An Improved Particle Swarm Optimization Algorithm for the Hybrid Flowshop Scheduling to Minimize Total Weighted Completion Time in Process Industry

Lixin Tang, *Member, IEEE*, and Xianpeng Wang

***Abstract*—In this paper, we present an improved particle swarm optimization (PSO) algorithm for the hybrid flowshop scheduling (HFS) problem to minimize total weighted completion time. This problem has a strong practical background in process industry. For example, the integrated production process of steelmaking, continuous-casting, and hot rolling in the iron and steel industry, and the short-term scheduling problem of multistage multiproduct batch plants in the chemical industry can be reduced to a HFS problem. To make PSO applicable in the HFS problem, we use a job permutation that is the processing order of jobs in the first stage to represent a solution, and construct a greedy method to transform this job permutation into a complete HFS schedule. In addition, a hybrid variable neighborhood search (VNS) incorporating variable depth search, a hybrid simulated annealing incorporating stochastic local search, and a three-level population update method are incorporated to improve the search intensification and diversification of the proposed PSO algorithm. Computational experiments on practical production data and randomly generated instances show that the proposed PSO algorithm can obtain good solutions compared to the lower bounds and other metaheuristics.**

***Index Terms*—Hybrid flowshop scheduling (HFS), hybrid simulated annealing, hybrid variable neighborhood search, improved particle swarm optimization, three-level population update method.**

I. Introduction

THE hybrid flowshop scheduling (HFS) problem is a typical discrete combinatorial optimization problem. This problem consists of n jobs, S successive processing stages with m_j parallel identical machines in each stage j $(j = 1, \ldots, S)$, and infinite buffer capacities between consecutive stages. Each job is processed by one machine in each stage and it must go through all these S successive stages (see Fig. 1). Upon completion in stage j, a job can either be delivered to the immediately subsequent stage to process when one of the machines in that stage is available, or stay in the following buffer. The objective is to find a schedule so as to minimize a given performance measure such as makespan (Brah and Hunsucker [1]), total weighted completion time (Tang and Xuan [2]), maximum lateness (Vignier [3]), and so on.

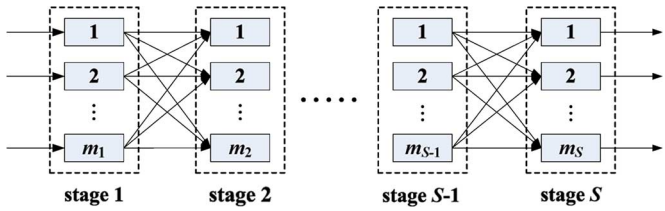


Fig. 1. Structure of the hybrid flowshop.

HFS problem has a strong practical background in process industry such as the iron and steel industry and the chemical industry. For example, Fig. 2 illustrates the integrated production process in the iron and steel industry, which can be classified into three sections (i.e., the primary steelmaking section, the hot rolling section, and the cold rolling section) according to the output products. In the primary steelmaking production, raw materials such as iron ore and coal are transformed sequentially into liquid iron in the blast furnaces, into liquid steel in the steel making furnaces, and into large solid steel slabs in the continuous casters. Then the slabs are transferred to the hot strip mill and stored in a slab yard. Before rolling, these slabs need to be reheated to a required temperature in the reheating furnaces, and then the reheated slabs are rolled into hot strip coils in the hot rolling lines. To further improve the mechanical quality of these coils, they are sent to the cold rolling section. In this section, the surface oxidation on these coils are first eliminated by pickling and then transformed into thinner cold coils in the cold rolling lines. To eliminate the inside stress caused by the cold rolling and make the coils anticorrosive, the cold coils are then annealed and galvanized in the galvanizing lines. Finally, the galvanized coils are color-coated in the color-coating lines to meet some special requirements. To keep the continuous production of all stages and reduce the impact of bottleneck stages on the shop floor capacities, at each stage the facilities are duplicated in parallel. Therefore, each section of the integrated production process in the iron and steel industry can be seen as a two-stage or three-stage HFS problem, while the whole integrated production process can be reduced to a multistage HFS with identical parallel machines in each stage.

Manuscript received April 01, 2009; revised July 23, 2009. Manuscript received in final form November 09, 2009. First published December 22, 2009; current version published October 22, 2010. Recommended by Associate Editor S.-L. Jamsa-Jounela. This work was supported by the National Natural Science Foundation for Distinguished Young Scholars of China under Grant 70425003, by the National 863 High-Tech Research and Development Program of China under Grant 2006AA04Z174, and by the National Natural Science Foundation of China under Grant 60674084.

The authors are with the Liaoning Key Laboratory of Manufacturing System and Logistics, The Logistics Institute, Northeastern University, Shenyang 110004, China (e-mail: qhjytlx@mail.neu.edu.cn; wangxianpeng@ise.neu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCST.2009.2036718

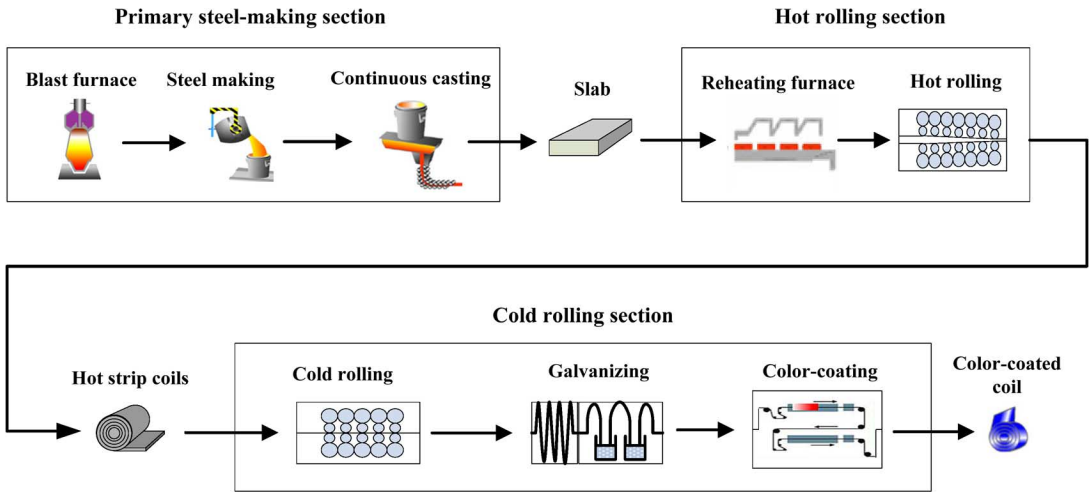


Fig. 2. Illustration of the integrated production process in the iron and steel industry.

Besides the integrated production scheduling of the iron and steel industry, many practical production problems in the chemical industry can also be reduced to a HFS problem. For example, Pinto and Grossmann [4] and Castro and Grossmann [5] studied the short-term scheduling problem of multistage multiproduct batch plants in chemical industry, and formulated this kind of problem as a mixed integer linear programming model, which can be seen as a hybrid flowshop model. Ždánský and Požívil [6] studied the scheduling problems of batch plants in chemical industry and proposed an HFS model for this problem. Therefore, the efficient scheduling of HFS problems is very important for many practitioners of enterprises in different industries, and it can help the enterprises to reduce production cost, to improve productivity, and so on.

The HFS problem can be seen as a mixture of the flowshop scheduling and the parallel machine scheduling, so it is more complex and difficult to be solved. Even when there are only two stages in the HFS, it is also NP-hard [7]. Therefore, efficiently solving the HFS problem has been a challenging area and numerous research works have been devoted to it. A review of related research on HFS, including processing complexity, scheduling criterion and scheduling approaches, can be found in Linn and Zhang [8]. Branch and bound (B&B) and heuristics are the two most commonly employed approaches to HFS scheduling. Brah and Hunsucker [1] proposed a B&B algorithm including three parts: lower bound calculation, branching and node elimination. Moursli and Pochet [9] introduced a B&B algorithm to minimize makespan. They adopted several heuristics for the computation of upper bounds and single-stage subproblem relaxation for lower bounds. Néron *et al.* [10] proposed a B&B algorithm in which energetic reasoning and global operations are adopted to improve the efficiency of the B&B algorithm. Tang and Xuan [2] presented a mixed integer programming for the HFS problem to minimize the total weighted completion time and developed a Lagrangian relaxation algorithm in which a dynamic programming was designed to solve the parallel identical machine subproblems. The computational results show that this algorithm can obtain good lower bounds. Since the B&B algorithm is very time consuming and can only obtain optimal solutions for small size HFS problems, heuristics are often employed to solve large size problems. Nowicki and Smutnicki [11] proposed a tabu search heuristic for the HFS problem. Six heuristics were given in Botta–Genoulaz [12] to solve the HFS problem with precedence constraints and time lags to minimize maximum lateness. Yang *et al.* [13] analyzed three heuristics based on decomposition methods and local search to minimize the total weighted tardiness of jobs with release dates for the HFS problem. Engin and Döyen [14] presented an artificial immune system for the HFS problem to minimize makespan. Allaoui and Artiba [15] proposed a heuristic that integrated the simulation and optimization to solve the HFS with maintenance constraints. A genetic algorithm was proposed in Ruiz and Maroto [16] for the HFS with sequence dependent setup times and machine eligibility. Bellanger and Oulamara [17] investigated a special two-stage HFS in which the first stage contains several identical discrete machines and the second stage contains several identical batching machines (that can simultaneously process several jobs in a batch mode). Several heuristics were proposed along with their worst cases analysis. Jungwattanakit *et al.* [18] compared the performance of several metaheuristics such as simulated annealing, tabu search and genetic algorithm for the HFS with sequence-dependent and machine-dependent setup times.

As described above, most of the researches on the HFS are focused on the criterion of minimizing makespan. However, in recent years the criterion of minimizing total weighted completion time has started to draw more attention from researchers because it is more relevant and meaningful for today's dynamic production environment and directly related to important manufacturing logistics measures such as the work-in-process inventory level and on time delivery of products. That is, the efficient scheduling of HFS with the objective of minimizing total weighted completion time can help manufacturing enterprises to improve the balance of logistics of different production lines in each stage and make the time connection of consecutive stages more precise, which in turn will help to reduce the production cost, the work-in-process inventory level, and the delivery tardiness of final products. Therefore, in this paper we consider the HFS problem to minimize the total weighted completion

time (denoted as $\mathrm{HFS}/\sum W_jC_j$), which has been proven to be NP-hard by Lenstra *et al.* [19].

The main contribution of this paper is that we proposed an improved PSO algorithm to solve the $\mathrm{HFS}/\sum W_jC_j$. To make this algorithm applicable in HFS, we use a job permutation that is the processing order of jobs in the first stage to represent a solution, and construct a greedy method to transform this job sequence into a complete HFS schedule. To improve the search intensification, we developed two kinds of local search methods: the hybrid variable neighborhood search incorporating variable depth search, the hybrid simulated annealing incorporating stochastic local search, and apply them to particles in a systematical way. Furthermore, to improve the search diversification, a three-level population update scheme is incorporated in the PSO to diversify the particles to search more solution spaces.

The rest of this paper is organized as follows. Section II describes the proposed PSO algorithm, as well as the improvement strategies (local search and population update method) incorporated in the PSO. The computational results on practical production data and randomly generated instances for $\mathrm{HFS}/\sum W_jC_j$ are presented in Section IV. Finally, Section V concludes this paper.

II. PSO Algorithm for HFS

As one of the latest evolutionary optimization methods, the particle swarm optimization (PSO) has been successfully applied in many scheduling problems such as the single machine scheduling [20], flowshop scheduling [21], job shop scheduling [22], open shop scheduling [23]. However, in the literature there have been few PSO algorithms proposed for the HFS problem with multiprocessor tasks, which is different from the HFS problem considered in this paper. Ercan and Fung [24] developed a hybrid PSO algorithm for the HFS problem with multiprocessor tasks, and the computational results provided by the authors showed that the proposed PSO algorithm is superior to the tabu search and is comparable with the genetic algorithm and ant colony system. Ercan [25] proposed two kinds of PSO algorithms hybrid with tabu search and simulated annealing for the HFS problem with multiprocessor tasks. Tseng and Liao [26] also took into account the HFS problem with multiprocessor tasks, and developed a hybrid PSO with reduced variable neighborhood search. The computational results showed that the basic PSO algorithms have competitive or superior performance as compared to genetic algorithm, ant colony system, and tabu search, whereas the hybrid PSO with reduced variable neighborhood search has the best performance. Due to the promising performance of PSO, in this paper we develop an improved PSO algorithm for the $\mathrm{HFS}/\sum W_jC_j$ problem.

A. Brief Introduction of PSO

PSO algorithm is a population based metaheuristic method introduced by Kennedy and Eberhart [27], [28] based on the social behavior of bird flocking and fish schooling, as well as the means of information exchange between individuals, to solve optimization problems. In the PSO algorithm, a swarm consists of m individuals (called *particles*) flying around in an n-dimensional search space. The position of the ith particle at the tth iteration is used to evaluate the particle and represent the candidate solution for the optimization problem. It can be represented as $X_i^t = [x_{i1}^t, x_{i2}^t, \ldots, x_{\mathrm{in}}^t]$, where x_{ij}^t is position value of the ith particle with respect to the jth dimension $(j = 1, 2, \ldots, n)$. During the search process, the position of a particle is influenced by two factors: the best position visited by itself (p_{best}) denoted as $P_i^t = [p_{i1}^t, p_{i2}^t, \ldots, p_{\mathrm{in}}^t]$, and the position of the best particle found so far in the swarm (g_{best}) denoted as $G^t = [g_1^t, g_2^t, \ldots, g_n^t]$. The new velocity (denoted as $V_i^t = [v_{i1}^t, v_{i2}^t, \ldots, v_{\mathrm{in}}^t]$) and position of particle i at the next iteration are calculated according to

$$v_{ij}^{t+1} = w \cdot v_{ij}^t + c_1 r_1 \cdot \left(p_{ij}^t - x_{ij}^t\right) + c_2 r_2 \cdot \left(g_j^t - x_{ij}^t\right) \quad (1)$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \quad (2)$$

where w is called the *inertia* parameter, c_1 and c_2 are, respectively, *cognitive* and *social* learning parameter (usually $c_1 = c_2 = 2$), and r_1, r_2 are random numbers between (0,1). Based on the above equations, the particle can fly through search space toward p_{best} and g_{best} in a navigated way while still exploring new areas by the stochastic mechanism to escape from local optima. Among these parameters, the *inertia* parameter w controls the momentum of each particle. A larger w increases the global exploration ability while a smaller w tends to improve the local exploration ability of particles. Therefore, to balance these two abilities, the *inertia* parameter can be time-varying through the following formula:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{T_{\max}} \times t \quad (3)$$

where t and $T_{\max}$ are, respectively, the current and maximum number of iterations, and $w_{\max}$ and $w_{\min}$ are, respectively, the upper and lower bounds for w. By doing so, particles can have good global exploration ability at the beginning phase of the search process, and good local exploration ability as the current number of iterations increases.

B. Solution Representation

Due to the continuous characters of the position values of particles in the PSO, general solution representation of HFS cannot be directly adopted in the PSO algorithms. To overcome this obstacle, we propose the following method. First, let the given particle $X_i^t = [x_{i1}^t, x_{i2}^t, \ldots, x_{\mathrm{in}}^t]$ correspond to the continuous position values for n jobs in the HFS problems (that is, each dimension represents a job). Second, using the smallest position value (SPV) rule proposed in [20] to convert X_i^t into a discrete job permutation $\pi_i^t = \{\pi_{i1}^t, \pi_{i2}^t, \ldots, \pi_{\mathrm{in}}^t\}$, in which π_{ij}^t denote the job arranged in the jth position. Third, take π_i^t as the processing order of these n jobs in the first stage and construct the corresponding complete schedule of HFS using a greedy method. That is, in our PSO, the solution is represented as the job permutation instead of a complete complex HFS schedule. Through this method, the solution representation becomes simple and the PSO can then be applicable in HFS. Furthermore, the neighborhood used in the local search of PSO also becomes simple and easy to be implemented.

1) SPV Rule: A simple example is provided in Table I to illustrate the SPV rule. In this instance $(n = 8)$, the smallest position value is $x_{i6}^t = -1.20$, so job 6 is assigned to the first

TABLE I
SOLUTION REPRESENTATION AND THE CORRESPONDING JOB PERMUTATION

Dimension j	1	2	3	4	5	6	7	8
x_{ij}^t	0.54	–0.75	–1.02	–0.41	0.92	–1.20	0.23	0.12
Job, π_{ij}^t	6	3	2	4	8	7	1	5

position of the job permutation according to the SPV rule, then job 3 is assigned to the second position of the job permutation because the second smallest position value $x_{i3}^t = -1.02$. With the same way, other jobs are assigned in their corresponding position of the job permutation according to their position values. Thus, based on the SPV rule, the job permutation is obtained, i.e., $\pi_i^t = \{6, 3, 2, 4, 8, 7, 1, 5\}$.

2) Greedy Method for Constructing a Complete Schedule: To transform the job permutation π_i^t into a complete schedule of HFS, the following greedy construction method is used.

Step 1) Take π_i^t as the processing order of these n jobs in the first stage, and set $j = 1$.

Step 2) Set $k = M_j$, and assign the first k jobs in π_i^t on the M_j machines in stage j. Calculate the completion time of each job and update the first available machine in stage j.

Step 3) Set $k = k + 1$. If $k \leq n$, assign job π_{ik}^t on the first available machine in stage j. Calculate the completion time of this job and then reupdate the first available machine in stage j. Repeat this step until all jobs have been assigned in stage j.

Step 4) Sequence jobs in the ascending order of their completion time in stage j, and update π_i^t accordingly. Set $j = j+1$. If $j > S$, stop; otherwise, go to *Step* 2.

Since this construction method is a kind of greedy heuristic, the obtained result is only a near-optimal schedule. With the complete schedule, the objective value of this schedule can be then easily determined.

C. Population Initialization

The population with n_{pop} solutions is initialized by two kinds of heuristic procedures so that both the solution quality and diversity can be considered.

1) Deterministic Rules: The first solution is generated using the deterministic priority rule: the weighted shortest processing time (WSPT) rule that sequences jobs in nondecreasing order of p_i/w_i, where p_i is the total processing time through all stages of job i and w_i is the weight value of job i.

Based on the discrete job permutations π_1^0 obtained by the WSPT, the second discrete job permutations π_2^0 is then obtained using the following NEH rule.

Step 1) Copy the given job permutation π_1^0 to s, and set π_2^0 to be empty.

Step 2) Take the first two jobs in s, insert them into π_2^0, and determine their sequence to minimize the objective function as if the problem only consists of these two jobs. Delete these two jobs from s.

Step 3) Take the first job in s and insert it into the previously obtained partial solution π_2^0 at the position that gives the least increased objective function value. Delete the job from s. Repeat this step until all jobs in s have been inserted.

TABLE II
ILLUSTRATION OF THE PROPOSED POSITION VALUE ADJUSTMENT METHOD

Dimension j		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Before local search	x_{ij}^t	0.54	–0.75	–1.02	–0.41	0.92	–1.20	0.23	0.12
	Job, π_{ij}^t	6	3	2	4	8	7	1	5
After local search	Job, π_{ij}^t	6	7	2	5	4	8	1	3
	x_{ij}^t	0.54	–0.75	0.92	0.12	–0.41	–1.20	–1.02	0.23

Since the first two solutions are discrete job permutations, we should convert them into four particles with continuous position values. For a given job permutation π_i^0, the position value of job $x_{i,\pi_{ij}^0}^0$ (arranged in the jth position in π_i^0) is calculated by $x_{\min} + ((x_{\max} - x_{\min})/n) \times j$, where $x_{\min} = -4.0$ and $x_{\max} = 4.0$.

2) Randomized Rule: The continuous position values of the other $n_{\text{pop}} - 2$ solutions are randomly generated according to the following formula: $x_{ij}^0 = x_{\min} + (x_{\max} - x_{\min}) \times \text{rand}$, where rand denotes a random number uniformly distributed in interval [0, 1].

The initial velocities for the PSO particles are also generated by a randomly scheme that is similar to the position value formula: $v_{ij}^0 = v_{\min} + (v_{\max} - v_{\min}) \times \text{rand}$, where $v_{\min} = -4.0$ and $v_{\max} = 4.0$. Such a scheme can improve the diversity of the search directions of particles.

D. Local Search Procedure of the PSO Algorithm

To improve the search intensification of the PSO algorithm, we use two kinds of local search methods: a hybrid simulated annealing incorporating stochastic local search (denoted as HSA_SLS) and a hybrid variable neighborhood search incorporating variable depth search (HVNS_VDS), and systematically change the adoption sequence of them (details of the adoption scheme are described in Section II-F).

1) Neighborhood: Two neighborhoods are used in our local search methods. The neighborhood N_1 is the set of all possible insert moves, each of which removes a job from its current position in the solution and reinserts it at a different position. The neighborhood N_2 is the set of all possible swap moves, each of which swaps two jobs at two different positions in the solution.

In our PSO, the local search is not directly applied to continuous position values but to job permutations. To make the position values change accordingly after a local search operator such as insert or swap applied to the job permutation, a position value adjustment method is proposed in [20] to adjust the position value of each dimension. However, this adjustment may be very time-consuming when the neighborhood size is large. So in this paper we use another method. That is, we first apply the local search without adjusting the position values, and then adjust them based on the final improved solution obtained by the local search. For example, Table II shows the original position values of each job, and the corresponding position value changes according to the final improved solution $\pi = (6, 7, 2, 5, 4, 8, 1, 3)$ obtained by the local search. By doing this, we only need to apply the position value adjustment once.

Begin:
Initialization:
Let π_0 be the input initial solution. Set $\pi = \pi_0$, the acceptance threshold T=0.05, and *outerloop_counter*=0.
while (*outerloop_counter* < n/2) **do**
1. Generate a random number r in [0, 1], and two random integer numbers w and z in [1, n]. If r>0.5, generate $\pi' = insert(\pi, w, z)$; otherwise generate $\pi' = swap(\pi, w, z)$.
2. Set *inner_counter*=0.
3. **while** (*inner_counter*<$n\times(n-1)$) **do**
Set k=1.
while ($k \le 2$) **do**
(1) Generate two random integer numbers w, z in [1, n].
(2) If k=1, generate $\pi''=insert(\pi', w, z)$.
(3) If k=2, generate $\pi''=swap(\pi', w, z)$.
(4) If $f(\pi'') < f(\pi')$, set $\pi' = \pi''$ and k=1; otherwise set $k=k+1$.
end while
Set *inner_counter*++.
end while
4. If $f(\pi') < f(\pi_0)$, set $\pi_0 = \pi'$ and $\pi = \pi'$, and then go to step 6; otherwise go to step 5.
5. If $f(\pi') \ge f(\pi_0)$ but $(f(\pi')-f(\pi_0))/f(\pi_0) \le T$, set $\pi = \pi'$; otherwise, generate a random number r in [0, 1], and then set $\pi = \pi_0$ if r>0.5 and $\pi = \pi'$ if $r \le 0.5$.
6. Set $T=T\times0.95$ and *outerloop_counter*++.
end while
Report the improved solution π_0.
End

Fig. 3. Main procedure of HSA_SLS.

2) HSA_SLS Method: Typical local search methods generally search the entire neighborhood to find the best neighbor solution. However, this search scheme may be very time-consuming when the neighborhood size is large. Therefore, in our method we propose to use a stochastic local search (SLS), which searches the neighborhood in a stochastic way. One major difference of SLS from typical local search methods is that it does not search the entire neighborhood, but a part of the entire neighborhood through a sufficient number of random moves. To further improve the exploration ability of the SLS, we incorporate the solution acceptance scheme of simulated annealing into it, and thus obtain the HSA_SLS method. To reduce the computation time and make the search process focus on the intensification phase, we use a decreasing acceptance threshold to act as the cooling procedure of simulated annealing (for simplicity, we use a linearly decreasing temperature instead of the classical one used in traditional simulated annealing algorithm). For a given discrete job permutation π^t, let w and z denote two different random integer numbers generated in [1, n], and then the two stochastic neighborhood moves used in the HSA_SLS to generate a neighbor solution $\pi^{t'}$ are: 1) $\pi^{t'} = insert(\pi^t, w, z)$: remove the job at the wth position and insert it in the zth position; and 2) $\pi^{t'} = swap(\pi^t, w, z)$: interchange two jobs arranged at the wth position and the zth position. Then the procedure of the proposed HSA_SLS method can be described Fig. 3.

3) HVNS_VDS Method: VNS is a simple and effective metaheuristic recently proposed by Mladenović and Hansen [29], and Hansen and Mladenović [30] gave a detailed description. Different from most local search heuristics, the principle of VNS is to use two or more neighborhoods and systematically change the neighborhood within a local search algorithm. Therefore, besides the above HSA_SLS method that performs a random insert move or a random swap move on a solution at a time, we also propose a hybrid method based on VNS, in which a VDS is incorporated to find and perform compound moves on a solution at a time based on the fact that compound moves have shown great advantages over single move [31], [32] to act as the local search. Compound moves are referred to as a series of *independent moves* (note that the number of these moves is variable) to be simultaneously performed on a solution. If two moves are performed in two different regions of a solution that have no intersection, then they are called the *independent moves*. For example, for two swap moves (a_1, b_1) and (a_2, b_2), if $\max\{a_1, b_1\} < \min\{a_2, b_2\}$ then these two moves are called *independent moves*.

The procedure of the HVNS_VDS method follows the framework of [29] except that the *shaking* procedure, in which a random solution x' based on the current solution x is generated using a random move in the current neighborhood, is not adopted.

Step 1) Take the neighborhood structures N_1 (*insert move neighborhood*) and N_2 (*swap move neighborhood*) to be used in the search. Take the input X_k^t (the best particle among the population in the current

iteration) and the corresponding job permutation π_k^t as the initial solutions. Set $k = 1$.

Step 2) Repeat the following steps if $k \leq 2$:

Step 2.1—*Neighborhood Search*: Find the best compound moves in the neighborhood N_k of π_k^t.

Step 2.2—*Move or Not*: If the best compound moves found in N_k can improve π_k^t, apply them to π_k^t, adjust the position values of X_k^t correspondingly, and then set $k = 1$; otherwise set $k = k + 1$.

Step 3) Return the improved solutions X_k^t and π_k^t.

To avoid duplicated description, in the following we only take the swap move to describe this VDS method. Given a job permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, in which $\pi(i)$ denotes the index of a job arranged in the ith position. Let $m_k = (\pi(i_k), \pi(j_k))$, in which $j_k > i_k$, denote the profitable swap move found at stage k of the VDS procedure and g_k the corresponding improvement in the objective function value by performing this move. Note that $g_k > 0$ indicates that this move can improve the current solution. Then the procedure of the VDS can be described as follows.

Step 1) Set $k = 1$. Starting from $\pi(1)$, search for the first profitable swap move $m_1 = (\pi(i_1), \pi(j_1))$ satisfying $g_1 > 0$. If no such swap move exists, return π as the local optimum and stop.

Step 2) Set $k = k+1$. Starting from $\pi(j_{k-1}+1)$, search for the first profitable swap move $m_k = (\pi(i_k), \pi(j_k))$ satisfying $g_k > 0$. Repeat this step until no further profitable swap move exists.

Step 3) Simultaneously perform these k independent profitable swap moves $\{m_1, \ldots, m_k\}$ on π. Then go to *Step* 1 with the new solution π.

4) Speedups: As described above, the HSA_SLS is a kind of stochastic local search and is applied to the best particle in the current population at each iteration. If the quality of this particle is too low (namely its objective function value is much bigger than that of the global best solution g_{best}), then there will be little chance to find a new better global best solution by applying the HSA_SLS to this particle. Therefore, we introduce a threshold value to prevent the application of the HSA_SLS to low quality particles. That is, the HSA_SLS can be applied to a particle π only if $(f(\pi) - f(g_{\text{best}}))/f(g_{\text{best}}) \leq threshold\_value$.

Different from the HSA_SLS, the HVNS_VDS is a kind of deterministic local search method. If at a certain iteration we apply the HVNS_VDS to a particle π and obtain an improved solution π', then at later iterations the application of this method on the same particle π will also obtain the same improved solution π'. So this kind of situation should be avoided. To reach this aim, a tabu list of all searched particles by the HVNS_VDS is used. Since the Hash function value can be used to easily check the duplicated solutions (that is, if the Hash function values of two solutions are equal, then the two solutions are the same), we only memorize the Hash function value of each searched solution. If a particle has the same Hash function value stored in the tabu list, then it is omitted by the HVNS_VDS. The Hash function value of a particle $\pi = (\pi(1), \ldots, \pi(n))$ is calculated as $h(\pi) = \sum_{j=1}^{n} (j \times \pi(j) \times \pi(j))$.

E. Three-Level Population Update Scheme

As the evolution of the PSO processes, some particles may always fly around low quality regions for many consecutive iterations, or some particles may tend to converge quickly (i.e., the premature phenomenon), or some particles may be trapped in local optimum. To break through these obstacles for finding better solutions, a three-level population update method is developed to improve the search diversification of the PSO: 1) the update of particles with low probability of finding better solutions; 2) the update of particles with bad diversification; and 3) the update of the whole population.

1) Level 1: Update of Particles With Low Probability of Finding Better Solutions: To determine the particles with low probability of finding better solutions, we track the consecutive iterations without improvement on the personal best p_{best} for each particle X_i (i.e., denoted as $PbNO_i$). If $PbNO_i$ is bigger than a limit (i.e., ten consecutive iterations), then this particle i will be replaced with a new generated particle. To balance the particle quality and the diversification, the new particle is not generated randomly, but in a way that takes use of the previous search experience. To reach this aim, we use a reference set (i.e., Refset) that is a collection of b particles that are used to generate the new particle by means of a *solution combination method*. The reference set includes two parts: 1) $Refset_1$ with $b/2$ particles that focus on the objective function value and 2) Refset_2 with $b/2$ particles that focus on the diversification. To evaluate the diversification of particles, we define the distance between two particles X_i and X_j as $d(X_i, X_j) = d(\pi_i, \pi_j) = |h(\pi_i) - h(\pi_j)|$, where π is the corresponding job permutation (obtained by the SPV rule) of particle X and $h(\pi)$ is the Hash function value of π. We prefer to use the Hash function value to define the distance between two solutions because it needs not determine the difference between solutions, which is time-consuming for large size problems, and thus can help to reduce the computational efforts of calculating the distance of solutions.

Then at each iteration t, the generation method of the new particle that replaces the particle X_i with $PbNO_i > 10$ is as follows.

Step 1. Initialization. Copy the current population to P, and set $Refset$ to be empty.

Step 2. Build $Refset_1$. Generate a random number b in [1, 5]. Randomly select $b/2$ particles from the best n particles in P, add them to $Refset$, and delete these particles from P.

Step 2. Build $Refset_2$. Repeat the following steps until the other $b/2$ particles with good diversification from P are added to $Refset$.

Step 2.1 For each particle in P, compute the minimum value of its distances to the particles in $Refset$.

Step 2.2 Select the particle with the maximum value of the minimum distance from P, add it to $Refset$, and delete this particle from P. Then go to *Step 2.1*.

Step 3. Solution combination method

Step 3.1 Calculate the relative value r_j of each particle X_j in the Refset by $r_j = (1/f(X_j))/(\sum_{i\in \mathrm{Refset}} /f(X_i))$ (that is, $0 < rj < 1$ and $\sum_{j\in \mathrm{Refset}} r_j = 1$).

Step 3.2 Determine the value of each position k $(k = 1, 2, \ldots, n)$ of the new particle $X'_i = (x'_{i1}, \ldots, x'_{ik}, \ldots, x'_{\mathrm{in}})$ by $x'_{ik} = \sum_{j=1}^{|\mathrm{Refset}|} r_j \times x_{jk}$.

Step 3.3 Determine the new velocity V'_i of particle X'_i by $v'_{ik} = \sum_{j=1}^{|\mathrm{Refset}|} r_j \times v_{jk}$.

Step 4. Particle update. Replace X_i with X'_i, replace V_i with V'_i, and set the personal best solution found by particle X_i to be X_i.

2) Level 2: Update of Particles With Bad Diversification: As the PSO evolves, some particles may tend to converge, which will reduce the search diversification and consequently lead the search to be trapped in local optimum. To avoid this situation, we check the diversity of the population and replace some particles with bad diversity. This update procedure is applied every ten iterations or the global best solution found so far (g_{best})\ has not been improved for a number of consecutive iterations (i.e., ten consecutive iterations in our algorithm). This update method can be described as follows.

Step 1) For each particle in the current population, compute the average value of its distances to the other particles.

Step 2) Construct a particle index list L in which the particle indexes are sequenced in the nondecreasing order of their average distance value. Let $L(k)$ denote the index of the particle arranged at the kth position in L. Set $k = 1$.

Step 3) Repeat the following steps until $k > n$.

Step 3.1: Generate a random number p in (0, 1), if $p < 0.5$ or the personal best solution of the particle $X_{L(k)}$ is the global best solution, go to *Step 3.3*; otherwise go to *Step 3.2*.

Step 3.2: Replace the particle $X_{L(k)}$ in the current population with a new particle that is obtained using the same method described in the above Section II-E1.

Step 3.3: Set $k = k + 1$.

After one application of the above particle update procedure, at most n particles with bad diversification are replaced based on probability with new generated particles that consider both the particle quality and diversification.

3) Level 3: Update of the Whole Population: If the global best solution found so far (g_{best}) has not been improved for a number of consecutive iterations (i.e., 20 consecutive iterations in our algorithm), the whole population will be reinitialized using the initialization method described in Section II-C. However, to inherit the previous search result, the personal best of each particle and the global best particle will be the same obtained in the previous iteration.

F. Complete Procedure of the PSO Algorithm

Let $PbNO_i$ denote the consecutive iterations without improvement on the personal best P_i^t for each particle X_i^t. Since the consecutive iterations without improvement for the global best solution is used in both the level 2 and the level 3 update method, we used $GbNO_1$ and $GbNO_2$ to denote the consecutive iterations that the global best solution G^t has not been updated. Let $SLSNO$ denote the consecutive iterations that the HSA_SLS method does not improve the global best solution G^t. Then the complete procedure of the proposed PSO algorithm can be summarized as follows.

Step 1) *Initialization*.

Step 1.1: Set $t = 0$, $n_{\mathrm{pop}} = 10\,\mathrm{n}$, $GbNO_1 = 0$, $GbNO_2 = 0$, $SLSNO = 0$, and $PbNO_i = 0$ for each particle X_i^t $(i = 1, 2, \ldots, n_{\mathrm{pop}})$. Set the tabu list to be empty. Set initial values for the *inertia* weight, the *cognitive*, and *social* parameters. Create the initial population and the initial velocities for each particle using the method described in Section II-C.

Step 1.2: Calculate the corresponding objective value $f(X_i^t)$ of each particle X_i^t using the method described in Section II-B.

Step 1.3: Set the personal best of each particle to be a copy of the particle itself $(P_i^t = X_i^t)$, and the global best to be the best one among the population $(G^t = X_k^t$, $k = \arg\min\{f(X_i^t), \forall i \le N\})$.

Step 2) *Update Particles*.

Step 2.1: Update iteration counter $t = t + 1$.

Step 2.2: Update *inertia* weight according to (3).

Step 2.3: For each particle X_i^t, update the velocity and position values according to (1) and (2).

Step 2.4: Calculate the corresponding objective value $f(X_i^t)$ of each particle X_i^t using the method described in Section II-B.

Step 2.5: For each particle X_i^t, if $f(X_i^t) < f(P_i^t)$, then update the personal best $P_i^t = X_i^t$ and set $PbNO_i = 0$; otherwise set $PbNO_i = PbNO_i + 1$.

Step 2.6: Find the best particle X_k^t with the minimum objective value (the best particle in the current population). That is, $k = \arg\min\{f(X_i^t), \forall i \le N\}$.

Step 2.7: Find the personal best particle P_m^t with the minimum objective value. That is, $m = \arg\min\{f(P_i^t), \forall i \le N\}$. If $f(P_m^t) < f(G^t)$, update the global best $G^t = P_m^t$, and set $GbNO_1 = GbNO_2 = 0$; otherwise, $GbNO_1 = GbNO_1 + 1$, and $GbNO_2 = GbNO_2 + 1$.

Step 3) *Local Search*.

Step 3.1: Improve the best particle in the current population. If $(f(X_k^t) - f(G^t))/f(G^t) > threshold\_value$, go to *Step 3.2:*; otherwise apply the *HSA_SLS* to improve X_k^t. If X_k^t is improved, add the Hash value of G^t to the tabu list, and then apply the *HVNS_VDS* to further

improve X_k^t, update the personal best $P_k^t = X_k^t$ and set $PbNO_k = 0$.

Step 3.2: Improve the global best solution. If $SLSNO \leq 3$, go to *Step 3.2.1*; otherwise go to *Step 3.2.2*.

Step 3.2.1: Apply the *HSA_SLS* to improve G^t. If G^t is improved, set $SLSNO = 0$, add the Hash value of G^t to the tabu list, and apply the *HVNS_VDS* to further improve G^t; otherwise set $SLSNO = SLSNO+1$.

Step 3.3.2: Set $SLSNO = 0$. If the Hash value of G^t is not stored in the tabu list, then add the Hash value of G^t to the tabu list, and apply the *HVNS_VDS* to improve G^t; otherwise, perform a random $insert$ move to a copy of G^t, and then apply the *HVNS_VDS* to improve the copy of G^t. If the improved copy is better than G^t, update G^t, and then apply the *HSA_SLS* to further improve G^t.

Step 3.3.3: If G^t is improved, set $GbNO_1 = GbNO_2 = 0$.

Step 4) *Population Update*.

Step 4.1: For each particle X_i^t, if $PbNO_i > 10$, then update this particle (as well as its personal best P_i^t) using the *level 1 update method* and set $PbNO_i = 0$.

Step 4.2: If t is a multiple of 10 or $GbNO_1 > 10$, then call the *level 2 update method* to replace particles with bad diversification, and set $GbNO_1 = 0$.

Step 4.3: If $GbNO_2 > 20$, then call the *level 3 update method* to re-initialize the whole population, and set $GbNO_1 = GbNO_2 = 0$. Remain the personal best particle and the global best particle unchanged so that the previous search results can be inherited in the next iteration.

Step 5) *Stopping Criterion*.

If the runtime has reached the limit, then stop; otherwise, go to *Step 2*.

III. Computational Experiments

To test the performance of our PSO algorithm, computational experiments were carried out for the $\mathrm{HFS}/\sum W_jC_j$ problem. Our improved PSO algorithm was implemented using C++, and tested on a personal PC with Pentium IV 3.0 GHz CPU and 512 MB memory. The performance of an algorithm is evaluated by the percentage relative deviation of the objective value f obtained by this algorithm from the ***lower bound*** (which is obtained by the LR algorithm proposed in [2]), that is, $deviation(\%) = ((f - f_{lowerbound})/f_{lowerbound}) \times 100$.

A. Test Instances

In the experiments, three types of instances are used. The first type of instances are the same ones used in [2], which are derived from the integrated primary steel production process in a major iron and steel enterprise in China. Since these instances are small size ones, we further generated other two types of large size random instances.

1) Type I: Practical Production Data: The structure of these instances is as follows:

1) the number of jobs is chosen from $n \in \{15, 25, 35\}$;
2) the number of stages is chosen form $S \in \{2, 3, 4\}$;
3) the number of machines at each stage is the same and varies at three levels {3, 4, 5}.

Therefore, there are 27 problem scenarios based on different configurations of the above three parameters. For each problem configuration, ten instances are randomly generated, therefore resulting in a total of 270 test problems. Based on practical production data, the processing time of each job is generated uniformly from [30, 50] and the weight value of each job is randomly generated from a uniform distribution [10, 15].

2) Type II—Random Instances With the Same Number of Machines in Each Stage: The structure of this type of problem instances is as follows:

1) the number of jobs is chosen from $N \in \{50, 70, 100\}$;
2) the number of stages is chosen form $S \in \{2, 3, 4\}$;
3) the number of machines at each stage is the same and varies at three levels {3, 4, 5}.

Therefore, there are 27 problem scenarios generated, and for each problem configuration, ten instances are randomly generated, thus resulting in a total of 270 test problems. The processing time of each job is generated uniformly from [1, 20] and the weight value of each job is randomly generated from a uniform distribution [1, 10].

3) Type III—Random Instances With the Random Number of Machines in Each Stage: The structure of problem instances of this type is the same as that in the above Section III-A2, except that the number of jobs is chosen form {20, 40, 60, 100} and that the number of machines in each stage is randomly chosen from {2, 3, 4, 5}. Therefore, there are 120 test problem instances are generated in this type.

B. Parameter Setting

To determine the parameter values used in our PSO, we first randomly select an instance from each test problem configuration of the three types of instances, and then test our PSO algorithm with different kinds of parameter setting on these selected test instances. When determining the value of a parameter (or several parameters in the same kind), we keep other parameters unchanged and do not use the local search procedure so as to amplify the effects of different values of parameters on the PSO algorithm. Based on the obtained results, the size of the population is set to $n_{\mathrm{pop}} = 10$ n, the upper bound and the lower bound of the *inertia* weight are set to $w_{\max} = 0.975$ and $w_{\min} = 0.4$, the maximum consecutive iterations without improvement on the personal best solution for each particle is set to 10, the maximum consecutive iterations without improvement on the global best solution is set to 10 in the level 2 population update method, while in the level 3 population update method it is set to 20, the $threshold\_value$ in the speedup strategy is set to 0.05.

C. Computational Results

To show the effectiveness of the solution representation method and the three-level population update method proposed

TABLE III
COMPUTATIONAL RESULTS OF PRACTICAL PRODUCTION DATA

Problem number	Structures ($n \times S \times m_j$)	Deviation (%)						CPU (seconds)
		PSO_{SPV}	PSO_{POP}	LR	TS	PSO_{VNS}	PSO^*	
1	15 × 2 × 3	1.23	1.12	1.26	1.27	0.89	**0.88**	2.73
2	15 × 2 × 4	1.03	0.83	0.87	1.19	0.67	**0.63**	1.75
3	15 × 2 × 5	0.49	0.41	0.38	0.54	**0.21**	0.22	1.97
4	15 × 3 × 3	2.16	1.76	2.27	1.72	**1.47**	**1.47**	4.85
5	15 × 3 × 4	1.03	0.85	1.33	0.77	**0.67**	**0.67**	4.32
6	15 × 3 × 5	0.63	0.56	0.80	0.72	**0.43**	0.44	4.86
7	15 × 4 × 3	0.85	0.77	1.19	0.71	**0.59**	**0.59**	6.92
8	15 × 4 × 4	1.10	0.96	1.46	0.88	0.68	**0.67**	7.15
9	15 × 4 × 5	0.87	0.71	1.14	0.63	**0.55**	**0.55**	8.95
10	25 × 2 × 3	1.71	1.59	2.05	1.67	1.18	**1.08**	5.95
11	25 × 2 × 4	1.52	1.16	1.37	2.06	0.95	**0.71**	5.84
12	25 × 2 × 5	0.91	0.70	0.80	1.71	0.60	**0.47**	4.96
13	25 × 3 × 3	2.38	2.20	2.53	2.03	1.65	**1.64**	9.36
14	25 × 3 × 4	1.80	1.55	2.16	1.40	1.12	**1.07**	10.49
15	25 × 3 × 5	0.95	0.82	1.13	0.67	0.42	**0.41**	9.51
16	25 × 4 × 3	3.07	2.78	3.62	2.82	2.05	**1.99**	12.12
17	25 × 4 × 4	2.13	1.93	2.59	1.62	1.36	**1.30**	13.79
18	25 × 4 × 5	2.01	1.94	2.50	1.49	1.18	**1.15**	15.78
19	35 × 2 × 3	2.38	2.23	2.92	2.05	1.80	**1.72**	10.34
20	35 × 2 × 4	1.47	1.25	1.44	1.28	0.87	**0.68**	10.44
21	35 × 2 × 5	1.31	1.03	0.88	1.06	0.54	**0.35**	10.18
22	35 × 3 × 3	3.17	2.89	3.47	2.50	2.05	**1.97**	14.77
23	35 × 3 × 4	2.42	2.17	2.66	1.79	1.47	**1.28**	16.33
24	35 × 3 × 5	1.76	1.55	1.87	1.37	1.01	**0.82**	16.64
25	35 × 4 × 3	3.79	3.69	4.38	3.01	2.61	**2.41**	19.12
26	35 × 4 × 4	2.83	2.60	3.05	2.20	1.58	**1.41**	21.25
27	35 × 4 × 5	2.31	2.12	2.51	1.58	1.36	**1.25**	23.82
Average		**1.75**	**1.56**	**1.95**	**1.51**	**1.11**	**1.03**	**10.16**

for the HFS, we tested two kinds of PSO algorithms: one without the population update method (denoted as $\mathrm{PSO_{SPV}}$) and one with the population update method (denoted as $\mathrm{PSO_{POP}}$). To amplify the effectiveness of the population update method, the local search procedure is not adopted in both $\mathrm{PSO_{SPV}}$ and $\mathrm{PSO_{POP}}$.

To show the effectiveness of the integrated adoption of the proposed local search and the three-level population update method, we also developed another kind of PSO. In this PSO, the three-level population update method is also adopted, but the local search only uses the VNS, which is similar to the HVNS_VDS except that it uses the single move but not the compound move in the neighborhood search. In the following, we denote this kind of PSO as $\mathrm{PSO_{VNS}}$, and denote our PSO as PSO^*. Furthermore, since tabu search (denoted as TS) has been successfully applied in many combinatorial optimization problems such as single machine and flowshop scheduling problems ([31], [33]) in the last decade, a TS algorithm is developed as a candidate for comparison with our PSO^*. All these algorithms use the same computation time as that of the LR algorithm in [2], and all the four kinds of PSO algorithms share the same parameter setting. Please note that the LR algorithm can also obtain near optimal feasible solutions for the problem.

1) Computational Results of Type I Instances: The comparison results of algorithms on the type I instances are given in Table III, in which all the values represent the average of performance measures for the ten instances of the corresponding problem configuration. In the column of structures $(n \times S \times m_j)$, n is the number of jobs, S is the number of stages, and m_j is the number of machines in each stage.

From Table III, it can be seen that the $\mathrm{PSO_{SPV}}$ and the $\mathrm{PSO_{POP}}$ algorithms can obtain better solutions than the LR algorithm even without the local search procedure, which shows that our solution representation method is effective for HFS problems. And with the three-level population update method, the $\mathrm{PSO_{POP}}$ outperforms the $\mathrm{PSO_{SPV}}$, which demonstrates the effectiveness of this kind of population update method for PSO. Furthermore, it can be also seen that with the same operation time, TS outperforms the LR algorithm for most instances, and both the $\mathrm{PSO_{VNS}}$ and PSO^* algorithms outperform the LR and the TS for all instances. With the increase of the problem size, the PSO^* algorithm obtains better and better solutions than the TS and $\mathrm{PSO_{VNS}}$, which shows that the proposed local search procedure is effective. Since the average deviation of solutions obtained by the PSO^* algorithm from the lower bounds is 1.03%, these solutions are very close to optimal.

TABLE IV
COMPUTATIONAL RESULTS OF LARGE SIZE INSTANCES WITH SAME NUMBER MACHINES IN EACH STAGE

Problem number	Structures ($n \times S \times m_j$)	Deviation (%)						CPU (seconds)
		PSO_{SPV}	PSO_{POP}	LR	TS	PSO_{VNS}	PSO^*	
1	50 × 2 × 3	4.87	4.25	3.10	3.02	2.47	**2.09**	25
2	50 × 2 × 4	4.03	3.55	2.03	2.54	1.76	**1.52**	26
3	50 × 2 × 5	4.33	3.67	2.24	2.15	1.77	**1.45**	28
4	50 × 3 × 3	8.49	7.59	6.19	6.00	4.76	**4.05**	36
5	50 × 3 × 4	7.51	6.75	5.18	5.05	4.10	**3.44**	38
6	50 × 3 × 5	5.75	5.19	3.04	2.88	2.48	**1.88**	40
7	50 × 4 × 3	9.28	8.50	7.14	7.90	5.86	**4.83**	47
8	50 × 4 × 4	7.12	6.44	5.07	5.06	3.66	**2.93**	49
9	50 × 4 × 5	6.55	5.76	4.04	3.79	2.96	**2.45**	52
10	70 × 2 × 3	4.52	3.98	2.34	2.62	2.03	**1.46**	90
11	70 × 2 × 4	4.32	3.75	2.42	2.95	1.96	**1.62**	94
12	70 × 2 × 5	4.62	3.96	2.86	3.11	2.57	**2.06**	100
13	70 × 3 × 3	7.90	7.08	5.18	5.16	4.18	**3.47**	133
14	70 × 3 × 4	7.61	6.71	4.92	5.15	3.87	**3.22**	139
15	70 × 3 × 5	6.07	5.39	3.41	3.57	2.88	**2.08**	144
16	70 × 4 × 3	9.86	8.92	6.65	6.38	5.61	**4.45**	172
17	70 × 4 × 4	8.91	7.91	6.08	6.03	4.94	**3.94**	179
18	70 × 4 × 5	8.21	7.37	5.40	6.03	4.65	**3.83**	189
19	100 × 2 × 3	4.54	3.75	2.25	2.34	1.82	**1.32**	554
20	100 × 2 × 4	5.14	4.38	2.49	2.75	2.03	**1.53**	568
21	100 × 2 × 5	4.63	3.84	2.18	2.34	1.87	**1.45**	587
22	100 × 3 × 3	7.69	6.78	5.23	5.17	4.20	**3.20**	805
23	100 × 3 × 4	9.05	7.89	6.09	5.60	5.20	**4.13**	835
24	100 × 3 × 5	7.44	6.79	5.12	5.38	4.52	**3.50**	868
25	100 × 4 × 3	10.92	9.84	8.48	9.39	6.99	**5.71**	1064
26	100 × 4 × 4	9.91	8.97	7.51	7.17	6.30	**5.01**	1090
27	100 × 4 × 5	8.87	8.05	5.32	5.49	4.55	**3.33**	1123
Average		**6.97**	**6.19**	**4.52**	**4.63**	**3.70**	**2.96**	**336.11**

2) Computational Results of Type II Instances: Table IV presents the computational results for the $\mathrm{PSO_{spv}}$, $\mathrm{PSO_{pop}}$, LR, TS, $\mathrm{PSO_{VNS}}$ and $\mathrm{PSO^*}$ algorithms using large size random instances with the same number of machines in each stage.

From this table, the following conclusions can be made.

1) With the proposed population update method, the $\mathrm{PSO_{POP}}$ still outperforms the $\mathrm{PSO_{SPV}}$, and it can obtain solutions with 6.19% average deviation from the lower bounds, which shows that this improvement method still works even for large size problems. However, it should be noted that without local search both the $\mathrm{PSO_{spv}}$ and $\mathrm{PSO_{pop}}$ algorithms obtain worse solutions than the LR algorithm.
2) Similar to the results shown in Table III, with the same operation time the $\mathrm{PSO_{VNS}}$ and $\mathrm{PSO^*}$ algorithms outperform the LR and the TS for all instances. Furthermore, the performance of TS is a litter worse than that of the LR algorithm.
3) For all instances, the $\mathrm{PSO^*}$ algorithm obtains better solutions than any other testing algorithm, which also shows the effectiveness of the improvement strategies we proposed for the PSO algorithm. The average deviation of solutions obtained by the $\mathrm{PSO^*}$ algorithm from the lower bounds (2.96%) are also very close to optimal.
4) For most problem instances with the same jobs and stages, the deviation from the lower bounds improves as the number of machines in each stage increases, because having more available machines can reduce the job competition and consequently make the problem easier to be solved.

3) Computational Results of Type III Instances: The performance of the $\mathrm{PSO_{spv}}$, LR, TS, $\mathrm{PSO_{VNS}}$, and $\mathrm{PSO^*}$ algorithms on large size random instances with random number of machines in each stage is shown in Table V. Based on this table, the following conclusions can be made accordingly.

1) The solution representation method and the three-level population update method still works well for the HFS problem.
2) With the same computation time the $\mathrm{PSO_{VNS}}$ and $\mathrm{PSO^*}$ algorithms outperform the LR and the TS algorithms for all instances. And the TS algorithm outperforms the LR algorithm for most instances.

TABLE V
COMPUTATIONAL RESULTS OF LARGE SIZE INSTANCES WITH RANDOM NUMBER MACHINES IN EACH STAGE

Problem number	Structures ($n \times S$)	Deviation (%)						CPU (seconds)
		PSO_{SPV}	PSO_{POP}	LR	TS	PSO_{VNS}	PSO^*	
1	20 × 2	3.85	3.50	3.25	3.22	**2.63**	**2.63**	6
2	20 × 3	6.07	5.68	6.10	5.46	4.95	**4.91**	14
3	20 × 4	7.82	7.27	7.93	6.33	5.82	**5.79**	18
4	40 × 2	5.33	4.26	2.14	2.92	1.62	**1.21**	25
5	40 × 3	8.76	7.87	5.67	4.96	3.97	**3.65**	47
6	40 × 4	10.33	9.53	8.48	7.36	6.35	**5.82**	62
7	60 × 2	5.38	4.69	2.21	1.76	1.61	**1.33**	65
8	60 × 3	9.24	8.55	5.58	5.04	4.52	**3.73**	94
9	60 × 4	11.68	10.85	7.68	6.92	6.02	**5.24**	121
10	100 × 2	5.63	4.82	1.57	1.60	1.11	**0.85**	545
11	100 × 3	9.30	8.50	3.85	2.79	2.59	**1.77**	781
12	100 × 4	11.81	10.94	6.42	4.87	4.46	**3.55**	1043
Average		**7.93**	**7.21**	**5.07**	**4.44**	**3.80**	**3.37**	**235.08**

3) The PSO* algorithm still outperforms any other testing algorithm for all instances. This result again demonstrates the effectiveness of the improvement strategies we proposed for the PSO algorithm.
4) As the number of jobs and the number of stages increase, the deviation and the CPU time increase because the increase in jobs and stages will lead to the increase in the number of operations, therefore making the problem more complicated.

IV. CONCLUSION

In this paper, we take into account the HFS problem with the criteria of minimizing the total weighted completion time. An improved continuous PSO algorithm is proposed to solve this problem. A mapping rule using the SPV rule and a greedy constructive method is developed between the continuous position values of a particle and the complete HFS schedule so that the continuous PSO algorithm can be effectively applied in the discrete HFS problems. A local search that adopts a hybrid simulated annealing incorporating stochastic local search and a hybrid variable neighborhood search incorporating variable depth search, and a three-level population update scheme are incorporated in the PSO algorithm to improve the search intensification and the search diversification, respectively. Numerical results on both practical production instances and randomly generated instances demonstrate that the proposed improvement strategies are effective for PSO algorithm. The results also shows that the improved PSO algorithm can obtain good solutions with comparison to the lower bounds, and this algorithm outperforms other algorithms such as the Lagrangian relaxation algorithm and the tabu search algorithm. Further research may be focused on the application of this method to practical production scheduling problems in the iron and steel industry, and the extension of this method to the multiobjective HFS problems. In the two future researches, the different types of machines in each stage and the machine eligibility constraints can be taken into account for the development of the greedy method to construct a complete schedule.

REFERENCES

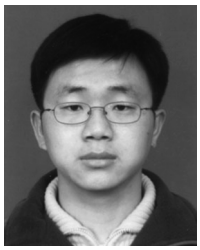
[1] S. A. Brah and J. L. Hunsucker, "Branch and bound algorithm for the flow shop with multiple processors," *Eur. J. Operat. Res.*, vol. 51, pp. 88–99, 1991.
[2] L. X. Tang and H. Xuan, "A new Lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time," *Comput. Operat. Res.*, vol. 33, pp. 3344–3359, 2006.
[3] A. Vignier, "Resolution of some 2-stage hybrid flowshop scheduling problems," in *Proc. IEEE Int. Conf. Syst., Man Cybern., Inf. Intell. Syst.*, vol. 4, pp. 2934–2941.
[4] J. M. Pinto and I. E. Grossmann, "A continuous time MILP model for short term scheduling of batch plants with pre-ordering constraints," *Comput. Chem. Eng.*, vol. 20, pp. 1197–1202, 1996.
[5] P. M. Castro and I. E. Grossmann, "New continuous-time MILP model for the short-term scheduling of multistage batch plants," *Ind. Eng. Chem. Res.*, vol. 44, no. 24, pp. 9175–9190, 2005.
[6] M. Ždánský and J. Poživil, "Combination genetic/tabu search algorithm for hybrid flowshops optimization," in *Proc. Algorithmy, Conf. Scientif. Comput.*, 2002, pp. 230–236.
[7] J. N. D. Gupta, "Two stage, hybrid flowshop scheduling problem," *J. Operat. Res. Soc.*, vol. 39, pp. 359–364, 1988.
[8] R. Linn and W. Zhang, "Hybrid flow shop scheduling: A survey," *Comput. Ind. Eng.*, vol. 37, pp. 57–61, 1999.
[9] O. Moursli and Y. Pochet, "A branch-and-bound algorithm for the hybrid flowshop," *Int. J. Prod. Econom.*, vol. 64, pp. 113–125, 2000.
[10] E. Neron, P. Baptiste, and J. N. D. Gupta, "Solving hybrid flow shop problem using energetic reasoning and global operations," *Omega*, vol. 29, pp. 501–511, 2001.
[11] E. Nowicki and C. Smutnicki, "The flow shop with parallel machines—A tabu search approach," *Int. J. Prod. Res.*, vol. 106, pp. 226–253, 1998.
[12] V. Botta-Genoulaz, "Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness," *Int. J. Prod. Econom.*, vol. 64, pp. 101–111, 2000.
[13] Y. Yang, S. Kreipl, and M. Pinedo, "Heuristics for minimizing total weighted tardiness in flexible flowshops," *J. Scheduling*, vol. 3, pp. 71–88, 2000.
[14] O. Engin and A. Döyen, "A new approach to solve hybrid flow shop scheduling problems by artificial immune system," *Future Generation Comput. Syst.*, vol. 20, pp. 1083–1095, 2004.
[15] H. Allaoui and A. Artiba, "Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints," *Comput. Ind. Eng.*, vol. 47, pp. 431–450, 2004.

[16] R. Ruiz and C. Maroto, "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility," *Eur. J. Oper. Res.*, vol. 169, pp. 781–800, 2006.
[17] A. Bellanger and A. Oulamara, "Scheduling hybrid flowshop with parallel batching machines and compatibilities," *Comput. Oper. Res.*, vol. 36, pp. 1982–1992, 2009.
[18] J. Jungwattanakit, M. Reodechaa, P. Chaovalitwongsea, and F. Werner, "A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria," *Comput. Oper. Res.*, vol. 36, pp. 358–378, 2009.
[19] J. K. Lenstra, K. A. H. G. Rinnooy, and P. Bruker, "Complexity of machine scheduling problems," *Annals Discrete Math.*, vol. 1, pp. 343–362, 1977.
[20] M. F. Tasgetiren, Y. C. Liang, M. Sevkli, and G. Gencyilmaz, "Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem," *Int. J. Prod. Res.*, vol. 44, no. 22, pp. 4737–4754, 2006.
[21] C. J. Liao, C. T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization for flowshop scheduling problems," *Comput. Oper. Res.*, vol. 34, pp. 3099–3111, 2007.
[22] D. Y. Sha and C. Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Comput. Ind. Eng.*, vol. 51, pp. 791–808, 2006.
[23] D. Y. Sha and C. Y. Hsu, "A new particle swarm optimization for the open shop scheduling problem," *Comput. Oper. Res.*, vol. 35, pp. 3243–3261, 2008.
[24] M. F. Ercan and Y. F. Fung, "Performance of particle swarm optimization in scheduling hybrid flow-shops with multiprocessor tasks," in *Proc. Comput. Sci. Its Appl. (ICCSA)*, 2007, pp. 309–318.
[25] M. F. Ercan, "Particle swarm optimization and other metaheuristic methods in hybrid flow shop scheduling problem," in *Particle Swarm Opt.*, A. Lazinica, Ed. Vienna, Austria: InTech Education and Publishing, Jan. 2009, pp. 155–168.
[26] C. T. Tseng and C. J. Liao, "A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks," *Int. J. Prod. Res.*, vol. 46, pp. 4655–4670, 2008.
[27] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.
[28] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Int. Symp. Micro Mach. Human Sci.*, 1995, pp. 39–43.
[29] N. Mladenović and P. Hansen, "Variable neighborhood search," *Comput. Oper. Res.*, vol. 24, pp. 1097–1100, 1997.
[30] P. Hansen and N. Mladenović, "Variable neighborhood search: Principles and applications," *Eur. J. Oper. Res.*, vol. 130, no. 3, pp. 449–467, 2001.
[31] W. Bożejko, J. Grabowski, and M. Wodecki, "Block approach-tabu search algorithm for single machine total weighted tardiness problem," *Comput. Ind. Eng.*, vol. 50, pp. 1–14, 2006.
[32] 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 J. Comput.*, vol. 14, no. 1, pp. 52–67, 2002.
[33] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flowshop problem," *Eur. J. Oper. Res.*, vol. 91, pp. 160–175, 1996.

Lixin Tang (M'09) received the B.Eng. degree in industrial automation, the M.Eng. degree in systems engineering, and the Ph.D. degree in control theory and application from Northeastern University, Shenyang, China, in 1988, 1991, 1996, respectively.

Currently he is a Chair Professor of "the Cheung Kong Scholars Programme of China", and the director of both the Liaoning Key Laboratory of Manufacturing System and Logistics, and the Logistics Institute, Northeastern University, China. His research interests include operations planning, production scheduling, logistics and supply chain management and combinational optimization. He has published a monograph and more than 50 papers in international journals such as *Computers & Operations Research, Computers and Industrial Engineering, European Journal of Operational Research, IEEE Transactions, Industrial & Engineering Chemistry Research, International Journal of Management Science, International Journal of Production Economics, International Journal of Production Research, International Transactions in Operational Research, Journal of the Operational Research Society*.

Dr. Tang was a recipient of the National Natural Science Foundation for Distinguished Young Scholars of China, State Youth Science and Technology Award of China, Outstanding Young Faculty Award Program of the Ministry of Education and Fok Ying Tung Education-Foundation Reward.



Xianpeng Wang received the B.S. degree in materials and control engineering from Shenyang University, Shenyang, China, and the Ph.D. degree in systems engineering from Northeastern University, Shenyang, China, in 2002 and 2007, respectively.

He is currently a lecturer with the Liaoning Key Laboratory of Manufacturing System and Logistics, and the Logistics Institute, Northeastern University, China. His current research interests include production scheduling, modeling and optimization in process industries, decision support systems, and intelligent optimization algorithms. His research was featured in *European Journal of Operational Research, Computers & Operations Research, Journal of the Operational Research Society*, IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, *and International Journal of Advanced Manufacturing Technology*.