INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH
https://doi.org/10.1080/00207543.2020.1788732

Taylor & Francis
Taylor & Francis Group

An improved gravitational search algorithm to the hybrid flowshop with unrelated parallel machines scheduling problem

Cuiwen Cao [a], Yao Zhang[a], Xingsheng Gu[a], Dan Li[b] and Jie Li [b]

[a]Key Laboratory of Advanced Control and Optimization for Chemical Processes, Ministry of Education, East China University of Science and Technology, Shanghai, People's Republic of China; [b]Centre for Process Integration, Department of Chemical Engineering and Analytical Science, The University of Manchester, Manchester, UK

ABSTRACT

The hybrid flowshop scheduling problem with unrelated parallel machines exists in many industrial manufacturers, which is an NP-hard combinatorial optimisation problem. To solve this problem more effectively, an improved gravitational search (IGS) algorithm is proposed which combines three strategies: generate new individuals using the mutation strategy of the standard differential evolution (DE) algorithm and preserve the optimal solution via a greedy strategy; substitute the exponential gravitational constant of the standard gravitational search (GS) algorithm with a linear function; improve the velocity update formula of the standard GS algorithm by mixing an adaptive weight and the global search strategy of the standard particle swarm optimisation (PSO) algorithm. Benchmark examples are solved to demonstrate the proposed IGS algorithm is superior to the standard genetic algorithm, DE, GS, DE with local search, estimation of distribution algorithm and artificial bee colony algorithms. Two more examples from a real-world water-meter manufacturing enterprise are effectively solved.

ARTICLE HISTORY
Received 6 June 2018
Accepted 20 June 2020

KEYWORDS
Improved gravitational search algorithm; hybrid flowshop; unrelated parallel machines; water-meter manufacturing enterprise

1. Introduction

The hybrid flowshop (HFS), also known as the 'multiprocessor' or the 'flexible' flowshop problem (Hoogeveen, Lenstra, and Veltman 1996; Ruiz and Vázquez-Rodríguez 2010; Feng, Zheng, and Xu 2016), is widely used in the industrial fields such as iron and steel production (Li, Huo, and Tang 2011), plastics manufacturing (Baxendale et al. 2020), machinery manufacturing (Liu et al. 2020), semiconductor (Yu, Huang, and Lee 2017) and chemical production (Borisovsky, Eremeev, and Kallrath 2020). In the HFS, parallel machines are allowed to be operated at each stage in the flowshop. The HFS scheduling problems involve the allocation of jobs to machines and the determination of job sequences on a machine, which has been proved to be an NP-hard problem. The state-of-the-art comprehensive reviews on the HFS scheduling can be referred to Ruiz and Vázquez-Rodríguez (2010), Imma, Rainer, and Jose (2010) and Lin and Mitsuo (2018).

Although there are many types of the deterministic HFS scheduling problems, they can be divided into two categories based on whether the parallel machines at a stage are identical or nonidentical (unrelated). The HFS scheduling problem with identical parallel machines is denoted as HFS-IPM where identical parallel machines at a stage are available to perform the same operation (Portmann et al. 1998). The algorithms that could be used to solve the HFS-IPM problems mainly include three types (Ruiz and Vázquez-Rodríguez 2010): exact algorithms (Rao 1970; Gooding, Pekny, and McCroskey 1994; Choi and Lee 2009), deterministic heuristic search algorithms (Gupta 1988; Guinet and Solomon 1996; Kyparisis and Koulamas 2001), and metaheuristics (Xiao et al. 2000; Engin and Döyen 2004; Ying and Lin 2006; Xu et al. 2013). The HFS scheduling problem with unrelated parallel machines is denoted as HFS-UPM which can be further classified into two types based on whether the processing time of a job on a machine is inversely proportional to the velocity of the machine (Lu and Liu 2016) or is unrelated to that on other machines (Figielska 2009). The algorithms that could be used to solve the HFS-UPM scheduling problems also include exact methods, deterministic heuristic search algorithms and metaheuristics. The exact methods including mixed-integer linear

CONTACT Cuiwen Cao caocuiwen@ecust.edu.cn Key Laboratory of Advanced Control and Optimization for Chemical Processes, Ministry of Education, East China University of Science and Technology, Shanghai 200237, People's Republic of China; Jie Li jie.li-2@manchester.ac.uk Centre for Process Integration, Department of Chemical Engineering and Analytical Science, The University of Manchester, Manchester M139PL, UK

Supplemental data for this article can be accessed here. https://doi.org/10.1080/00207543.2020.1788732

programming (MILP) (Meng et al. 2019; Meng et al. 2020; Pei et al. 2020) and Lagrangian relaxation-based algorithms (Brah, Hunsucker, and Shah 1991; Martello, Soumis, and Toth 1997) are suitable for solving small or medium scale problems. They often fail to generate a feasible schedule for large-scale problems as the computational complexity of these algorithms increases exponentially with the problem size. In the deterministic heuristic search algorithms, local search (Piersma and Dijk 1996; Fanjul-Peyro and Ruiz 2010; Ramezanian, Vali-Siar, and Jalalian 2019) and heuristic strategies (Kim and Lee 2009; Yang-Kuei and Chi-Wei 2013) are commonly used. Compared to the exact methods, these heuristic algorithms are much more efficient. However, the deterministic heuristic search algorithms are in fact traversal algorithms under particular strategies, making them still suitable for small-scale problems. The metaheuristics such as simulated annealing (SA) algorithm (Low 2005), genetic algorithm (GA) (Jungwattanakit et al. 2008; Cui, Li, and Zhang 2005; Zhou and Tang 2009; Meng et al. 2019), ant colony optimisation (ACO) algorithm (Lin, Lin, and Hsieh 2013), discrete differential evolution algorithm (Zhang and Chen 2018), differential evolution with local search (DELS) algorithm (Xu and Wang 2011), artificial bee colony (ABC) algorithm (Wang et al. 2012), particle swarm optimisation (PSO) algorithm (Torabi et al. 2013), and fruit fly optimisation algorithm (Zheng and Wang 2016) have been successfully applied for the HFS-UPM scheduling problems.

Besides the aforementioned metaheuristics, the nature-inspired metaheuristic algorithms, also known as adaptive stochastic search strategies, can also be used to deal with practically complicated black-box problems in a simpler manner than many deterministic global optimisation algorithms. Although these nature-inspired metaheuristic techniques can identify local solutions rather than global ones, they are more and more popular due to their high efficiency (Ren et al. 2017). A significant subset of the nature-inspired metaheuristics consists of the so-called swarm intelligence algorithms (Dmitri and Marat 2018). The standard gravitational search (GS) algorithm is a novel swarm intelligence algorithm for nonlinear functions optimisation (Rashedi, Nezamabadi-Pour, and Saryazdi 2009). It has been successfully applied to various research fields such as path planning (Li and Duan 2012), clustering problems (Bahrololoum et al. 2012) and multimodal optimisation (Yazdani, Nezamabadi-pour, and Kamyab 2014; Fan 2014). To address the profit-oriented partial disassembly line balancing problem, Ren et al. (2017) proposed an improved GS algorithm that shows advantages in translating mutually between continuous and integer variables. To the best of our knowledge, very few applications of the GS for the HFS-UPM scheduling problems have been reported. Li et al. (2010) proposed an effective GS algorithm with three strategies including the largest-ranked-value rules, SA-based local search and individual improvement-based local search strategies, and successfully applied them to solve the permutation flowshop scheduling problem, demonstrating the great potential of GS in solving HFS scheduling problem. Therefore, it is worth further investigating the potential of GS in solving HFS-UPM. Inspired by Li et al. (2010) and others work, in this paper we firstly apply the standard GS algorithm for the HFS-UPM scheduling problem to find some room for improvement. Then we develop an improved GS algorithm (IGS) with the combination of three strategies: (1) new individuals are generated and the optimal solution is preserved via greedy strategy based on the mutation strategy of the standard DE algorithm; (2) the linear function from Dai (2014) is adopted instead of the exponential gravitational constant of the standard GS algorithm; (3) the velocity update formula in the standard GS algorithm is improved through mixing the adaptive weight approach of Han, Li, and Wei (2006) and the global search strategy of the standard PSO algorithm. Benchmark HFS-UPM case studies are tested to compare the performance of the proposed IGS algorithm with the standard GA, DE, GS, DELS, EDA, and ABC algorithms. The computational results demonstrate that the IGS algorithm is more effective than others. Finally, two HFS-UPM problems from a real-world water-meter manufacturing enterprise are solved to further illustrate the effectiveness of the proposed IGS algorithm.

This paper is organised as follows: Section 1 is the introduction followed by Section 2 which is the statement of the HFS-UPM problem. Then, Section 3 introduces the standard GS algorithm for the HFS-UPM problem. In Section 4, the standard GS algorithm is improved and the complexity of the IGS algorithm is analysed. After that, Section 5 explores the parameter-settings through orthogonal tests and benchmark cases studies. Following that, Section 6 shows calculation results of the two HFS-UPM problems of a real-world water-meter manufacturing enterprise. Finally, the conclusion is provided in Section 7.

2. The hybrid flowshop with unrelated parallel machines (HFS-UPM)

2.1. Description of the HFS-UPM

Our description of the HFS-UPM scheduling problem is written in the similar way as Cui and Gu's work (Cui and Gu 2015) for the HFS-IPM model. Givens and assumptions are as below.

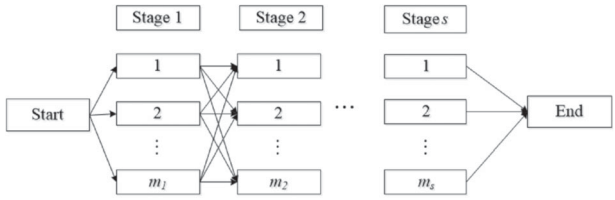

Figure 1. Description of HFS-UPM.

(1) A set of n jobs $\mathbf{J}$, $\mathbf{J} = \{1, 2, \ldots, i, \ldots, n\}$;
(2) A set of s stages $\mathbf{S}$, $\mathbf{S} = \{1, 2, \ldots, j, \ldots, s\}$;
(3) s sets $\mathbf{M}$, $\mathbf{M} = \{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_j, \ldots, \mathbf{M}_s \mid j \in \mathbf{S}\}$, and $\mathbf{M}_j = \{1, 2, \ldots, k, \ldots, m_j \mid m_j \geq 1, j \in \mathbf{S}\}$ is one set of unrelated parallel machines at the stage j and the processing time for each machine is irrelevant.
(4) One or more machines exist at each stage, and at least one stage j must have more than one machine;
(5) All machines are available at all the time without breakdown;
(6) A machine can process at most one job at a time and a job can be processed by at most one machine at a time;
(7) The setup, release, and transportation times of all jobs are negligible;
(8) Preemption is not allowed and the intermediate buffer capacities between two successive stages are unlimited.

There are n jobs to be processed on s stages. Every job has to visit all of the stages in the same order from stage 1 through stage s and is processed by exactly one machine at every stage. The HFS-UPM scheduling problem is to allocate a machine at each stage for each job and determine the sequence of jobs on each machine so as to minimise the makespan, which is the maximum completion time of all jobs at the final stage s. The brief description of the HFS-UPM is shown in Figure 1. Please see Cui and Gu's work (Page 250; Cui and Gu 2015) for a similar simple explanation ($n = 4$, $s = 3$, $m_1 = m_2 = m_3 = 2$) of the problem.

2.2. HFS-UPM mathematical model

In general, there are two formats to represent a solution for HFS problems: the matrix representation and the vector representation. Since the vector representation has been proved to be more effective than the matrix (Cui and Gu 2015), we choose the vector representation in this paper. And we consider the position of the jobs at the first stage to be the elements of the vector representation.

The HFS-UPM model is formulated as follows.

Objective: Our objective is to minimise the makespan $C_{\max}(\boldsymbol{\pi}_s)$.

$$\text{Minimize} \quad C_{\max}(\boldsymbol{\pi}_s) = \max_{i=1,2,\ldots,n} \{C_{\pi_s(i),s,k_s}\} \tag{1}$$

where s is the number of the stage and it also represents the final stage, $\boldsymbol{\pi}_s$ ($= \{\boldsymbol{\pi}_s(1), \boldsymbol{\pi}_s(2), \ldots, \boldsymbol{\pi}_s(i), \ldots, \boldsymbol{\pi}_s(n)\}$) is the sequence of jobs processed at the final stage s, k_s is the number of the unrelated parallel machines at the final stage s; $C_{\pi_s(i),s,k_s}$ ($\in C_{\pi_s(i),s}$) is the completion time of job $\boldsymbol{\pi}_s(i)$ at the final stage s, and $\max_{i=1,2,\ldots,n} \{C_{\pi_s(i),s,k_s}\}$ is the maximum completion time of all jobs at the final stage s.

Subject to:

$$\begin{cases} C_{\pi_1(i),1,k_1} \\ \quad = p_{\pi_1(i),1,k_1} \\ IM_{k_1,1} = C_{\pi_1(i),1,k_1} \end{cases} \qquad i = 1, 2, \ldots m_1; k_1 = 1, 2, \ldots m_1 \tag{2}$$

Equation (2) computes the completion time $C_{\pi_1(i),1,k_1}$ of the job $\boldsymbol{\pi}_1(i)$ ($i = 1, 2, \ldots, m_1$) processed on the machine k_1 ($k_1 \in \mathbf{M}_1$) at the stage one. m_1 is the number of unrelated parallel machines at the stage one. $P_{\pi_1(i),1,k_1}$ is the processing time of the job $\boldsymbol{\pi}_1(i)$ on the machine k_1 at the stage one. $IM_{k_1,1}$ ($\in \mathbf{IM}_1$) is the idle moment of the machine k_1 at the stage one.

$$\begin{cases} NM_1 = \arg\min_{k_1=1,2,\ldots m_1}^{\{IM_{k_1,1}\}} \\ C_{\pi_1(i),1,NM_1} \\ \quad = IM_{NM_1,1} + p_{\pi_1(i),1,NM_1} \\ IM_{NM_1,1} = C_{\pi_1(i),1,NM_1} \end{cases} \qquad \begin{array}{l} i = m_1 + 1, m_1 \\ \quad +2, \ldots, n; k_1 \in \mathbf{M}_1 \end{array} \tag{3}$$

Equation (3) computes the completion time $C_{\pi_1(i),1,NM_1}$ of the job $\boldsymbol{\pi}_1(i)$ ($i = m_1+1, m_1+2, \ldots, n$) processed on the machine NM_1 ($\in \mathbf{NM}$) and allocates the job $\boldsymbol{\pi}_1(i)$ to the corresponding machine NM_1 at the stage one. NM_1 ($\in \mathbf{NM}$) denotes the serial number of the earliest available machine at the stage one, and argmin{}stands for the serial number of the given expression in '{}' attains its minimum value. If there is more than one available machine, we use the rule (Wang et al. 2012) shown in Section 3.1. $IM_{NM_1,1}$ ($\in \mathbf{IM}_1$) is the idle moment of the machine NM_1 at the stage one. This idle moment should be updated after the new completion time $C_{\pi_1(i),1,NM_1}$ being computed.

$$\boldsymbol{\pi}_j = \{g(C_{\pi_{j-1}(i),j-1,k_{j-1}}) \mid i = 1, 2, \ldots n; j = 2, \ldots s; \\ k_{j-1} \in \mathbf{M}_{j-1}\} \tag{4}$$

Equation (4) means the processing sequence of jobs at the stage j ($j = 2, \ldots, s$). $g(C_{\pi_{j-1}(i),j-1,k_{j-1}})$ is the function of sorting the jobs at the stage j based on the ascending order of the completion times at the stage $(j-1)$, where

$C_{\pi_{j-1}(i),j-1,k_{j-1}}$ stands for the completion time of the job $\boldsymbol{\pi}_{j-1}(i)$ at the stage $(j-1)$ processed on the machine k_{j-1} ($\in \mathbf{M}_{(j-1)}$). For the processing sequence at the stage j, we use the first-come-first-service rule (Cui and Gu 2015; Oğuz et al. 2004; Oğuz and Ercan 2005) in which the jobs with the shortest completion time from the previous stage should be scheduled as early as possible.

$$\begin{cases} C_{\pi_j(i),j,k_j} & i = 1, 2, \ldots m_j; \\ \quad = C_{\pi_j(i),j-1,k_{j-1}} + p_{\pi_j(i),j,k_j} & j = 2, 3, \ldots s \\ IM_{k_j,j} = C_{\pi_j(i),j,k_j} & k_j \in \mathbf{M}_j; k_{j-1} \in \mathbf{M}_{j-1} \end{cases} \tag{5}$$

Equation (5) computes the completion time $C_{\pi_j(i),j,k_j}$ of the job $\boldsymbol{\pi}_j(i)$ $(i = 1, 2, \ldots, m_j)$ processed on the machine k_j ($\in \mathbf{M}_j$) at the stage ($j = 2, 3, \ldots, s$). $m_j$stands for the number of unrelated parallel machines at the stage j; $C_{\pi_j(i),j-1,k_{j-1}}$ is the completion time of the job $\boldsymbol{\pi}_j(i)$ at the stage $(j-1)$processed on the previous machine k_{j-1} ($\in \mathbf{M}_{(j-1)}$); $p_{\pi_j(i),j,k_j}$ is the processing time of the job $\boldsymbol{\pi}_j(i)$ on the machine $k_j$at the stage j. $IM_{k_j,j}$ ($\in \mathbf{IM}_j$) is the idle moment of the machine k_j at the stage j.

Equation (6) is similar to Equation (3). It computes the completion time $C_{\pi_j(i),j,NM_j}$ of the job $\boldsymbol{\pi}_j(i)$ $(i = m_j+1, m_j+2, \ldots, n)$ processed on the machine NM_j ($\in \mathbf{NM}$) and allocates the job $\boldsymbol{\pi}_j(i)$ to the corresponding machine NM_j at the stage j ($= 2, 3, \ldots, s$). NM_j ($\in \mathbf{NM}$) computes the serial number of the earliest available machine at the stage j, if there are more than one available machine, we use the same rule as Equation (3). $IM_{NM_j,j}$ ($\in \mathbf{IM}_j$) is the idle moment of the machine NM_j at the stage j. This idle moment should be updated after the new completion time $C_{\pi_j(i),j,NM_j}$ being computed.

$$\begin{cases} NM_j = \arg \min\limits_{k_j=1,2,\ldots m_j} \{IM_{k_j,j}\} & \\ C_{\pi_j(i),j,NM_j} & \\ \quad = \max\limits_{k_{j-1} \in \mathbf{M}_{j-1}} \{C_{\pi_j(i),j-1,k_{j-1}}, IM_{NM_j,j}\} & i = m_j + 1, m_j \\ \quad + p_{\pi_j(i),j,NM_j} & +2, \ldots, n; j = 2, 3, \ldots, s \\ IM_{NM_j,j} = C_{\pi_j(i),j,NM_j} & \end{cases} \tag{6}$$

3. Standard GS algorithm for the UFS-UPM scheduling problem

The standard GS algorithm was originated by Rashedi, Nezamabadi-Pour, and Saryazdi (2009) to solve the nonlinear function optimisation problems in 2009, inspired by the gravitational phenomenon in nature. The main mechanism of this algorithm is that individuals with different qualities are attracted to each other, and the performance of these individuals is determined by their qualities. All these objects attract each other by the gravity forces, and these forces result in a global movement of all objects towards the objects with heavier masses. When this algorithm satisfies the termination condition, the position of the individual with the largest inertia mass is the optimal solution of the problem. In this section, we use the standard GS algorithm to solve the HFS-UPM scheduling problem.

3.1. Encoding method and population initialisation

The encoding method for the standard GS algorithm is the same vector representation mentioned in Section 2.2, and each of the vectors corresponds to the position of an individual. For the HFS-UPM scheduling problem, this vector representation must ensure the feasibility of solutions (Onwubolu and Davendra 2006).

When the population initialisation process begins, we generate the set $\mathbf{X}(0)$ of P_{size} individuals (masses) $\mathbf{X}(0) = \{\mathbf{X}_1(0), \ldots, \mathbf{X}_{i_a}(0), \ldots, \mathbf{X}_{j_a}(0), \ldots, \mathbf{X}_{P_{size}}(0)\}$, where $\boldsymbol{X}_{i_a}(0)(= \{x^1_{i_a}(0), x^2_{i_a}(0), \ldots, x^i_{i_a}(0), \ldots, x^n_{i_a}(0) i_a = 1, 2, \ldots, P_{size}; i \in \boldsymbol{J}\})$ is the initial position information of individual i_a, n is the total number of the jobs to be machined, and $x^i_{i_a}(0)$ presents the initial position of the job i in the individual i_a.

For the standard GS algorithm, the individuals $\boldsymbol{X}_{i_a}(0)(i_a = 1, 2, \ldots, P_{size})$ $(i_a = 1, 2, \ldots, P_{size})$ in the initial population are generated randomly as Equation (7).

$$x^i_{i_a}(0) = rand \cdot (up - down) + down \tag{7}$$

where 'rand' represents a random number between 0 and 1; *up* and *down* represent the upper and the lower bounds respectively. For the HFS-UPM problem, *up* is the number of unrelated parallel machines ($= m_1$) at the first stage, and *down* is an integer with the value 1. This equation ensures the integer part of the position $x^i_{i_a}(0)$ranging from 1 to m_1, that is, every job is processed on an available machine at the first stage after decoding. If the individuals' positions beyond the upper bound or less than the lower bound, we still use Equation (7) to change the position values to feasible values.

The solutions generated by the vector representation stand for the sequence of the jobs at the first stage ($j = 1$; $j \in \mathbf{S}$). For example, there are 10 jobs of the HFS-UPM, the first stage has 3 groups of unrelated parallel machines, and the vector representation is $\underset{job}{\overset{[3.92\ 3.10\ 3.21\ 2.25\ 2.67\ 3.76\ 3.98\ 1.81\ 2.32\ 1.09]}{1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 10}}$. The integer part of each real number stands for the class ID of machines (shown in Table 1), and the fractional part is used to arrange the job sequence by the method described in Section 3.2.

Table 1. Decoding results of vector representation.

Class	1			2				3		
Job	8	10	4	5	9	1	2	3	6	7
Position	1.81	1.09	2.25	2.67	2.32	3.92	3.1	3.21	3.76	3.98
Machine	1	1	2	2	2	3	3	3	3	3
Processing Sequence	2	1	1	3	2	4	1	2	3	5

3.2. Decoding method

For the first stage ($j = 1$; $j \in \mathbf{S}$), the decoding process includes two steps: the allocation of the machines and the sorting of the jobs. Before decoding, the vector representation should be divided into integer part and the fractional part. Those jobs with the same integer part are allocated to the identical machine and the sequence of jobs processed on the same machine depends on the ascending order of the fractional parts.

For the vector representation in Table 1, the decoding steps are as follows:

(1) The jobs with the same integer part should be arranged to the same machine. So, the job 8 and 10; the job 4, 5 and 9; the job 1, 2, 3, 6 and 7 are divided into three classes, they are processed on the first, the second, the third machine respectively.
(2) Sort the jobs processed on the same machine in terms of the ascending order of the fractional parts. In consequence, on the first machine, the processing sequence of the jobs is [10 8]; on the second machine, the order is [4 9 5]; and on the third machine, the order is [2 3 6 1 7]. Table 1 presents the decoding results of the vector representation in Section 3.1.

For the decoding method after the first stage ($j > 1$; $j \in \mathbf{S}$), we adopt the first-come-first-service rule. The decoding steps are as follows:

(1) Selection of the jobs: sort the jobs according to the completion times of the jobs at the previous stage $(j-1)$ from small to large, and then select the jobs in sequence.
(2) Selection of the machines: after selecting a job $\pi_j(i)$ to be processed, the earliest allowable starting time at the stage j on the machine k_j should be considered. It is the maximum time between the release time $(= IM_{k_j,j})$ of the machine k_j and the completion time $(= C_{\pi_{j-1}(i),j-1,k_{j-1}})$ of the selected job at the stage $(j-1)$. If there is more than one available machine $NM_j (\in \mathbf{M}_j)$, we use the rule proposed by Wang, Zhou et al.'s work (2012), that is, calculate the completion time $C_{\pi_j(i),j,NM_j}$ of Equation (6) for each available machine NM_j at the stage j and select the machine with minimum value to process the job $\pi_j(i)$. Finally, update the completion time of the job $\pi_j(i)$ at the stage j and the release time of the chosen machine NM_j.

3.3. The operation of the standard GS algorithm for the UFS-UPM scheduling problem

The operation of the standard GS algorithm mainly follows two laws: Newton law of gravity and Newton's second law. Each individual attracts every other one with a 'gravitational force'. The gravitational force between two individuals is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. In the proposed algorithm, individuals are considered as objects and their performance is measured by their masses. All these objects attract each other by the gravity force, and this force causes a global movement of all objects towards the objects with heavier masses. The heavier masses which correspond to better solutions move more slowly than lighter ones, and this rule guarantees the exploitation step of the algorithm. (Rashedi, Nezamabadi-Pour, and Saryazdi 2009)

In one iteration calculation (expressed as 'at time t'), Equation (8) defines the force action of i-th dimension space (the job i ($\in \mathbf{J}$)) on the mass 'i_a' from the mass 'j_a' ($F^i_{i_a j_a}(t)$).

$$F^i_{i_a j_a}(t) = G(t)\frac{Mp_{i_a}(t) \cdot Ma_{j_a}(t)}{R_{i_a j_a}(t) + \varepsilon}[x^i_{j_a}(t) - x^i_{i_a}(t)] \quad (8)$$

where Ma_{j_a} is the active gravitational mass related to the individual j_a, it is a measure of the strength of the gravitational field due to a particular object; Mp_{i_a} is the passive gravitational mass related to the individual i_a, it is a measure of the strength of an object's interaction with the gravitational field (Rashedi, Nezamabadi-Pour, and Saryazdi 2009); $G(t)$ is the gravitational constant at time t (calculated in Equation (10)), ε is a small positive constant which is used to avoid the denominator becoming zero, and $R_{i_a j_a}(t)$ is the Euclidian distance (calculated in Equation (9)) between the two individuals i_a and j_a.

$$R_{i_a j_a}(t) = ||\mathbf{X}_{i_a}(t), \mathbf{X}_{j_a}(t)||_2 \quad (9)$$

The gravitational constant function $G(t)$ is initialised at the beginning and will be reduced with time to control

the search accuracy. It is computed by Equation (10).

$$G(t) = G_0 \cdot e^{-\alpha \frac{t}{t_{\max}}} \tag{10}$$

where G_0 and α are constants with 100 and 20, respectively. t_{max} is the maximum number of iterations.

Gravitational and inertia masses are computed by the fitness evaluation of optimisation problems. The heavier the mass, the more efficient the individual. This means that better individuals have higher attractions and move more slowly. Rashedi, Nezamabadi-Pour, and Saryazdi (2009) assumed the gravitational masses were equal to inertia masses, and the values of them were computed by using the map of fitness. Equations (11)–(13) update the gravitational and inertial masses.

$$Ma_{i_a}(t) = Mp_{i_a}(t) = M_{i_a}(t), i_a = 1, 2, \cdots, P_{size} \tag{11}$$

$$m_{i_a}(t) = \frac{f_{i_a}(t) - worst(t)}{f_m(t) - worst(t)} \tag{12}$$

$$M_{i_a}(t) = \frac{m_{i_a}(t)}{\sum_{j_a=1}^{P_{size}} m_{j_a}(t)} \tag{13}$$

where $f_{i_a}(t)$ represents the fitness value (is equal to the minimised makespan calculated by Equation (1)) of the individual $i_a$at time t, $f_m(t)$ and $worst(t)$ are defined as the best and the worst fitness values in the population at time t.

To give a stochastic characteristic to the standard GS algorithm, the force that acts on the job i in individual i_a is supposed to be a randomly weighted sum of forces exerted from the other individuals. And to improve the performance of GS algorithm by controlling exploration and exploitation, only K_{best} individuals are chosen to exert the forces to the individual i_a. K_{best} represents the number of individuals whose masses at the time t are relatively larger than those without being chosen and K_{best} is the initial value K_0 at the beginning and decreasing with time. In such a way, at the beginning, all individuals apply the forces, and as time passes, K_{best} is decreased linearly and at the end there will be just one individual applying force to the others. Equation (14) is the formula of the forces acting on the job i in individual i_a.

$$F^i_{i_a}(t) = \sum_{j_a \in Kbest, j_a \neq i_a} rand \cdot F^i_{i_a j_a}(t) \tag{14}$$

where *rand* is a random number between 0 and 1.

By Newton's second law, the acceleration of the object is proportional to the force it receives, and is inversely proportional to its mass. Hence, the acceleration of the job i in individual i_a is defined as Equation (15).

$$a^i_{i_a}(t) = \frac{F^i_{i_a}(t)}{M_{i_a}(t)} \tag{15}$$

Finally, the velocity and position of the individual $i_a$is updated as Equations (16) and (17).

$$v^i_{i_a}(t+1) = rand_{i_a} \cdot v^i_{i_a}(t) + a^i_{i_a}(t) \tag{16}$$

$$x^i_{i_a}(t+1) = x^i_{i_a}(t) + v^i_{i_a}(t+1) \tag{17}$$

3.4. The flowchart of the standard GS algorithm

The flow chart of the standard GS algorithm for solving the HFS-UPM scheduling problem is shown in Figure 2.

4. The improved GS algorithm for the HFS-UPM problem

In the improved GS (IGS) algorithm, the encoding and decoding processes are the same as the standard GS algorithm in Section 3, and it also uses Equation (7) to initialise its population.

4.1. Generate new individuals for the IGS algorithm

For the standard GS algorithm, after population initialisation, it enters the update step by using Equation (10). For the IGS algorithm, we add the following two strategies after population initialisation.

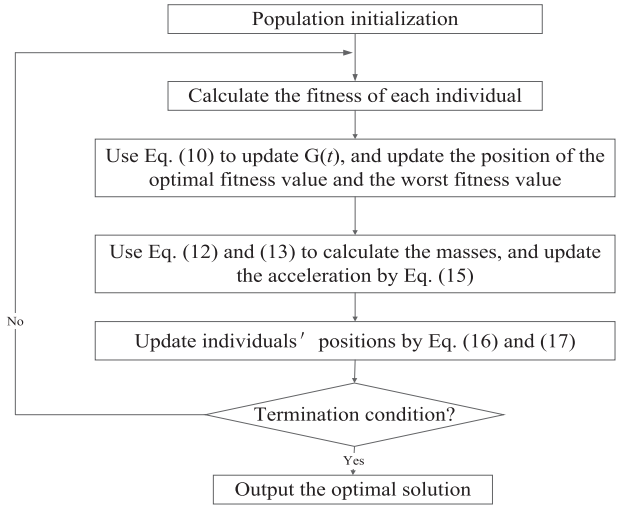

Figure 2. Flow chart of the standard GS algorithm for the HFS-UPM scheduling problem.

4.1.1. Mutation strategy of the standard DE algorithm

The DE algorithm was firstly proposed by Storn and Price (1997). It is a population-based intelligence algorithm with the efficient global search ability. The operators of the DE algorithm consist of mutation, crossover and selection. Mutation strategy is the main novelty and includes two steps which are parent choosing and individual reproduction, that is, each individual in the current generation is allowed to breed through mating with the parent individuals by their difference. To complement the differential mutation search strategy, the DE algorithm then uses a crossover operation to generate the trial individual (Pant, Ali, and Singh 2009).

In order to improve the global search of the standard GS algorithm, the mutation strategy and crossover strategy in the standard DE algorithm are used to generate new individuals at the current time/iteration t to improve the search range and efficiency. The mutation and crossover strategies are presented as Equations (18) and (19), respectively.

$$\mathbf{T}_{i_a}(t) = \mathbf{X}_{r3}(t) + F_{DE} \cdot [\mathbf{X}_{r1}(t) - \mathbf{X}_{r2}(t)] \tag{18}$$

$$\begin{aligned}\mathbf{Trial}_{i_a}(t) = \{&Trial^1_{i_a}(t), Trial^2_{i_a}(t), \cdots, Trial^i_{i_a}(t),\\ &\cdots Trial^n_{i_a}(t) | i_a = 1, 2, \ldots, P_{size}; i \in \mathbf{J}\}\end{aligned} \tag{19}$$

$$Trial^i_{i_a}(t) = \begin{cases} T^i_{i_a}(t) & if \quad rand > CR \\ x^i_{i_a}(t) & otherwise \end{cases}$$

where $\mathbf{T}_{i_a}(t)$ $(= \{T^1_{i_a}(t), T^2_{i_a}(t), \cdots, T^i_{i_a}(t), \cdots, T^n_{i_a}(t) | i_a = 1, 2, \ldots, P_{size}; i \in \mathbf{J}\})$ represents the individual i_a after the mutation strategy carried out; $\mathbf{X}_{r1}(t)$, $\mathbf{X}_{r2}(t)$, and $\mathbf{X}_{r3}(t)$ are three random individuals in the whole population; r_1, r_2, and r_3 are three different random integers among the interval $[1, P_{size}]$. $\mathbf{Trial}_{i_a}(t)$ is the new i_a-th individual after crossover operation via Equations (18) and (19), and $Trial^i_{i_a}(t)$ is the position of the job i in individual i_a. 'rand' is a random number between 0 and 1. CR (the crossover rate) and F_{DE} (the scaling factor) are the key parameters which determine the quality of the new individuals. They are obtained by orthogonal tests (Montgomery 2005) and examples are given in Section 5.1.

4.1.2. Greedy strategy

After the above mutation strategy, the fitness value of the i_a-th individual is calculated, and a greedy strategy is used to obtain the better individuals between the individuals without update and the individuals after update. The strategy is given as Equation (20).

$$\mathbf{X}_{i_a}(t) = \begin{cases} \mathbf{Trial}_{i_a}(t) & \text{if } fitness_{\mathbf{Trial}_{i_a}}(t) < f_{i_a}(t) \\ \mathbf{X}_{i_a}(t) & \text{otherwise} \end{cases} \tag{20}$$

where $fitness_{\mathbf{Trial}_{i_a}}(t)$ and $f_{i_a}(t)$ represent the fitness values of the i_a-th new individual after the greedy strategy and the old one before the mutation and crossover operations, respectively.

4.2. Improved gravitational constant G(t) for the IGS algorithm

The gravitational constant $G(t)$ affects the balance between the global search and the local search of the GS algorithm, which is an important parameter in the GS algorithm. $G(t)$ in the standard GS algorithm is defined as Equation (10), in which it is in the form of exponential function. However, Dai (Dai 2014) found the original descending velocity $G(t)$ is too fast and the GS algorithm is easy to fall into local optima, which is very similar to what we found in the HFS-UPM scheduling problems. Therefore, we use the $G(t)$ function from Dai (Dai 2014) as shown in Equation (21).

$$G(t) = G_0(t) \cdot (1 - t/t_{\max}) \tag{21}$$

where t is the current iteration number and $t_{\max}$ is the maximum number of iterations. The values of $G_0(t)$ are dynamically adjusted according to the positions of the individuals in the whole population (Dai 2014) as given in Equation (22).

$$G_0(t) = \gamma \cdot \max_{i=1,2,\ldots,n} (|x^i_{\max}(t) - x^i_{\min}(t)|) \tag{22}$$

where γ is a constant which is set to 2.5,$x^i_{\max}(t)$ and$x^i_{\min}(t)$ represent the maximum and the minimum position of the job $i(\in \mathbf{J})$among all the individuals in the current population at time t.

4.3. Improved velocity update strategy for the IGS algorithm

For the standard GS algorithm, the velocities of the next generation are affected by the velocities and the accelerations of the current generation, and the accelerations determine the exploration ability of the algorithm to a large extent.

Inspired by the work of Han, Li, and Wei (2006) on the PSO algorithm, we use their adaptive weight method to dynamically regulate the influence of the acceleration on the next generation in the IGS algorithm. The adaptive weight (see Equation (23)) is adjusted according to the

premature convergence status and the fitness values.

$$w(t) = \begin{cases} w_{\max} - (w_{\max} - w_{\min}) \\ \quad \cdot \left| \dfrac{f_{i_a}(t) - f_{avg}{}'(t)}{\Delta} \right| & \text{if } f_{i_a}(t) < f'_{avg}(t) \\ 1.5 - \dfrac{1}{1 + k_{1a} \cdot} & \text{else if } f_{i_a}(t) > f_{avg}(t) \\ \quad \exp(-k_{2a} \cdot \Delta) & \\ 1 & \text{else} \end{cases} \tag{23}$$

The main rules are as follows: (1) if the current individual performs well, the search step length is reduced and the next individual's position is generated near the superior individual to enhance the local search capability; (2) if the current individual's fitness value is poor, then increase the exploration step length in order to jump out of the current area and enhance the global search capability.

In Equation (23), w_{max} and w_{min} represent the maximum and minimum values of the weight in the whole search, and the orthogonal tests (Montgomery 2005) are used to select the maximum and minimum values. $f_{i_a}(t)$ is the fitness of i_a-th individual at t-th iteration. $f_{avg}(t)$ is the average fitness at t-th iteration, $f_m(t)$ is the best fitness at t-th iteration, $f'_{avg}(t)$ is the average fitness among the individuals whose fitness values are better than $f_i(t)$ at t-th iteration. k_{1a} and k_{2a} are both controlling parameters, k_{1a} is used to control the upper bound of $w(t)$ while k_{2a} mainly controls the adjustment ability of the second equation in Equation (23). In this paper, we assume $k_{1a} = 2$, $k_{2a} = 1$ after the primary experiments. Δ is defined as $\Delta = |f_m(t) - f'_{avg}(t)|$, it is proposed to evaluate premature convergence of the whole population. If the individuals distribution is scattered, the value of Δ is larger while $w(t)$ becomes smaller, and its local search ability is enhanced; If the distribution of the individuals is gathered, the value of Δ is smaller while $w(t)$ becomes larger, the exploration of the individuals is strengthened, and thus the individuals are able to jump out of the local optimal. Finally, if $f_{i_a}(t)$ is better than f_{avg} (t) but inferior to$f'_{avg}(t)$, then we define $w(t) = 1$, which is the same parameter value as the standard GS algorithm.

The standard GS algorithm has good searching ability in early stage, but it is easy to fall into the local optimum. In addition, only the position information of the current generation plays the role in its search process, so the standard GS algorithm is also a type of algorithm that lacks memory. Thus, in order to enhance the global exploration capacity in our IGS algorithm, the standard GS algorithm is improved by making reference to the global search operation of the standard PSO algorithm in the middle and late iterations.

Since the position where the global search operation adding will both influence the global and local search abilities, how to choose the proper location is an import task. We assume that the ($p\%{\cdot}t_{\max}$) iteration number is the position to add the global search of the standard PSO algorithm. When the iteration t is less than ($p\%{\cdot}t_{\max}$), we choose Equation (24) to update the velocity, otherwise take Equation (25). The orthogonal tests (Montgomery 2005) are used to choose the value p, and examples are introduced in Section 5.1.

$$v^i_{i_a}(t+1) = rand \cdot v^i_{i_a}(t) + w(t) \cdot a^i_{i_a}(t) \tag{24}$$

$$v^i_{i_a}(t+1) = rand \cdot v^i_{i_a}(t) + w(t) \cdot a^i_{i_a}(t) + c(t) \cdot rand \cdot [bestX^i(t) - x^i_{i_a}(t)] \tag{25}$$

where *rand* represents a random number between 0 and 1,$c(t)$ is an impact factor which determines the influence of the global optimal solution, **bestX**(t) $(= \{bestX^1(t), bestX^2(t), \ldots, bestX^i(t), \ldots, bestX^n(t)\})$ is the best individual among the whole population. In order to keep the local optimisation capacity in early stage and avoid the algorithm falling into local optimum in late stage, $c(t)$ is varied with the current number of iterations and the maximum number of iterations as Equation (26).

$$c(t) = 0.5 + \frac{t^3}{{t_{\max}}^3} \tag{26}$$

From Equation (26) we can find that $c(t)$ is ascending values ranging from 0.5 to 1.5.

4.4. The flowchart of the IGS algorithm

The flow chart of the IGS algorithm for solving the HFS-UPM scheduling problem is shown in Figure 3.

From Figure 3 we can see that, for the strategies proposed in the IGS algorithm, there is no increase in the complexity compared with the standard GS algorithm, and it only increases the amount of computation.

5. Simulation results and comparisons for benchmark case studies

Two benchmark case studies belonging to the HFS-UPM scheduling problems from Cui, Li, and Zhang (2005), Zhou and Tang (2009), Xu and Wang (2011), Wang et al. (2012), and Wang et al. (2012) are solved to illustrate the capability of the proposed IGS algorithm. Case study 1 is a scheduling problem of a refinement processing workshop in a car engine factory with three processes of turner-planer-grinder. The number of unrelated parallel machines is 3, 2, and 4, respectively. Case study 2 is a scheduling problem of a steelmaking factory with four processing steps of steelmaking-refining-continuous casting-rolling. The number of unrelated parallel machines is 3, 3, 2, and 2, respectively. Detailed data

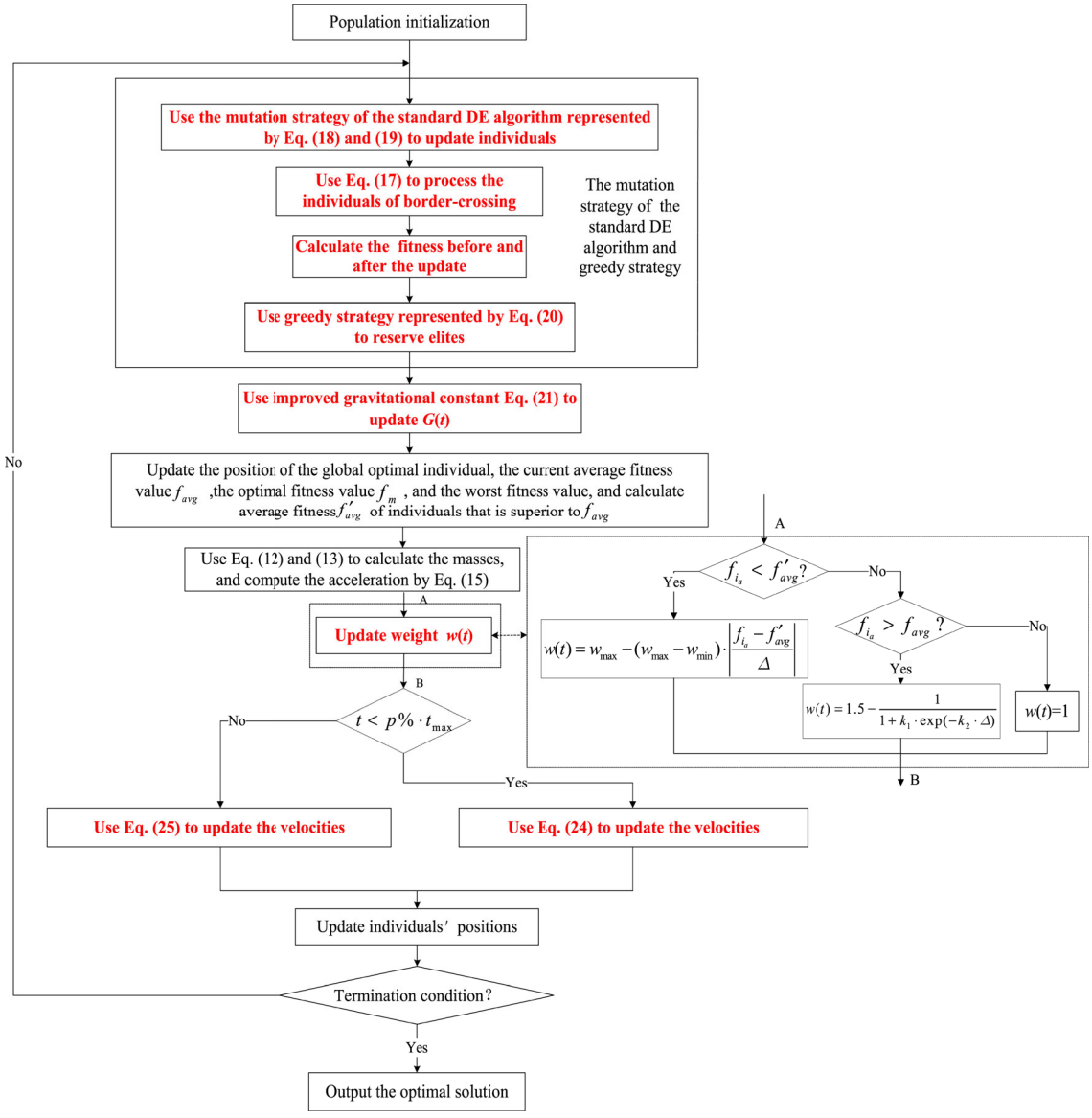

Figure 3. Flow chart of the IGS algorithm for the HFS-UPM scheduling problem.

of the two case studies are shown in Tables S1–S2 in **Supplementary Material 1**. Note that both case studies have been solved using GA (Cui, Li, and Zhang 2005; Zhou and Tang 2009), DELS (Xu and Wang 2011), EDA (Wang et al. 2012) and ABC (Wang et al. 2012) algorithms before.

5.1. Parameters discussion through orthogonal tests

Some parameters in the IGS algorithm have important influences on the optimal solutions. Therefore, it is necessary to obtain proper values of these parameters via a large number of experiments. The key parameters involved in this paper are the population size (P_{size}), the crossover rate (CR) and the scaling factor (F_{DE}) of the DE algorithm, the maximum weight (w_{max}) and the minimum weight (w_{min}) in Section 4.3 (the difference between the maximum weight w_{max} and the minimum weight w_{min} is set as 0.6 in this paper, so we only consider w_{max}). In addition, in order to verify the rationality of the global search strategy's start point of the iterative position ($p\% \cdot t_{\max}$) in Section 4.3, p is also set as a parameter.

We take the case study 1 to thoroughly explain the orthogonal test from Montgomery (2005). After preliminary test, five parameters are set to four levels where P_{size} is set to 20, 30, 40 and 50, CR is set to 0.1, 0.158, 0.2 and 0.3, F_{DE} is set to 0.4, 0.5, 0.6 and 0.7, w_{max} is set to 0.7, 0.8, 0.9 and 1, and p is set to 0, 20, 40 and 60. Then, the experimental design method (Montgomery 2005) is used

Table 2. Orthogonal tests and average fitness statistics (AVG) for case study 1.

	Level					
Group	P_{size}	CR	F_{DE}	w_{max}	p	AVG
1	1	1	1	1	1	24.040
2	1	2	2	2	2	23.960
3	1	3	3	3	3	23.900
4	1	4	4	4	4	24.000
5	2	1	2	3	4	23.960
6	2	2	1	4	3	23.920
7	2	3	4	1	2	23.900
8	2	4	3	2	1	23.880
9	3	1	3	4	2	23.940
10	3	2	4	3	1	23.880
11	3	3	1	2	4	23.920
12	3	4	2	1	3	23.920
13	4	1	4	2	3	23.860
14	4	2	3	1	4	23.880
15	4	3	2	4	1	23.840
16	4	4	1	3	2	23.640

Table 3. Average fitness statistics of the AVG in Table 2 for each level.

Level	P_{size}	CR	F_{DE}	w_{max}	p
1	23.975	23.950	**23.880**	23.935	23.910
2	23.915	23.910	23.920	23.905	**23.860**
3	23.915	23.890	23.900	**23.845**	23.900
4	**23.805**	**23.860**	23.910	23.925	23.940
Range	0.170	0.090	0.040	0.090	0.080
Rank	1	2	5	2	4

to explore the influence of the IGS algorithm on the performance. Table 2 shows the orthogonal testing results for 16 groups of testing samples. Each group has been tested for fifty times with the maximum iteration number of 100 ($t_{max} = 1000$) as the stopping criteria.

The average fitness statistics of the AVG in Table 2 for each level, the ranges, and the ranks are shown in Table 3. The smallest values of the AVG are 23.805, 23.860, 23.880, 23.845 and 23.860 which corresponding to level 4 ($P_{size} = 50$), level 4 ($CR = 0.3$), level 1 ($F_{DE} = 0.4$), level 3 ($w_{max} = 0.9$ ($w_{min} = 0.3$)), and level 2 ($p = 20$), respectively. We also find that the population size P_{size} has the greatest influence on the IGS algorithm, followed by the crossover rate CR and the maximum weight w_{max}. The position parameter p is ranked fourth (when the value of p is 20, the global search strategy is added at the iteration number of $20\% \cdot t_{\max} = 20\% \cdot 1000 = 200$), and the scaling factor F_{DE} is the fifth.

Taking both the optimisation quality and efficiency into account, when the maximum number of iterations (t_{max}) is 1000, we set the five key parameters for the case study 1 as follows: $P_{size} = 50$, $CR = 0.3$, $F_{DE} = 0.4$, $w_{max} = 0.9$ ($w_{min} = 0.3$), $p = 20$.

Similarly, when the maximum number of iterations (t_{max}) is 5000, we set the five key parameters for the case study 2 as follows: $P_{size} = 50$, $CR = 0.158$, $F_{DE} = 0.5$, $w_{max} = 0.7$ ($w_{min} = 0.1$), $p = 30$. If the HFS-UPM scheduling models have different scales as the case study 1 and case study 2, their maximum number of iterations should be retested, and the test standard is to keep the distinctions among the orthogonal test.

5.2. Experimental results and comparisons

To compare the performance of the IGS algorithm with the standard GA (Cui, Li, and Zhang 2005; Zhou and Tang 2009), DELS (Xu and Wang 2011), GS, EDA (Wang et al. 2012) and ABC (Wang et al. 2012) algorithms, we

Table 4. Results and comparison of benchmark case studies 1 and 2 (Unit: Minute).

Algorithm/Case study 1	1	2	3	4	5	6	7	8	9	10	CPU time (s)
GA (Zhou and Tang 2009)	25	25	24	26	25	25	26	25	24	26	5.425
	25	**26**	**25**	**24**	**25**	**26**	**25**	**25**	**25**	**25**	**30.227**
DE	27	27	25	27	26	27	27	27	26	27	16.82
	26	**26**	**27**	**26**	**27**	**25**	**27**	**26**	**27**	**27**	**30.226**
DELS (Xu and Wang 2011)	24	24	24	24	24	25	24	24	24	25	Not found
GS	24	24	24	24	24	24	24	24	24	24	28.217
	24	**24**	**24**	**24**	**24**	**24**	**24**	**23**	**24**	**24**	**30.212**
EDA (Wang et al. 2012)	23	24	23	23	23	23	24	24	23	24	Not found
ABC (Wang et al. 2012)	23	23	23	24	23	23	23	24	23	23	Not found
IGS	23	23	23	23	23	23	23	24	23	23	30.212
	23	**23**	**23**	**23**	**23**	**23**	**23**	**24**	**23**	**23**	**30.212**
Algorithm / **Case study 2**	1	2	3	4	5	6	7	8	9	10	CPU time (s)
GA (Cui, Li, and Zhang 2005)	304	305	308	306	303	304	303	303	300	307	11.734
	306	**307**	**303**	**299**	**304**	**301**	**306**	**304**	**305**	**300**	**74.820**
DE	313	314	307	309	308	300	312	306	308	312	37.262
	310	**308**	**305**	**305**	**306**	**312**	**305**	**306**	**308**	**308**	**74.820**
DELS (Xu and Wang 2011)	313	319	313	302	302	315	315	319	299	299	Not found
GS	299	299	299	298	300	299	298	299	298	300	51.446
	299	**299**	**298**	**300**	**299**	**297**	**299**	**298**	**297**	**299**	**74.821**
EDA (Wang et al. 2012)	297	297	297	297	298	297	297	298	298	298	Not found
ABC (Wang et al. 2012)	297	297	297	297	297	297	297	297	297	297	Not found
IGS	297	297	297	297	297	297	297	297	297	297	74.814
	297	**297**	**297**	**297**	**297**	**297**	**297**	**297**	**297**	**297**	**74.814**

also use them to solve the case studies 1 and 2. Each algorithm runs independently for 10 times. We use the Visual C++ 2017 on a Laptop (Intel(R) Core (TM) i5 2.67 GHz, 6 GB RAM) running on Windows 10. The comparative results are given in Table 4.

For case study 1, we set the maximum number of iterations of 10000 ($t_{max} = 10000$) as the termination condition for the standard GA, DE, DELS, GS, EDA and ABC algorithms. For the IGS algorithm, it can find the same best solutions when t_{max} is 8000, and the results are shown in the upper part of Table 4. In this table, we list the results of GA (Cui, Li, and Zhang 2005; Zhou and Tang 2009), DE, DELS (Xu and Wang 2011), EDA (Wang et al. 2012) and ABC (Wang et al. 2012), in which the results of GA, DE, and GS algorithm are obtained by recoding. Then we set the termination condition of once calculation is the CPU time of 30.212 s (i.e. the time which the IGS algorithm runs case 1 once) and keeping the complete loop for case study 1 and get the results in bold. The Gantt chart of the best solution of IGS for case 1 is shown in Figure 4. The IGS algorithm can find nine optimal solutions in ten independent runs, while the ABC algorithm (Wang et al. 2012) is the best one among GA, DE, DELS, GS, and EDA algorithms, and it find optimal solutions eight times. Because the CPU times of once computing cannot be found in some literatures, we only list the algorithms we have coded.

For case study 2, when t_{max} is 18000 and the other parameters are all the same with the corresponding algorithms, the results are listed in the lower part of Table 4. When the termination condition is the CPU time of 74.814 s (i.e. the time which the IGS algorithm runs case study 2 once) and keeping the complete loop for once calculation, we get the results in bold. The performance of our IGS algorithm is consistent with the ABC algorithm and is superior to others with regard to the number of the best solutions found. The Gantt chart of the optional solution for case study 2 from the IGS algorithm is shown in Figure 5.

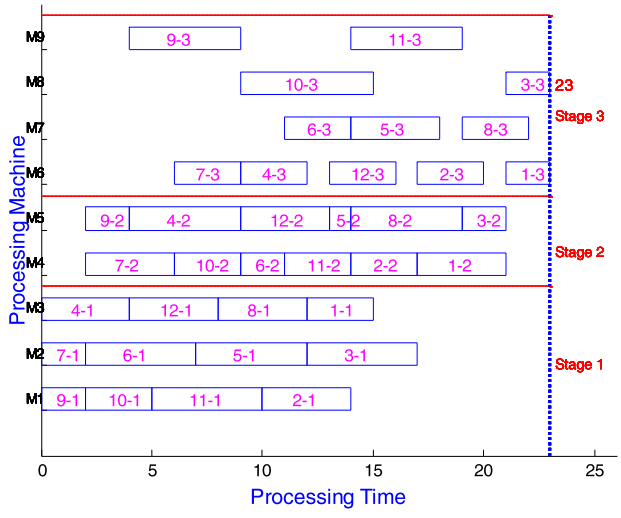

Figure 4. Gantt chart of the best solution of IGS for the benchmark case study 1.

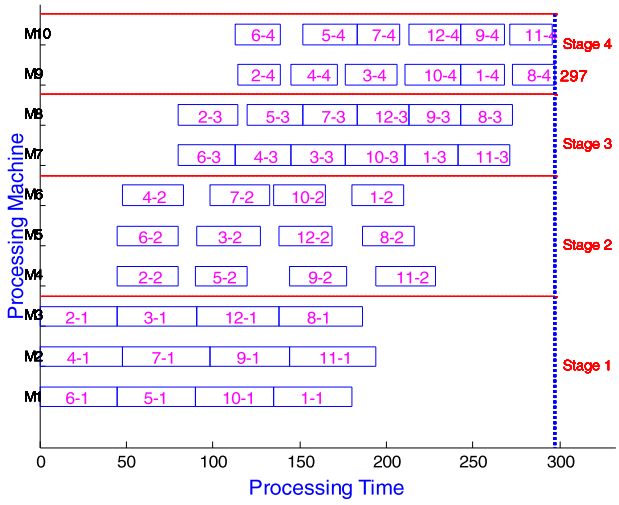

Figure 5. Gantt chart of the best solution of IGS for the benchmark case study 2.

6. Application in a real-world water-meter manufacturing enterprise

In order to enhance the production efficiency and response sensitivity to markets of a real-world water-meter manufacturing enterprise, the proposed IGS algorithm is applied to two workshop production scheduling problems from this enterprise. The case study 1 is composed of four stages called electroplating-printing-assembling-manual processing in sequence. It has 12 jobs with the processing time data given in Table 5. The case study 2 is composed of six stages called crushing-forging-injection-plating-printing-assembling in turn. It involves 50 jobs with the processing time data shown in Table 6. The complete data for the case study 2 is provided in Table S1 in the **Supplementary Material 2** due to its more complexity.

To compare the performance of the proposed IGS algorithm with the standard GA, DE and GS algorithms, the maximum number of iterations (t_{max}) is firstly set as 6000. After 50 independent runs, the computational results are recorded as shown in Tables 7 and 8. Then we set the termination condition of once calculation is the CPU time of 25.362 and 215.390 s (while keeping the

Table 5. Processing time for the real-world case study 1 (Unit: Minute).

	Electroplating			Printing		Assembling			Manual processing	
Job	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
1	68	54	74	52	52	52	61	62	60	67
2	48	49	88	63	57	63	62	49	33	40
3	43	84	50	100	51	97	63	82	98	44
4	99	100	81	30	36	68	49	85	33	93
5	51	88	45	94	78	81	90	38	79	83
6	94	88	69	76	99	66	100	102	95	105
7	66	40	78	85	52	53	53	63	20	59
8	70	66	61	57	49	51	28	22	58	43
9	64	49	52	73	77	47	52	73	59	64
10	79	52	68	62	57	64	73	87	57	61
11	47	40	36	51	37	46	44	91	30	82
12	58	73	66	60	84	72	54	62	48	69

Table 6. Processing time for the real-world case study 2 (Unit: Minute).

	Crushing			Plating			Printing			Assembling		
Job	M1	M2	M3	M14	M15	M16	M17	M18	M19	M20	M21	M22
1	96	94	86	97	96	93	95	90	80	52	57	61
2	95	92	88	95	89	93	85	85	86	57	56	58
3	91	86	92	88	97	85	85	80	81	64	67	61
4	100	91	92	95	90	83	80	100	87	63	67	66
5	97	91	81	98	89	82	86	88	91	59	56	59
46	87	92	92	80	98	95	82	100	95	66	63	64
47	93	99	97	86	93	98	93	95	99	63	57	65
48	94	87	87	99	81	86	99	85	86	63	57	53
49	90	92	93	98	90	87	97	84	82	59	60	60
50	90	89	86	84	99	86	86	96	100	63	66	63

Table 7. Computational results for the real-world case study 1. (Unit: Minute).

	Minimum	Mean	Maximum	Median	Variance	CPU Time (s)
DE	515	529.18	552	529.5	58.43	13.932
	507	**528.76**	**546**	**529.5**	**73.49**	**25.387**
GA	501	512.24	528	510	45.70	4.497
	495	**508.42**	**523**	**508**	**37.31**	**25.383**
GS	482	493.34	502	494	20.30	17.476
	479	**490.96**	**500**	**490.5**	**19.10**	**25.391**
IGS	479	486.82	493	488	10.60	25.362
	479	**486.82**	**493**	**488**	**10.60**	**25.362**

Table 8. Computational results for the real-world case study 2 (Unit: Minute).

	Minimum	Mean	Maximum	Median	Variance	CPU Time (s)
DE	2000	2033.88	2057	2032	161.17	161.710
	2002	**2029.74**	**2047**	**2028**	**104.73**	**215.471**
GA	1989	2003.92	2020	2003	54.93	58.028
	1988	**2002.72**	**2017**	**2003**	**48.86**	**215.434**
GS	1956	1980.78	1998	1980	121.07	109.389
	1952	**1981.66**	**1999**	**1983**	**114.80**	**215.433**
IGS	1948	1966.70	1983	1967	38.17	215.390
	1948	**1966.70**	**1983**	**1967**	**38.17**	**215.390**

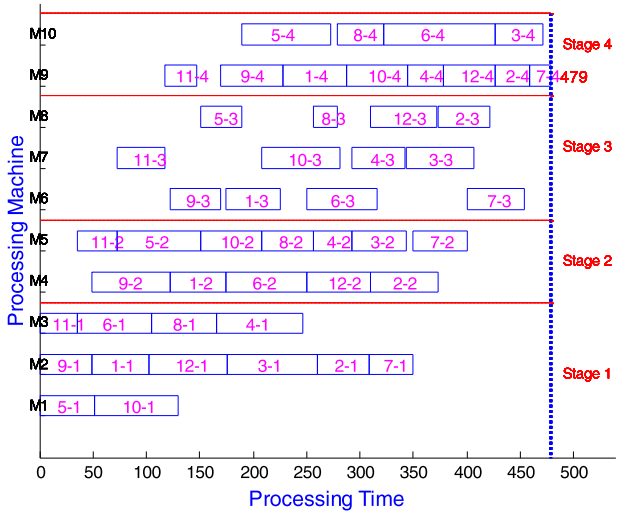

Figure 6. Gantt chart of the best solution of IGS for the real-world case study 1.

complete loop) for the real-world case studies 1 and 2, respectively, and get the results in bold.

It can be seen from Tables 7 and 8 that different makespan and the variance values obtained by the IGS algorithm are all the best among the results obtained by the standard DE, GA and GS algorithms. The Gantt chart of the best solution from the IGS algorithm for the real-world case study 1 and case study 2 is shown in Figures 6 and 7, respectively. According to the mean solutions and variance values, the IGS algorithm is much more stable and efficient.

Compared with the hour-based schedule in real practice, our optimal strategy greatly improves their efficiency and accuracy as our schedule is generated on second

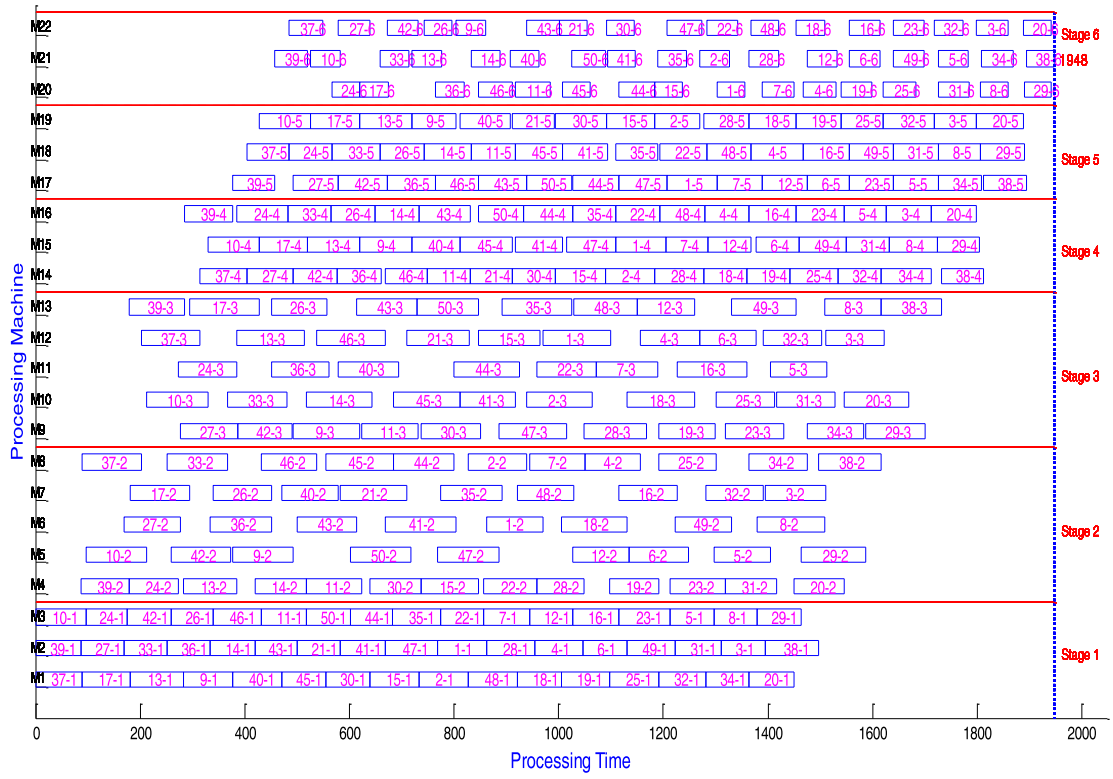

Figure 7. Gantt chart of the best solution of IGS for the real-world case study 2.

basis. The proposed IGS is ready to be directly applied to generate best schedules in real practice once the processing time for each stage is known.

7. Conclusions

The hybrid flowshop with unrelated parallel machines (HFS-UPM) scheduling problem addressed in this paper is an NP-hard problem. In order to solve this type of scheduling problems efficiently, an improved gravitational search (IGS) algorithm was developed. Compared with the standard GA, standard DE, DELS, GS, EDA and ABC algorithms, the IGS shows outstanding computing ability. The IGS mixed three strategies listed as generating new individuals by mutation strategy of the standard DE algorithm and preserve the optimal solution via greedy strategy, substituting the exponential gravitational constant of the standard GS algorithm with a linear function, and improving the velocity update formula of the standard GS algorithm by mixing the adaptive weight approach and the global search strategy of the standard PSO algorithm. After the benchmark and real-world case studies, the IGS algorithm proves to perform better than the other algorithms with regard to the best results finding ability and stability. There are other operational features such as sequence-dependent setup time, machine eligibility constraint, no wait HFS-UPM, and HFS-UPM with blocking which will be addressed using the IGS algorithm in the future.

Notation

Parameters

HFS-UPM Model

i	the serial number of the i-th job
n	the serial number of the final job or the number of jobs
j	the serial number of the j-th stage
s	the serial number of the final stage or the number of stages
k_j	the number of the unrelated parallel machines at stage $j(j \in \mathbf{S})$
m_j	the serial number of the final machine at stage j or the number of unrelated parallel machines at stage $j(j \in \mathbf{S})$
$p_{\pi_1(i),1,k_1}$	the processing time of job $\pi_1(i)$ on the k-th machine at the stage one $(i \in \mathbf{J}, k_1 \in \mathbf{M}_1)$
$p_{\pi_j(i),j,k_j}$	the processing time of job $\pi_j(i)$ on the k-th machine at the stage $j(i \in \mathbf{J}, j \in \mathbf{S}, k_j \in \mathbf{M}_j)$

GS/IGS algorithms

P_{size}	the population size, which means the total number of the individuals
up	the number of machines at the first stage
down	an integer with the value 1
t_{max}	maximum number of iterations
CR	the crossover rate
F_{DE}	the scaling factor
$w_{\max}$	the maximum value of the weight
$w_{\min}$	the minimum value of the weight
i_a *and* j_a	the serial number of the *i*-th and *j*-th individuals in algorithms($i_a \neq j_a$);
r_1, r_2, r_3	three different integers ranging from 1 to P_{size}.
i	the serial number of the *i*-th job ($i \in \mathbf{J}$),(in the original standard GS algorithm, it means *i*-dimension space)
n	the original meaning is the dimension of the search space, here it means the total number of the jobs to be machined.

Sets

HFS-UPM Model

$\mathbf{J}$	the set of jobs, $\mathbf{J} = \{1,2,\ldots,i,\ldots n\}$
$\mathbf{S}$	the set of stages, $\mathbf{S} = \{1,2,\ldots,j,\ldots,s\}$
$\mathbf{M}$	s sets of machines, $\mathbf{M} = \{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_j, \ldots, \mathbf{M}_s \mid j \in S\}$
$\mathbf{M}_j$	the set of machines at the stage j, $\mathbf{M}_j = \{1, 2, \ldots, k_j, \ldots, m_j | 1 \leq m_j < n\}$
$\boldsymbol{\pi}_j$	the processing sequence of jobs at the stage j, $\boldsymbol{\pi}_j = \{\pi_j(1), \pi_j(2), \ldots, \pi_j(i), \ldots, \pi_j(n) | j \in \mathbf{S}\}$
$\boldsymbol{\pi}_1$	the processing sequence of jobs at the stage one, $\boldsymbol{\pi}_1 = \{\pi_1(1), \pi_1(2), \ldots, \pi_1(i), \ldots, \pi_1(n)\}$
$\boldsymbol{\pi}_s$	the processing sequence of jobs at the final stage s, $\boldsymbol{\pi}_s = \{\pi_s(1), \pi_s(2), \ldots, \pi_s(i), \ldots, \pi_s(n)\}$
$\mathbf{C}_{\pi_s(i),s}$	the completion time set of the job $\pi_s(i)$ at the final stage s, $\mathbf{C}_{\pi_s(i),s} = \{C_{\pi_s(1),s,k_s}, C_{\pi_s(2),s,k_s}, \ldots, C_{\pi_s(i),s,k_s}, \ldots, C_{\pi_s(n),s,k_s} | k_s \in \mathbf{M}_s\}$
$\mathbf{IM}_j$	the idle moment set of the machine k_j at the stage j, $\mathbf{IM}_j = \{IM_{1,j}, IM_{2,j}, \ldots, IM_{k_j,j}, \ldots, IM_{m_j,j} | j \in \mathbf{S}\}$
$\mathbf{NM}$	the serial numbers of the available machines that can be used at the earliest at each stage, $\mathbf{NM} = \{NM_j | j \in \mathbf{S}\}$

GS/IGS algorithms

$\mathbf{J}$	the set of jobs, $\mathbf{J} = \{1,2,\ldots,i, \ldots, n\}$
$\mathbf{X}(t)$	the set of all the individuals' position information, $\mathbf{X}(t) = \{\mathbf{X}_1(t), \ldots, \mathbf{X}_{i_a}(t), \ldots, \mathbf{X}_{j_a}(t), \ldots, \mathbf{X}_{P_{size}}(t)\}$
$\mathbf{V}(t)$	the set of all the individuals' velocity information, $\mathbf{V}(t) = \{\mathbf{V}_1(t), \ldots, \mathbf{V}_{i_a}(t), \ldots, \mathbf{V}_{j_a}(t), \ldots, \mathbf{V}_{P_{size}}(t)\}$
$\mathbf{X}_{i_a}(t)$	the position information of individual ia at time t, $\mathbf{X}_{i_a}(t) = \{x^1_{i_a}(t), x^2_{i_a}(t), \cdots, x^i_{i_a}(t), \cdots x^n_{i_a}(t) | i_a = 1, 2, \ldots, P_{size}; i \in \mathbf{J}\}$
$\mathbf{V}_{i_a}(t)$	the velocity information of individual ia at time t, $\mathbf{V}_{i_a}(t) = \{v^1_{i_a}(t), v^2_{i_a}(t), \ldots, v^i_{i_a}(t), \ldots v^n_{i_a}(t) | i_a = 1, 2, \ldots, P_{size}; i \in \mathbf{J}\}$
$\mathbf{T}_{i_a}(t)$	new individuals generated by the mutation strategy of the standard DE algorithm at time t, $\mathbf{T}_{i_a}(t) = \{T^1_{i_a}(t), T^2_{i_a}(t), \cdots, T^i_{i_a}(t), \cdots, T^n_{i_a}(t) | i_a = 1, 2, \ldots, P_{size}; i \in \mathbf{J}\}$
$\mathbf{Trial}_{i_a}(t)$	new individuals after crossover operation at time t, $\mathbf{Trial}_{i_a}(t) = \{Trial^1_{i_a}(t), Trial^2_{i_a}(t), \cdots, Trial^i_{i_a}(t), \cdots, Trial^n_{i_a}(t) | i_a = 1, 2, \ldots, P_{size}; i \in \mathbf{J}\}$
$\mathbf{X}_{r_1}(t), \mathbf{X}_{r_2}(t), \mathbf{X}_{r_3}(t)$	three different random individuals in the whole population at time t, $\mathbf{X}_{r_1}(t) = \{x^1_{r_1}(t), x^2_{r_1}(t), \cdots, x^i_{r_1}(t), \cdots, x^n_{r_1}(t)\}$, $\mathbf{X}_{r_2}(t) = \{x^1_{r_2}(t), x^2_{r_2}(t), \cdots, x^i_{r_2}(t), \cdots, x^n_{r_2}(t)\}$, $\mathbf{X}_{r_3}(t) = \{x^1_{r_3}(t), x^2_{r_3}(t), \cdots, x^i_{r_3}(t), \cdots, x^n_{r_3}(t)\}$
$\mathbf{bestX}(t)$	the best individual in the whole population at time t, $\mathbf{bestX}(t) = \{bestX^1(t), bestX^2(t), \ldots, bestX^i(t), \ldots, bestX^n(t)\}$

Variables

HFS-UPM Model

$IM_{k_j,j}$	the idle time of the machine k_j at the stage j ($k_j \in \mathbf{M}_j, j \in \mathbf{S}$)
NMj	the serial number of the available machines that can be used at the earliest at the stage j ($j \in \mathbf{S}$)
$C_{\pi_1(i),1,k_1}$	the completion time of the job $\pi_1(i)(i \in \mathbf{J})$ on the machine $k_1(k_1 \in \mathbf{M}_1)$ at the stage one
$C_{\pi_j(i),j,k_j}$	the completion time of the job $\pi_j(i)(i \in \mathbf{J})$ on the machine $k_j(k_j \in \mathbf{M}_j)$at the stage j($j \in \mathbf{S}$)
$C_{\pi_{j-1}(i),j-1,k_{j-1}}$	the completion time of the job $\pi_{j-1}(i)(i \in \mathbf{J})$ on the machine $k_{j-1}(k_{j-1} \in \mathbf{M}_{j-1})$ at the stage (*j*-1)($j = 2, \ldots, s$)

$C_{\pi_j(i),j-1,k_{j-1}}$	the completion time of the job $\pi_j(i)(i \in \mathbf{J})$ on the machine $k_{j-1}(k_{j-1} \in \mathbf{M}_{j-1})$ at the stage $(j\text{-}1)(j = 2, \ldots, s)$

GS/IGS algorithms

t	current time/iteration
$x_{i_a}^i(t)$	the position of the job i in the individual ia at time t
$x_{j_a}^i(t)$	the position of the job i in the individual ja at time t
$Ma_{j_a}(t)$	the active gravitational mass of the individual ja at time t
$Mp_{i_a}(t)$	the passive gravitational mass of the individual ia at time t
$M_{i_a}(t)$	the inertia mass of the individual ia at time t
$F_{i_a j_a}^i(t)$	the gravitation of the job i between individual ia and individual ja at time t
$F_{i_a}^i(t)$	the whole gravitation of the job i in the individual ia from the other individuals at time t
$R_{i_a j_a}(t)$	Euclidean distance between individual ia and individual ja at time t
$a_{i_a}^i(t)$	the acceleration of the job i in the individual ia at time t
$worst(t)$	the worst fitness value in the population at time t
$G(t)$	gravitational constant at time t
$G_0(t)$	the parameter in gravitational constant $G(t)$ at time t
$Trial_{i_a}^i(t)$	the position of the job i in the newly generated individual ia after crossover operation at time t
$T_{i_a}^i(t)$	the position of the job i in the individual ia generated by the mutation strategy of standard DE algorithm at time t
$x_{\max}^i(t)$	the maximum position of the job i among all the individuals at time t
$x_{\min}^i(t)$	the minimum position of the job i among all the individuals at time t
$w(t)$	the adaptive weight at time t
$f_{i_a}(t)$	the fitness of ia-th individual at time t
$f_{avg}(t)$	the average fitness value among the whole individuals at time t
$f_m(t)$	the best fitness value among the whole individuals at time t
$f_{avg}'(t)$	the average fitness value among the individuals whose fitness values are better than $f_{i_a}(t)$
$fitness_{\mathbf{Trial}_{i_a}}(t)$	the fitness value of the ia-th new individual after greedy strategy
P	the position to add the global search of PSO algorithm
$c(t)$	the impact factor which determines the influence of the global optimal solution at time t

Acknowledgements

This work was supported by the National Nature Science Foundation of China [grant numbers 61673175, 61973120, 61773165, 61603139]; and the Fundamental Research Funds for the Central Universities [grant number 222201717006].

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the National Nature Science Foundation of China [grant numbers 61673175, 61973120, 61773165, 61603139]; and the Fundamental Research Funds for the Central Universities [grant number 222201717006].

ORCID

Cuiwen Cao http://orcid.org/0000-0003-0662-217X
Jie Li http://orcid.org/0000-0001-5196-2136

References

Bahrololoum, A., H. Nezamabadi-Pour, H. Bahrololoum, and M. Saeed. 2012. "A Prototype Classifier Based on Gravitational Search Algorithm." *Applied Soft Computing* 12 (2): 819–825.

Baxendale, M., J. M. McGree, A. Bellette, and P. Corry. 2020. "Machine-based Production Scheduling for Rotomoulded Plastics Manufacturing." *International Journal of Production Research*, doi:10.1080/00207543.2020.1727046.

Borisovsky, P., A. Eremeev, and J. Kallrath. 2020. "Multi-product Continuous Plant Scheduling: Combination of Decomposition, Genetic Algorithm, and Constructive Heuristic." *International Journal of Production Research* 58 (9): 2677–2695.

Brah, S., J. Hunsucker, and J. Shah. 1991. "Mathematical Modeling of Scheduling Problems." *Journal of Information and Optimization Sciences* 12 (1): 113–137.

Choi, H. S., and D. H. Lee. 2009. "Scheduling Algorithms to Minimize the Number of Tardy Jobs in Two-Stage Hybrid Flow Shops." *Computers & Industrial Engineering* 56 (1): 113–120.

Cui, Z., and X. S. Gu. 2015. "An Improved Discrete Artificial bee Colony Algorithm to Minimize the Makespan on Hybrid Flow Shop Problems." *Neurocomputing* 148: 248–259.

Cui, J., T. Li, and W. Zhang. 2005. "Hybrid Flowshop Scheduling Model and its Genetic Algorithm." [In Chinese.] *Journal of University of Science & Technology Beijing* 27 (5): 623–626.

Dai, J. 2014. "The Improved and Applied Research of Gravitational Search Algorithm." [In Chinese.] Master diss., Jiang Nan University.

Dmitri, E. K., and S. M. Marat. 2018. "Metaheuristic vs. Deterministic Global Optimization Algorithms: The Univariate Case." *Applied Mathematics and Computation* 318: 245–259.
Engin, O., and A. Döyen. 2004. "A New Approach to Solve Hybrid Flow Shop Scheduling Problems by Artificial Immune System." *Future Generation Computer Systems* 20 (6): 1083–1095.
Fan, W. F. 2014. "Analysis and Improvement of the Gravitational Search Algorithm." [In Chinese.] Master diss., Guangdong University of Technology.
Fanjul-Peyro, L., and R. Ruiz. 2010. "Iterated Greedy Local Search Methods for Unrelated Parallel Machine Scheduling." *European Journal of Operational Research* 207 (1): 55–69.
Feng, X., F. Zheng, and Y. Xu. 2016. "Robust Scheduling of a Two-Stage Hybrid Flow Shop with Uncertain Interval Processing Times." *International Journal of Production Research* 54 (12): 3706–3717.
Figielska, E. 2009. "A Genetic Algorithm and a Simulated Annealing Algorithm Combined with Column Generation Technique for Solving the Problem of Scheduling in the Hybrid Flowshop with Additional Resources." *Computers & Industrial Engineering* 56 (1): 142–151.
Gooding, W. B., J. F. Pekny, and P. S. McCroskey. 1994. "Enumerative Approaches to Parallel Flowshop Scheduling via Problem Transformation." *Computers & Chemical Engineering* 18 (10): 909–927.
Guinet, A. G. P., and M. M. Solomon. 1996. "Scheduling Hybrid Flowshops to Minimize Maximum Tardiness or Maximum Completion Time." *International Journal of Production Research* 34 (6): 1643–1654.
Gupta, J. N. D. 1988. "Two-stage, Hybrid Flowshop Scheduling Problem." *Journal of the Operational Research Society* 39 (4): 359–364.
Han, J. H., Z. R. Li, and Z. C. Wei. 2006. "Adaptive Particle Swarm Optimization Algorithm and Simulation." [In Chinese.] *Journal of System Simulation* 10 (18): 2969–2971.
Hoogeveen, J. A., J. K. Lenstra, and B. Veltman. 1996. "Preemptive Scheduling in a Two-Stage Multiprocessor Flow Shop is NP-Hard." *European Journal of Operational Research* 89 (1): 172–175.
Imma, R., L. Rainer, and M. F. Jose. 2010. "Review and Classification of Hybrid Flow Shop Scheduling Problems From a Production System and a Solutions Procedure Perspective." *Computers & Operations Research* 37: 1439–1454.
Jungwattanakit, J., M. Reodecha, P. Chaovalitwongse, and F. Werner. 2008. "Algorithms for Flexible Flow Shop Problems with Unrelated Parallel Machines, Setup Times, and Dual Criteria." *The International Journal of Advanced Manufacturing Technology* 37 (3): 354–370.
Kim, H. W., and D. H. Lee. 2009. "Heuristic Algorithms for re-Entrant Hybrid Flow Shop Scheduling with Unrelated Parallel Machines." *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 223 (4): 433–442.
Kyparisis, G. J., and C. Koulamas. 2001. "A Note on Weighted Completion Time Minimization in a Flexible Flow Shop." *Operations Research Letters* 29 (1): 5–11.
Li, P., and H. Duan. 2012. "Path Planning of Unmanned Aerial Vehicle Based On Improved Gravitational Search Algorithm." *Science China Technological Sciences* 55: 2712–2719.
Li, L., J. Huo, and O. Tang. 2011. "A Hybrid Flowshop Scheduling Problem for a Cold Treating Process in Seamless Steel Tube Production." *International Journal of Production Research* 49 (15): 4679–4700.
Li, X., J. Wang, J. Zhou, and M. Yin. 2010. "An Effective GSA Based Memetic Algorithm for Permutation Flow Shop Scheduling." *Evolutionary Computation* 37: 1–6.
Lin, C. W., Y. K. Lin, and H.-T. Hsieh. 2013. "Ant Colony Optimization for Unrelated Parallel Machine Scheduling." *The International Journal of Advanced Manufacturing Technology* 67 (1-4): 35–45.
Lin, L., and G. Mitsuo. 2018. "Hybrid Evolutionary Optimization with Learning for Production Scheduling: State-of-the-art Survey on Algorithms and Applications." *International Journal of Production Research* 56 (1-2): 193–223. doi:10.1080/00207543.2018.1437288.
Liu, M., X. Liu, F. Chu, F. Zheng, and C. Chu. 2020. "Profit-oriented Distributionally Robust Chance Constrained Flowshop Scheduling Considering Credit Risk." *International Journal of Production Research* 58 (8): 2527–2549.
Low, C. 2005. "Simulated Annealing Heuristic for Flow Shop Scheduling Problems with Unrelated Parallel Machines." *Computers & Operations Research* 32 (8): 2013–2025.
Lu, X. R., and Z. H. Liu. 2016. "An Optimal Online Algorithm for Fractional Scheduling on Uniform Machines with Three Hierarchies." *Journal of Systems Science and Complexity* 29 (6): 1650–1657.
Martello, S., F. Soumis, and P. Toth. 1997. "Exact and Approximation Algorithms for Makespan Minimization on Unrelated Parallel Machines." *Discrete Applied Mathematics* 75 (2): 169–188.
Meng, L., C. Zhang, X. Shao, Y. Ren, and C. Ren. 2019. "Mathematical Modelling and Optimisation of Energy-Conscious Hybrid Flow Shop Scheduling Problem with Unrelated Parallel Machines." *International Journal of Production Research* 57 (4): 1119–1145.
Meng, L., C. Zhang, X. Shao, B. Zhang, Y. Ren, and W. Lin. 2020. "More MILP Models for Hybrid Flow Shop Scheduling Problem and its Extended Problems." *International Journal of Production Research.* doi:10.1080/00207543.2019.1636324.
Montgomery, D. C. 2005. *Design and Analysis of Experiments*. Hoboken: John Wiley & Sons.
Oğuz, C., and M. F. Ercan. 2005. "A Genetic Algorithm for Hybrid Flow-Shop Scheduling with Multiprocessor Tasks." *Journal of Scheduling* 8 (4): 323–351.
Oğuz, C., Y. Zinder, A. Janiak, and M. Lichtenstein. 2004. "Hybrid Flow-Shop Scheduling Problems with Multiprocessor Task Systems." *European Journal of Operational Research* 152 (1): 115–131.
Onwubolu, G., and D. Davendra. 2006. "Scheduling Flow Shops Using Differential Evolution Algorithm." *European Journal of Operational Research* 171 (2): 674–692.
Pant, M., M. Ali, and V. P. Singh. 2009. "Parent-centric Differential Evolution Algorithm for Global Optimization Problems." *Opsearch* 46 (2): 153–168.
Pei, Z., X. Zhang, L. Zheng, and M. Wan. 2020. "A Column Generation-Based Approach for Proportionate Flexible two-Stage no-Wait job Shop Scheduling." *International Journal of Production Research* 58 (2): 487–508.
Piersma, N., and V. W. Dijk. 1996. "A Local Search Heuristic for Unrelated Parallel Machine Scheduling with Efficient

Neighborhood Search." *Mathematical and Computer Modelling* 24 (9): 11–19.

Portmann, M. C., A. Vignier, D. Dardilhac, and D. Dezalay. 1998. "Branch and Bound Crossed with GA to Solve Hybrid Flowshops." *European Journal of Operational Research* 107 (2): 389–400.

Ramezanian, R., M. M. Vali-Siar, and M. Jalalian. 2019. "Green Permutation Flowshop Scheduling Problem with Sequence-Dependent Setup Times: A Case Study." *International Journal of Production Research* 57 (10): 3311–3333.

Rao, T. B. K. 1970. "Sequencing in the Order A, B, with Multiplicity of Machines for a Single Operation." *Journal of the Operational Research Society of India* 7: 135–144.

Rashedi, E., H. Nezamabadi-Pour, and S. Saryazdi. 2009. "GSA: A Gravitational Search Algorithm." *Information Sciences* 179 (13): 2232–2248.

Ren, Y., D. Yu, C. Zhang, G. Tian, L. Meng, and X. Zhou. 2017. "An Improved Gravitational Search Algorithm for Profit-Oriented Partial Disassembly Line Balancing Problem." *International Journal of Production Research* 55 (24): 7302–7316.

Ruiz, R., and J. A. Vázquez-Rodríguez. 2010. "The Hybrid Flow Shop Scheduling Problem." *European Journal of Operational Research* 205 (1): 1–18.

Storn, R., and K. Price. 1997. "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces." *Journal of Global Optimization* 11 (4): 341–359.

Torabi, S. A., N. Sahebjamnia, S. A. Mansouri, and M. A. Bajestani. 2013. "A Particle Swarm Optimization for a Fuzzy Multi-Objective Unrelated Parallel Machines Scheduling Problem." *Applied Soft Computing* 13 (12): 4750–4762.

Wang, S. Y., L. Wang, Y. Xu, and G. Zhou. 2012. "An Estimation of Distribution Algorithm for Solving Hybrid Flow-Shop Scheduling Problem." [In Chinese.] *Acta Automatica Sinica* 38 (3): 437–443.

Wang, L., G. Zhou, Y. Xu, and S. Y. Wang. 2012. "An Artificial Bee Colony Algorithm for Solving Hybrid Flow-Shop Scheduling Problem with Unrelated Parallel Machines." [In Chinese.] *Control Theory & Applications* 29 (12): 1551–1556.

Xiao, W., P. Hao, S. Zhang, and X. Xu. 2000. "Hybrid Flow Shop Scheduling Using Genetic Algorithms." In *Proceedings of the 3rd World Congress on Intelligent Control and Automation*. Vol. 1, 537–541. Hefei, China: IEEE.

Xu, Y., and L. Wang. 2011. "Differential Evolution Algorithm for Hybrid Flow-Shop Scheduling Problems." *Journal of Systems Engineering and Electronics* 22 (5): 794–798.

Xu, Y., L. Wang, S. Wang, and M. Liu. 2013. "An Effective Shuffled Frog-Leaping Algorithm for Solving the Hybrid Flow-Shop Scheduling Problem with Identical Parallel Machines." *Engineering Optimization* 45 (12): 1409–1430.

Yang-Kuei, L., and L. Chi-Wei. 2013. "Dispatching Rules for Unrelated Parallel Machine Scheduling with Release Dates." *The International Journal of Advanced Manufacturing Technology* 67 (1): 269–279.

Yazdani, S., H. Nezamabadi-pour, and S. Kamyab. 2014. "A Gravitational Search Algorithm for Multimodal Optimization." *Swarm and Evolutionary Computation* 14: 1–14.

Ying, K. C., and S. W. Lin. 2006. "Multiprocessor Task Scheduling in Multistage Hybrid Flow-Shops: An Ant Colony System Approach." *International Journal of Production Research* 44 (16): 3161–3177.

Yu, J., R. Huang, and D. Lee. 2017. "Iterative Algorithms for Batching and Scheduling to Minimise the Total Job Tardiness in Two-Stage Hybrid Flow Shops." *International Journal of Production Research* 55 (11): 3266–3282.

Zhang, X., and L. Chen. 2018. "A Re-Entrant Hybrid Flow Shop Scheduling Problem with Machine Eligibility Constraints." *International Journal of Production Research* 56 (16): 5293–5305.

Zheng, X. L., and L. Wang. 2016. "A Two-Stage Adaptive Fruit Fly Optimization Algorithm for Unrelated Parallel Machine Scheduling Problem with Additional Resource Constraints." *Expert Systems with Applications* 65: 28–39.

Zhou, H. R., and W. S. Tang. 2009. "Optimize Flexible Flow-Shop Scheduling Using Genetic Algorithm." [In Chinese.] *Computer Engineering & Applications* 45 (30): 224–227.