Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 8870917, 16 pages
https://doi.org/10.1155/2020/8870917

Research Article

Energy-Saving Production Scheduling in a Single-Machine Manufacturing System by Improved Particle Swarm Optimization

Qingquan Jiang, Xiaoya Liao, Rui Zhang, and Qiaozhen Lin

School of Economics & Management, Xiamen University of Technology, Xiamen 361024, China

Correspondence should be addressed to Xiaoya Liao; lxyjiang@gmail.com, Rui Zhang; r.zhang@ymail.com, and Qiaozhen Lin; 1174656792@qq.com

Received 22 September 2020; Revised 11 October 2020; Accepted 15 October 2020; Published 5 November 2020

Academic Editor: Chenxi Huang

A single-machine scheduling problem that minimizes the total weighted tardiness with energy consumption constraints in the actual production environment is studied in this paper. Based on the properties of the problem, an improved particle swarm optimization (PSO) algorithm embedded with a local search strategy (PSO-LS) is designed to solve this problem. To evaluate the algorithm, some computational experiments are carried out using PSO-LS, basic PSO, and a genetic algorithm (GA). Before the comparison experiment, the Taguchi method is used to select appropriate parameter values for these three algorithms since heuristic algorithms rely heavily on their parameters. The experimental results show that the improved PSO-LS algorithm has considerable advantages over the basic PSO and GA, especially for large-scale problems.

1. Introduction

Manufacturing is an important industry that consumes about one-third of the world's energy [1] and 56% of China's energy [2]. Saving energy not only reduces production cost but also protects the environment. Achieving energy savings through low-cost production scheduling is important [3] although it is possible to replace old machines with higher-priced and energy-efficient ones.

Energy-saving scheduling has received increasing attention in recent years. The literature on energy-efficient scheduling has considerably expanded since 2013 [4].

Lee et al. [5] studied single-machine scheduling with energy efficiency under a policy of time-varying electricity price and solved the problem using a dynamic control approach. Módos et al. [6] studied robust single-machine scheduling to minimize release times and total tardiness with periodic energy consumption limits. They proposed an efficient algorithm to obtain the optimal robust solution of given processing orders and used it in two exact algorithms and Tabu search. Che [7] focused on biobjective optimization to minimize total energy consumption and maximum tardiness in a single-machine scheduling problem considering a power-down mechanism. The problem was built as a mixed integer linear programming (MILP) model. They developed exact and approximate algorithms for small- and large-scale problems accordingly. Che et al. [8] established a new continuous-time mixed integer programming model for an energy-sensitive single-machine scheduling problem and proposed a greedy insertion algorithm to reduce the total electricity cost of production, which provided energy-saving solution for a large-scale single-machine scheduling problem, such as 5000 jobs, in a few tens of seconds. Chen et al. [9] studied the energy consumption of production activities from the point of view of machine reliability. They examined the actual problem in a rotor production workshop. The problem was a single-machine scheduling problem that optimized delay costs and energy costs. The ant colony algorithm embedded with a modified Emmons rule was used to solve this problem. In addition, they used sensitivity analysis to discuss special cases. By determining the start time of job processing, the idle time, and the time when the machine must be shut down, Shrouf et al. [10] aimed to minimize the cost of the energy consumption in a

single-machine scheduling problem in the production process, and the energy price was varied throughout a day. For large-scale problems, a genetic algorithm (GA) was used to obtain a satisfactory solution.

Interval number theory was used to describe the uncertainty of renewable energy such as wind energy and solar energy [11], and two new biobjective optimization problems for interval single-machine scheduling were studied. Through parameter analysis, the authors provided decision-makers with guidelines for practical production environment. In [12], a memetic differential evolution (MDE) algorithm with superior performances to strength Pareto evolutionary algorithm II (SPEA-II) and nondominated sorting genetic algorithm II (NSGA-II) was proposed to solve an energy-saving biobjective unrelated parallel-machine scheduling problem to minimize maximum completion time and total energy consumption. The MDE algorithm was enhanced by integrating the combination of list scheduling heuristic and local search.

Given the difficulty in the energy-concerned production scheduling problem, solving most of them by exact algorithms is unrealistic, especially for large-scale instances. Heuristic algorithms are popular for solving these problems [4].

Particle swarm optimization (PSO) is one of the heuristic algorithms, and it has been widely used in many fields. A new bare-bones multiobjective particle swarm optimization algorithm was firstly proposed by Zhang et al. [13] in order to deal with environmental/economic multiobjective optimization problems. There were three innovations in an updating method for particles which do not need to adjust parameters, a dynamic mutation operator to improve search ability, and a global particle leader updating method based on particle diversity. The results of experiments demonstrated that the algorithm can obtain a good approximation of the Pareto front.

Song et al. [14] proposed a variable-size cooperative coevolutionary particle swarm optimization algorithm (VS-CCPSO) for evolutionary feature selection (FS) methods to deal with "curse of dimensionality" on high-dimensional data. The idea of "divide and conquer" was used to cooperative coevolutionary approach, and several strategies tailored according to properties of problems were developed including a space division strategy, an adaptive adjustment mechanism of subswarm size, a particle deletion strategy, and a particle generation strategy. The experimental results illustrated that VS-CCPSO can obtain a good subset of features, which indicated that vS-CCPSO is competitive in dealing with high-dimensional FS problems.

The paper by Zhang et al. [15] presented the first study of multiobjective particle swarm optimization (PSO) for cost-based feature selection in classification in order to obtain nondominated solutions. Some adjustments, including a probability-based encoding method, a hybrid operator, ideas of the crowding distance, the external archive, and the Pareto domination relationship, were embedded in PSO to improve its search capability. Experimental results illustrated that this algorithm can automatically evolve a Pareto front, and it is a competitive feature selection method to solve the cost-based feature selection problem.

A fuzzy multiobjective FS method with particle swarm optimization, called PSOMOFS, was developed in the literature by Hu et al. [16], aiming at the feature selection (FS) problem with fuzzy cost. Specifically, a fuzzy dominance relationship, a fuzzy crowding distance measure, and a tolerance coefficient were used in the algorithm. Compared with other evolutionary methods and classical multiobjective FS methods, experimental results indicated that the proposed algorithm can obtain feature sets with better performances in approximation, diversity, and feature cost.

In our earlier work [17], we studied a biobjective optimization problem. The drawbacks of the biobjective approach can be identified from two aspects. Firstly, it is difficult and sometimes puzzling for practitioners to choose the most suitable solution from the set of nondominated solutions output by the scheduling algorithm. Secondly, the optimization capability of multiobjective evolutionary algorithms is often not robust enough, which means the obtained nondominated solutions can be far from the true optimum.

In this paper, a single-machine scheduling problem considering energy consumption based on real production environment is studied. Numerous industries involving continuous production processes (e.g., continuous casting and hot rolling in the steel industry) are limited in their energy consumption over time by contracts with power suppliers. In particular, the amount of energy expended in each successive period must not exceed a certain limit. Production scheduling under such constraints is much more complex than under normal production conditions. Therefore, we focus on minimizing total weighted tardiness with energy consumption constraints. The total weighted tardiness captures the requirement of meeting the delivery time specified by customers. In other words, we are treating the problem as a single-objective optimization model, with energy-saving goals defined as constraints. This approach is closer to reality because energy considerations are usually expressed as constraints.

In order to solve our problem, a local search strategy was designed to be embedded in the basic particle swarm optimization (PSO) algorithm [18] as the proposed PSO-LS. Besides, a constraint handling process was tailored for this problem, which was used to determine the start processing time and completion time of each job to obtain total weighted tardiness as an objective function value. To make the problem more realistic, the starting time of each job is as low as seconds without violation of constraints. The experimental results show that our enhanced algorithm PSO-LS is statistically superior than that of the basic PSO and GA algorithms especially for large-scale problems.

The remainder of this paper is arranged as follows: in Section 2, we define the problem and analyze its difficulty and then provide a small-scale example to facilitate understanding. Then, a PSO embedded with a local search strategy (PSO-LS) algorithm with design details, such as the decoding method, is proposed to solve this problem in Section 3. In Section 4, many computational experiments are executed using PSO-LS, basic PSO, and GA to evaluate PSO-LS. At last, we outline the conclusion and a prospect for future studies in Section 5.

2. Problem Definition

2.1. Problem Statement. At the beginning, there are n jobs marked as $\{1, 2, 3, \ldots, n\}$ waiting to be processed on one machine. Job i requires a processing time of p_i (h) and consumes A_i (kWh) per hour, and it cannot start to be processed before the former job is finished. The total energy consumption of the jobs processed in the same processing time window shall not exceed Z (kWh). Time window Θ's time interval is $[(\Theta - 1)T, \Theta T]$, where $\Theta \in \{1, 2, \ldots, \Theta_{\max}\}$ is the label of time window and T is the length of processing time window.

Job i also has due date d_i (h) and importance w_i due to customer requirements. We need to decide the starting time st_i of job i so that we can determine completion time $C_i = st_i + p_i$ and the weighted tardiness $wt_i = w_i \times \max(0, C_i - d_i)$. Our target is to minimize the sum of weighted tardiness of all the jobs, i.e., $\sum_{i=1}^{n} wt_i$, which is called total weighted tardiness (TWT).

The following assumptions are considered in this problem:

(i) There are no uncertainties.

(ii) The length of processing time window $T \geq \max(p_i)$ so that the time window is reasonable.

(iii) The electricity limit $Z \geq 0.5 \times \max(A_i \times p_i)$ guarantees the problem is solvable.

The following mathematical model can be built to describe the problem in detail:

$$\min \sum_{i=1}^{n} wt_i, \tag{1}$$

$$\sum_{i=1}^{n} \max\{\min\{\Theta \times T - st_i, st_i + p_i - (\Theta - 1) \times T\}, 0\} \times A_i \leq Z, \quad \Theta = 1, 2, 3, \ldots, 2n, \tag{2}$$

$$\left(st_i + p_i - st_j\right) \times \left(st_j + p_j - st_i\right) \leq 0, i \neq j,\ i = 1, 2, \ldots n,\ j = 1, 2, \ldots n, \tag{3}$$

$$w_i \times \left(st_i + p_i - d_i\right) \leq wt_i,\ i = 1, 2, \ldots n, \tag{4}$$

$$0 \leq st_i \leq (2n - 1)T + p_i/2,\ i = 1, 2, \ldots n, \tag{5}$$

$$wt_i \geq 0,\ i = 1, 2, \ldots, n. \tag{6}$$

Expression (1) is the objective function used to minimize TWT. Expression (2) is for the energy constraint. Expression (3) is used to express the constraint that one job at most is being processed at a time. Expression (4) calculates the weighted tardiness of each job. Expressions (5) and (6) are the ranges of the starting time and weighted tardiness, respectively.

Noticing that Expression (2) is a max-min constraint and Expression (3) is a quadratic constraint, this model is nonlinear.

2.2. Analysis of the Problem. According to Graham et al. [19], our problem can be expressed as $1|Z|$ TWT. If the energy consumption constraint is not considered, the problem can be relaxed to $1\|$ TWT, which is nondeterministic polynomial-time hardness (NP-hard) [20]. This means an NP-hard problem can be reduced to our problem. So, in this paper, the problem $1|Z|$ TWT is at least NP-hard.

There are an infinite number of possible solutions to this problem. Due to energy consumption limits in each time window, some jobs may have to wait for some time after the previous job has been completed. Therefore, during certain time windows, the machine may need to be idle for a period of time. Jobs to be fully processed in such time windows have an infinite number of starting times to choose from because there are an infinite number of points on a line segment.

When jobs are processed as early as possible without violating the constraint conditions, one permutation can only represent one feasible solution. Considering the objective function with the minimum TWT, i.e., $\sum_{i=1}^{n} \max(0, st_i + p_i - d_i)$, the value of the objective function at an early starting time must not be worse than that at a late starting time in the same situation. Therefore, the optimal solution to the scheduling problem can be included in the permutations.

Based on the above analysis, the job processing sequences are used to represent the solutions of the single-machine scheduling problem with energy consumption constraints. When the order of jobs is determined, the starting time of each job is also determined based on the principle of as early as possible. For a problem with n jobs, there are $n!$ possible solutions in the solution space.

2.3. A Small-Scale Example. A simple example is used to facilitate understanding of the solution. Considering 6 jobs at hand needing to be scheduled, the length of time window equals 30 and the energy consumption limit per hour T is equal to 189. Other figures are provided in Table 1.

Table 1: Job data in an instance.

i	p_i	A_i	d_i	w_i
1	8	12	28	3
2	4	7	15	2
3	10	7	15	2
4	4	11	28	6
5	9	12	10	3
6	10	11	10	6

To facilitate the actual production operation, the time value of starting machining operation is accurate to the second. Since the processing time is measured in hours, we need to reserve the starting processing time to two decimal places without violating production constraints.

To find the optimal solution, recursion and backtracking are used to enumerate all the permutations of the job processing order. Then, we have 6! = 720 solutions that are labelled from 1 to 720, as shown in Figure 1. The elbow points are captured during the enumeration process.

The optimal solution whose objective function value is 262.63 appears at the 637th feasible solution, whose processing order is (6, 2, 4, 1, 3, 5) and the starting processing time for each job is (0, 10, 14, 29.42, 37.42, 57.51).

The schedule and energy consumption of the optimal solution are shown in Figures 2(a) and 2(b).

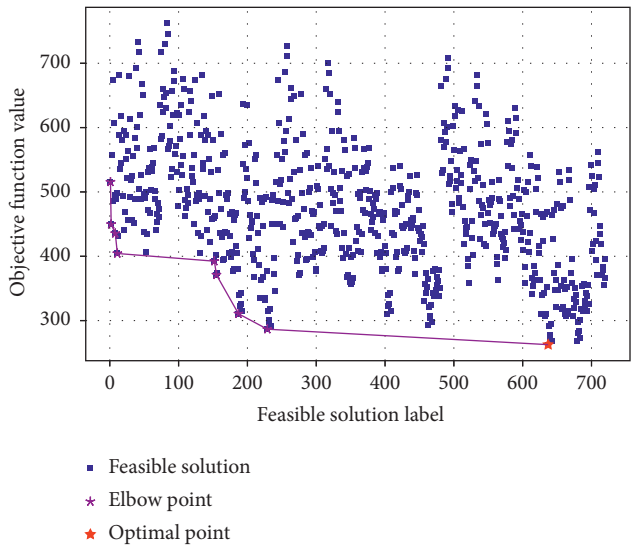

Figure 1: The feasible solutions of an instance.

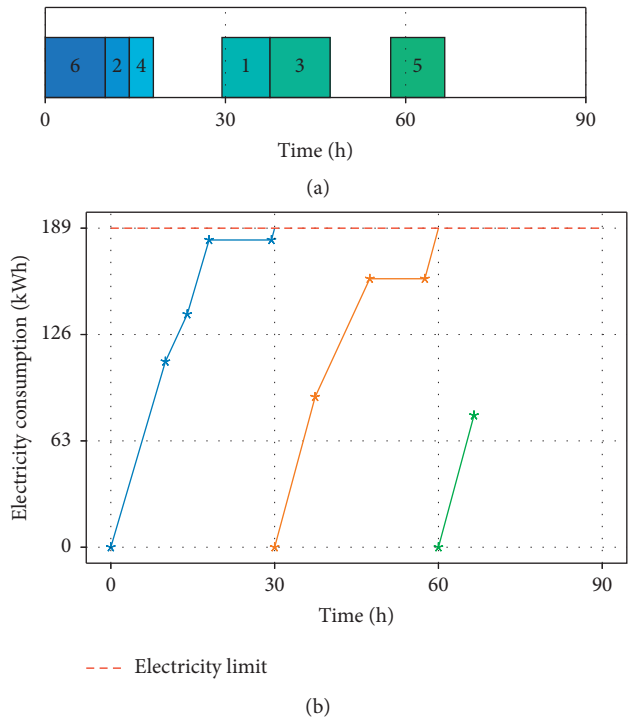

Figure 2: The optimal solution: (a) the schedule; (b) electricity consumption.

3. Heuristic Algorithms

Due to the difficulty in the single-machine scheduling problem with energy consumption constraints, it is impossible to quickly obtain an exact solution with the increase in the number of jobs.

To solve this problem, it is necessary to use a fitting algorithm based on a heuristic algorithm to obtain satisfactory solutions. There are many heuristic algorithms but we choose PSO from them as our framework for three reasons. Firstly, PSO relies on only a few parameters. Secondly, it has good performance in both the convergence speed and the diversity of solutions. Thirdly, its process is simple and effect is good.

In this section, the basic PSO algorithm is first introduced. Then, the encoding and decoding algorithms for this problem are designed in detail. To improve the PSO algorithm, a local search based on the nature of the problem is proposed. Finally, the PSO-LS framework is described.

3.1. Basic PSO Algorithm. There are a bunch of particles in the solution space looking for a target. Every particle from the beginning has a position (pos) and a velocity (vel). The best previous location in its memory is called pbest. The best particle in the swarm is called gbest. During each flight, as the particles reach their target, they need to determine their speed based on the speed of the last flight, the previous pbest, and the previous gbest. Their pbest and gbest are reselected based on new locations. This iteration runs until the end of time. The gbest resulting from the final iteration will be the result of the problem produced by PSO:

$$\begin{aligned} \text{vel}^{ite} = {} & \text{weight} \times \text{vel}^{ite-1} + c_1 \times r_1 \times \left(\text{pbest}^{ite-1} - \text{pos}^{ite-1}\right) \\ & + c_2 \times r_2 \times \left(\text{gbest}^{ite-1} - \text{pos}^{ite-1}\right), \end{aligned} \tag{7}$$

$$\text{pos}^{ite} = \text{pos}^{ite-1} + \text{vel}^{ite}. \tag{8}$$

The PSO algorithm can be described in detail as follows:

(1) Initialize the particle swarm. Specifically, initialize the position and velocity of each particle.

(2) Evaluate all the particles in the swarm according to their own positions.

(3) Initialize pbest and gbest. Assign the *pos* value of each particle to its own pbest, and the best one of all the particles is used to assign gbest as its initial value.

(4) Repeating steps 5 to 8 until the stop criteria are satisfied.

(5) Update particle swarm. The new velocity and new position of each particle can be calculated by Expressions (7) and (8), respectively, where weights c_1 and c_2 are parameters that need to be set in advance and r_1 and r_2 are two different random real numbers with values ranging from 0.0 to 1.0. *ite* is the number of the current iteration.

(6) Adjust the particle swarm. The position of the particle is in the range of $[pos_{\min}, pos_{\text{Max}}]$, where both are parameters of the PSO. Particles beyond this limit are adjusted to meet this constraint.

(7) Evaluate the swarm like in step 2.

(8) Obtain new pbest and gbest.

(9) Obtain the gbest of the last iteration as the result of the problem by PSO.

3.2. Encoding and Decoding. To apply the particle swarm algorithm to our problem, for each particle, the position and velocity have n dimensions according to the number of jobs to be processed. In addition, the position and velocity of each particle are calculated by formulas (9) and (10), respectively, where k is the dimension index:

$$\text{vel}^{ite}[k] = \text{weight} \times \text{vel}^{ite-1}[k] + c_1 \times r_1 \times \left(\text{pbest}^{ite-1}[k] - \text{pos}^{ite-1}[k]\right) + c_2 \times r_2 \times \left(\text{gbest}^{ite-1}[k] - \text{pos}^{ite-1}[k]\right), \tag{9}$$

$$\text{pos}^{ite}[k] = \text{pos}^{ite-1}[k] + \text{vel}^{ite}[k]. \tag{10}$$

Since the basic PSO algorithm was originally designed to solve continuous problems, the smallest position value (SPV) was used to convert the positions into integer sequences as the decoding method, which can be expressed as job processing sequences.

The core of SPV process is a sorting algorithm. It is simple, effective, and easy to implement, so we use it as the decoding method.

For example, given a 5-dimensional position [3.1, 4.3, 2.7, 2.2, 1.9], [1.9, 2.2, 2.7, 3.1, 4.3] can be obtained by placing the values of the different dimensions in ascending order. The values on each dimension correspond to the original dimension indexes that are extracted to produce sequences (5, 4, 3, 1, 2).

The decoding method (SPV) is shown in Algorithm 1.

3.3. Constraint Handling and Solution Obtaining. To ensure these permutations have the probability of containing optimal solutions, jobs are scheduled as early as possible according to processing orders in compliance with the energy consumption constraint.

Specifically, we need to decide where to place the job π in a given processing order Π on the processing schedule in the current processing time window Θ. The jobs $(\pi - 1)$, $(\pi - 2)$, etc., are scheduled before job π.

To make the problem easier to solve, we only need to decide how long the job π costs in the current time window Θ and the next time window $(\Theta + 1)$. Each job only needs to make this decision, then the starting processing time of each job can be determined, and finally, the objective function value of this processing sequence can be used to evaluate the quality of this given processing sequence.

The time the job costs in the current time window Θ and the next time window $\Theta + 1$, expressed as tpc and tpn, respectively, can be calculated by formulas (11) and (12), respectively, where rZ is the rest energy that can be used in the current time window, A_π is the energy consumption per hour to process the job π, and p_π is the processing time of job π:

$$tpc = \min\left\{\frac{rZ}{A_\pi}, rT, p_\pi\right\}, \tag{11}$$

$$tpn = \min\left\{\frac{Z}{A_\pi}, T, p_\pi - tpc\right\}. \tag{12}$$

We enumerate all the meaningful situations based on Expressions (11) and (12) to obtain the rest T and rest Z,

Require: position, n
Ensure: order
(1) $order \longleftarrow \varnothing$
(2) $multimap < double, int > \longleftarrow \varnothing$
(3) **for** each i: $= 1, 2, \ldots, n$ **do**
(4) $\quad pair \longleftarrow makepair(position[i], i)$
(5) $\quad multimap \longleftarrow multimap \cup \{pair\}$
(6) **end for**
(7) **for** each i: $= 1, 2, \ldots, n$ **do**
(8) $\quad order \longleftarrow order \cup \{multimap[i].second\}$
(9) **end for**
(10) **return** order

Algorithm 1: The decoding method (SPV).

which are expressed as rZ' and rT' for the job in the next position, respectively, and to determine the completion time of the current job.

All the cases are shown in Figure 3.

(1) $tpc = p_\pi$ and $tpn = 0$. This means the job π will be scheduled to be processed entirely in the current time window. The completion time of the current job can be calculated by $\Theta \times T - rT + tpc$.

(a) $tpc = p_\pi < \min\{rZ/A_\pi, rT\}$. $rZ' = rZ - p_\pi \times A_\pi$ and $rT' = rT - p_\pi$ are for the next job. This situation is shown in Figure 3(a).

(b) $tpc = p_\pi = rZ/A_\pi < rT$. The energy consumption runs out after processing job π. So, the next job must be scheduled in the next time window, and $rZ' = Z$ and $rT' = T$ are included in the next job. Figure 3(b) displays this case.

(c) $tpc = p_\pi = rT < rZ/A_\pi$. So, the job $(\pi + 1)$ has to be processed in the next time window $(\Theta + 1)$, and $rZ' = Z$ and $rT' = T$ are for the next job. Figure 3(c) demonstrates this situation.

(d) $tpc = p_\pi = rZ/A_\pi = rT$. The next job is scheduled to be processed in the next time window with $rZ' = Z$ and $rT' = T$. This case is described in Figure 3(d).

(2) $tpc < p_\pi$ and $tpn = p_\pi - tpc$. The job π cannot be scheduled entirely in the time window Θ. The completion time of the current job can be calculated by $\Theta \times T + tpn$. This current job takes tpn hours to be processed in the next time window. We can obtain $rT' = Z - tpn$ and $rZ' = Z - tpn \times A_\pi$ for the next job.

(a) $tpc = rZ/A_\pi < rT$. After producing the job π, the electricity in time window Θ runs out. There is a gap between the starting time of job π and the completion time of job $(\pi - 1)$. This situation is shown in Figure 3(e).

(b) $tpc = rT < rZ/A_\pi$. The current job can begin processing when the production of job $(\pi - 1)$ is complete. Figure 3(f) displays this case.

(c) $tpc = rZ/A_\pi = rT$. This is similar to 2(b). The energy in the current time window runs out. Figure 3(g) shows this situation in detail.

(3) $tpc < p_\pi$ and $tpn = Z/A_\pi < p_\pi - tpc$. The job π has to start being processed in the time window $(\Theta + 1)$ with $rZ = Z$ and $rT = T$. Then, we need to reschedule the job π in the next iteration, which may be consistent with situation 2(a) discussed above. An example of this situation is shown in Figure 3(h).

The procedure above repeats from $\pi = 1$ until $\pi = n$ for the number of jobs. We can determine the completion time of each job so that TWT can be calculated as the fitness of a particle.

Based on the above description, the constraint handling process is described in Algorithm 2. To determine the completion time precision down to seconds, lines 7 and 17 to 22 were added to handle the tpn with more than two decimals. In line 7, we leave tpn with 2 decimal places to obtain tpn_{round}. For example, if $tpn = 3.91 + 1 \times 10^{-100}$, then we have $tpn_{round} = 3.92$. In lines 17 through 22, rZ and rT are calculated based on tpn_{round} instead of tpn, and the schedule is adjusted as well.

3.4. Local Search. To improve the searching ability of the basic PSO, an insertion operator is used in the gbest of the particle swarm in each iteration. This insertion operator can considerably increase the diversity of solutions, thus increasing the probability of significantly improving gbest. In addition, it is simple to implement and has low time and memory overhead.

The insertion operator for the processing order of gbest is described in Figure 4. It aims to insert the job π_1 from the time window Θ_1 immediately after the job π_2 in time window Θ_2.

This insertion operator can be easily applied to the job processing order.

Firstly, we need to select two different time windows. Time windows Θ_1 and Θ_2 are random integers selected from the range of $[1, \Theta_{\max} - 1]$ and $[\Theta_1 + 1, \Theta_{\max}]$. Then, two jobs need to be chosen from those two time windows. Jobs π_1 and π_2 are chosen randomly from the jobs processed in the selected time windows. At this moment, job π_1 must be processed before job π_2 because time window Θ_1 is in front of time window Θ_2. Thirdly, for jobs from job π_2 to the job immediately after job π_1, their ordinal numbers need to be subtracted by 1 in this processing sequence. Finally, we change the ordinal number of job π_1 in the processing order. The only task to perform is to place job π_1 after job π_2.

Because the position of gbest is used to update particles in the next iteration, we need to change the position of gbest to maintain the consistency of the position and processing order of gbest. A small example is provided in Table 2 to illustrate the process of adjusting the position of gbest according to the insertion operator.

Assume that the original gbest has a 5-dimensional position $pos_0 = (3.1, 4.3, 2.7, 2.2, 1.9)$ and the original order $order_0 = (5, 4, 3, 1, 2)$. If $\pi_1 = 4$ and $\pi_2 = 2$, we can determine the changed order $target = (5, 3, 1, 2, 4)$ according to insertion operator for order, as shown in Figure 4. For this instance, step 1 shown in Table 2 involves exchanging the job π_1 to the job to the right. So we need to swap jobs 4 and 3,

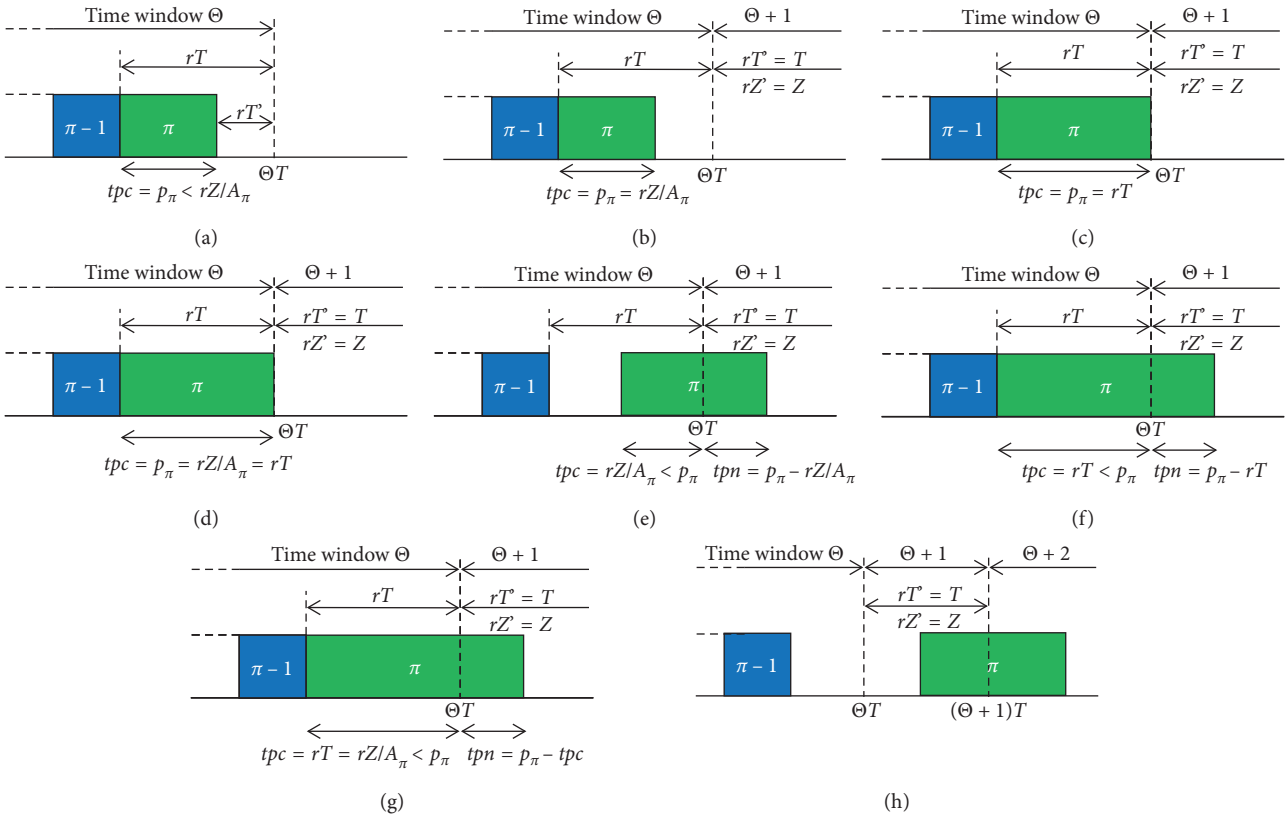

Figure 3: Energy constraint handling cases: (a) case 1; (b) case 2; (c) case 3; (d) case 4; (e) case 5; (f) case 6; (g) case 7; (h) case 8.

and we can obtain pos_1 by swapping the value of dimensions 3 and 4 of pos_0. At this moment, we have $pos_1 = [3.1, 4.3, 2.2, 2.7, 1.9]$ and $ord_1 = (5, 3, 4, 1, 2)$. In step 2, we also swap the job π_1 to the right job. This time, we need to exchange job 4 with job 1 according to ord_1 so that ord_2 can be obtained. To obtain pos_2, we can swap the value of dimensions 4 and 1 of pos_1, that is, exchanging 3.1 with 2.7. Step 3 is also based on its previous step; this time, we need to exchange job π_1 and job π_2 because job π_2 is to the right of job π_1 in ord_2. Similar with step 1 and step 2, pos_3 is obtained by swapping the values of dimensions 4 and 2 of pos_2. Finally, we get $pos_3 = [2.7, 3.1, 2.2, 4.3, 1.9]$. To verify its correctness, we can decode pos_3 using SPV so that we can determine its order (5, 3, 1, 2, 4), which is equal to the target order.

Based on all descriptions above, for each iteration, we applied the insertion operator shown in Algorithm 3 to the original gbest five times to obtain five different new gbests. The original gbest was replaced by the best gbest among these five gbests with the smallest objective function value if it was better than the original gbest. Based on the above, the local search process is described in Algorithm 4.

3.5. PSO-LS Framework. The PSO-LS framework structure is shown in Algorithm 5. As described in the previous sections, the state of the swarm is initialized by the first three lines. The main loop of the PSO algorithm consists of lines from 4 through 9. Based on the fact that the behavior of each particle depends heavily on the global leader gbest, the eighth line which aims to improve the quality of the global optimal particle is added before the end of each iteration by calling Algorithm 4.

4. Computational Experiments

To evaluate PSO-LS proposed in the previous chapter, we performed many experiments for comparison with the genetic algorithm (GA) [21].

To be specific, we used the main algorithm framework in a previous study [22] that uses GA to solve the scheduling problem. The initial population was generated randomly. We used C1 from [22] for chromosomal representation and crossover operator. For selection mechanism, we used a hybrid of roulette wheel selection [23] with an elitist strategy [23]. A shift mutation was applied to the GA according to the literature [22]. The probability of mutation p_m was dynamic, and at the beginning, $p_m := p_m^0$. For each iteration, p_m decreases by a rate of θ. In the population, when the value of the minimum fitness over the mean of fitness is larger than a constant real number D, i.e., $v_{\min}/v_{mean} > D$, p_m takes the initial value p_m^0.

Before the comparison, experiments for parameter selection were needed because the performance of heuristic algorithms depends strongly on their parameters. We performed the same parameter selection experiment on PSO and GA.

4.1. Problem Data Setting. Before we started the experiment, the data of problems are given as follows:

Require: position and problemData
Ensure: schedule, starting time st, completion time C, and TWT
(1) $\{p, A, w, d, T, Z, n\} \longleftarrow problem\ Data$
(2) $order \longleftarrow SPV\ (position, n)$
(3) $schedule \longleftarrow \varnothing$
(4) $\Theta \longleftarrow 1, rT \longleftarrow T, rZ \longleftarrow Z, schedulep \longleftarrow \varnothing, scheduleporder \longleftarrow \{0, 0\}, TWT \longleftarrow 0$
(5) **if** each $i \coloneqq 1, \ldots, n$ **do**
(6) reschedule:
(7) $k \longleftarrow order[i]$
(8) $tpc \longleftarrow \min\{rZ/A[k], rT, p[k]\}[k]; tpn \longleftarrow \min\{Z/A[k], T, p[k] - tpc\}$
(9) $tpn_{round} \longleftarrow uproun\,d\,(tpn, 0.01)$
(10) **if** $(|p[k] - tpc - tpn| = 0)$**and** $(Z = tpn_{round} \times A[k])$ **then**
(11) $rZ \longleftarrow rZ - A[k] \times tpc; rT \longleftarrow rT - tpc$
(12) $scheduleporder[0] \longleftarrow i; scheduleporder[1] \longleftarrow tpc/p[k]$
(13) $schedulep \cup \{scheduleporder\}$
(14) $C[i] \longleftarrow \Theta \times T - rT$
(15) **if** $(rZ = 0)$**or** $(rT = 0)$ **then**
(16) $schedule \cup \{schedulep\}; schedulep \longleftarrow \varnothing; \Theta \longleftarrow \Theta + 1$
(17) $rZ \longleftarrow Z - tpn \times A[k]; rT \longleftarrow T - tpn$
(18) **if** $tpn > 0$ **then**
(19) **if** $tpn \neq tpn_{round}$ **then**
(20) $rZ \longleftarrow Z - tpn_{round} \times A[k]; rT \longleftarrow T - tpn_{round}$
(21) $scheduleporder[0] \longleftarrow i; scheduleporder[1] \longleftarrow tpn_{round}/p[k]$
(22) $schedulep \cup \{scheduleporder\}$
(23) $C[i] \longleftarrow \Theta \times T - rT$
(24) $(*\ (schedule[\Theta - 2].end - 1)).at\,(1) \longleftarrow 1.0 - scheduleporder[1]$
(25) **else**
(26) $scheduleporder[0] \longleftarrow i; scheduleporder[1] \longleftarrow tpn/p[k]; schedulep \cup \{scheduleporder\}$
(27) $C[i] \longleftarrow \Theta \times T - rT$
(28) **end if**
(29) **end if**
(30) **end if**
(31) $TWT \longleftarrow TWT + w[k] \times \max(0.0, C[i] - d[k]);$
(32) $st[i] \longleftarrow C[i] - p[k]$
(33) **else**
(34) $schedulep \cup \{scheduleporder\}; schedulep \longleftarrow \varnothing; \Theta \longleftarrow \Theta + 1;$
(35) $rZ \longleftarrow Z; rT \longleftarrow T$
(36) go to reschedule
(37) **end if**
(38) **end for**
(39) **if** $\Theta > schedule.size$ **then**
(40) $schedulep \cup \{scheduleporder\}$
(41) **end if**
(42) **return** schedule, st, C and TWT

Algorithm 2: Constraint handling process to obtain a solution.

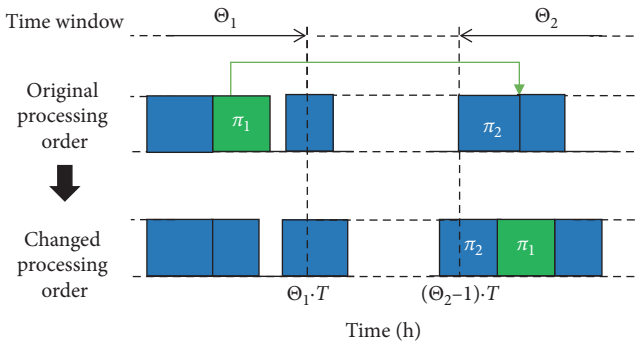

Figure 4: An example for insertion operator for job processing order.

(i) The processing time of job i, p_i, is uniformly generated in the range $unif\,(1, 10)$.

(ii) The electricity consumption of job i, A_i, is distributed from the uniform $unif\,(3, 15)$.

(iii) The due time of job i, d_i, is distributed from the uniform $unif\,(5, 5 \times n)$.

(iv) The importance of job i, w_i, is distributed from the uniform $unif\,(1, n)$.

All the problems handled in the experiments were generated randomly based on the distributions.

Table 2: Insertion operation example for position of gbest.

Original	$order_0$	(5,	$\pi_1 = 4$,	3,	1,	$\pi_2 = 2$)
Step 1	Target	(5,	3,	1,	$\pi_2 = 2$,	$\pi_1 = 4$)
	Dimension	1	2	3	4	5
	pos_0	[3.1,	4.3,	2.7,	2.2,	1.9]
	Swap	$\pi_1 = 4$	3			
	Dimension	1	2	3	4	5
	pos_1	[3.1,	4.3,	2.2,	2.7,	1.9]
	$order_1$	(5,	3,	4,	1,	2)
Step 2	Swap	$\pi_1 = 4$	1			
	Dimension	1	2	3	4	5
	pos_2	[2.7,	4.3,	2.2,	3.1,	1.9]
	$order_2$	(5,	3,	1,	4,	2)
Step 3	Swap	$\pi_1 = 4$	$\pi_2 = 2$			
	Dimension	1	2	3	4	5
	pos_3	[2.7,	3.1,	2.2,	4.3,	1.9]
	$order_3^* = target$	(5,	3,	1,	2,	4)

Require: gbest, problemData
Ensure: $gbest_{new}$
(1) time window $\Theta_1 \longleftarrow random(1, gbest.schedule.size - 1)$
(2) time window $\Theta_2 \longleftarrow random(\Theta_1 + 1, gbest.schedule.size)$
(3) $temp1 \longleftarrow random(1, gbest.schedule[\Theta_1].size)$
(4) $temp2 \longleftarrow random(1, tgnbqehs_i x.7sCc; hedule[\Theta_2].size)$
(5) $\pi_1 \longleftarrow gbest.schedule[\Theta_1][temp1]$
(6) $\pi_2 \longleftarrow gbest.schedule[\Theta_2][temp2]$
(7) initialize $gbest_{new} \longleftarrow gbest$
(8) **for** each $ite: = \pi_2, \pi_2 - 1, \ldots, \pi_1 + 1$ **do**
(9) $\quad swap(gbest_{new}.pos[gbest.order[ite]], gbest_{new}.pos[gbest.order[ite - 1]])$
(10) **end for**
(11) **for** each $ite: = \pi_1, \pi_1 + 1, \ldots, \pi_2 - 1$ **do**
(12) $\quad swap(gbest_{new}.order[ite], gbest_{new}.order[ite + 1])$
(13) **end for**
(14) $evaluate(gbest_{new}, problemData)$
(15) **return** $gbest_{new}$

Algorithm 3: Insertion operator for gbest.

Require: gbest, problemData
Ensure: gbest
(1) $arrgbest_{new} \longleftarrow \varnothing$
(2) **for** each $ite: = 1, 2, \ldots, maxite$ **do**
(3) $\quad$ apply insertion operator to gbest to get a $gbest_{new}$
(4) $\quad arrgbest_{new} \longleftarrow (arrgbest_{new} \cup \{gbest_{new}\})$
(5) **end for**
(6) obtain the best $gbest_{new}$ from $arrgbest_{new}$ as $bestgbest_{new}$
(7) **if** $bestgbest_{new}$ is better than gbest **then**
(8) $\quad$ gbest $\longleftarrow bestgbest_{new}$
(9) **end if**
(10) **return** gbest

Algorithm 4: Local search process.

```
Require: parameter, problemData
Ensure: solution
(1) initializeSwarm (swarm, parameter, n)
(2) evaluateSwarm (swarm, problemData)
(3) initialize pbest and gbest
(4) while parameter.stopCriteria is not satisfied do
(5)     updateSwarm (swarm, parameter)
(6)     adjustSwarm (swarm, parameter)
(7)     evaluateSwarm (swarm, problemData)
(8)     localSearch (swarm.gbest, problemData)
(9) end while
(10) solution = swarm.gbest.solution
(11) return solution
```

Algorithm 5: PSO-LS framework.

4.2. Algorithm Parameters Setting. We used the Taguchi method [24] for parameter selection.

Taguchi method is also called the orthogonal experimental design which is used to determine the value of each factor especially when the number of factors and factor levels are greater than 3. Based on orthogonal tables, the number of experiments can be greatly decreased. The brief steps of the Taguchi method are as follows: firstly, determine responses, factors, and levels; secondly, select appropriate orthogonal tables; thirdly, execute the experiments according to orthogonal tables and fill the tables with experimental results; then, analyze the results and determine the factor level combination; and at last, verify the effectiveness of the selected levels of factors.

We took the size of swarm, weight, c_1, c_2, and $V_{\max}$ as factors for the PSO algorithm. We also took the size of population, the probability of crossover p_c, the probability of mutation p_m, and the other two parameters related to mutations θ and D as the factors for the GA algorithm. Each factor of both algorithms took 4 levels, as shown in Tables 3 and 4.

We conducted 16 experiments according to the orthogonal tables of parameters of each algorithm, as shown in Tables 5 and 6.

Considering that the problems on different scales may take different parameter levels to obtain effective search capabilities, we set the means of TWT of a random small-scale problem and a random large-scale problem as responses. A problem with 30 jobs was used as the small-scale problem and 70 jobs as the large-scale problem. To eliminate as much random interference as possible, the average value of the objective function value resulting from 20 executions of each algorithm was taken as the response.

After the tables were designed and filled, we needed to select parameters for the PSO and GA algorithms by analyzing Tables 5 and 6.

Then, we needed to decide which level would be selected for each factor. We selected the factor of the size of the particle swarm for the small-scale problem as an example, as shown in the line on the top left of Figure 5. Firstly, let us consider the situation where size equals 50. According to Table 5, the average of TWT for a size of 50 for the small-scale problem

Table 3: Levels of parameters for PSO.

Parameter				
Size	50	100	150	200
Weight	0.6	0.7	0.8	0.9
c_1	0.01	0.05	0.1	0.15
c_2	0.3	0.6	0.8	0.95
$V_{\max}$	1	2	3	4

Table 4: Levels of parameters for GA.

Parameter				
Size	50	100	150	200
p_c	0.3	0.5	0.7	0.9
p_m	0.3	0.5	0.7	0.9
θ	0.69	0.79	0.89	0.99
D	0.65	0.75	0.85	0.95

Table 5: Orthogonal table for parameters for PSO.

No.	Size	Weight	c_1	c_2	$V_{\max}$	Small-scale	Large-scale
1	50	0.6	0.01	0.3	1	6150.0400	125,285.8610
2	50	0.7	0.05	0.6	2	4532.9490	91,016.6945
3	50	0.8	0.1	0.8	3	3702.4335	73,910.1090
4	50	0.9	0.15	0.95	4	3018.0550	38,671.7300
5	100	0.6	0.05	0.8	4	3867.6865	89,791.1670
6	100	0.7	0.01	0.95	3	3531.7860	71,033.7450
7	100	0.8	0.15	0.3	2	4092.9150	84,425.5930
8	100	0.9	0.1	0.6	1	3051.1515	52,046.6500
9	150	0.6	0.1	0.95	2	3695.0365	75,719.6005
10	150	0.7	0.15	0.8	1	3517.0280	67,949.1885
11	150	0.8	0.01	0.6	4	3122.3110	60,491.8475
12	150	0.9	0.05	0.3	3	3132.7980	57,114.9990
13	200	0.6	0.15	0.6	3	4070.9355	81,141.6650
14	200	0.7	0.1	0.3	4	3676.8785	83,755.1865
15	200	0.8	0.05	0.95	1	3137.8420	49,046.6995
16	200	0.9	0.01	0.8	2	2761.6215	34,490.5440

Table 6: Orthogonal table for parameters for GA.

No.	Size	p_c	p_m	θ	D	Small-scale	Large-scale
1	50	0.3	0.3	0.69	0.65	4220.1760	117,609.0645
2	50	0.5	0.5	0.79	0.75	3879.6515	108,405.6915
3	50	0.7	0.7	0.89	0.85	3905.2175	102,019.3425
4	50	0.9	0.9	0.99	0.95	3706.7560	93,161.1750
5	100	0.3	0.5	0.89	0.95	3317.0760	92,678.0220
6	100	0.5	0.3	0.99	0.85	3505.2000	108,444.7035
7	100	0.7	0.9	0.69	0.75	3510.1470	89,561.1925
8	100	0.9	0.7	0.79	0.65	3656.2845	106,062.7730
9	150	0.3	0.7	0.99	0.75	3064.3005	79,132.2530
10	150	0.5	0.9	0.89	0.65	3145.2140	77,560.3880
11	150	0.7	0.3	0.79	0.95	3470.5435	111,862.9095
12	150	0.9	0.5	0.69	0.85	3331.9765	101,187.5760
13	200	0.3	0.9	0.79	0.85	2951.0995	72,115.2650
14	200	0.5	0.7	0.69	0.95	3014.6595	80,158.9815
15	200	0.7	0.5	0.99	0.65	3223.1805	90,552.2890
16	200	0.9	0.3	0.89	0.75	3291.8940	113,233.6705

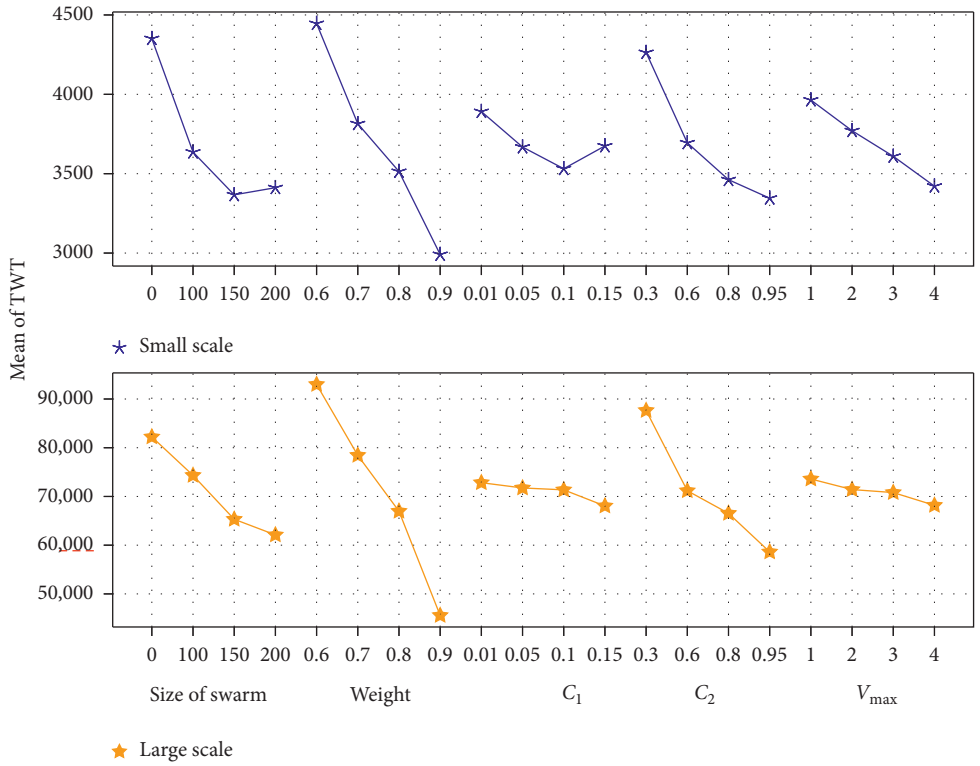

Figure 5: Influence of the key parameters for PSO.

can calculated by the mean of the small-scale response of the first, second, third, and fourth experiments. That is, (6150.0400 + 4532.9490 +3702.4335 + 3018.0550)/4 = 4350.869375. Secondly, similarly, we can obtain 3635.88475 for level 100, 3366.793375 for level 150, and 3411.819375 for level 200. Thirdly, we determine the best parameter of 150 for the small-scale problem for PSO because level 150 has the smallest mean TWT of 3366.793375.

The average responses at different levels for each factor need to be calculated, and then, the level of each factor with the smallest mean TWT is chosen. We performed this procedure on other factors for PSO and all the factors for GA for both the small- and large-scale problems. The main effect plots for PSO and GA are shown in Figures 5 and 6, respectively. The lowest points of these lines are taken as the value for each parameter for the algorithms.

As such, we determined all the parameter values for each algorithm. For the PSO algorithm, the values of the size of swarm, weight, c_1, c_2, m, and $V_{\max}$ are 150, 0.9, 0.1, 0.95, and 4 for the small-scale problem and 200, 0.9, 0.15, 0.95, and 4 for the large-scale problem, respectively. For the GA algorithm, similarly, the size of population, the probability of crossover p_c, the probability of mutation p_m, and the other two parameters relating to mutations θ and D are 200, 0.5, 0.9, 0.99, and 0.95 for the small-scale problem and 200, 0.3, 0.9, 0.99, and 0.95 for the large-scale problem, respectively.In order to verify the effectiveness of selected levels of factors by the Taguchi method, the responses of the selected parameter values need to be compared with that of all the level combinations displayed in Figures 5 and 6. Problems of size 30 and size 70 are generated randomly, and their results are displayed in Tables 7 and 8. The values of line 17 in both figures are the levels for factors selected for small-scale problems and line 18 for large-scale problems. It can be seen from the two tables that the fitness obtained by the levels of factors selected for the small-scale problems is the smallest among the other values in the same column. The TWT obtained by the levels of factors chosen for the large-scale problems is also the smallest among the other data in the same column. Therefore, the parameter selections for PSO and GA by the Taguchi method are effective.

4.3. Performance Comparison. We compared the results of PSO-LS, PSO, and GA for problems on six different scales: 10, 30, 50, 70, 100, and 300 jobs. Problems with fewer than 70 jobs were considered small-scale; otherwise, they were considered large-scale.

When solving these two sets of problems on different scales, the algorithms choose the values of parameters of the corresponding scale, which was obtained in the previous section.

For each scale, five different instances were randomly generated. When solving each problem, each algorithm was independently executed 20 times, and the average value of these 20 results was recorded to reduce the differences in the random initial solutions.

Each algorithm executes the same duration for each run. All algorithms were executed in 1, 10, 30, 50, 100, and 100 s, which correspond to solving problems with 10, 30, 50, 70, 100, and 300 jobs, respectively. For small-scale problems, the algorithms almost converge within the setting time, so we compare the results of each algorithm when the algorithms

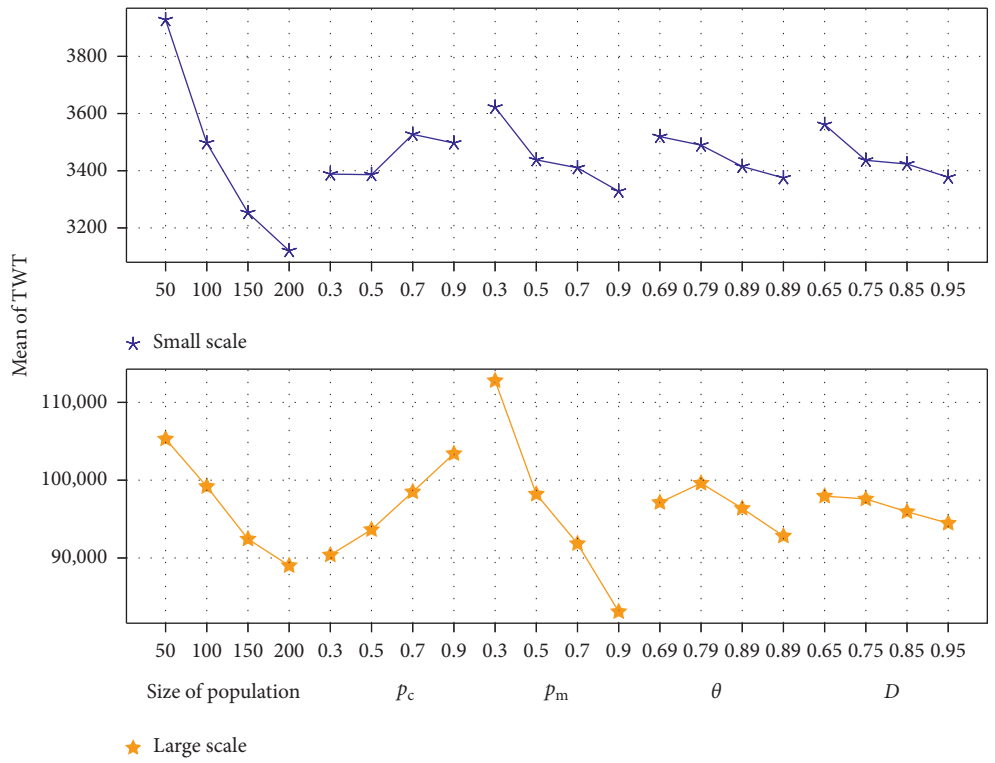

Figure 6: Influence of the key parameters for GA.

Table 7: The effectiveness of parameter selection for PSO.

No.	Size	Weight	c_1	c_2	V_{max}	Small-scale	Large-scale
1	50	0.6	0.01	0.3	1	7802.857	781944.0955
2	50	0.7	0.05	0.6	2	5892.0415	692195.5385
3	50	0.8	0.1	0.8	3	5343.4035	642169.6365
4	50	0.9	0.15	0.95	4	4433.615	504850.2295
5	100	0.6	0.05	0.8	4	5659.695	658437.448
6	100	0.7	0.01	0.95	3	5036.7085	603161.166
7	100	0.8	0.15	0.3	2	5346.9595	631647.512
8	100	0.9	0.1	0.6	1	4541.963	546953.6015
9	150	0.6	0.1	0.95	2	5399.3495	625364.341
10	150	0.7	0.15	0.8	1	4936.6025	620230.085
11	150	0.8	0.01	0.6	4	4620.1865	570452.559
12	150	0.9	0.05	0.3	3	4676.6135	554409.8515
13	200	0.6	0.15	0.6	3	5393.4215	624672.711
14	200	0.7	0.1	0.3	4	5250.421	635595.3965
15	200	0.8	0.05	0.95	1	4489.3905	538767.257
16	200	0.9	0.01	0.8	2	4074.7515	487169.7495
17	200	0.9	0.1	0.95	4	**4046.7475**	483691.6605
18	200	0.9	0.15	0.95	4	4124.0595	**479002.7685**

Table 8: The effectiveness of parameter selection for GA.

No.	Size	p_c	p_m	θ	D	Small-scale	Large-scale
1	50	0.3	0.3	0.69	0.65	5877.824	814077.7725
2	50	0.5	0.5	0.79	0.75	5532.903	781673.413
3	50	0.7	0.7	0.89	0.85	4947.04	750002.8675
4	50	0.9	0.9	0.99	0.95	5080.9145	768124.5555
5	100	0.3	0.5	0.89	0.95	4915.209	738638.408
6	100	0.5	0.3	0.99	0.85	5123.6145	791738.016
7	100	0.7	0.9	0.69	0.75	4637.9835	751141.2055
8	100	0.9	0.7	0.79	0.65	4943.807	778180.741
9	150	0.3	0.7	0.99	0.75	4400.202	720077.3955
10	150	0.5	0.9	0.89	0.65	4431.645	691519.7225
11	150	0.7	0.3	0.79	0.95	4864.1745	786077.9255
12	150	0.9	0.5	0.69	0.85	4664.906	792386.35
13	200	0.3	0.9	0.79	0.85	4357.5805	684881.422
14	200	0.5	0.7	0.69	0.95	4431.7005	722812.895
15	200	0.7	0.5	0.99	0.65	4511.614	747950.5375
16	200	0.9	0.3	0.89	0.75	5038.0215	769743.4985
17	200	0.5	0.9	0.99	0.95	**4296.492**	686883.08
18	200	0.3	0.9	0.99	0.95	4335.9145	**681536.872**

converge. For large-scale problems, because of the difficulty in the problems, algorithms cannot converge in a short time, so we let the algorithm run enough time to see, which makes the results closer to the optimal solutions. Since time is vital to production environment, it is important to get a better satisfactory solution in a shorter time. Therefore, the computational time rather than the number of iterations is controlled for the computational experiments.

For the 20 runs, the comparison results of the average TWT obtained by PSO-LS, PSO, and GA are shown in Table 9.

Paired t-tests were to be conducted with a significance level $\alpha = 0.05$ to analyze the performance of PSO-LS compared with both basic PSO and GA.

For the same scale of problems, the differences in the objective function values obtained by PSO-LS and PSO shown in Table 9 can be obtained by $D_i = \text{PSO}_i - \text{PSO-LS}_i$,

Table 9: Comparison of the mean TWT.

Scale-instance	PSO-LS	PSO	GA
10-1	**666.8100**	669.9850	**666.8100**
10-2	**186.3200**	186.4200	**186.3200**
10-3	**273.2200**	274.3000	273.6150
10-4	232.3720	**229.3980**	236.8330
10-5	**1,114.7300**	1,115.6540	1,117.5780
30-1	4,956.7030	4,938.7220	**4,724.9780**
30-2	**18,087.6580**	18,563.0790	19,007.3440
30-3	**6,810.1500**	6,919.5320	7,286.0000
30-4	**9,355.2740**	9,612.9000	9,701.4130
30-5	**10,393.8770**	10,603.2970	11,060.3790
50-1	**47,373.0790**	49,662.6280	50,611.0100
50-2	**68,891.7570**	70,747.0140	79,298.5780
50-3	**61,826.2370**	64,119.7600	67,243.0500
50-4	**68,777.6710**	70,193.7050	77,970.5510
50-5	**29,073.9680**	30,776.0000	33,192.1830
70-1	**135,271.7270**	143,512.1820	158,553.0030
70-2	**164,900.6270**	171,377.4080	183,845.9250
70-3	**128,195.6660**	135,871.2800	150,625.1680
70-4	**110,219.4880**	118,058.2520	138,250.7180
70-5	**161,958.1090**	166,567.7950	192,447.0210
100-1	**287,348.7000**	308,337.3160	452,220.2050
100-2	**316,831.4730**	339,679.4730	491,640.0870
100-3	**363,915.9090**	392,756.4790	550,393.6550
100-4	**512,024.2470**	541,724.8160	705,187.0550
100-5	**352,616.9280**	384,112.2770	588,789.3920
300-1	**13,726,081.6490**	14,313,663.8880	35,374,531.8320
300-2	**12,896,793.1160**	13,395,467.6770	34,768,966.6680
300-3	**10,358,806.2330**	10,809,516.3790	30,715,756.5610
300-4	**13,422,518.7670**	14,022,110.8640	35,002,552.1430
300-5	**10,851,867.1710**	11,269,214.0880	29,338,328.5100

where $i = 1, \ldots, 5$, which are normally distributed; thus, $D \sim N(u_D, s_D^2)$. We wish to test the hypothesis:

H_0: $u_D \leq 0$,

H_1: $u_D > 0$.

The number of samples, the sample mean, and the sample standard deviation are n, $\overline{d}$, and s_d accordingly. If $t - statistic = \overline{d}/(s_D/\sqrt{n}) > t_{\alpha=0.05}(n-1)$, H_0 is rejected and $u_d > 0$. That is, PSO has a greater fitness than PSO-LS, so the performance of PSO-LS is better than PSO. Otherwise, PSO-LS is considered to have no obvious advantages.

We used the above method to test the performance of PSO-LS compared to PSO at all scales. Firstly, subtract the value of the PSO-LS column from that of the PSO column in Table 9 and obtain the data of the second column in Table 10. Values greater than 0 indicate that PSO-LS is better. Then, the mean and standard deviation of differences on different problem sales are calculated including 10, 30, 50, 70, 100, and 300, and the results are recorded in the third and fourth columns in Table 10, respectively. T-statistics on different scales was finally obtained according to the mean differences and standard deviations which were displayed in the fifth column. Similarly, columns 2 through 5 are shown in Table 11 to demonstrate the differences between PSO-LS and GA on all scales. When the computed t-statistics was greater than $t_{\alpha=0.05}(n-t1) = 2.1318$, PSO-LS was considered to have better performance than PSO for problems of the corresponding numbers of jobs. In the fifth column of the two tables, the data greater than 2.1318 have been highlighted to show that PSO-LS is better for problems with corresponding number of jobs.

In addition, to analyze PSO-LS performance in small- and large-scale problems, the sample size is $n = 15$, and when $t - \text{statistic} = \overline{d}/(s_d/\sqrt{n}) > t_{\alpha=0.05}(n-1) = 1.7613$, PSO-LS is considered statistically better in large- or small-scale problems significantly. The figures were placed in columns 6 through 8 of Tables 10 and 11. Column 8 was the t-statistics for small-scale problems and large-scale problems, and the data greater than $t_{\alpha=0.05}(n-1) = 1.7613$ were shown in bold.

In order to analyze the differences between PSO-LS and PSO and GA in general, when $t - statistic = \overline{d}/(s_d/\sqrt{n}) > t_{\alpha=0.05}(n-1) = 1.6991$, PSO-LS was considered statistically better. The mean of differences, the standard deviations, and t-statistics are shown in the last three columns of Tables 10 and 11. The numbers greater than 1.6991 were bold.

In order to analyze the effectiveness of PSO-LS against basic PSO, the differences, the averages of differences, and t-statistics are shown in Table 10. The gap is the difference of TWT obtained by PSO and by PSO-LS, $\overline{d}_1$ is the mean of the differences calculated from 5 instances with the same job size, s_{D1} is the standard deviation related to $\overline{d}_1$, and then, the t-statistics of each scales, t_1, can be obtained by $\overline{d}_1/(s_{D1}/\sqrt{5})$. We can also calculate the t-statistics by the small-scale and the large-scale, t_2, by $\overline{d}_2/(s_{D2}/\sqrt{15})$, where $\overline{d}_2$ is the mean of the gaps from the small-scale category and large-scale, while s_{D2} is their related standard deviation. Similarly, we can obtain $t = 1.26$ by $\overline{d}/(s_D/\sqrt{30})$, where $\overline{d} = 684,186.52$ and $s_D = 2,984,291.44$ for the overall data obtained from PSO and PSO-LS.

Similar to Table 10, Table 11 is also given to analyze the performances of PSO-LS compared with GA.

The observations from the displayed results are shown as follows.

As for the comparison between PSO and PSO-LS, PSO-LS is statistically better than the basic PSO significantly according to Table 10 because the differences between the PSO and PSO-LS were almost greater than zero (except for 10-4 and 30-1 in Table 9) and the t-statistics for all the differences was 2.58 which was greater than $t_{\alpha=0.05}(29)$ (1.6991). As for performance of solving problems with the same number of jobs, t-statistics in the t_1 column was greater than $t_{\alpha=0.05}(4)$ (2.1318), only except for the problems with 10 jobs, which demonstrated the superior performance of PSO-LS than that of PSO except for the 10-scale problems. T-statistics in t_2 is both larger than $t_{\alpha=0.05}(14)$ (1.7613). This can be attributed to the effectiveness of the local search strategy, by which a better gbest, as the global leader which the whole particle swarm depends on, may be found in each iteration with just a little more time cost because the insertion operator is simple, efficient, and time-saving. Besides, there was a increasing tendency of t-statistics with the increase in the number of jobs, starting with 0.46 and reaching the top at 14.09 by the scale 300, which meant the local search strategy improved the search capability as the complex of problems increases, although on small scales such as scale 10, it did not have

Table 10: Statistical analysis between PSO and PSO-LS.

Scale	PSO-PSO-LS	$\overline{d}_1$	s_{D1}	t_1	$\overline{d}_2$	s_{D2}	t_2	$\overline{d}$	s_D	t
10	3.18 0.10 1.08 −2.97 0.92	0.46	2.23	0.46	706.17	914.75	**2.99**	91,107.10	193,487.23	**2.58**
30	−17.98 475.42 109.38 257.63 209.42	206.77	183.55	**2.52**						
50	2,289.55 1,855.26 2,293.52 1,416.03 1,702.03	1,911.28	381.24	**11.21**						
70	8,240.45 6,476.78 7,675.61 7,838.76 4,609.69	6,968.26	1,473.18	**10.58**	181,508.02	245,021.92	**2.87**			
100	20,988.62 22,848.00 28,840.57 29,700.57 31,495.35	26,774.62	4,582.87	**13.06**						
300	587,582.24 498,674.56 450,710.15 599,592.10 417,346.92	510,781.19	81,041.41	**14.09**						

significant advantages. Moreover, t-statistics in column t_2 and t is much smaller than 14.09 obtained by scale 300 because the differences in objective function values of problems in different scales vary dramatically, giving rise to large standard deviations which can bring a smaller t-statistics, according to the equation of t-statistic, i.e., $\overline{d}/(S_D/\sqrt{n})$. Standard deviation depends on the difference in instances, which can be considered as a property of problems.

As for the comparison between GA and PSO-LS, t-statistics was equal to 2.43 for all the gaps between GA and PSO-LS and the t-statistics is larger than $t_{\alpha=0.05}(29)$ (1.6991). So PSO-LS was significantly better than GA generally. PSO framework was time-saving, which gave rise to less time cost within each iteration so that within the executing time limit, the number of iterations of PSO-LS can be more, compared with the GA algorithm with relatively complex operators of selection, crossover, and mutation. Moreover, the local search strategy with the efficient insertion operator tailored specifically for this problem improved gbest which all the particles rely heavily on to update their states, leading to improvement in almost each iteration. Because of the energy constraints, the insertion operator can cause great changes. However, the operators of GA, such as crossover and mutation, can bring less diversity of offspring, and good genes might be likely to be lost in inheritance.

There are more information between PSO-LS and GA in detail. Firstly, when the number of jobs was 10, GA had a slightly larger average TWT than PSO-LS at most instances, except for instances 10-1 and 10-2 in Table 9, but the gaps between the objective function values obtained by PSO-LS and GA were smaller than 5.00 in all instances, which is shown in Table 11. The t-statistics of the gaps between GA and PSO-LS was 1.71, which was smaller than $t_{\alpha=0.05}(4) = 2.1318$. So, when the number of jobs was 10, PSO-LS was slightly better than GA and there was no statistically significant differences between them. This was because the problems with 10 jobs were not difficult and algorithms had converged or was close to converging within 1 second.

Secondly, when the number of jobs was 30, PSO-LS obtained the fewest points compared with PSO and GA except for 30-1, as shown in Table 9. So the differences between GA and PSO-LS were mostly nonnegative in Table 11. The t-statistics of the gaps was 2.26, which was greater than $t_{\alpha=0.05}(4) = 2.1318$, according to which the PSO-LS was statistically better than GA.

Thirdly, when the number of jobs was larger than 30, PSO-LS was much more effective than PSO and GA, as displayed in Table 9.

According to the t_1 column in Table 11, the t-statistics of gaps of TWT obtained from GA and PSO-LS was 4.58, 11.96,

Table 11: Statistical analysis between GA and PSO-LS.

Scale	GA-PSO-LS	$\overline{d}_1$	s_{D1}	t_1	$\overline{d}_2$	s_{D2}	t_2	$\overline{d}$	s_D	t
10	0.00 0.00 0.39 4.46 2.85	1.54	2.02	1.71	2,303.79	3,501.79	**2.55**	3,501,909.83	7,880,984.74	**2.43**
30	−231.73 919.69 475.85 346.14 666.50	435.29	430.87	**2.26**						
50	3,237.93 10,406.82 5,416.81 9,192.88 4,118.22	6,474.53	3,162.23	**4.58**						
70	23,281.28 18,945.30 22,429.50 28,031.23 30,488.91	24,635.24	4,606.30	**11.96**	7,001,515.88	10,119,889.35	**2.68**			
100	164,871.51 174,808.61 186,477.75 193,162.81 236,172.46	191,098.63	27,433.97	**15.58**						
300	21,648,450.18 21,872,173.55 20,356,950.33 21,580,033.38 18,486,461.34	20,788,813.76	1,416,510.18	**32.82**						

15.58, and 32.82 for scale from 50 to 300, which were all significantly larger than $t_{\alpha=0.05}(4) = 2.1318$. And t-statistics in column t_1 had an upward trend with the increase in jobs, which illustrated the superior performance of PSO-LS than GA especially for large-scale problems.

As for t-statistics of differences between GA and PSO for the small-scale and large-scale problems, according to column t_2 in Table 11, they were 2.55 and 2.68, respectively, which were both greater than $t_{\alpha=0.05}(14) = 1.7613$. It shows good global searching capability of PSO-LS against GA.

In conclusion, PSO-LS has a strong search capability for finding more optimal solutions than those obtained by basic PSO and GA. The larger the number of jobs, the larger the gap in the results between PSO-LS and GA. Because of local search strategy, PSO-LS is statistically better than the basic PSO and GA significantly.

5. Conclusion

In this paper, we studied a single-machine scheduling problem with energy consumption limits to minimize total weighted tardiness. Based on the basic PSO algorithm, a local search was designed to be embedded in the algorithm as PSO-LS to improve the performance of the original algorithm, which is one aspect of our main contribution. Another aspect of our contribution is the constraint handling process tailored for this problem, which was used to determine the start processing time and completion time of each job to obtain total weighted tardiness as the objective function value. The constraint handling process considers the actual production environment, reserving two efficiencies for starting processing time to be accurate to the seconds and using as much energy in each time window as possible without violating energy constraints.

Experimental results showed that this enhanced algorithm, PSO-LS, is more effective than the original PSO and GA algorithms especially for large-scale problems, which demonstrated the effectiveness of our algorithm design.

This research can be extended in the following aspects: first, more operators inside PSO could be developed to improve the performance of local search; second, other methods such as the adaptive large neighborhood search algorithm (ALNS) [25] could be used to bring more diversity of particles; third, initialization methods of PSO could be taken into account to improve both the diversity of particles and the goodness of initial solution; fourth, energy consumption limits with uncertainty can be taken into account to meet more complex production environment; fifth, other PSO variants such as binary PSO [26] can be considered to be used as the algorithm framework to handle the discrete problem. These issues will be addressed in the further studies.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Authors' Contributions

Qingquan Jiang and Xiaoya Liao contributed equally to this work.

Acknowledgments

This research was funded by the Natural Science Foundation of China (Grant no. U1660202), Project of Ministry of Education of China (Grant no. 2019ITA01018), Social Science Foundation of Fujian Province (Grant no. FJ2019B101), and Fujian Provincial Department of Science and Technology-Soft Science Research Plan Project (Grant nos. 2019R0093 and 2019R0094).

References

[1] L. Meng, C. Zhang, X. Shao, and Y. Ren, "MILP models for energy-aware flexible job shop scheduling problem," *Journal of Cleaner Production*, vol. 210, pp. 710–723, 2019.

[2] R. Zhang and R. Chiong, "Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption," *Journal of Cleaner Production*, vol. 112, no. 4, pp. 3361–3375, 2016.

[3] Y. He, F. Liu, H.-j. Cao, and C.-b. Li, "A bi-objective model for job-shop scheduling problem to minimize both energy consumption and makespan," *Journal of Central South University of Technology*, vol. 12, pp. 167–171, 2005.

[4] K. Gao, Y. Huang, A. Sadollah, and L. Wang, "A review of energy-efficient scheduling in intelligent production systems," *Complex & Intelligent Systems*, vol. 6, pp. 237–249, 2019.

[5] S. Lee, B. Do Chung, H. W. Jeon, and J. Chang, "A dynamic control approach for energy-efficient production scheduling on a single machine under time-varying electricity pricing," *Journal of Cleaner Production*, vol. 165, pp. 552–563, 2017.

[6] I. Módos, P. Šůcha, and Z. Hanzálek, "Algorithms for robust production scheduling with energy consumption limits," *Computers & Industrial Engineering*, vol. 112, pp. 391–408, 2017.

[7] A. Che, X. Wu, J. Peng, and P. Yan, "Energy-efficient bi-objective single-machine scheduling with power-down mechanism," *Computers & Operations Research*, vol. 85, pp. 172–183, 2017.

[8] A. Che, Y. Zeng, and K. Lyu, "An efficient greedy insertion heuristic for energy-conscious single machine scheduling problem under time-of-use electricity tariffs," *Journal of Cleaner Production*, vol. 129, pp. 565–577, 2016.

[9] L. Chen, J. Wang, and X. Xu, "An energy-efficient single machine scheduling problem with machine reliability constraints," *Computers & Industrial Engineering*, vol. 137, Article ID 106072, 2019.

[10] F. Shrouf, J. Ordieres-Meré, A. García-Sánchez, and M. Ortega-Mier, "Optimizing the production scheduling of a single machine to minimize total energy consumption costs," *Journal of Cleaner Production*, vol. 67, pp. 197–207, 2014.

[11] C.-H. Liu, "Mathematical programming formulations for single-machine scheduling problems while considering renewable energy uncertainty," *International Journal of Production Research*, vol. 54, no. 4, pp. 1122–1133, 2016.

[12] X. Wu and A. Che, "A memetic differential evolution algorithm for energy-efficient parallel machine scheduling," *Omega*, vol. 82, pp. 155–165, 2019.

[13] Y. Zhang, D.-W. Gong, and Z. Ding, "A bare-bones multi-objective particle swarm optimization algorithm for environmental/economic dispatch," *Information Sciences*, vol. 192, pp. 213–227, 2012.

[14] X.-f. Song, Y. Zhang, Y.-n. Guo, X.-y. Sun, and Y.-l. Wang, "Variable-size cooperative coevolutionary particle swarm optimization for feature selection on high-dimensional data," *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 882–895, 2020.

[15] Y. Zhang, D.-w. Gong, and J. Cheng, "Multi-objective particle swarm optimization approach for cost-based feature selection in classification," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 14, pp. 64–75, 2015.

[16] Y. Hu, Y. Zhang, and D. Gong, "Multiobjective particle swarm optimization for feature selection with fuzzy cost," *IEEE Transactions on Cybernetics*, vol. 50, pp. 2168–2275, 2020.

[17] X. Liao, R. Zhang, and R. Chiong, "Multi-objective optimization of single machine scheduling with energy consumption constraints," in *Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Honolulu, HI, USA, November 2017.

[18] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the ICNN'95-International Conference on Neural Networks*, pp. 1942–1948, Perth, WA, Australia, November 1995.

[19] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[20] M. Pinedo, *Scheduling*, Springer, New York, NY, USA, 2012.

[21] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.

[22] T. Murata, H. Ishibuchi, and H. Tanaka, "Genetic algorithms for flowshop scheduling problems," *Computers & Industrial Engineering*, vol. 30, pp. 1061–1071, 1996.

[23] K. A. De Jong, *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[24] G. Taguchi and M. S. Phadke, *Quality Engineering through Design Optimization*, Springer, New York, NY, USA, 1989.

[25] X. Liu, G. Laporte, Y. Chen, and R. He, "An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time," *Computers & Operations Research*, vol. 86, pp. 41–53, 2017.

[26] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 4104–4108, Computational Cybernetics and Simulation, Orlando, FL, USA, October 1997.