Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 687968, 11 pages
http://dx.doi.org/10.1155/2015/687968

Research Article

An Improved Particle Swarm Optimization for Selective Single Machine Scheduling with Sequence Dependent Setup Costs and Downstream Demands

Kun Li[1] and Huixin Tian[2]

[1]*School of Management, Tianjin Polytechnic University, Tianjin 300387, China*
[2]*School of Electrical Engineering & Automation, Tianjin Polytechnic University, Tianjin 300387, China*

Correspondence should be addressed to Kun Li; lk_neu@163.com

Received 15 October 2014; Revised 4 May 2015; Accepted 28 May 2015

Academic Editor: Mirko Viroli

This paper investigates a special single machine scheduling problem derived from practical industries, namely, the selective single machine scheduling with sequence dependent setup costs and downstream demands. Different from traditional single machine scheduling, this problem further takes into account the selection of jobs and the demands of downstream lines. This problem is formulated as a mixed integer linear programming model and an improved particle swarm optimization (PSO) is proposed to solve it. To enhance the exploitation ability of the PSO, an adaptive neighborhood search with different search depth is developed based on the decision characteristics of the problem. To improve the search diversity and make the proposed PSO algorithm capable of getting out of local optimum, an elite solution pool is introduced into the PSO. Computational results based on extensive test instances show that the proposed PSO can obtain optimal solutions for small size problems and outperform the CPLEX and some other powerful algorithms for large size problems.

1. Introduction

The single machine scheduling problem (SMSP) is one of the classical scheduling problems that have been widely researched in the literature. For example, the survey made by Koulamas [1] shows that a great deal of effort has been devoted to the single machine total weighted tardiness (SMTWT) problem. In this problem, there is a set of jobs that have arrived at time zero (denoted as $N = \{1, 2, \ldots, n\}$) and a single machine to process these jobs. Each job i has a positive processing time p_i, a positive weight w_i, and a due date d_i. It is assumed that the processing of a job cannot be interrupted once it starts. If job i is completed after its due date, then a penalty of weighted tardiness $w_i T_i$ (T_i is the tardiness of job i) will occur. The task of the SMTWT problem is to determine a job sequence so that the total weighted tardiness of all jobs can be minimized. Due to the fact that this problem is strongly NP-hard (Lawler [2]), many kinds of algorithms including both exact algorithms and meta-heuristics have been proposed in the literature [3–8].

Due to the fact that setup times often need to be considered in many practical industries (Yang and Liao [9] and Allahverdi et al. [10]), many researchers incorporated the sequence dependent setup time or setup cost s_{ij} between the processing of two adjacent jobs (namely, i and j) into the SMSP. For example, Das et al. [11] and França et al. [12] derived the SMSP with sequence dependent setup times from the plastic industry where a setup was required when the type of plastic changed. Gravel et al. [13] derived a similar problem from the aluminum industry where a setup was needed between casting operations whenever the type of alloys changed. Since then, the SMSP with sequence dependent setup times has drawn a great deal of attention from researchers and many kinds of algorithms have been proposed in the literature [14–16]. In particular, if a job is viewed as a customer and the sequence dependent setup time between jobs is viewed as the traveling distance between customers, then the SMSP with sequence dependent setup times can be transformed into a classical traveling salesman problem (Allahverdi et al. [17]). Recently, some researchers

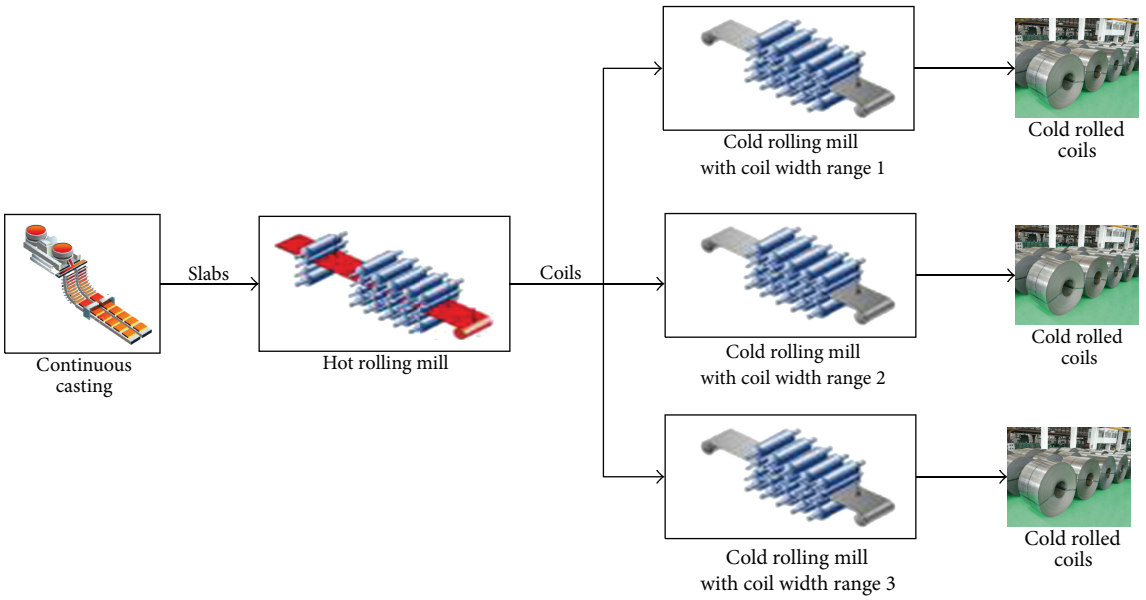

Figure 1: Illustration of this problem in iron and steel industry.

began to further consider more practical constraints into this problem; for example, Jula and Rafiey [18] incorporated the time window constraint on the start times of jobs into this problem and presented a network-based algorithm to solve it.

Although there have been many papers focusing on the SMSP with sequence dependent setup times (or costs), most of these researches assumed that all the jobs could be completed within a time horizon and did not consider the consistent production process in practical industries. That is, they did not consider the selection of jobs caused by processing capacity of machines within the time horizon and the demands of downstream production lines. Although some researchers have noticed the decision on job selection due to the processing capacity of the single machine (Wang and Tang [19]), current researches still do not consider the demands of downstream production lines. In practical production scheduling, the demands of downstream production lines are very important because the logistics quality of the entire production shop may be still inefficient even if the scheduling quality of a single machine (or production line) is optimal without considering the demands of downstream production lines. In addition, the requirement for considering the demands of downstream production lines is widespread in practical manufacturing industries. For example, in the equipment manufacturing industry jobs generally need a preprocessing on a production line and subsequently the completed jobs will be delivered to different downstream lines for different kinds of further processing. Due to the different dimensions of jobs, a sequence dependent setup time (e.g., time for replacing processing tools) will be needed. Because each job has been assigned to a specified downstream production line and the single machine has a limited processing capacity within a given scheduling time horizon, the schedulers in the preprocessing production line should make two decisions: which jobs to be included in the schedule and how to determine the processing sequence of jobs in the schedule. The objective is to ensure that the total tardiness of jobs can be minimized and at the same time the downstream production lines can receive sufficient jobs so as to guarantee a consistent production. Another typical example is the well-known hot rolling production in the iron and steel industry (Figure 1). In the hot rolling mill, slabs are delivered to the front warehouse from the continuous casting production line, and the coils rolled from slabs through the hot rolling line will be then transferred to the downstream production lines such as different cold rolling mills with different ranges of coil width. Since the width of coils after the hot rolling has been determined by contracts, the flow direction of each coil has been also preassigned according to the processing ability (i.e., the range of coil width) and the warehouse of the downstream cold rolling lines. In addition, the transition of different adjacent slabs will affect the coil quality and such a transition cost can be viewed as a kind of sequence dependent setup cost. When schedulers establish the hot rolling schedule with a limited processing capacity, they should determine which slabs to be included in the rolling schedule so as to satisfy the demands of downstream cold rolling lines and the processing sequence of selected slabs in the schedule so as to minimize the total transition cost.

Therefore, in this paper we investigate a new variant of the single machine scheduling problem by further considering the sequence dependent setup costs and the demands of downstream production lines. The two major characteristics of this problem are as follows:

(1) The single machine can only process a subset of all jobs because of its processing capacity within a given time horizon.

(2) Each job has been allocated to a downstream production line and the total weights of jobs allocated to a downstream production line should be within the range defined by the demand of this production line and the available storage capacity of warehouse before it (please note that in practical production the available storage capacity of warehouse before a downstream production line is usually much larger than the demand of this line with a given time horizon).

This paper is organized as follows. In Section 2 we give the description of our new variant and its mathematical optimization model. Section 3 is devoted to describing the proposed improved particle swarm optimization algorithm. In Section 4 we will provide the results of computational experiments and the corresponding analysis. Finally, this paper is concluded in Section 5.

2. Problem Description and MIP Model

2.1. Problem Description. The selective SMSP with sequence dependent setup costs and downstream demands can be briefly described as follows. There is a set of jobs $N = \{1, 2, \ldots, n\}$ that have arrived at time zero, a single machine with a maximum capacity of Q within a scheduling time horizon T to process these jobs (i.e., it is not possible for the single machine to process all the jobs within the scheduling time horizon T), and a set of downstream production lines with different demands. Each job i has a processing time p_i and a weight w_i, and the processing of a job cannot be interrupted once it starts. In addition, each job has been allocated to a specified downstream production line according to its dimension. A positive setup time s_{ij} is required between any two adjacent jobs. The objective is to select a subset of jobs to satisfy the demands of downstream lines and at the same time determine their processing sequence so as to maximize the utilization of the machine's capacity within the given time horizon.

2.2. Mathematical Model

2.2.1. Parameters

i and j: index of jobs.

k: index of downstream production lines.

N: the set of candidate jobs, $N = \{0, 1, 2, \ldots, n\}$, where 0 denotes a dummy job that must be selected in the schedule.

K: the set of downstream production lines.

T: a given scheduling time horizon.

Q: processing capacity of the machine within a given time horizon T.

D_k: demand of the downstream line k.

U_k: available storage capacity of the warehouse before the downstream line k.

p_i: processing time of job i.

w_i: weight of job i.

s_{ij}: the sequence dependent setup time between any two adjacent jobs i and j ($i \neq j$).

M: a very big number that is set to be the total processing time of all jobs.

$r_{ik} = \{1$, if job i is assigned to downstream production line k; 0, otherwise$\}$; please note that in our problem r_{ik} is predetermined by the dimensions of job i and not a decision variable.

2.2.2. Decision Variables

$x_{ij} = \{1$, if job j is processed immediately after job i; 0, otherwise$\}$, $i, j \in N, i \neq j$.

C_i: completion time of selected job i, $i \in N$.

2.2.3. Model. With the above parameters and decision variables, we can present a mixed integer linear programming model for this problem as follows:

$$\text{Minimize} \quad Q - \sum_{i \in N} \sum_{\substack{j \in N \\ j \neq i}} w_i x_{ij}, \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{0i} = \sum_{i=1}^{n} x_{i0} = 1, \tag{2}$$

$$\sum_{j=0}^{n} x_{ij} \leq 1, \quad i \in N \setminus \{0\}, \tag{3}$$

$$\sum_{j=0}^{n} x_{ji} = \sum_{j=0}^{n} x_{ij}, \quad i \in N \setminus \{0\}, \tag{4}$$

$$C_j \geq C_i + s_{ij} + p_j - M\left(1 - x_{ij}\right), \quad i, j \in N \setminus \{0\},\ i \neq j, \tag{5}$$

$$0 \leq C_i \leq T, \quad i \in N \setminus \{0\}, \tag{6}$$

$$\sum_{i \in N} \sum_{\substack{j \in N \\ j \neq i}} w_i x_{ij} \leq Q, \tag{7}$$

$$D_k \leq \sum_{i \in N} \sum_{\substack{j \in N \\ j \neq i}} w_i r_{ik} x_{ij} \leq U_k, \quad k \in K, \tag{8}$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in N,\ i \neq j, \tag{9}$$

$$C_i \in \Re, \quad i \in N. \tag{10}$$

Objective (1) is to maximize the utilization of the machine's capacity. Constraint (2) requires that the schedule should start from the dummy job and end at it finally. Constraints (3) ensure that each job can be selected and processed at most once. Constraints (4) guarantee the flow conservation of processing; that is, if job i is selected then

there must be a job before it and another job after it in the schedule; on the contrary, if it is not selected, then there are no jobs arranged before and after it. Constraints (5) give the completion time relationship between any two adjacent jobs. Constraints (6) require that the processing should be completed within the given scheduling time horizon. Constraint (7) ensures that the total weight of selected jobs cannot exceed the machine's capacity. Constraints (8) are the demand requirements of downstream production lines. Constraints (9) and (10) specify the values of decision variables.

The above mathematical model is similar to that proposed by Wang and Tang [19], in which the authors illustrated the difference of single machine scheduling with selective jobs from the other relative problems such as the selective traveling salesman problem (Gendreau et al. [20]) and the prize collecting traveling salesman problem (Dell'amico et al. [21]). However, in our model we further consider the demands of downstream production lines (constraints (8)) and the machine' capacity (constraint (7)). The incorporation of demands of downstream production lines makes the decision of selecting jobs quite different from that in [19]. In the problem considered in [19], the jobs have no flow directions and only the time horizon limitation is considered, and consequently the selection of jobs is only affected by the available time horizon; however, the selection of jobs in our problem is affected by three factors: one is the available time horizon, another is the capacity of the single machine, and the last one is the demand of each downstream production line. The different combinations of these three factors will make the search space much bigger than that in [19]. Due to the fact that the problem considered in [19] is NP-hard, the problem investigated in this paper is also NP-hard.

3. Improved Particle Swarm Optimization Algorithm

In recent years, evolutionary algorithms have been widely adopted to solve both the continuous optimization problems ([22–24]) and the combinatorial optimization problems derived in iron and steel industry ([25–27]). The particle swarm optimization (PSO) algorithm is a kind of evolutionary algorithm proposed by Kennedy and Eberhart [28].

The main idea of the PSO simulates the social behavior of bird flocking or fish schooling and their information exchange method. In the PSO, each solution is denoted as a particle and there is a swarm of particles flying and searching the solution space. During the search process, each particle updates itself under the guidance of the direction of its original flying track, the personal best solution it has found (p_{best}), and the global best solution found by the whole swarm (g_{best}). Let x_{ij}^t denote the jth dimension of the position value of particle i in current generation t, then the position update equation of each particle can be given as follows:

$$\begin{aligned} v_{ij}^{t+1} &= w \cdot v_{ij}^t + c_1 r_1 \cdot \left(p_{ij}^t - x_{ij}^t\right) + c_2 r_2 \cdot \left(g_j^t - x_{ij}^t\right), \\ x_{ij}^{t+1} &= x_{ij}^t + v_{ij}^{t+1}, \end{aligned} \tag{11}$$

where w, c_1, and c_2 are parameters and r_1 and r_2 are random numbers between (0, 1).

Since proposed, PSO has been successfully applied to solve many kinds of complicated combinatorial optimization problems, especially the scheduling problems (Nanvala [29]). The PSO hybrid with variable neighborhood search (VNS) proposed by Tasgetiren et al. [30] for permutation flow shop scheduling problem also illustrated its efficiency for scheduling problems. Therefore, in this paper we also prefer to adopt PSO to solve our problem. Although the PSO algorithm developed in this paper is inspired by the one proposed in Tasgetiren et al. [30], they are quite different in the following three aspects:

(i) First, a special decoding method based on the characteristics of our problem is presented and adopted in our algorithm.

(ii) Second, we develop an adaptive VNS in which the neighborhood is adaptively selected based on the analysis of their search result, while the VNS in Tasgetiren et al. [30] used a traditional predetermined neighborhood sequence.

(iii) Third, an elite pool strategy was developed to improve the search diversity while the local search in Tasgetiren et al. is mainly applied on the global best solution found so far at each generation.

3.1. Solution Representation and Repair Method in Our PSO. Due to the fact that classical PSO can only be used to solve optimization problems in continuous space, researchers have designed some decoding method to transform a solution from a continuous space to a discrete space. In our algorithm, we prefer to adopt the smallest position value (SPV) rule proposed in Tasgetiren et al. [30] in which this decoding rule had been proven to be very effective for scheduling problems. Taking into account that only a subset of all candidate jobs can be selected in the schedule, we develop the following encoding and decoding methods, as well as a repair method, used in our algorithm by extending the SPV rule.

Step 1 (encoding). In our algorithm, a solution is represented by a particle consisting of n dimensions; that is, $X = (x_1, x_2, \ldots, x_n)$ in which x_i is the continuous position value of job i. The domain for each x_i is $[-10.0, 10.0]$.

Step 2 (decoding). Since we should decide which job should be selected and at the same time determine the processing sequence of selected jobs, we extended the SPV rule to transform a solution in continuous space to a feasible solution in discrete space. In the decoding rule, we first use the SPV rule to obtain a sequence of all jobs, and then from the first job in the sequence we iteratively select jobs until constraints (6) and (7) are not satisfied.

Step 3 (repair). Since constraints (8) are not considered in the decoding procedure, a solution obtained by the decoding may not be feasible. So we propose a repair method to restore the feasibility of the obtained solution. If a solution after decoding

Table 1: Job information and the decoding method based on SPV rule.

Job	1	2	3	4	5	6	7	8	9	10
p_i	3	5	3	2	7	4	2	1	8	5
w_i	12	14	8	7	16	10	3	8	18	12
r_{ik}	1	2	1	1	2	1	2	2	2	1
x_i	0.7	1.5	0.1	−0.2	0.9	0.3	0.8	−0.7	−0.4	1.2
Permutation	8	9	4	3	6	1	7	5	10	2

is infeasible, the repair method consisting of the following steps will be applied to restore its feasibility.

Step 3.1. Calculate the total weights of jobs assigned to each downstream line and denote it as W_k.

Step 3.2. For each downstream line, if $W_k > U_k$, then remove the job that is assigned to line k and has the largest value of p_i/w_i. Repeat this process until W_k is not more than U_k for each downstream production line.

Step 3.3. For each downstream line, if $W_k < D_k$, then we will add a job that is assigned to line k and can bring the least value of p_i/w_i to the end of the sequence, and if the resulting solution violates constraints (6) or (7), then we will select a line k for which the total weight of assigned jobs in the sequence is the largest and subsequently we will remove a job that is assigned to it and has the largest value of p_i/w_i. Repeat this process until W_k is not less than D_k for each downstream production line.

Based on the above method, a solution consisting of all candidate jobs can be transformed into a feasible solution for our problem. In the experiment, the computational results show that feasible solutions always exist for each problem. This is also true in practical production because there are a large number of candidate jobs and the number of downstream lines is relatively very small.

An example including 10 jobs for the problem can be given as follows. The job information is given in Table 1, the demands of downstream lines are 25 and 30, respectively, and the available capacities are 35 and 50, respectively.

First, the representation of the solution $X = (0.7, 1.5, 0.1, -0.2, 0.9, 0.3, 0.8, -0.7, -0.4, 1.2)$ is transformed into a discrete job sequence $X' = (8, 9, 4, 3, 6, 1, 7, 5, 10, 2)$ by the SPV rule that always assigns a job with a smaller position value to the front of a job with a larger position value. Subsequently we select jobs from this sequence until the time horizon or the machine's capacity is reached. For simplicity, we assume the obtained job sequence is $(8, 9, 4, 3, 6, 1, 7)$. Then the total weight of jobs for downstream lines 1 and 2 is 37 and 29, respectively; that is, we have $W_1 > U_1$ and $W_2 < D_2$ which make the solution infeasible. So we first deal with the condition $W_1 > U_1$ by removing job 6 because it has the largest value of p_i/w_i and then this infeasibility is restored. Then we turn to deal with the condition $W_2 < D_2$ by inserting job 2 into the end of the sequence because it has the least value of p_i/w_i. At last, we obtain a feasible solution $(8, 9, 4, 3, 1, 7, 2)$ whose $W_1 = 27$ and $W_2 = 43$. Please note that in this example we assume that insertion of a job does not violate constraints (6) and (7) so that the explanation can be simple and clear.

3.2. Population Initialization. The initial population whose size is N is created with two methods: (1) $N - 1$ random solutions are generated according to $x_{ij} = x_{\min} + \text{rand}(0, 1) \times (x_{\max} - x_{\min})$, in which $\text{rand}(0, 1)$ is a random number uniformly generated in $[0, 1]$ and we use $x_{\min} = -10.0$ and $x_{\max} = 10.0$ and (2) one solution is generated based on a heuristic by extending the famous NEH method (Nawaz et al. [31]). The extended NEH method used in our algorithm consists of three steps: first sequence all the candidate jobs using the classical NEH method with the modified objective of minimizing the total setup times; then use the feasibility repair method described in the above Section 3.1 to obtain a feasible solution; and at last code this solution in a continuous space by uniformly assigning the nondescending order of positions values for jobs in the sequence in $[x_{\min}, x_{\max}]$. In addition, the velocity of each particle is randomly generated through the same way; that is, $v_{ij} = v_{\min} + \text{rand}(0, 1) \times (v_{\max} - v_{\min})$, in which we set $v_{\min} = -2.0$ and $v_{\max} = 2.0$.

3.3. Adaptive Variable Neighborhood Search. To improve the search efficiency and diversity, we present two kinds of improvement strategies: (1) an adaptive variable neighborhood search (AVNS) based on the job-block based neighborhood that can dynamically adjust the search depth is designed and (2) an elite solution pool is adopted to store good quality solutions (e.g., the best ten solutions found so far) and an adaptive strategy is used to select an appropriate solution for the local search by AVNS. In the following, we first describe the elite pool strategy, the neighborhood used in the algorithm, and at last the AVNS algorithm.

3.3.1. Elite Solution Pool. As mentioned above, we use an elite solution pool E consisting of the best ten solutions found so far by the algorithm. Then at each generation, we apply the AVNS to further improve a solution randomly selected from E instead of the best solution found so far. Such a strategy can help to improve the search diversity and at the same time can maintain a good probability of finding new better solutions through local search because the selected solution has similar quality to the best solution.

3.3.2. Job-Block Based Neighborhood and Local Search Method. In the single machine scheduling or the permutation flow shop scheduling problems, traditional neighborhoods are based on the insertion move that removes a job from its current position and then inserts it into another position and the swap move that swaps two different jobs in the sequence (Tasgetiren et al. [30]). Since the compound moves have been proven to be more effective than a single move (Bozejko et al. [5]), we design for the VNS a job-block based neighborhood, which is a compound-move based neighborhood.

If the size of the job-block is b, then the job-block move will remove the successive b jobs from the job sequence and

```
Let s be the input initial solution and NH_b (b = 1, ..., b_max) the candidate neighborhoods.
l := 1
while l ≤ l_max do
    b := AdaptiveSelection (s)          //adaptively select a neighborhood
    s' := Shaking (s, b)                //generate a random new solution in NH_b
    s'' := LocalSearch (s', N_b)        //perform local search on s' in neighbourhood NH_b
    if C_max(s'') < C_max(s)            //neighborhood change checking
        s := s''
    end if
    UpdateProbability (NH_b)            //update the selection probability of NH_b
end while
```

Algorithm 1: Main procedure of the AVNS.

then reinsert them into the job sequence according to their original sequence (note that if $b = 1$, then the neighborhood becomes the classical *insertion* neighborhood). For a given solution $s = (s(1), \ldots, s(n))$, let the size of the job-block be b and the corresponding neighborhood NH_b, then the local search procedure in neighborhood NH_b can be described as follows.

Step 1. Set the iteration index $r = 1$ and the best solution to be $s_b = s$.

Step 2. Randomly remove b adjacent jobs from s and denote this job-block as $s(d_1), s(d_2), \ldots, s(d_b)$ and the left partial solution as s'.

Step 3. Set $j = 1$.

Step 4. Insert $s(d_j)$ into the best position in s'.

Step 5. Set $j = j + 1$. If $j \le b$, go to *Step 4*; otherwise, obtain a new solution s'.

Step 6. If $C_{\max}(s') < C_{\max}(s_b)$, then set $s = s_b = s'$.

Step 7. Set $r = r + 1$. If $r \le r_{\max}$ (maximum iteration), go to *Step 2*; otherwise, terminate and output s_b.

In our problem, the selection of jobs should be considered. From the description of the decoding method in Section 3.1, it is clear that the decoding procedure can help change the selection of jobs. So we prefer to set the focus of the local search on the improvement of the sequence of selected jobs, that is, to reduce the total setup times of selected jobs. Therefore, in the above search procedure in neighborhood NH_b, the objective is the *makespan* (denoted as $C_{\max}$) of the selected jobs. In addition, it is clear that the local search is not performed on the entire neighborhood but only a number of $r_{\max}$ random searches. Such a strategy can help save computational time while maintaining the solution quality.

3.3.3. AVNS Algorithm. For a given solution $s = (s(1), \ldots, s(n'))$, $n' < n$, let the size of the job-block be b and the corresponding neighborhood is dented as NH_b, then the AVNS procedure can be described in Algorithm 1, in which the *LocalSearch* method is the one described in Section 3.3.2 and the shaking function is used to generate a new random solution based on a random insertion move. The AVNS is applied to a solution s selected from E at each generation of PSO. Whenever s is improved, we will update the global best solution with it.

In traditional VNS, the sequence of neighborhood used is predetermined, that is, from the first one to the last one. However, in the AVNS the sequence of neighborhoods is not predetermined but dynamically changed based on the search results of each neighborhood. The reason we adopt such a strategy is that we want to allocate more local search chances to more promising neighborhoods within the limited computational time. In addition, the AVNS shown in Algorithm 1 is simpler than the traditional VNS because whenever a better solution is found the AVNS does not go back to the first neighborhood.

To define the selection probability of each neighborhood, we introduce the concept of *success* and *failure* of a neighborhood search. The application of a neighborhood search is considered as successful only if it can result in a better solution s'' with comparison to the input solution s; otherwise it is viewed as unsuccessful.

In the implementation, the selection probability of each neighborhood is set to be equal at the beginning, that is, $1/B$ ($B = 4$ in our algorithm). In our algorithm, we memorize the successful count succ_b and the unsuccessful count fail_b of each neighborhood NH_b. After the evaluation of s'', the selection probability of the selected neighborhood $\mathrm{NH}_b(P_b)$ will be updated by

$$P_b = \frac{S_b}{\sum_{i=1}^{B} S_b}, \quad g = 1, 2, \ldots, b_{\max}, \tag{12}$$

where $S_b = \mathrm{succ}_b/(\mathrm{succ}_b + \mathrm{fail}_b) + 0.01$ is called the success ratio. We add 0.01 to each S_b in order to avoid the condition that some neighborhoods may have a selection probability of zero if they cannot bring better solutions than s. Based on this calculation method, it is clear that the neighborhood with a larger number of successful applications will have a larger selection probability of applying it to generate new solutions.

4. Computational Experiments and Results

4.1. Test Problems and Parameter Setting. To test the performance of the method, computational experiment has been carried out on a set of randomly generated instances. The generation process is based on the practical hot rolling production and can be described as follows: the number of jobs n is selected from $\{50, 80, 100, 120, 150, 200\}$, the number of downstream lines is selected from $\{3, 5, 7\}$, the processing time and weight of each job are randomly generated in $[1, 20]$ and $[5, 20]$, respectively, the sequence dependent setup times between two jobs are randomly generated in $[1, 10]$, the capacity of the single machine is set to be 60 percent of the total weight of all jobs, the time horizon is set to be $0.6 \times (\sum_{i \in N} p_i + (n-1) \times \bar{s})$, where $\bar{s}$ is the average setup time between adjacent jobs, the jobs are randomly assigned to each downstream line according to a uniform distribution, and the demand and available storage capacity of each downstream line are then generated as 40 percent and 80 percent of the total weights of jobs assigned to this downstream production line, respectively. Based on these settings, there is $6 \times 3 = 18$ scenarios and for each scenario we generated 10 instances; consequently there is a total of 180 instances generated for the experiment. Please note that when the capacity of the single machine increases (at the same time the demands of downstream lines remains unchanged), the problem tends to be simple because more feasible solutions can be found since with more selected jobs constraint (8) will become loose.

The proposed PSO algorithm was implemented in C++ language and tested on a personal computer with Intel Core i5-3.2 GHz CPU and 4 GB memory. Through experiment, the following setting of algorithm parameters is adopted: the size of swarm is set to 50, the maximum iteration of AVNS is set to $l_{\max} = 20$, the maximum iteration of the neighborhood search $r_{\max}$ is set to 5, and the stopping criterion is set to the maximum computational time of $m \times n/2$ seconds (m is the number of downstream lines). However, whenever an optimal solution is obtained for small size problems, the algorithm also terminates (please note that optimal solutions can be obtained by CPLEX for small size problems and these optimal solutions can be used to terminate the algorithm before the stopping criterion is reached). The maximum size of job-block is set to 5 (i.e., $b_{\max} = 5$) and the impact of b on the performance of our algorithm is analyzed in the following Section 4.2.1. Please note that all the test algorithms in the following sections share the same stopping criterion. That is, we run each algorithm with the stopping criterion that is the maximum computational time of $m \times n/2$ seconds. In the implementation, we will check the computational time elapsed until now whenever the function of evaluating a solution (i.e., calculating the objective value of a solution) is called. During experiment, it is found that the testing algorithms can terminate once this stopping criterion is reached for most instances and that the difference of used computational times between different algorithms is rather small (the difference is often within 0.05 seconds). Therefore, in the following sections the used computational time is given as $m \times n/2$ seconds for medium and large size instances for simplicity.

4.2. Computational Results and Analysis. To evaluate the performance of our algorithm, the CPLEX 12.4 was adopted to solve the random instances. For small size problems, CPLEX can obtain optimal solutions based on the MILP models (1)–(10). Although for large size problems, CPLEX cannot provide optimal solutions with a maximum runtime of 1600 seconds, it can provide a lower bound for each problem instance. So use the gap between a solution obtained by an algorithm and the corresponding lower bound obtained by CPLEX 12.4 to act as the evaluation metric; that is, $\text{Gap}(\%) = 100 \times [f(s) - \text{LB}]/\text{LB}$ in which $f(s)$ is the objective value of a solution s obtained by an algorithm and LB is the corresponding lower bound for a given problem instance.

In the following sections, we first carried out experiments to analyze the impact of the maximum neighborhood size $b_{\max}$ on the performance of AVNS and then illustrate the effectiveness of proposed improvement strategies. At last we will compare our improved PSO algorithm with the PSO_{VNS} algorithm.

4.2.1. Impact of $b_{\max}$ on the Performance of AVNS. As described in Section 3.3.2, the job-block size b determines the complexity of local search and thus $b_{\max}$ has a significant impact on the performance of AVNS. So in this section an experiment was carried out to analyze its impact and subsequently an appropriate value can be determined.

In this experiment, the candidate values of $b_{\max}$ are selected from $\{2, 3, 4, 5, 6\}$. The computational results are presented in Table 2, in which the results for an instance scenario are the average of 10 instances and a better result is shown in a bold style (please note that in the following experiments we always use this setting). In this table, Gap(%) means the gap between the solution obtained by an algorithm and the corresponding lower bound obtained by CPLEX 12.4.

From the comparison results, it can be found that for most of the large size problems the solution quality first improves but then deteriorates quickly as $b_{\max}$ increases. The main reason behind this phenomenon is that for small size problems the larger values of $b_{\max}$ can provide larger neighborhoods that can help reach the optimal solutions more quickly. But for large size problems, as $b_{\max}$ increases the size of neighborhood will become too large, which in turn reduces the number of practical evolution generations of PSO within a given computational time and consequently deteriorates its performance. So there is a tradeoff between the solution quality and computational time, though larger value of $b_{\max}$ can generally achieve better solutions if enough computational time is available. Based on the results shown in Table 2, it is clear that $b_{\max} = 4$ is an appropriate choice.

4.2.2. Effective Analysis of Adaptive Strategy Used in AVNS. To show the effectiveness of the adaptive strategy of neighborhood selection used in the AVNS, we compared our PSO algorithm ($\text{PSO}_{\text{adaptiveVNS}}$) with a simplified version of it by using a random selection strategy. That is, in the simplified PSO algorithm ($\text{PSO}_{\text{randomVNS}}$) a neighborhood is randomly selected instead of selection based onits selection probability.

The computational results are given in Table 3. From the comparison results, it can be seen that the adaptive strategy

Table 2: Impacts of different values of b_{max} on the performance of the proposed PSO algorithm.

m	n	$b_{max} = 2$		$b_{max} = 3$		$b_{max} = 4$		$b_{max} = 5$		$b_{max} = 6$	
		Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)
3	50	0.02	24	0.02	17	**0.00**	12	**0.00**	**11**	**0.00**	**11**
	80	0.11	96	0.08	80	**0.06**	71	**0.06**	**65**	**0.06**	70
	100	0.62	150	0.50	150	0.46	150	**0.44**	150	0.46	150
	120	0.58	180	0.45	180	**0.41**	180	**0.41**	180	0.45	180
	150	1.47	225	1.38	225	**1.34**	225	**1.34**	225	1.36	225
	200	2.81	300	2.72	300	**2.69**	300	2.71	300	2.75	300
5	50	0.00	43	0.00	26	**0.00**	22	**0.00**	**18**	**0.00**	19
	80	0.19	200	**0.17**	200	**0.17**	200	**0.17**	200	0.19	200
	100	0.77	250	0.65	250	**0.62**	250	**0.62**	250	0.64	250
	120	1.21	300	**1.15**	300	**1.15**	300	**1.15**	300	**1.15**	300
	150	1.78	375	1.69	375	**1.66**	375	1.69	375	1.72	375
	200	3.63	500	3.42	500	**3.34**	500	3.40	500	3.47	500
7	50	0.05	36	0.02	29	0.02	24	**0.00**	**21**	0.02	23
	80	0.97	280	**0.86**	280	**0.86**	280	**0.86**	280	**0.86**	280
	100	1.44	350	1.38	350	**1.34**	350	**1.34**	350	1.41	350
	120	1.35	420	**1.31**	420	**1.31**	420	**1.31**	420	**1.31**	420
	150	2.26	525	2.09	525	**1.97**	525	2.06	525	2.14	525
	200	4.53	700	4.25	700	**4.18**	700	4.33	700	4.45	700
Average		1.32	275	1.23	273	**1.20**	**271**	1.22	**271**	1.25	**271**

Table 3: Comparison results for the effectiveness of adaptive strategy in VNS.

m	n	$PSO_{randomVNS}$		$PSO_{adaptiveVNS}$	
		Gap (%)	CPU (s)	Gap (%)	CPU (s)
3	50	**0.00**	17.21	**0.00**	12.74
	80	**0.06**	83.67	**0.06**	71.35
	100	0.54	150.00	**0.46**	150.00
	120	0.52	180.00	**0.41**	180.00
	150	1.43	225.00	**1.34**	225.00
	200	2.85	300.00	**2.69**	300.00
5	50	**0.00**	38.92	**0.00**	22.06
	80	**0.17**	200.00	**0.17**	200.00
	100	0.69	250.00	**0.62**	250.00
	120	1.24	300.00	**1.15**	300.00
	150	1.71	375.00	**1.66**	375.00
	200	3.58	500.00	**3.34**	500.00
7	50	**0.02**	29.05	**0.02**	24.86
	80	**0.86**	280.00	**0.86**	280.00
	100	1.39	350.00	**1.34**	350.00
	120	1.37	420.00	**1.31**	420.00
	150	2.14	525.00	**1.97**	525.00
	200	4.42	700.00	**4.18**	700.00

can further improve the performance of VNS. More specifically, it appears that for small size problems the $PSO_{adaptiveVNS}$ can obtain the optimal solutions with a shorter runtime. The main reason is that the adaptive strategy can help select the neighborhoods that have a higher probability of finding better solutions, which in turn helps improve the search efficiency within limited computational time.

4.2.3. Effective Analysis of Elite Solution Pool. To show the effectiveness of the incorporation of elite solution pool used in the PSO, we compared our PSO algorithm with a version in which the AVNS is always applied to the global best solution ($PSO_{globalbest}$).

The computational results are given in Table 4, from which it can be found that the elite solution pool can help improve the performance of PSO for 7 out of the 18 instance scenarios and that these 7 instance scenarios are all large size instances. For the small size instances, $PSO_{globalbest}$ can obtain the optimal solutions faster than $PSO_{adaptiveVNS}$ because the local search is performed on the global best solution. However, for large size instance scenarios $PSO_{globalbest}$ algorithm shows an inferior performance compared to the $PSO_{adaptiveVNS}$. The main reason is that the incorporation of elite solution pool can provide different start points with similar quality to the global best solution but with better diversity for the AVNS and consequently the PSO can achieve a better ability of getting out of local optimum.

4.2.4. Comparison with the Other Powerful Algorithms. To evaluate the performance of our algorithm, we compare it with the CPLEX 12.4 and another powerful algorithm named PSO_{VNS} proposed by Tasgetiren [30]. The PSO_{VNS} was proposed to solve the permutation flow shop scheduling problem, whose solution is also represented by a job sequence. So we modified this algorithm to solve our problem: first, the solution decoding method described in Section 3.1 is incorporated into PSO_{VNS}; second, the neighborhood search used

Table 4: Comparison results for the effectiveness of elite solution pool in PSO.

m	n	$PSO_{globalbest}$		$PSO_{adaptiveVNS}$	
		Gap (%)	CPU (s)	Gap (%)	CPU (s)
3	50	**0.00**	10.16	**0.00**	12.74
	80	**0.06**	63.54	**0.06**	71.35
	100	**0.46**	150.00	**0.46**	150.00
	120	**0.41**	180.00	**0.41**	180.00
	150	1.37	225.00	**1.34**	225.00
	200	2.73	300.00	**2.69**	300.00
5	50	**0.00**	19.05	**0.00**	22.06
	80	**0.17**	200.00	**0.17**	200.00
	100	**0.62**	250.00	**0.62**	250.00
	120	1.19	300.00	**1.15**	300.00
	150	**1.66**	375.00	**1.66**	375.00
	200	3.42	500.00	**3.34**	500.00
7	50	**0.02**	21.97	**0.02**	24.86
	80	**0.86**	280.00	**0.86**	280.00
	100	**1.34**	350.00	**1.34**	350.00
	120	1.33	420.00	**1.31**	420.00
	150	2.07	525.00	**1.97**	525.00
	200	4.27	700.00	**4.18**	700.00

in the VNS of PSO_{VNS} is the one described in Section 3.3.2, and the neighborhoods used are also the job-block based neighborhood NH_b and the neighborhood sequence is from $b = 1$ to $b = 4$. In the experiment, the PSO_{VNS} algorithm shares the same stopping criterion with our algorithm and the CPLEX has a maximum runtime of 1600 seconds. However, it should be noted that these three algorithms will terminate whenever an optimal solution is found.

In addition, although Wang and Tang [19] dealt with a similar problem and proposed a scatter search algorithm hybrid with VNS (SS_{VNS}); we did not compare our algorithm with theirs based on the following reason: in the VNS method used in SS_{VNS}, the neighborhood moves are mainly focused on the change of selection of jobs. When applied in our problem, a large proportion of new solutions generated by these moves may be infeasible since they may not satisfy constraints (7) and (8). The repair of these infeasible solutions during neighborhood search will need a great deal of computational efforts and repaired solution may not have a good enough quality. On the contrary, in our algorithm the local search is just focused on the change of job sequence that will not result in infeasible solutions and the repair of infeasible solutions is just applied for population update (i.e., the number of infeasible solutions to be repaired is far less than that in SS_{VNS}). So the SS_{VNS} algorithm will suffer from low search efficiency during local search and the comparison with it seems unfair.

The computational results of our algorithm with comparison to CPLEX and PSO_{VNS} are given in Table 5, in which the superscript "$*$" denotes that the performance difference between $PSO_{adaptiveVNS}$ and PSO_{VNS} is significantly for a certain problem size based on the *Pairwise-Samples t Test* with a confidence level of 95%. In addition, the column Gains% denotes the average improvement of the results obtained by our algorithm over that obtained by PSO_{VNS} and it is calculated as

$$\begin{aligned}&\text{Gains}(\%)\\&\quad = 100\\&\qquad \times \frac{\left[\text{Gap}\left(\text{PSO}_{\text{adaptiveVNS}}\right)\% - \text{Gap}\left(\text{PSO}_{\text{VNS}}\right)\%\right]}{\text{Gap}\left(\text{PSO}_{\text{VNS}}\right)\%}\end{aligned} \tag{13}$$

in which Gap(A)% means the value of Gap(%) obtained by a certain testing algorithm denoted by A (e.g., A can be $PSO_{adaptiveVNS}$ or PSO_{VNS}).

Based on these results, we can obtain the following observations:

(1) With the increase of instance size, the performance of all of the three algorithms tends to deteriorate because the problems become more and more difficult as their sizes increase.

(2) CPLEX 12.4 can obtain optimal solutions for small size instances, but for large size instances its performance deteriorates quickly. The main reason is that for large size problems the search space (i.e., the number of branching nodes) becomes too large for CPLEX to search and consequently its performance becomes much worse within the given computational time.

(3) The PSO_{VNS} algorithm shows much better results than CPLEX within a much shorter computational time, and it can achieve the best results for 3 out of the 18 instance scenarios.

(4) The proposed $PSO_{adaptiveVNS}$ algorithm obtains the best average results and it can provide 15 out of the 18 instance scenarios. In addition, the performance difference between our algorithm and the PSO_{VNS} algorithm is significant for 6 large size problems. For medium and large size instances (e.g., $m = 3$ and $n \geq 120$, $m = 5$ and $n \geq 100$, and $m = 7$ and $n \geq 80$), the improvement of our algorithm over the PSO_{VNS} varies from 5.63% to 16.53% and the average improvement for all test instances is about 5.61%. Therefore, it can be concluded that the proposed $PSO_{adaptiveVNS}$ algorithm is very effective for this problem.

5. Conclusions

Unlike previous literatures dealing with the single machine scheduling problem, this paper further incorporates two constraints derived from practical manufacturing industries, that is, the capacity of machine within a given scheduling time horizon and the demands of the downstream production lines, to ensure the consistent production. For the new variant of single machine scheduling problem, a mixed integer linear

Table 5: Comparison results for different kinds of PSO algorithms.

m	n	CPLEX		PSO_{VNS}		$PSO_{adaptiveVNS}$		
		Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	Gains%	CPU (s)
3	50	**0.00**	32.54	**0.00**	17.01	**0.00**	0.00	12.74
	80	**0.00**	48.75	0.06	87.35	0.06	0.00	71.35
	100	1.14	1637.63	**0.46**	150.00	**0.46**	0.00	150.00
	120	0.95	1615.27	0.45	180.00	**0.41**	8.89	180.00
	150	2.56	1608.52	1.42	225.00	**1.34**	5.63	225.00
	200	5.39	1605.28	2.86	300.00	**2.69***	5.94	300.00
5	50	**0.00**	74.07	**0.00**	27.76	**0.00**	0.00	22.06
	80	**0.09**	581.50	0.17	200.00	0.17	0.00	200.00
	100	1.68	1617.86	0.68	250.00	**0.62**	8.82	250.00
	120	2.36	1612.21	1.24	300.00	**1.15**	7.26	300.00
	150	2.81	1607.12	1.79	375.00	**1.66***	7.26	375.00
	200	6.86	1603.56	3.66	500.00	**3.34***	8.74	500.00
7	50	**0.00**	71.64	0.02	26.54	0.02	0.00	24.86
	80	1.04	1606.09	0.94	280.00	**0.86**	8.51	280.00
	100	2.77	1624.07	1.47	350.00	**1.34***	8.84	350.00
	120	2.51	1602.53	1.39	420.00	**1.31**	5.76	420.00
	150	4.36	1609.16	2.36	525.00	**1.97***	16.53	525.00
	200	8.42	1615.82	4.58	700.00	**4.18***	8.73	700.00

*Denotes that the performance difference between $PSO_{adaptiveVNS}$ and PSO_{VNS} is significantly for a certain problem size based on the *Pairwise-Samples t Test* with a confidence level of 95%.

programming model is presented and an improved particle swarm optimization algorithm is developed. The proposed PSO algorithm has two main improvement strategies: the first one is the elite solution pool used to improve the search diversity and the second one is the adaptive variable neighborhood search based on a job-block neighborhood used to improve the search intensification. In addition, an adaptive strategy is used in the AVNS to select neighborhoods that have higher probability of finding better solutions. Extensive experiments have been carried out to illustrate the effectiveness of the proposed improvement strategies and computational results show that the proposed PSO algorithm is able to achieve solutions whose average gap over the lower bound is about 1.2% for all 180 test instances. Future research can be the application of this algorithm to some practical scheduling problems in some manufacturing industries.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (Grants 61403277) and Humanities and Social Science Research Projects for Colleges in Tianjin (Grants 20132151).

References

[1] C. Koulamas, "The single-machine total tardiness scheduling problem: review and extensions," *European Journal of Operational Research*, vol. 202, no. 1, pp. 1–7, 2010.

[2] E. L. Lawler, "A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness," *Annals of Discrete Mathematics*, vol. 1, pp. 331–342, 1977.

[3] T. S. Abdul-Razaq, C. N. Potts, and L. N. van Wassenhove, "A survey of algorithms for the single machine total weighted tardiness scheduling problem," *Discrete Applied Mathematics*, vol. 26, no. 2-3, pp. 235–253, 1990.

[4] S. Avci, M. S. Akturk, and R. H. Storer, "A problem space algorithm for single machine weighted tardiness problems," *IIE Transactions*, vol. 35, no. 5, pp. 479–486, 2003.

[5] W. Bozejko, J. Grabowski, and M. Wodecki, "Block approach-tabu search algorithm for single machine total weighted tardiness problem," *Computers & Industrial Engineering*, vol. 50, no. 1-2, pp. 1–14, 2006.

[6] O. Holthatis and C. Rajendran, "A fast ant-colony algorithm for single-machine scheduling to minimize the sum of weighted tardiness of jobs," *Journal of the Operational Research Society*, vol. 56, no. 8, pp. 947–953, 2005.

[7] X. P. Wang and L. X. Tang, "A population-based variable neighborhood search for the single machine total weighted tardiness problem," *Computers & Operations Research*, vol. 36, no. 6, pp. 2105–2110, 2009.

[8] R. K. Congram, C. N. Potts, and S. L. van de Velde, "An iterated dynasearch algorithm for the single-machine total

weighted tardiness scheduling problem," *INFORMS Journal on Computing*, vol. 14, no. 1, pp. 52–67, 2002.

[9] W.-H. Yang and C.-J. Liao, "Survey of scheduling research involving setup times," *International Journal of Systems Science*, vol. 30, no. 2, pp. 143–155, 1999.

[10] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan, "A review of scheduling research involving setup considerations," *Omega*, vol. 27, no. 2, pp. 219–239, 1999.

[11] S. R. Das, J. N. D. Gupta, and B. M. Khumawala, "Saving index heuristic algorithm for flowshop scheduling with sequence dependent set-up times," *Journal of the Operational Research Society*, vol. 46, no. 11, pp. 1365–1373, 1995.

[12] P. M. França, M. Gendreau, G. Laporte, and F. M. Müller, "A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times," *International Journal of Production Economics*, vol. 43, no. 2-3, pp. 79–89, 1996.

[13] M. Gravel, W. L. Price, and C. Gagné, "Scheduling jobs in an Alcan aluminium foundry using a genetic algorithm," *International Journal of Production Research*, vol. 38, no. 13, pp. 3031–3041, 2000.

[14] X. Q. Sun, J. S. Noble, and C. M. Klein, "Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness," *IIE Transactions*, vol. 31, no. 2, pp. 113–124, 1999.

[15] C. Gagné, W. L. Price, and M. Gravel, "Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times," *Journal of the Operational Research Society*, vol. 53, no. 8, pp. 895–906, 2002.

[16] N. Al Theeb and T. Alhwiti, "Solving single-machine weighted tardiness and total setup costs scheduling problem with sequence dependent setup times and sequence dependent setup cost using discrete particle swarm," in *Proceedings of the Industrial and Systems Engineering Research Conference*, Montreal, Canada, 2014.

[17] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *European Journal of Operational Research*, vol. 187, no. 3, pp. 985–1032, 2008.

[18] P. Jula and A. Rafiey, "Coordinated scheduling of a single machine with sequence-dependent setup times and time-window constraints," *International Journal of Production Research*, vol. 50, no. 8, pp. 2304–2320, 2012.

[19] X. P. Wang and L. X. Tang, "A hybrid metaheuristic for the prize-collecting single machine scheduling problem with sequence-dependent setup times," *Computers and Operations Research*, vol. 37, no. 9, pp. 1624–1640, 2010.

[20] M. Gendreau, G. Laporte, and F. Semet, "A tabu search heuristic for the undirected selective travelling salesman problem," *European Journal of Operational Research*, vol. 106, no. 2-3, pp. 539–545, 1998.

[21] M. Dell'amico, F. Maffioli, and A. Sciomachen, "A lagrangian heuristic for the prize collecting traveling salesman problem," *Annals of Operations Research*, vol. 81, pp. 289–305, 1998.

[22] L. X. Tang and X. P. Wang, "A hybrid multi-objective evolutionary algorithm for multi-objective optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 20–45, 2013.

[23] L. Chen, X. P. Wang, and L. X. Tang, "Operation optimization in the hot-rolling production process," *Industrial & Engineering Chemistry Research*, vol. 53, no. 28, pp. 11393–11410, 2014.

[24] L. X. Tang, Y. Dong, and J. Y. Liu, "Differential evolution with an individual-dependent mechanism," *IEEE Transactions on Evolutionary Computation*, 2014.

[25] L. Tang, G. Wang, and Z. Chen, "Integrated charge batching and casting width selection at Baosteel," *Operations Research*, vol. 62, no. 4, pp. 772–787, 2014.

[26] L. X. Tang, Y. Zhao, and J. Y. Liu, "An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 209–225, 2014.

[27] Q. X. Guo and L. X. Tang, "An improved scatter search algorithm for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times," *Applied Soft Computing*, vol. 29, pp. 184–195, 2015.

[28] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, December 1995.

[29] H. B. Nanvala, "A hybrid metaheuristic for the prize-collecting single machine scheduling problem with sequence-dependent setup times," *International Journal of Engineering and Technology*, vol. 3, no. 2, pp. 80–86, 2011.

[30] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1930–1947, 2007.

[31] M. Nawaz, E. E. Enscore Jr., and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.