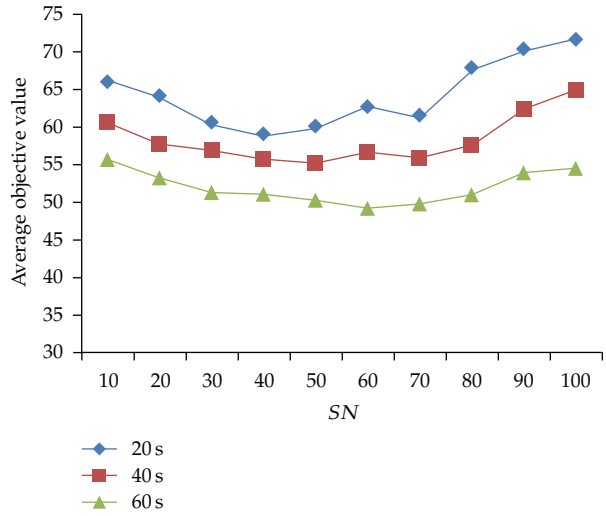

Figure 4: The influence of parameter SN.

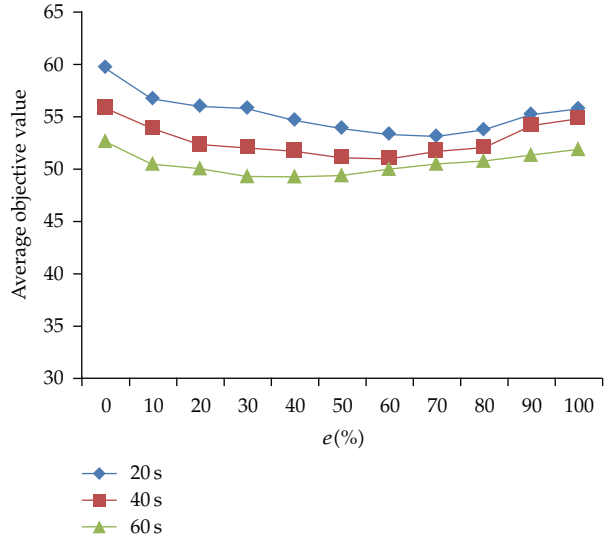

Figure 5: The influence of parameter e.

before termination, which results in solution quality decline. On the other hand, when L and η_1 are both set large, the overall performance also worsens because the exploitation is consuming too much time and crowds out the exploration process of DE.

(2) The value of L (representing the depth of the search tree) and the value of η_1 (representing the width of the search tree) should be coordinated in order to guarantee satisfactory solution quality. When the tree is growing too wide (resp., deep) but not sufficiently deep (resp., wide), the overall effectiveness will

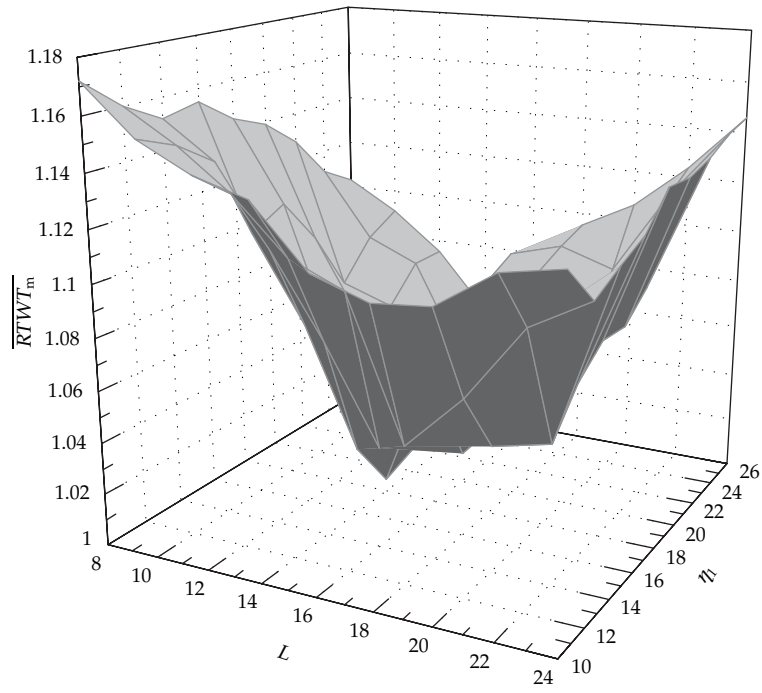

Figure 6: The influence of parameters L and $\eta_1 (= 2\eta_2)$.

deteriorate. Based on the experiments, the recommended settings are $L = 14$ and $\eta_1 = 18$ (thus $\eta_2 = 9$).

6. Conclusion

In this paper, we propose a hybrid differential evolution algorithm for the job shop scheduling problem with the objective of minimizing total weighted tardiness. Based on the mathematical programming models, we show a neighborhood property for the swap of adjacent operations in a critical block, which can be used to exclude some nonimproving neighborhood moves. Then, a tree-based local optimizer is designed and embedded into the DE algorithm in order to promote the exploitation function. The tree search algorithm generates compound neighborhood for the selected solution and therefore it helps DE to exploit the relatively high-quality solutions. Finally, the computational results verify the effectiveness and efficiency of the proposed hybrid approach.

The future research can be carried out from the following aspects.

(1) It is worthwhile to investigate the new and more effective neighborhood properties for TWT-JSSP. This could provide a deeper insight into the inherent nature of TWT-JSSP and facilitate the design of metaheuristics like DE.

(2) It is worthwhile to consider other encoding schemes and mutation/crossover mechanisms of the DE algorithm (such as the discrete DE), which may be more suitable for the application of the neighborhood properties.

Appendices

A. Proof of Theorem 3.1

Proof. We prove the theorem by construction.

First, it is noticeable that the unit cost of each arc in $A \cup \sigma$ is equal to the processing time of the operation that is connected to the tail of the arc, that is, $c_{i_1,i_2} = p_{i_1}$, which means the largest cost path from node 0 to node u_j is exactly the same as the critical path from operation 0 to operation u_j, and the total cost of this path equals the length of the critical path.

Then, because constraint (f) requires that the total input flow should equal the total output flow for every node, a feasible solution $\mathbf{F}$ to this problem must be composed of several cycle flows. However, we know that $G(N, A \cup \sigma)$ does not contain any cycles, so each cycle in $\mathbf{F}$ must contain at least one arc from U. On the other hand, we can see that the arcs in U all point to the same node 0, and because a cycle cannot include duplicate nodes, it is clear that each cycle in $\mathbf{F}$ contains at most one arc from U. Finally, it is concluded that each cycle contains exactly one arc from U.

Based on the previous discussions, we can construct an optimal solution $\mathbf{F}^*$ by the following steps.

Step 1. Initialize the flow on each arc to be 0, and let $j = 1$.

Step 2. Find the unique critical path from operation 0 to u_j, denoted by $P^*(0, u_j)$ (the uniqueness is guaranteed by a simple rule detailed in the following).

Step 3. If the total unit cost (i.e., length) of the path $P^*(0, u_j)$ plus the unit cost of arc $(u_j, 0)$ is greater than 0, then add a flow amount of w_{u_j} to arc $(u_j, 0)$ and each arc in $P^*(0, u_j)$ (notice that the capacity limit for arc $(u_j, 0)$ is w_{u_j}).

Step 4. Set $j \leftarrow j + 1$. If $j \leqslant n$, then return to Step 2; otherwise, terminate the algorithm.

In such a constructed solution $\mathbf{F}^*$, each node except 0 has at most one positive-flow incoming arc. This is because the flows are only distributed on the arcs belonging to critical paths, and meanwhile, only one critical path is considered from node 0 to any other node i.

The last issue is how to maintain the uniqueness of the critical path to a certain node. Here we use a rule called "machine predecessor first." Indeed, when looking for a critical path from 0 to u_j, we begin from u_j and move backward. In each step, we must select an operation whose completion time equals the starting time of the current operation i from its immediate job predecessor $\mathrm{JP}(i)$ and its immediate machine predecessor $\mathrm{MP}(i)$. Then, if $C_{\mathrm{JP}(i)} = s_i$ and $C_{\mathrm{MP}(i)} = s_i$ both hold, the rule requires that the machine predecessor should be selected as the next current operation. This tie-breaking rule guarantees the uniqueness of the critical path to any node. □

B. Proof of Theorem 3.3

Proof. As Figure 1 shows, the flows in the initial network are denoted by x_i $(x_i > 0)$, so we have the flow equilibrium condition:

$$\begin{aligned} x_{\alpha-1} &= F_\alpha + x_\alpha, \\ x_\alpha &= F_\beta + x_\beta, \\ x_\beta &= F_{\beta+1} + x_{\beta+1}. \end{aligned} \tag{B.1}$$

Now we are swapping the operations α and β. After the SWAP operation is performed, the flows on the relevant arcs are denoted by y. So the following equilibrium equations must be satisfied:

$$\begin{aligned} y_{\alpha-1} &= F_\beta + y_\beta, \\ y_\beta &= F_\alpha + y_\alpha, \\ y_\alpha &= F_{\beta+1} + y_{\beta+1}. \end{aligned} \tag{B.2}$$

It is noticed that, in order to keep the equilibrium of the remaining nodes $\alpha - a$ ($1 \leqslant a < \alpha$) and $\beta + b$ ($1 < b \leqslant z - \beta$), we have $y_{\alpha-a} = x_{\alpha-a}$ and $y_{\beta+b-1} = x_{\beta+b-1}$.

Solving (B.2) together with (B.1) yields

$$\begin{aligned} y_\beta &= x_{\alpha-1} + x_\beta - x_\alpha, \\ y_\alpha &= x_\beta. \end{aligned} \tag{B.3}$$

We can ensure $y_\beta \geqslant 0$, because Theorem 3.1 implies $x_{\alpha-1} \geqslant x_\alpha$.

The difference in the total cost of the flows $\{y_i\}$ and the original flows $\{x_i\}$ is

$$\begin{aligned} \Delta C = C(y) - C(x) &= \sum_{i=\alpha}^{\beta} p_i y_i - \sum_{i=\alpha}^{\beta} p_i x_i \\ &= p_\alpha x_\beta + p_\beta(x_{\alpha-1} + x_\beta - x_\alpha) - p_\alpha x_\alpha - p_\beta x_\beta \\ &= p_\beta(x_{\alpha-1} - x_\alpha) - p_\alpha(x_\alpha - x_\beta) \\ &= p_\beta F_\alpha - p_\alpha F_\beta. \end{aligned} \tag{B.4}$$

Since the flows outside this block are all kept unchanged, the difference in the total cost of the whole network resulted from executing SWAP(α, β) is the same as ΔC calculated for this subnetwork, that is, $\mathrm{TC}(y) - \mathrm{TC}(x) = C(y) - C(x)$.

Therefore, if $\Delta C \geqslant 0$, then $\mathrm{TC}_{\sigma'}^{\max} \geqslant TC(y) \geqslant \mathrm{TC}(x) = \mathrm{TC}_{\sigma}^{\max}$ (σ denotes the original set of directed disjunctive arcs while σ' denotes the new arc set obtained after performing SWAP(α, β)). The last "=" is because we assume the original network is related with the optimal solution to the dual problem under σ. In fact, $\mathrm{TC}_{\sigma}^{\max} = \mathrm{TWT}_{\sigma}^{\min}$ and $\mathrm{TC}_{\sigma'}^{\max} = \mathrm{TWT}_{\sigma'}^{\min}$. So it concludes that, when $\Delta C \geqslant 0$ ($\Leftrightarrow F_\alpha/p_\alpha \geqslant F_\beta/p_\beta$), swapping α and β will not improve the current solution ($\mathrm{TWT}_{\sigma'}^{\min} \geqslant \mathrm{TWT}_{\sigma}^{\min}$). □

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant nos. 61104176, 60874071.

References

[1] J. K. Lenstra, A. H. G. R. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[2] I. Essafi, Y. Mati, and S. Dauzère-Pérès, "A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem," *Computers and Operations Research*, vol. 35, no. 8, pp. 2599–2616, 2008.

[3] H. Zhou, W. Cheung, and L. C. Leung, "Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm," *European Journal of Operational Research*, vol. 194, no. 3, pp. 637–649, 2009.

[4] E. Nowicki and C. Smutnicki, "An advanced tabu search algorithm for the job shop problem," *Journal of Scheduling*, vol. 8, no. 2, pp. 145–159, 2005.

[5] J. Q. Li, Q. K. Pan, P. N. Suganthan, and T. J. Chua, "A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 52, no. 5–8, pp. 683–697, 2011.

[6] D. Y. Sha and C.-Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers & Industrial Engineering*, vol. 51, no. 4, pp. 791–808, 2006.

[7] G. Moslehi and M. Mahnam, "A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search," *International Journal of Production Economics*, vol. 129, no. 1, pp. 14–22, 2011.

[8] M. Seo and D. Kim, "Ant colony optimisation with parameterised search space for the job shop scheduling problem," *International Journal of Production Research*, vol. 48, no. 4, pp. 1143–1154, 2010.

[9] L. N. Xing, Y. W. Chen, P. Wang, Q. S. Zhao, and J. Xiong, "A knowledge-based ant colony optimization for flexible job shop scheduling problems," *Applied Soft Computing Journal*, vol. 10, no. 3, pp. 888–896, 2010.

[10] K. L. Huang and C. J. Liao, "Ant colony optimization combined with taboo search for the job shop scheduling problem," *Computers and Operations Research*, vol. 35, no. 4, pp. 1030–1046, 2008.

[11] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers & Operations Research*, vol. 35, no. 9, pp. 2892–2907, 2008.

[12] R. Tavakkoli-Moghaddam, M. Azarkish, and A. Sadeghnejad-Barkousaraie, "A new hybrid multi-objective Pareto archive PSO algorithm for a bi-objective job shop scheduling problem," *Expert Systems with Applications*, vol. 38, no. 9, pp. 10812–10821, 2011.

[13] M. Singer and M. Pinedo, "A computational study of bound techniques for the total weighted tardiness in job shops," *IIE Transactions*, vol. 30, no. 2, pp. 109–118, 1998.

[14] E. Kutanoglu and I. Sabuncuoglu, "An analysis of heuristics in a dynamic job shop with weighted tardiness objectives," *International Journal of Production Research*, vol. 37, no. 1, pp. 165–187, 1999.

[15] S. J. Mason, J. W. Fowler, and W. M. Carlyle, "A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops," *Journal of Scheduling*, vol. 5, no. 3, pp. 247–262, 2002.

[16] L. Mönch and R. Drießel, "A distributed shifting bottleneck heuristic for complex job shops," *Computers and Industrial Engineering*, vol. 49, no. 3, pp. 363–380, 2005.

[17] S. Kreipl, "A large step random walk for minimizing total weighted tardiness in a job shop," *Journal of Scheduling*, vol. 3, no. 3, pp. 125–138, 2000.

[18] M. Singer, "Decomposition methods for large job shops," *Computers & Operations Research*, vol. 28, no. 3, pp. 193–207, 2001.

[19] K. M. J. Bontridder, "Minimizing total weighted tardiness in a generalized job shop," *Journal of Scheduling*, vol. 8, no. 6, pp. 479–496, 2005.

[20] R. Tavakkoli-Moghaddam, M. Khalili, and B. Naderi, "A hybridization of simulated annealing and electromagnetic-like mechanism for job shop problems with machine availability and sequence-dependent setup times to minimize total weighted tardiness," *Soft Computing*, vol. 13, no. 10, pp. 995–1006, 2009.

[21] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[22] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "Modified differential evolution for constrained optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 25–32, July 2006.

[23] L. Wang and L.-P. Li, "Fixed-structure H∞ controller synthesis based on differential evolution with level comparison," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 120–129, 2011.

[24] M. Vasile, E. Minisci, and M. Locatelli, "An inflationary differential evolution algorithm for space trajectory optimization," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 267–281, 2011.

[25] M. Sharma, M. Pandit, and L. Srivastava, "Reserve constrained multi-area economic dispatch employing differential evolution with time-varying mutation," *International Journal of Electrical Power and Energy Systems*, vol. 33, no. 3, pp. 753–766, 2011.

[26] V. C. Mariani, L. S. Coelho, and P. K. Sahoo, "Modified differential evolution approaches applied in exergoeconomic analysis and optimization of a cogeneration system," *Expert Systems with Applications*, vol. 38, no. 11, pp. 13886–13893, 2011.

[27] N. Noman and H. Iba, "Accelerating differential evolution using an adaptive local search," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 107–125, 2008.

[28] C.-W. Chiang, W.-P. Lee, and J.-S. Heh, "A 2-Opt based differential evolution for global optimization," *Applied Soft Computing Journal*, vol. 10, no. 4, pp. 1200–1207, 2010.

[29] W. Gong, Z. Cai, and L. Jiang, "Enhancing the performance of differential evolution using orthogonal design method," *Applied Mathematics and Computation*, vol. 206, no. 1, pp. 56–69, 2008.

[30] G. Onwubolu and D. Davendra, "Scheduling flow shops using differential evolution algorithm," *European Journal of Operational Research*, vol. 171, no. 2, pp. 674–692, 2006.

[31] B. Qian, L. Wang, R. Hu, W.-L. Wang, D.-X. Huang, and X. Wang, "A hybrid differential evolution method for permutation flow-shop scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 7, pp. 757–777, 2008.

[32] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete differential evolution algorithm for the permutation flowshop scheduling problem," *Computers & Industrial Engineering*, vol. 55, no. 4, pp. 795–816, 2008.

[33] L. Wang, Q.-K. Pan, P. N. Suganthan, W.-H. Wang, and Y.-M. Wang, "A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems," *Computers &Operations Research*, vol. 37, no. 3, pp. 509–520, 2010.

[34] S. D. Wu, E. S. Byeon, and R. H. Storer, "A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness," *Operations Research*, vol. 47, no. 1, pp. 113–124, 1999.

[35] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, Springer, Berlin, Germany, 2005.

[36] C. Rego and R. Duarte, "A filter-and-fan approach to the job shop scheduling problem," *European Journal of Operational Research*, vol. 194, no. 3, pp. 650–662, 2009.

[37] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[38] U. K. Chakraborty, *Advances in Differential Evolution*, Springer, Berlin, Germany, 2008.