Research Article

A Decomposition-Based Two-Stage Optimization Algorithm for Single Machine Scheduling Problems with Deteriorating Jobs

Yueyue Liu, Rui Zhang, Miaomiao Wang, and Xiaoxi Zhu

School of Economics and Management, Nanchang University, Nanchang 330031, China

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

Received 14 February 2015; Revised 7 May 2015; Accepted 12 May 2015

Academic Editor: Marco Mussetta

Copyright © 2015 Yueyue Liu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper studies a production scheduling problem with deteriorating jobs, which frequently arises in contemporary manufacturing environments. The objective is to find an optimal sequence of the set of jobs to minimize the total weighted tardiness, which is an indicator of service quality. The problem belongs to the class of NP-hard. When the number of jobs increases, the computational time required by an optimization algorithm to solve the problem will increase exponentially. To tackle large-scale problems efficiently, a two-stage method is presented in this paper. We partition the set of jobs into a few subsets by applying a neural network approach and thereby transform the large-scale problem into a series of small-scale problems. Then, we employ an improved metaheuristic algorithm (called GTS) which combines genetic algorithm with tabu search to find the solution for each subproblem. Finally, we integrate the obtained sequences for each subset of jobs and produce the final complete solution by enumeration. A fair comparison has been made between the two-stage method and the GTS without decomposition, and the experimental results show that the solution quality of the two-stage method is much better than that of GTS for large-scale problems.

1. Introduction

The problem of scheduling jobs on a single machine to minimize total weighted tardiness is extensively studied by many researchers, which also occurs as a subproblem in other scheduling environments such as job shops. The classic scheduling models routinely assume that the job processing times are known and fixed, which however may not be satisfied in many real-world situations. For example, the deterioration effect is known as the fact that a job may need longer processing time if its starting time is postponed. This paper considers the single machine total weighted tardiness problem with proportional deterioration, which is common in steel manufacturing, plastic processing and medical treatments, and so forth [1].

Lawler [2] and Lenstra et al. [3] have shown that the total weighted tardiness problem is NP-hard. Congram et al. [4] proposed an effective local search method for this kind of problem. Bozejko et al. [5] presented a fast local search procedure based on a tabu search approach which employs blocks of jobs and compound moves and applied it to the total weighted tardiness problem.

J. N. D. Gupta and S. K. Gupta [6] and Browne and Yechiali [7] first introduced deterioration into scheduling problems. Recently, there have been growing interests in studying scheduling problems with deteriorating jobs [8]. Bachman and Janiak [9] considered the problems of minimizing maximum lateness under linear deterioration which is NP-hard and presented two heuristic algorithms. Bachman et al. [10] dealt with the single machine scheduling problem with start time dependent job processing times. Cheng et al. [11] introduced a class of machine scheduling problems in which the processing time of a task is dependent on its starting time in a schedule. Hsu and Lin [12] designed a branch-and-bound algorithm for deriving exact solutions according to several properties concerning dominance relations and lower bounds for the single machine problem with deteriorating jobs to minimize the maximum lateness. Wang et al. [13] considered single machine scheduling problems where the processing time is the increasing function of their starting times and the jobs are related by a series-parallel graph. More recent contributions in this line of research can be referred to [14–18], where integrated scheduling problems with deteriorating jobs are studied.

In terms of general single machine scheduling, many different solution approaches have been proposed in the existing literature. Sels and Vanhoucke [19] developed a hybrid dual-population genetic algorithm for the single machine maximum lateness problem which took some specific characteristics into account. Their work has an important implication on the balance between intensification and diversification in the design of search algorithms. Voutsinas and Pappis [20] proposed a branch-and-bound algorithm which uses the suboptimal solution of a heuristic as initial solution for solving the single machine scheduling problem with deteriorating jobs. Jolai et al. [21] focused on the bicriteria scheduling problem of minimizing the number of tardy jobs and maximum earliness for single machine scheduling without allowing idle times. Pakzad-Moghaddam et al. [22] presented a mixed-integer mathematical programming model for a single machine scheduling problem with deteriorating and learning effects. Xu et al. [23] solved the single machine scheduling problem with sequence-dependent setup times and conducted a systematic comparison of hybrid evolutionary algorithms (HEAs), which independently used the six combinations of three crossover operators and two population updating strategies. More recent works that deal with advanced single machine scheduling problems can be found in [24–28].

This paper considers the single machine total weighted tardiness problem with proportional deterioration. Our objective is to minimize the total weighted tardiness of jobs. The problem belongs to the class of NP-hard problems. With the increase of the job number, the time an algorithm consumes to solve the problem increases exponentially. Hence, a two-stage method based on decomposition is presented. First, we perform partition using neural networks to transfer the large-scale problem into smaller-scale subproblems. Then, we find the solution for each subproblem by a hybrid metaheuristic algorithm. Indeed, we employ an improved genetic algorithm, called genetic tabu search (GTS), which uses tabu search (TS) as the mutation operator to increase the performance of local search. Genetic algorithm (GA) is an optimization method based on the principles of genetics and natural selection. However, it is weak in local search and has premature convergence, so we combine it with a tabu search approach to enhance the algorithm. Finally, we combine the obtained sequence for each partition of jobs and give the final solution by means of enumeration.

This paper is organized as follows. In Section 2, the basic definitions and notations of the problem are presented. Section 3 describes the detailed algorithms in the two-stage method, including the neural network, the genetic algorithm, and tabu search. Computational results are shown in Section 4. Section 5 presents the conclusion.

2. Problem Description

We consider the problem of scheduling a set of n jobs on a single machine with the following assumptions:

- (i) All jobs are ready at time zero.
- (ii) The machine can handle only one job at a time.

- (iii) There are no precedence relations between jobs and preemption is not allowed.
- (iv) Setup times are not explicitly considered.
- (v) The basic processing times, the latest starting times, and the deterioration factor are known.

The single machine total weighted tardiness problem addressed in this paper is affected by the deterioration effect, which is a phenomenon commonly observed in many cases. For example, in the process of steel-making, as the temperature of an ingot drops below a specified level, it must be heated again to the temperature required for rolling.

The problem can be described as follows. There is a set $N = \{1, 2, \dots, n\}$ of n jobs (index $i, i = 1, 2, \dots, n$) that have to be scheduled on a single machine. Each job i is characterized by a basic processing time t_{bi} , a latest starting time t_{di} , a weight w_i , and a due date d_i . For a given sequence of jobs, the (earliest) completion time C_i , tardiness $T_i = \max\{0, C_i - d_i\}$, and cost $f_i(C_i) = w_i T_i$ of job $i \in N$ can be computed. The objective is to find a job sequence in which the given set of jobs are scheduled such that the total weighted tardiness $\sum_{i=1}^n f_i(C_i) = \sum_{i=1}^n w_i T_i$ with respect to the given due dates is minimized.

Besides, the actual processing time of each job is a nondecreasing function of its starting time. When the starting time of job i (t_{si}) is earlier than or equal to the latest starting time t_{di} which is given, the actual processing time t_{pi} equals the basic processing time t_{bi} . Otherwise, t_{pi} is a function of t_{si} as defined below:

$$t_{pi} = \begin{cases} t_{bi}, & \text{if } t_{si} \leq t_{di}, \\ t_{bi} + b_i(t_{si} - t_{di}), & \text{if } t_{si} > t_{di}, \end{cases} \quad (1)$$

where b_i is the deterioration factor.

The notations that will be used throughout this paper are summarized in Notation section.

3. A Two-Stage Solution Method Based on Decomposition

3.1. Methodology Overview. To tackle complex optimization problems efficiently, the utilization of problem-specific properties is highly important in the design of optimization algorithms. As for the problem studied in this paper, since each job has several features including a basic processing time t_{bi} , a latest starting time t_{di} , a weight w_i , a due date d_i , and a deterioration factor b_i , the distributions of these data tend to have certain characteristics which can be extracted and utilized as special information to guide the search behavior of optimization algorithms. To improve the time efficiency of the optimization process, we transform the original large-scale scheduling problem into several small-scale scheduling problems based on the features of each job using a back-propagation (BP) neural network.

Firstly, we get the optimal solutions of some small-scale problems for training the neural network. After establishing the neural network which can roughly predict the position of each job, we divide the original set of jobs into several

- (1) **Input:** N
- (2) **Output:** the best job sequence found so far
- (3) Population Initialization
- (4) **Repeat**
- (5) Randomly choose two individuals from the current population
- (6) Crossover
- (7) Mutation (tabu_search)
- (8) Fitness_Evaluation
- (9) Selection
- (10) **Until** the stop criterion is met

ALGORITHM 1: Pseudocode of the GTS.

subsets, each corresponding to a scheduling subproblem. For example, if we are to have 4 subproblems, then we could use the neural network to predict which jobs will belong to the first 1/4 in the final schedule (and thus will be placed into the first subset), which will belong to the next 1/4 (and thus will be placed into the second subset), and so forth. Then, we use the proposed GTS to obtain the solution for each subproblem and combine them to yield the final solution.

3.2. The First Stage. The BP neural network is a typical artificial neural network that can approximate complex nonlinear mapping functions. It has been widely adopted in application areas like classification, fitting, and compression. A typical BP neural network consists of three layers including an input layer, a hidden layer, and an output layer. The number of nodes in the input layer depends on the dimension of the input vector. In this paper, each job has five features including a basic processing time t_{bi} , a latest starting time t_{di} , a weight w_i , a due date d_i and a deterioration factor b_i . Generally, a single hidden layer can realize the arbitrary nonlinear mapping by increasing the number of neuron nodes appropriately. The number of nodes in the hidden layer, which can adjust the accuracy of the neural network, affects the training results and the training time. With an increase of the node number, the training results improve while the training time increases correspondingly. The number of neurons in the output layer is decided according to the practical problem, and, in this study, it is equal to the number of subproblems we are planning to divide into. The Quasi-Newton method, which is a fast optimization technique based on Taylor series expansion, is applied to train the neural network.

The network is trained based on sample data with known output derived from small-scale problem instances. The data consist of the information of each job with the above features and the sequential index of the subset it belongs to. The data is grouped into two parts, the training data for training the neural network and the test data for testing the correctness of the classification.

3.3. The Second Stage. Although GA can be directly applied to complex combinatorial optimization problems, each generation of the algorithm must maintain a large population size. With the expansion of the problem size, the computational

time needed will increase dramatically. Besides, GA usually converges prematurely, which is mainly caused by a lack of diversity in the population. In addition, the mutation operator is inadequate for a systematic local search. Compared with GA, TS has faster convergence rate. However, the search performance of TS greatly depends on the initial solution.

Population-based GA and single-trajectory TS have complementary characteristics. GA explores well the search space while TS intensifies the search in promising regions. According to the strengths and weaknesses of these two algorithms, we apply TS to replace the mutation operator in GA.

We present the general framework of the GTS in Algorithm 1. GTS starts from an initial random population (line 3). Then, the crossover operator is employed to generate new offspring solutions (line 6). Besides, the mutation operator is implemented with TS to enhance the local search performance (line 7). Subsequently, the population updating rule decides whether such a mutated solution should be inserted into the population and which existing individual should be replaced (line 9).

3.3.1. Encoding and Decoding. A chromosome is represented by a sequence of jobs and each gene is denoted by the index of a job. For example, chromosome $V = (3, 7, 1, 10, 4, 8, 2, 9, 6, 5)$ means that job 3 is processed first and job 7 comes second and so on.

3.3.2. Initial Population. Usually, we use specific heuristic rules combined with random methods to generate the initial population. However, since in this study the GTS is only used to solve each subproblem with limited size, the individuals of the initial population are generated randomly to encourage solution diversity.

3.3.3. Fitness and Selection. Here the fitness is calculated simply according to the formula of the total weighted tardiness. We select better offspring based on their fitness and the individual whose fitness is better will be selected with higher probability.

3.3.4. Crossover Operator. Crossover is used to generate new offspring by recombining the selected parents, so it

- (1) Solution Initialization
- (2) **Repeat**
- (3) Generate a candidate set of solutions
- (4) Select a solution and update the tabu list
- (5) **Until** the stop criterion is met

ALGORITHM 2: Pseudocode of tabu search.

is the key operator in GA. There are several crossover methods for combinatorial problems including linear order crossover (LOX), position-based crossover (PBX), and partially mapped crossover (PMX). We employ the LOX method in GTS.

3.3.5. Tabu Search Based Mutation. The mutation operator is reimplemented with a tabu search algorithm. Here, we take the solution obtained by GA as the initial solution for the TS. The neighborhood structure is defined as swap and we can obtain a neighborhood solution when we choose two different genes randomly and then exchange their locations. Consequently, the two genes which have been exchanged are recorded as the element in the tabu list. Besides, when the new solution is better than the best-so-far solution, we accept it no matter whether the move exists in the tabu list. The procedure of the tabu search is presented in Algorithm 2.

The fundamental idea underlying tabu search is to avoid repeated search in the same area of the solution space. To this end, tabu search involves some essential concepts such as tabu list, tabu length, and aspiration criterion. To make a clearer description of the tabu search module, we provide a specific example detailed as follows.

Suppose that we are dealing with a sequencing problem in which a solution is represented by a permutation of $\{1, 2, \dots, 7\}$. The initial solution is $(2, 5, 7, 3, 4, 6, 1)$, with an objective value of 10. The tabu list is initialized as an empty set (i.e., $H = \emptyset$). The tabu length is set as 3. The neighborhood solutions are generated by the swap operator which exchanges two elements in the current permutation. In the first iteration, the 5 best candidate solutions in the neighborhood are first identified (they are listed in Figure 1 in terms of the elements to be swapped). The best one, $(5, 4)$, is selected, and the new solution is obtained by swapping 5 and 4. Now, the pair $(5, 4)$ is put into the tabu list (i.e., $H = \{(5, 4; 3)\}$), which means the swap of 5 and 4 is not allowed in the subsequent 3 iterations (unless it leads to a new best solution). In the second iteration, the current solution has an objective value of 4, and, also, the 5 best candidate solutions are identified. The best one, $(3, 1)$, is chosen, and then the pair $(3, 1)$ must be put into the tabu list. The updated tabu list is therefore $H = \{(5, 4; 2), (3, 1; 3)\}$. In the third iteration, it is found that all the neighborhood solutions are inferior to the current solution (with an objective value of 2). However, according to the tabu search principles, the best solution that is not tabooed should be selected as the new solution. Therefore, $(2, 4)$ is

TABLE 1: Data generation rules.

Item	Distribution
Basic processing times	$t_{bi} \sim U(1, 100)$
Deterioration factors	$b_i \sim U(0, 1)$
Latest starting times	$t_{di} \sim 30n \times U(0, 1)$
Job numbers	$n \in \{25, 50, 100, 200\}$

selected because $(1, 3)$ is in the tabu list, and afterwards the tabu list should be updated as $H = \{(5, 4; 1), (3, 1; 2), (2, 4; 3)\}$. In the fourth iteration, the current objective value is 6, and the 5 best neighborhood solutions are identified as usual. It is noteworthy that, although $(4, 5)$ is still in the tabu list at this stage, swapping 4 and 5 can lead to an improved upper-bound solution with a new best objective value of 0. The aspiration criterion is activated at this moment, which means the solution obtained by swapping 4 and 5 should be selected as the new current solution for the next iteration. Meanwhile, we also need to update the tabu list as $H = \{(3, 1; 1), (2, 4; 2), (5, 4; 3)\}$.

3.3.6. Stopping Condition. GTS is terminated if the best-so-far solution does not change for 50 consecutive generations or the given execution time is exhausted, whichever comes first.

4. Computational Experiments

Because the problem has not been investigated in the literature, there is no standard data set available. Nevertheless, the literature on the single machine scheduling problem with deteriorating jobs without considering the weight of the jobs provides us with several data generation techniques. The basic processing times t_{bi} are uniformly distributed integers between 1 and 100. The deterioration factors are uniformly distributed in the range $U(0, 1)$ and the latest starting times are $30n \times U(0, 1)$, where n is the number of jobs. The parameters are summarized in Table 1.

In order to refine the two-stage method, the number of subproblems, the tabu length, and the mutation probability are chosen to be fine-tuned by computational tests. In each cycle, a single parameter was chosen to be fine-tuned, while the other parameters were set to their recommend values.

4.1. Number of Subproblems. The number of subproblems that the original problem is divided into can affect the solution quality, because the optimization accuracy will be decreased if the number is too large, and there will be little effect on reducing the computational complexity if the number is too small. For convenience, the number of subproblems is tested at integer values which exactly divide the number of jobs. Here, ten different instances with 100 jobs are tested, and the number of subproblems is set as 2, 4, and 5, respectively. Each test is repeated 30 times to reduce random errors.

We use ten randomly generated instances of 100 jobs and divided the jobs into 2 subsets (each with 50 jobs),

<div>Iteration 1: Current solution (obj. = 10)</div> <table><tr><td>2</td><td>5</td><td>7</td><td>3</td><td>4</td><td>6</td><td>1</td></tr></table> <div>Candidate solutions</div> <table><tr><th>Swap</th><th>Δobj.</th></tr><tr><td>(5, 4)</td><td>-6</td></tr><tr><td>(7, 4)</td><td>-4</td></tr><tr><td>(3, 6)</td><td>-2</td></tr><tr><td>(2, 3)</td><td>0</td></tr><tr><td>(4, 1)</td><td>1</td></tr></table>	2	5	7	3	4	6	1	Swap	Δ obj.	(5, 4)	-6	(7, 4)	-4	(3, 6)	-2	(2, 3)	0	(4, 1)	1	<div>Iteration 2: Current solution (obj. = 4)</div> <table><tr><td>2</td><td>4</td><td>7</td><td>3</td><td>5</td><td>6</td><td>1</td></tr></table> <div>Candidate solutions</div> <table><tr><th>Swap</th><th>Δobj.</th></tr><tr><td>(3, 1)</td><td>-2</td></tr><tr><td>(2, 3)</td><td>-1</td></tr><tr><td>(3, 6)</td><td>1</td></tr><tr><td>(7, 1)</td><td>2</td></tr><tr><td>(6, 1)</td><td>4</td></tr></table>	2	4	7	3	5	6	1	Swap	Δ obj.	(3, 1)	-2	(2, 3)	-1	(3, 6)	1	(7, 1)	2	(6, 1)	4	<div>Iteration 3: Current solution (obj. = 2)</div> <table><tr><td>2</td><td>4</td><td>7</td><td>1</td><td>5</td><td>6</td><td>3</td></tr></table> <div>Candidate solutions</div> <table><tr><th>Swap</th><th>Δobj.</th></tr><tr><td>(1, 3)</td><td>2</td></tr><tr><td>(2, 4)</td><td>4</td></tr><tr><td>(7, 6)</td><td>6</td></tr><tr><td>(4, 5)</td><td>7</td></tr><tr><td>(5, 3)</td><td>9</td></tr></table>	2	4	7	1	5	6	3	Swap	Δ obj.	(1, 3)	2	(2, 4)	4	(7, 6)	6	(4, 5)	7	(5, 3)	9
2	5	7	3	4	6	1																																																					
Swap	Δ obj.																																																										
(5, 4)	-6																																																										
(7, 4)	-4																																																										
(3, 6)	-2																																																										
(2, 3)	0																																																										
(4, 1)	1																																																										
2	4	7	3	5	6	1																																																					
Swap	Δ obj.																																																										
(3, 1)	-2																																																										
(2, 3)	-1																																																										
(3, 6)	1																																																										
(7, 1)	2																																																										
(6, 1)	4																																																										
2	4	7	1	5	6	3																																																					
Swap	Δ obj.																																																										
(1, 3)	2																																																										
(2, 4)	4																																																										
(7, 6)	6																																																										
(4, 5)	7																																																										
(5, 3)	9																																																										
<div>Iteration 4: Current solution (obj. = 6)</div> <table><tr><td>4</td><td>2</td><td>7</td><td>1</td><td>5</td><td>6</td><td>3</td></tr></table> <div>Candidate solutions</div> <table><tr><th>Swap</th><th>Δobj.</th></tr><tr><td>(4, 5)</td><td>-6</td></tr><tr><td>(5, 3)</td><td>-2</td></tr><tr><td>(7, 1)</td><td>0</td></tr><tr><td>(1, 3)</td><td>3</td></tr><tr><td>(2, 6)</td><td>6</td></tr></table>	4	2	7	1	5	6	3	Swap	Δ obj.	(4, 5)	-6	(5, 3)	-2	(7, 1)	0	(1, 3)	3	(2, 6)	6	<div>Iteration 5: Current solution (obj. = 0)</div> <table><tr><td>5</td><td>2</td><td>7</td><td>1</td><td>4</td><td>6</td><td>3</td></tr></table> <div>Candidate solutions</div> <table><tr><th>Swap</th><th>Δobj.</th></tr><tr><td>(7, 1)</td><td>0</td></tr><tr><td>(4, 3)</td><td>3</td></tr><tr><td>(6, 3)</td><td>5</td></tr><tr><td>(5, 4)</td><td>6</td></tr><tr><td>(2, 6)</td><td>8</td></tr></table>	5	2	7	1	4	6	3	Swap	Δ obj.	(7, 1)	0	(4, 3)	3	(6, 3)	5	(5, 4)	6	(2, 6)	8																				
4	2	7	1	5	6	3																																																					
Swap	Δ obj.																																																										
(4, 5)	-6																																																										
(5, 3)	-2																																																										
(7, 1)	0																																																										
(1, 3)	3																																																										
(2, 6)	6																																																										
5	2	7	1	4	6	3																																																					
Swap	Δ obj.																																																										
(7, 1)	0																																																										
(4, 3)	3																																																										
(6, 3)	5																																																										
(5, 4)	6																																																										
(2, 6)	8																																																										

FIGURE 1: Illustration of the tabu search algorithm.

4 subsets (each with 25 jobs), and 5 subsets (each with 20 jobs) in the three experiments. We obtain the solution for each subproblem and then combine the solutions to obtain the final result. The computational time allowed for each instance is equal and here we set the time limit as 1 minute. Based on the averaged total weighted tardiness, we can judge the best number of subproblems.

An example under the setting of 2 subproblems is described as follows. First, we divide the given jobs into 2 subsets of 50 jobs and record the top 20 solutions for each subproblem after applying the GTS within the time limit of 30 seconds. Next, we combine the solutions by a full enumeration. A total of 400 (20×20) combinations are considered and the best one is selected. We assume the time needed to validate the combined solutions is negligible compared to the time consumed by GTS.

In Table 2, the test results are given in terms of the averaged total weighted tardiness. Overall, when the number of subproblems is set to 4, the final solution quality is the best.

4.2. Tabu Length. In the TS module, the tabu length has great influence on the search speed and the solution quality. If the length of tabu list is too small, repeated search may frequently happen. On the contrary, it will exclude the search

TABLE 2: Comparison among different numbers of subproblems.

Problem	Number of subproblems		
	2	4	5
1	604434	500550	553985
2	418412	329170	407383
3	602420	530410	628836
4	476975	336679	512190
5	417457	367804	506563
6	389990	334427	441047
7	570347	475935	612574
8	481801	403357	533737
9	459356	411443	554240
10	414181	471123	551213

paths which may lead to good solutions. Consequently, it is of great significance to confirm a suitable tabu length. Here, ten different instances of 50 jobs are tested with the tabu length set as 5, 7, 11, and 14, respectively, and each test is performed 30 times.

It is obvious from Table 3 that the results under tabu length 5 are much better than the others. Consequently, the tabu length is determined as 5.

TABLE 3: Comparison among different tabu lengths.

Problem	Tabu length			
	5	7	11	14
1	134613	184317	183208	180838
2	104881	136679	136514	133930
3	117146	153793	152919	153162
4	121507	171547	170033	170376
5	96158	137546	137103	134584
6	1039234	146959	145926	145900
7	108086	145278	146827	142996
8	118044	160532	159950	156173
9	58751	86418	87289	86187
10	128365	1680323	167237	164579

TABLE 4: Comparison among different crossover probabilities.

Problem	Crossover probability				
	0.5	0.6	0.7	0.8	0.9
1	156498	149962	148261	149655	110132
2	158193	152260	153507	154429	115348
3	144359	137106	137950	137782	97555
4	104388	99124	97431	98004	67944
5	181635	176879	176791	175853	145718
6	199300	192442	192860	193136	155364
7	123019	123814	121339	119890	100275
8	98601	98554	97450	95932	73561
9	108456	107456	104860	105493	85837
10	114432	113560	112900	112718	78344

4.3. Crossover Probability. In genetic algorithm, the crossover operator is one of the most significant factors to determine the quality of new offspring individuals. Similarly, ten different instances of 50 jobs are tested with the crossover probability set as 0.5, 0.6, 0.7, 0.8, and 0.9, respectively. According to the results presented in Table 4, 0.9 is the best setting for this parameter.

4.4. Comparative Evaluation of the Proposed Algorithm. The number of jobs is considered at four levels, that is, $n = 100, 200, 500$, and 1000 . Because each problem size is associated with 10 instances, the total number of instances is 40, and each instance is solved for 30 independent times by each algorithm tested. In this paper, the improvement rate is defined as $(TWT_{GTS} - TWT_{2-S})/TWT_{2-S}$, where TWT_{GTS} represents the objective value achieved by GTS alone (without decomposition), while TWT_{2-S} denotes the solution value obtained by the proposed two-stage algorithm. The BP neural network was implemented under Matlab R2013a and GTS was coded in C++ under Microsoft Visual C++ 2010 on an Intel Core i3 CPU and Windows 7 platform.

Table 5 shows the computational results of the two-stage method and GTS in terms of the mean objective values.

According to the table, the average percentage improvement of the proposed method over GTS is 4.36%, 10.05%, 14.21%, and 7.94% under $n = 100, 200, 500$, and 1000 , respectively.

It is noted that when the number of jobs is small the two-stage method sometimes performs not as well as GTS. For example, in the 10 instances of 100 jobs, there are 3 instances for which the proposed method performs even worse than the GTS without job partition. This is because the relative positions of some jobs will be fixed after applying the partition and thus the search domain of the optimization algorithm will be limited. The partition of jobs is conducted based on the selected features of the jobs and is used to guide the search algorithm. However, to some extent, it limits the search capability of the algorithm in some directions. Thus, we believe that the effects of job partition can be divided into two parts including a positive guiding effect and a negative limiting effect. When the number of jobs is small, the positive effect of guiding may have less influence than the negative effect of limiting the search. However, with the increase of jobs, the positive effect becomes overwhelming, and the two-stage method outperforms the GTS in a very considerable manner, which reveals the valuable contribution of the job partition methodology.

5. Conclusion

This paper studies the single machine total weighted tardiness scheduling problem with deterioration. Deterioration is related to the processing time and starting time of jobs. When the starting time of a job is later than the latest starting time which is given, the processing time will increase. It is common in most just-in-time production environments with human participation.

We propose a hybrid algorithm called GTS which combines GA and TS to solve the problem. In order to tackle large-scale instances, we conduct job partition by means of a neural network approach and then find the solution for each partition by GTS. Finally, we combine the job sequence for each partition and give the final solution by enumeration. We named the GTS with partition the two-stage method. To demonstrate whether the two-stage method performs better than GTS for large-scale problems, we use 40 instances for the comparison purpose. The average improvement rate is considerable as revealed by the computational results.

Notation

- n : Number of jobs
- J_i : Job i , $i = 1, 2, \dots, n$
- t_{pi} : Actual processing time of J_i
- t_{bi} : Basic processing time of J_i
- t_{si} : Starting time of J_i
- t_{di} : Latest starting time for J_i
- w_i : Weight of J_i
- d_i : Due date for J_i
- C_i : Completion time of J_i
- b_i : Deterioration factor of J_i .

TABLE 5: Comparison between the performances of the two-stage method and GTS.

Problem	100 jobs			Problem	200 jobs		
	Two-stage method	GTS	Improvement (%)		Two-stage method	GTS	Improvement (%)
1	529251	543701	2.73	1	2772482	2693236	-2.86
2	580368	561897	-3.18	2	2254343	2646620	17.40
3	471791	531532	12.66	3	2549536	2808366	10.15
4	467193	552274	18.21	4	2701978	2582402	-4.43
5	498979	502002	0.61	5	2214216	2653940	19.86
6	592447	542141	-8.49	6	2991533	3263023	9.08
7	458492	486609	6.13	7	2066705	2344396	13.44
8	590894	554199	-6.21	8	2408844	2555122	6.07
9	572005	636890	11.34	9	2331178	2802221	20.21
10	366301	402192	9.80	10	2401338	2679454	11.58
Problem	500 jobs			Problem	1000 jobs		
	Two-stage method	GTS	Improvement (%)		Two-stage method	GTS	Improvement (%)
1	16526240	18984561	14.88	1	82521970	89691988	8.69
2	16917170	19958887	17.98	2	78710890	84133722	6.89
3	17430260	19615766	12.54	3	83526650	89416688	7.05
4	17960130	19779356	10.13	4	82014700	86538534	5.52
5	17383250	19476219	12.04	5	76615210	84045424	9.70
6	16520930	18978797	14.88	6	79475530	86345229	8.64
7	15694230	18608654	18.57	7	82407780	84327182	2.33
8	18541230	21285024	14.80	8	73426980	80531920	9.68
9	20040810	22896113	14.25	9	81376060	90411751	11.10
10	17562700	19684769	12.08	10	72945010	80092001	9.80

Conflict of Interests

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

Acknowledgment

This work is supported by the Natural Science Foundation of China under Grants nos. 61473141 and 61273233.

References

- [1] B. Alidaee and N. K. Womer, "Scheduling with time dependent processing times: review and extensions," *Journal of the Operational Research Society*, vol. 50, no. 7, pp. 711–720, 1999.
- [2] E. L. Lawler, "A pseudo polynomial algorithm for sequencing jobs to minimize total tardiness," *Annals of Discrete Mathematics*, vol. 1, pp. 331–342, 1997.
- [3] J. K. Lenstra, A. G. H. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1997.
- [4] 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.
- [5] W. Bozejko, J. Grabowski, and M. Wodecki, "Block approach-tabu search algorithm for single machine total weighted tardiness problem," *Computers and Industrial Engineering*, vol. 50, no. 1-2, pp. 1–14, 2006.
- [6] J. N. D. Gupta and S. K. Gupta, "Single facility scheduling with nonlinear processing times," *Computers & Industrial Engineering*, vol. 14, no. 4, pp. 387–393, 1988.
- [7] S. Browne and U. Yechiali, "Scheduling deteriorating jobs on a single processor," *Operations Research*, vol. 38, no. 3, pp. 495–498, 1990.
- [8] J.-B. Wang and J.-J. Wang, "Single-machine scheduling problems with precedence constraints and simple linear deterioration," *Applied Mathematical Modelling*, vol. 39, no. 3-4, pp. 1172–1182, 2015.
- [9] A. Bachman and A. Janiak, "Minimizing maximum lateness under linear deterioration," *European Journal of Operational Research*, vol. 126, no. 3, pp. 557–566, 2000.
- [10] A. Bachman, T. C. E. Cheng, A. Janiak, and C. T. Ng, "Scheduling start time dependent jobs to minimize the total weighted completion time," *Journal of the Operational Research Society*, vol. 53, no. 6, pp. 688–693, 2002.
- [11] T. C. E. Cheng, Q. Ding, and B. M. T. Lin, "A concise survey of scheduling with time-dependent processing times," *European Journal of Operational Research*, vol. 152, no. 1, pp. 1–13, 2004.
- [12] Y. S. Hsu and B. M. T. Lin, "Minimization of maximum lateness under linear deterioration," *Omega*, vol. 31, no. 6, pp. 459–469, 2003.
- [13] J.-B. Wang, C. T. Ng, and T. C. E. Cheng, "Single-machine scheduling with deteriorating jobs under a series-parallel graph constraint," *Computers & Operations Research*, vol. 35, no. 8, pp. 2684–2693, 2008.
- [14] K. Rustogi and V. A. Strusevich, "Single machine scheduling with time-dependent linear deterioration and rate-modifying

- maintenance,” *Journal of the Operational Research Society*, vol. 66, no. 3, pp. 500–515, 2014.
- [15] J. Liu, Y. Wang, and X. Min, “Single-machine scheduling with common due-window assignment for deteriorating jobs,” *Journal of the Operational Research Society*, vol. 65, no. 2, pp. 291–301, 2014.
 - [16] D. Oron, “Scheduling controllable processing time jobs in a deteriorating environment,” *Journal of the Operational Research Society*, vol. 65, no. 1, pp. 49–56, 2014.
 - [17] K. Rustogi and V. A. Strusevich, “Single machine scheduling with general positional deterioration and rate-modifying maintenance,” *Omega*, vol. 40, no. 6, pp. 791–804, 2012.
 - [18] S. Gawiejnowicz, W.-C. Lee, C.-L. Lin, and C.-C. Wu, “Single-machine scheduling of proportionally deteriorating jobs by two agents,” *Journal of the Operational Research Society*, vol. 62, no. 11, pp. 1983–1991, 2011.
 - [19] V. Sels and M. Vanhoucke, “A hybrid dual-population genetic algorithm for the single machine maximum lateness problem,” in *Evolutionary Computation in Combinatorial Optimization*, vol. 6622 of *Lecture Notes in Computer Science*, pp. 14–25, Springer, Berlin, Germany, 2011.
 - [20] T. G. Voutsinas and C. P. Pappis, “A branch and bound algorithm for single machine scheduling with deteriorating values of jobs,” *Mathematical and Computer Modelling*, vol. 52, no. 1-2, pp. 55–61, 2010.
 - [21] F. Jolai, M. Rabbani, S. Amalnick, A. Dabaghi, M. Dehghan, and M. Y. Parast, “Genetic algorithm for bi-criteria single machine scheduling problem of minimizing maximum earliness and number of tardy jobs,” *Applied Mathematics and Computation*, vol. 194, no. 2, pp. 552–560, 2007.
 - [22] S. H. Pakzad-Moghaddam, H. Mina, and R. Tavakkoli-Moghaddam, “An approach for modeling a new single machine scheduling problem with deteriorating and learning effects,” *Computers & Industrial Engineering*, vol. 78, pp. 33–43, 2014.
 - [23] H. Xu, Z. Lü, A. Yin, L. Shen, and U. Buscher, “A study of hybrid evolutionary algorithms for single machine scheduling problem with sequence-dependent setup times,” *Computers and Operations Research*, vol. 50, pp. 47–60, 2014.
 - [24] D. Xu, L. Wan, A. Liu, and D.-L. Yang, “Single machine total completion time scheduling problem with workload-dependent maintenance duration,” *Omega*, vol. 52, pp. 101–106, 2015.
 - [25] E. R. Gafarov, A. A. Lazarev, and F. Werner, “Single machine scheduling problems with financial resource constraints: some complexity results and properties,” *Mathematical Social Sciences*, vol. 62, no. 1, pp. 7–13, 2011.
 - [26] Y. Yin, T. C. E. Cheng, C.-J. Hsu, and C.-C. Wu, “Single-machine batch delivery scheduling with an assignable common due window,” *Omega*, vol. 41, no. 2, pp. 216–225, 2013.
 - [27] D. Xu, M. Liu, Y. Yin, and J. Hao, “Scheduling tool changes and special jobs on a single machine to minimize makespan,” *Omega*, vol. 41, no. 2, pp. 299–304, 2013.
 - [28] Y. Yin, S.-R. Cheng, T. C. E. Cheng, W.-H. Wu, and C.-C. Wu, “Two-agent single-machine scheduling with release times and deadlines,” *International Journal of Shipping and Transport Logistics*, vol. 5, no. 1, pp. 75–94, 2013.