COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 05/06 pp 292–302
Volume 4, Number 3, September 2024
DOI: 10.23919/CSMS.2024.0014

An Adaptive Shuffled Frog-Leaping Algorithm for Hybrid-Flow Shop Scheduling with No Precedence Between Some Stages

Zhenghui Yin, Deming Lei*, and Bo Yang

Abstract: Hybrid flow shop scheduling problem (HFSP) has been extensively considered, however, some real-life conditions are seldom investigated. In this study, HFSP with no precedence between some stages is solved and an adaptive shuffled frog-leaping algorithm (ASFLA) is developed to optimize makespan. A new solution representation and a decoding procedure are presented, an adaptive memeplex search and dynamical population shuffling are implemented together. Many computational experiments are implemented. Computational results prove that the new strategies of ASFLA are effective and ASFLA is very competitive in solving HFSP with no precedence between some stages.

Key words: hybrid-flow shop scheduling; shuffled frog-leaping algorithm; precedence

1 Introduction

Hybrid flow shop (HFS) consists of at least two stages, and at least one stage is composed of some parallel machines. The main advantages of hybrid flow shop are the effective balance among machine workload and increasing production capacity. Hybrid flow shop possesses a variety of complex conditions and constraints such as no-wait, batch processing, uncertainty, reentrant, multiprocessor task, sequence dependent setup time, blocking, energy-efficient, and preventive maintenance. Hybrid flow shop scheduling problem (HFSP) with these complex conditions and constraints has attracted much attention, and many results are obtained[1–14].

For HFSP with batch processing, researchers presented an improved cuckoo algorithm based on separate superior and inferior populations[2] and an efficient genetic algorithm (GA) with adaptive crossover and mutation[3]. Li et al.[4] solved the green HFSP with fuzzy processing time by using discrete artificial bee colony algorithm (ABC). Li et al.[5] designed the multi-objective optimization algorithm by selecting related neighborhood structures for the problem with setup energy consumptions. Wang et al.[6] devised a novel shuffled frog-leaping algorithm (SFLA) for minimizing makespan and total tardiness for reentrant problem. Guan et al.[7] presented an improved GA with multi-crossover-operator for the problem with multiprocessor task and minimization of makespan. Wu et al.[8] applied a multi-objective evolutionary algorithm with adaptive neighborhood updating strategy to solve reentrant problem with job release. Wang and Liu[9] solved the problem with preventive maintenance by applying multi-objective tabu search. Wang et al.[10] handled blocking problem and devised an improved iterative greedy algorithm. Li et al.[11] developed a two-level imperialist competitive algorithm to solve energy-efficient problem with relative importance of objectives. Qin et al.[12] devised an improved iterated greedy algorithm for the blocking HFSP with energy-efficient. Lei and Guo[13] presented a novel SFLA with tournament selection based population division to solve the problem with two agents. Wang et al.[14] studied green HFSP with consistent sublots and applied a multi-objective discrete ABC algorithm. Wu and Liu[15] proposed an improved memetic algorithm to solve green HFSP with sequence dependent setup time and transportation time.

• Zhenghui Yin, Deming Lei, and Bo Yang are with the School of Automation, Wuhan University of Technology, Wuhan 430070, China. E-mail: 334932@whut.edu.cn; deminglei11@163.com; byang@whut.edu.cn.
* To whom correspondence should be addressed.
※ This article was recommended by Associate Editor Xinyu Li
Manuscript received: 2024-01-23; revised: 2024-05-17; accepted: 2024-06-06

HFSP with limited buffers is solved by discrete whale swarm algorithm (DWSA) with a deduplication strategy[16]. HFSP with device dynamic reconfiguration is also solved by an improved whale optimization algorithm (WOA)[17]. Marichelvam et al.[18] presented improved particle swarm optimization algorithm with variable neighborhood search for HFSP with effect of human factors.

As stated above, the previous works are mainly about HFSP with various conditions and constraints such as no-wait, batch processing, and reentrance; however, some conditions in real-world manufacturing processes are seldom considered, for example, no precedence relation between some stages attracted little attention, which exists in hot and cold casting shop. In this shop, there are eight stages, Stages 2 and 3 are core making and mold assembling, and there is no precedence relation between them, that is, processing sequence of job on Stages 2 and 3 can be (2, 3) or (3, 2). This constraint will lead to stage determination sub-problem, the number of sub-problems increases and optimization difficulties will enlarge; meanwhile, HFSP with this constraint is refined from the real-word casting shop and the corresponding results have high application possibility, thus, it is necessary to handle HFSP with no precedence relation between some stages.

SFLA is a metaheuristic[19] algorithm with some remarkable features of simple concept, the fewer parameters, the faster computation speed and strong global optimization ability. There are some application of SFLA to scheduling problems[6, 13, 20–29]. It can be found that SFLA has been used to solve HFSP with complex conditions and constraints[6, 13], showing significant advantages in solving HFSP[25, 26]. It is rarely used to solve HFSP with no precedence between some stages, which is an extension of HFSP. There are great similarities between HFSP and HFSP with no precedence. However, HFSP with no precedence between some stages needs consideration on the visiting sequence of each job at stages with no precedence. The successful applications of SFLA to HFSP show that SFLA is an optimization algorithm with potential advantages for solving HFSP with no precedence between some stages.

In this study, HFSP with no precedence between some stages is solved, and an adaptive shuffled frog-leaping algorithm (ASFLA) is used to optimize makespan. A new solution representation and decoiding are proposed, an adaptive memeplex search process and dynamical population shuffling are implemented together. Extensive experiments are carried out to test the performance of ASFLA by comparing it with other existing algorithms. The effectiveness of new strategies are proved, and notable search advantages of ASFLA are validated in solving HFSP with no precedence between some stages.

2 Problem Description

θ_l: number of machines at stage l.

M_{lk}: the k-th machines at stage l.

p_{ilk}: processing time of J_i on $M_{lk} \in \theta_l$.

C_i: completion time of job J_i.

$C_{\max}$: the maximum completion time of all jobs.

gen: the number of current generation.

μ: number of searches in each memeplex.

Ω: memory solution set.

β: number of solution selected to execuse memeplex search.

meq_i: evolutionary quality of memeplex $\mathcal{M}_i$.

η: evaluation index of memeplex.

ϑ: an integer, $\vartheta \in \{1,2,3\}$.

s: number of memeplexes.

HFSP, with no precedence between some stages possesses, consists of n jobs $J_1, J_2, \ldots, J_n$ and H processing stages. Each stage l has θ_l unrelated parallel machines, $\theta_l = \{M_{l1}, M_{l2}, \ldots, M_{l|\theta_l|}\}$, $\forall 1 \leqslant l \leqslant H$, $|\theta_l| > 1$. J_i is processed in terms of production flow: Stage 1, Stage 2,..., Stage H. There is no precedence between stages σ and $\sigma+1$, each job J_i can be processed at stage σ then at stage $\sigma+1$, or vice versa, where $1 \leqslant \sigma < H$.

In some real-life hybrid flow shop manufacturing processes, there is no precedence between some stages. For example, in a casting hybrid flow shop, there are 8 stages, and there is no precedence between Stages 2 and 3. HFSP with no precedence between some stages is seldom investigated.

The problem can be divided into three sub-problems: scheduling, machine assignment, and stage determination. Stage determination is used to decide the visiting sequence between stages σ and $\sigma+1$ for each job.

The following objective is minimized.

$$C_{\max} = \max_{i=1,2,\ldots,n} \{C_i\} \tag{1}$$

where $C_{\max}$ indicates the maximum completion time among all jobs, and C_i is the completion time of job J_i.

Table 1 shows an illustrative example with 5 jobs

Table 1 Data on p_{ilk} on the example.

J_i	Processing time (s)									
	M_{11}	M_{12}	M_{21}	M_{22}	M_{23}	M_{31}	M_{32}	M_{41}	M_{42}	M_{43}
1	5	8	7	8	6	4	2	4	5	8
2	4	3	6	4	4	1	3	7	6	8
3	1	3	8	5	3	4	5	3	7	9
4	8	5	3	7	4	3	6	5	3	4
5	6	4	5	4	3	2	5	4	6	3

and 4 stages. Stages 1 and 3 have two unrelated parallel machines, Stages 2 and 4 have three unrelated parallel machines and there is no precedence between Stages 2 and 3. A schedule of the example is listed in Fig. 1.

3 ASFLA for HFSP with No Precedence Between Some Stages

3.1 Encoding, decoding, and initialization

A two-string representation and a heuristic for the machine assignment is proposed. For HFSP with no precedence between stages σ and $\sigma+1$, a solution is denoted by a scheduling string $[\gamma_1,\gamma_2,\ldots,\gamma_n]$ and a stage determination string $[\varphi_1,\varphi_2,\ldots,\varphi_n]$, where $\gamma_i\in\{1,2,\ldots,n\}$, $\varphi_i\in\{\sigma,\sigma+1\}$, if $\varphi_i=\sigma$, the visiting sequence for job J_i is stages σ, $\sigma+1$; otherwise, the above sequence is stages $\sigma+1$, σ.

Decoding is done as follows:

Step 1: Let $l=1$, start with the first job π_1 from scheduling string, for each job π_i, determine a machine M_{1k} based on the heuristic and accomplish processing of π_i on the machine.

Step 2: Repeat the following steps until $l>H$.

(1) If $1<l<\sigma$ or $l>\sigma+2$, produce permutation $\gamma'_1,\gamma'_2,\ldots,\gamma'_n$ by ranking n jobs in ascending order of their completion time at stage $l-1$, start with job γ'_1. For each γ'_i, determine a machine M_{lk} by the heuristic and process γ'_i on the machine.

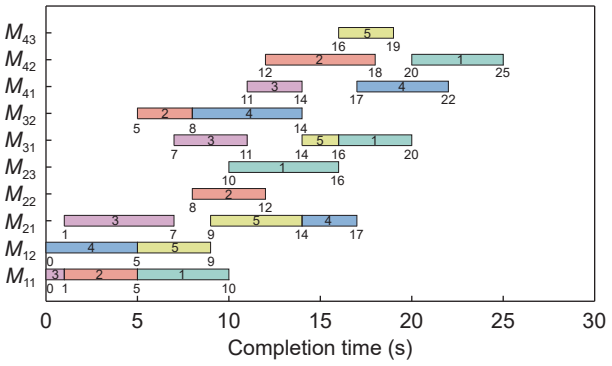

Fig. 1 Schedule of the example.

(2) If $l=\sigma$, obtain a job permutation $\gamma'_1,\gamma'_2,\ldots,\gamma'_n$ using the same process of Step (1), starting with job γ'_1. For each job γ'_i, let $\delta=\varphi_{\gamma_i'}$, determine the visiting sequence of stages σ, $\sigma+1$ according to $\varphi_{\gamma_i'}$, and select a machine $M_{\delta k}$ and a machine $M_{(2\sigma+1-\delta)k'}$ by the heuristic and process γ'_i on them sequentially.

(3) If $l=\sigma+2$, sort n jobs in ascending order of their completion time at stage $2\sigma+1-\varphi_{\gamma_i}$, and obtain a permutation $\gamma'_1,\gamma'_2,\ldots,\gamma'_n$, starting with job γ'_1. For each job γ'_i, select a machine M_{lk} according to the heuristic and process γ'_i on the machine sequentially.

(4) $l=l+1$

The heuristic for machine assignment is described below. For each J_i, calculate the earliest beginning time on each M_{lg}, $g=1,2,\ldots,\theta_l$ at stage l, select M_{lk} with the smallest beginning time. If there exist more than one machine with same beginning time, select one with the smallest index number.

For the example in Section 2, the solution consists of $[3,4,2,1,5]$ and a stage determination string $[2,3,2,3,2]$. When $l=1$, Jobs 3, 4, 2, 1, and 5 are assigned to machines M_{11}, M_{12}, M_{11}, M_{11}, and M_{12} by the heuristic sequentially. When $l=2$, a permutation $[3,2,4,5,1]$ is obtained by ranking their completion times at stage 1, take Job 3 as an example, $\varphi_3=2$, the visiting sequence of job 3 is Stages 2 and 3, and Job 3 is processed sequentially on M_{21} and M_{31} based on the heuristic. When $l=4$, a permutation $[3,2,5,4,1]$ is formed. The related schedule is displayed in Fig. 1.

s memeplexes $\mathcal{M}_1,\mathcal{M}_2,\ldots,\mathcal{M}_s$ are produced by dividing a randomly generated initial population P with N initial solutions[19].

3.2 Adaptive memeplex search and dynamical population shuffling

In existing SFLA[27, 28], optimization object is x_w, x_b and x_g are used in memeplex search and other solutions are seldom involved with memeplex search. In ASFLA, $\beta>1$ solutions are used as optimization object, and adaptive memeplex search and dynamical population shuffling are as shown in Algorithm 1.

(1) If gen = 1, for each memeplex $\mathcal{M}_i$, determine β solutions with the smallest makespan, and let them into set ϕ, for each $x\in\phi$, repeat the following step μ/β times: randomly select a solution $y\in\phi$, $y\neq x$, execute GS(x,y), if x is not updated, conduct multiple neighborhood search on x.

(2) If gen > 1, then

(a) Sort all memeplexes with $\text{meq}_i\geqslant\eta$ in descending

Algorithm 1 Adaptive memeplex search and dynamical population shuffling

if gen = 1 **then**
  for each memeplex $\mathcal{M}_i$ **do**
    put the first β solution into set ϕ
    for each $x \in \phi$ **do**
      randomly select a solution $y \in \phi$, $y \neq x$, execute GS(x,y) and multiple neighborhood search on x
    end for
  end for
end if
if gen > 1 **then**
  if $\vartheta \leqslant 2$ **then**
    for each $\mathcal{M}_i$ with $\text{meq}_i \geqslant \eta$ **do**
      execute GS(x,y) and multiple neighborhood search on x
    end for
  end if
  if $\vartheta \geqslant 2$ **then**
    for each $\mathcal{M}_i$ with $\text{meq}_i < \eta$ **do**
      randomly select $y \in \mathcal{M}^*$, GS(x_b,y) and multiple neighborhood search on x are execute μ times
    end for
    execute population division by using $P' \cup \Omega$ to form g memeplexes
  end if
end if
Compute evolutionary quality meq_i of each memeplex

order of meq_i, suppose that $\text{meq}_1 \geqslant \text{meq}_2 \geqslant \cdots \geqslant \text{meq}_u$, for each memeplex $\mathcal{M}_i$, execute global search and multiple neighborhood as done in Step (1), where y is chosen: $y \in \mathcal{M}_l \subset \{\mathcal{M}_g \,|\text{meq}_g > \text{meq}_i\}$ or $y \in \mathcal{M}_i$, if $\{\mathcal{M}_g \,|\text{meq}_g > \text{meq}_i\}$ is empty, $1 \leqslant i \leqslant u$ and $1 \leqslant g \leqslant u$.

(b) If $\vartheta \geqslant 2$, select $\mathcal{M}_i$ with $\text{meq}_i < \eta$. For $x_b \in \mathcal{M}_i$, randomly select $y \in \mathcal{M}^*$, GS(x_b,y) and multiple neighborhood search on x are execute μ times.

(3) Compute evolutionary quality meq_i of each memeplex, if s memeplexes have $\text{meq}_i \geqslant \eta$, $\vartheta = 1$; if at least one memeplex has $\text{meq}_i \geqslant \eta$, $\vartheta = 2$; if s memeplexes have $\text{meq}_i < \eta$, $\vartheta = 3$.

(4) If $\vartheta \geqslant 2$, obtain population P' by all memeplexes with $\text{meq}_i < \eta$, suppose g memeplexes with $\text{meq}_i < \eta$, and execute population division by using $P' \cup \Omega$ to form g memeplexes. If $\vartheta = 2$, select memeplex with the biggest meq_i as $\mathcal{M}^*$. If $\vartheta = 3$, $\mathcal{M}^*$ consists of all updated solutions in P.

$$\text{meq}_i = \text{ES}_i / \text{TS}_i \tag{2}$$

where ES_i and TS_i are integers and computed as follows: when the search of $\mathcal{M}_i$ starts, $\text{ES}_i = 1$, $\text{TS}_i = 1$, when a solution z is given by global search or multiple neighborhood search on x, $\text{TS}_i = \text{TS}_i + 1$; if $C^z_{\max} < C^x_{\max}$, $\text{ES}_i = \text{ES}_i + 1$, that is, solution x is updated.

Global search GS(x',y') is described below. Conduct order-based crossover[30] on the scheduling string of x' and y', get a solution z', if $C^z_{\max}{}' < C^x_{\max}{}'$, update Ω with x' and replace x' with z'; otherwise, produce z'' by performing two-point crossover on the stage determination string of x' and y'. If $C^{z''}_{\max} < C^x_{\max}{}'$, Ω is updated with x' and z'' substitutes for x'.

Ω has maximum capacity of N/s. When Ω is updated by x, if $|\Omega|$ exceeds N/s and x is better than the worst solution in Ω, put x into Ω to replace the worst solution; if $|\Omega| < N/s$, add x to Ω directly.

Five neighborhood structures $\mathcal{N}_1$–$\mathcal{N}_5$ are used. $\mathcal{N}_1$–$\mathcal{N}_4$ are related to scheduling string, $\mathcal{N}_1$ is the swapping of two randomly chosen solutions γ_i and γ_j. $\mathcal{N}_2$ is used to generate solutions by inserting γ_i into the position of γ_j. $\mathcal{N}_3$ and $\mathcal{N}_4$ are similar to $\mathcal{N}_1$ and $\mathcal{N}_2$. In $\mathcal{N}_3$ and $\mathcal{N}_4$, job γ_i has the maximum completion time, where $i > j$. $\mathcal{N}_5$ acts on stage determination string and is proposed for no precedence between some stages. It is described below. Determine the stage $h \in \{\sigma, \sigma+1\}$ that has the maximum load, randomly choose φ_i with $\varphi_i = h$, and let $\varphi_i = 2\sigma + 1 - h$.

Multiple neighborhood search is described as follows. $r = 1$, execute the following steps repeatedly until $r = 6$: produce $z' \in \mathcal{N}_r(x')$, if $C^z_{\max}{}' < C^x_{\max}{}'$, update Ω with x', z' substitutes for x', $r = 6$; otherwise, $r = r + 1$. $\mathcal{N}_r(x')$ is the set of neighbourhood solution of x' produced by using $\mathcal{N}_r$.

3.3 Algorithm description

ASFLA has simple structure and is given in Algorithm 2. After initial population being randomly generated, population division, and adaptive memeplex search and dynamical population shuffling are executed repeatedly until the stopping condition is met.

Flow chart of ASFLA is given in Fig. 2.

Unlike existing SFLA[6, 13, 20, 25, 26], β solutions are

Algorithm 2 ASFLA

Randomly produce initial population P, let gen = 1
Divide population P into s memeplexes
while stopping condition is not met **do**
  Execute adaptive memeplex search and dynamical population shuffling
  gen = gen + 1
end while

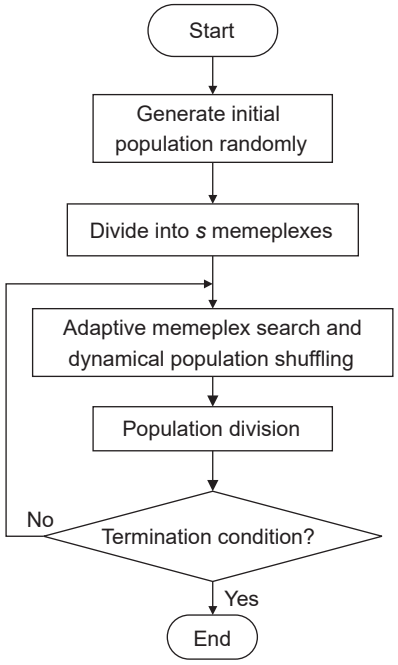

Fig. 2 Flow chart of ASFLA.

used as optimization object, and each of them is given μ/β times. Memeplexes with $\text{meq}_i \geqslant \eta$ are given different search strategies from memeplexes with $\text{meq}_i < \eta$. Moreover, memeplexes with $\text{meq}_i \geqslant \eta$ are not used for population division to sufficiently use the good solution structure of these memeplexes.

4 Computational Experiment

All experiments are implemented by using Microsoft Visual C++ 2022 and run on 8.0 GB RAM 2.4 GHz CPU PC.

4.1 Test instance and comparative algorithm

In this study, 60 instances are used. For each instance, use $n \times H$ to represent it, where $n \in \{10, 20, \ldots, 150\}$, $H \in \{2, 4, 6, 8\}$, $p_{ilk} \in [50, 100]$, and $|\theta_l| \in \{2, 3\}$. If $H = 2$, $\sigma = 1$; if $H = 4$, $\sigma = 2$; if $H = 6$, $\sigma = 5$; if $H = 8$, $\sigma = 6$. All of above data are integers.

Three algorithms are selected, which are improved gravitational search algorithm (IGSA)[31], improved particle swarm optimization algorithm (IPSOA)[18], and hybrid evolutionary algorithm (HEA)[32]. These algorithms can provide promising results for HFSP and can be applied to solve the HFSP with no precedence between some stages, so they are chosen as comparative algorithm.

Stage determination string of ASFLA is directly added into IGSA, IPSOA, and HEA. For IGSA and IPSOA, after using related formulas for scheduling string, $\mathcal{N}_5$ is applied for stage determination string. For HEA, after using its neighborhood structures for scheduling string, $\mathcal{N}_5$ is executed for stage determination string.

To show the effect of adaptive memeplex search and dynamical population shuffling, ASFLA is compared with SFLA[19], in which the memeplex search is as follows: for each $\mathcal{M}_i$, execute $\text{GS}(x_w, x_b)$, and obtain z. If x_w is not updated by z, apply $\text{GS}(x_w, x_g)$, and obtain z'; if x_w is not updated by z', conduct multiple neighborhood search on x_w. Random initialization is also used.

4.2 Parameter setting

ASFLA has the following parameters: N, s, μ, β, η, and the stopping condition. ASFLA can converge fully when $0.25 \times n \times H$ s CPU time is used; moreover, when $0.25 \times n \times H$ s CPU time is applied, IGSA, IPSOA, HEA, and SFLA also converge fully, so the above CPU time is given as stopping condition.

Taguchi method[33] on instance 20×4 is used to obtain settings on the remaining parameters of ASFLA. Table 2 reveals the levels of each parameter. 16 orthogonal experiments are executed. In each orthogonal experiment, ASFLA randomly runs on instance 20×4. Figure 3 gives related results. s/N ratio is $-10 \times \log_{10}(\text{MIN}^2)$, where MIN indicates the solution with the smallest makespan in 10 runs.

There are 16 parameter combinations used. The computational results reveal that when the following settings are used: $N = 96$, $s = 6$, $\mu = 40$, $\beta = 4$, and $\eta = 0.3$, ASFLA yields the smallest MIN, so these settings are adopted.

$N = 96$, $s = 4$, and $\mu = 40$ are given to SFLA. After giving the stopping condition, experiments show that parameter settings of IGSA[33], IPSOA[20], and HEA[32] are still effective, so they are still adopted.

4.3 Result and discussion

Each of ASFLA, SFLA, IGSA, IPSOA, and HEA runs randomly 10 times for each instance. x_g is produced in a run. MIN (MAX) denotes the best (worst) x_g found in

Table 2 Levels of each parameter.

Factor level	Parameter				
	N	s	μ	β	η
1	48	4	30	2	0.2
2	72	6	40	3	0.3
3	96	8	50	4	0.4
4	120	12	60	5	0.5

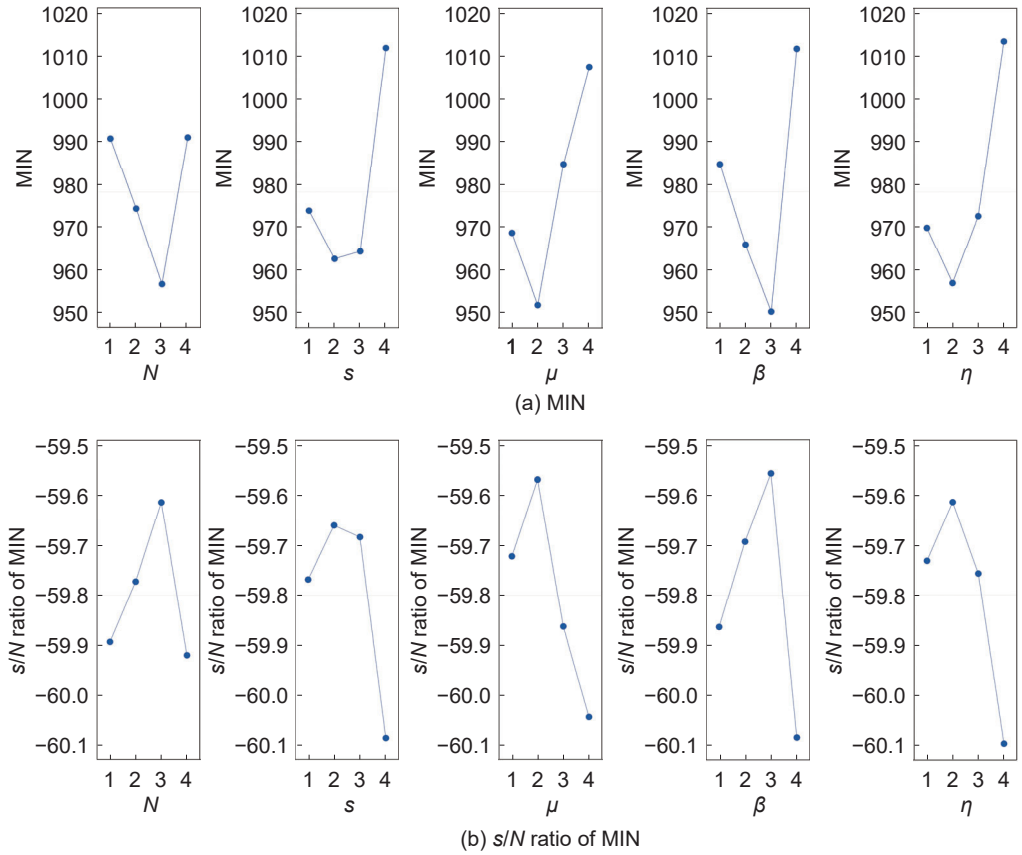

Fig. 3 Main effect plot for MIN and *s*/*N* ratio.

10 runs, and AVG denotes the average makespan of $10x_g$ in 10 runs. Tables 3−5 reveal their results on MAX, MIN, and AVG. Figure 4 depicts convergence curves for 50×4 and 70×6.

Figure 5 demonstrates box plots of ASFLA, SFLA, IGSA, IPSOA, and HEA. $\mathrm{RPD_{MIN}}$, $\mathrm{RPD_{MAX}}$, and $\mathrm{RPD_{AVG}}$ are as follows:

$$\mathrm{RPD_{MIN}} = \frac{\mathrm{MIN} - \mathrm{MIN}^*}{\mathrm{MIN}^*} \times 100 \tag{3}$$

where MIN^* is the smallest MIN produced by all algorithms. $\mathrm{RPD_{AVG}}$ and $\mathrm{RPD_{MAX}}$ are represented in the same way as $\mathrm{RPD_{MIN}}$.

Table 6 reports the related data on pair-sample t-test with the significance level of 0.05. t-test (B_1, B_2) means that a paired t-test is performed to judge whether algorithm B_1 gives a better sample mean than B_2. Statistically significant difference between algorithms B_1 and B_2 exists if the p-value is less than 0.05.

It can be found in Table 3 that ASFLA converges better than SFLA. As shown in Tables 4 and 5, ASFLA also produces better results on MAX and AVG than SFLA on at least 49 of 60 instances. The results in Table 6 and Figs. 4 and 5 also demonstrate the

Table 3 Computational results of ASFLA, SFLA, IGSA, IPSOA, and HEA on MIN.

Instance ($s \times H$)	MIN				
	ASFLA	SFLA	IGSA	IPSOA	HEA
10×2	**360**	372	360	363	360
10×4	**546**	597	567	570	579
10×6	**730**	764	736	733	742
10×8	**875**	893	881	878	886
20×2	**680**	712	690	688	696
20×4	951	1023	968	**943**	985
20×6	**1130**	1232	1175	1211	1217
20×8	**1257**	1481	1308	1356	1410
30×2	**1050**	1237	1154	1094	1192
30×4	**1299**	1420	1352	1342	1325
30×6	**1447**	1665	1537	1491	1563
30×8	**1633**	1822	1750	1690	1794
40×2	**1340**	1534	1393	1379	1473
40×4	1754	1822	**1727**	1778	1750
40×6	**1794**	1926	1864	1817	1895
40×8	**1976**	2231	2110	2064	2161
50×2	1766	1782	1786	**1723**	1887
50×4	**2307**	2436	2471	2409	2492
50×6	**2212**	2426	2384	2314	2403

(to be continued)

Table 3 Computational results of ASFLA, SFLA, IGSA, IPSOA, and HEA on MIN.

(continued)

Instance ($s \times H$)	MIN				
	ASFLA	SFLA	IGSA	IPSOA	HEA
50×8	**2575**	2758	2648	2624	2697
60×2	**2017**	2168	2085	2039	2137
60×4	**2976**	3093	3003	3035	3041
60×6	**2681**	2883	2742	2838	2868
60×8	**3319**	3539	3466	3454	3488
70×2	**2384**	2467	2425	2408	2478
70×4	3540	3665	**3530**	3612	3759
70×6	**3450**	3682	3489	3506	3577
70×8	**3769**	3953	3799	3884	3967
80×2	**2695**	2813	2746	2763	2854
80×4	4151	4382	4307	**4101**	4322
80×6	**3856**	4093	3957	3945	4088
80×8	**4418**	4672	4603	4569	4644
90×2	3186	3285	**3134**	3195	3242
90×4	**4674**	4895	4785	4736	4817
90×6	4678	4839	4752	**4552**	4808
90×8	**5377**	5547	5431	5401	5496
100×2	**3531**	3663	3604	3575	3637
100×4	**5574**	5747	5664	5627	5770
100×6	**4843**	4991	4887	4899	4943
100×8	**6172**	6402	6219	6324	6364
110×2	**3881**	4018	3926	3900	3953
110×4	**6225**	6375	6326	6284	6385
110×6	**5409**	5595	5492	5563	5542
110×8	**6840**	7046	6872	6922	7021
120×2	4172	4235	4193	4163	**4116**
120×4	**7110**	7320	7197	7187	7240
120×6	**6241**	6452	6348	6301	6376
120×8	7590	7792	7661	**7498**	7765
130×2	**4405**	4580	4496	4467	4521
130×4	**7641**	7839	7768	7669	7814
130×6	6965	7149	7016	**6933**	7071
130×8	**8195**	8388	8255	8213	8338
140×2	4959	5124	4988	**4926**	5048
140×4	**8319**	8506	8490	8476	8507
140×6	**7312**	7419	7398	7387	7482
140×8	**9026**	9211	9179	9103	9139
150×2	**5191**	5337	5225	5217	5272
150×4	9234	9367	9354	**9209**	9454
150×6	**7906**	8142	8081	8103	8099
150×8	**9962**	10117	10056	10024	10087

convergence advantage of ASFLA.

Table 3 reveals that ASFLA generates better MIN than IGSA, IPSOA, and HEA on 48 of 60 instances,

Table 4 Computational results of ASFLA, SFLA, IGSA, IPSOA, and HEA on MAX.

Instance ($s \times H$)	MAX				
	ASFLA	SFLA	IGSA	IPSOA	HEA
10×2	**366**	378	372	369	376
10×4	**613**	671	625	617	640
10×6	**746**	786	764	749	767
10×8	**889**	915	930	913	900
20×2	708	714	709	712	**701**
20×4	**1029**	1214	998	1050	1097
20×6	**1168**	1268	1331	1362	1322
20×8	**1346**	1565	1462	1427	1573
30×2	**1069**	1267	1169	1120	1214
30×4	1625	1751	1645	**1612**	1664
30×6	**1527**	1841	1622	1625	1617
30×8	1918	1989	**1814**	1849	1982
40×2	**1368**	1574	1422	1414	1502
40×4	2243	2373	2196	**2185**	2217
40×6	**2054**	2126	2085	2066	2093
40×8	**2257**	2348	2510	2475	2311
50×2	**1753**	1902	1816	1786	1851
50×4	**2816**	2930	2841	2852	2870
50×6	**2593**	2752	2637	2673	2713
50×8	**3068**	3293	3142	3224	3180
60×2	**2067**	2199	2119	2099	2083
60×4	3401	**3365**	3542	3625	3392
60×6	3080	3228	**3028**	3194	3211
60×8	**3710**	3899	3901	3821	3901
70×2	**2416**	2517	2426	2434	2524
70×4	**4146**	4497	4199	4243	4284
70×6	**3592**	3837	3796	3735	3882
70×8	**4502**	4774	4550	4598	4600
80×2	**2740**	2881	2774	2787	2880
80×4	**4857**	5047	4871	4901	4962
80×6	**3856**	4093	3957	3945	4088
80×8	**4418**	4672	4603	4569	4644
90×2	3186	3285	**3134**	3195	3242
90×4	**4674**	4895	4785	4736	4817
90×6	**4852**	5003	4875	4927	4962
90×8	**6058**	6415	6126	6211	6324
100×2	3562	3842	3642	**3550**	3690
100×4	**6200**	6461	6344	6406	6316
100×6	**5385**	5507	5410	5391	5460
100×8	**6731**	6950	6878	6896	7045
110×2	**3934**	4396	3995	4011	3980
110×4	7083	7268	**6947**	7214	7168
110×6	**5691**	6035	5825	5752	5929
110×8	**7615**	8172	7725	7694	7740
120×2	**4158**	4387	4188	4207	4234

(to be continued)

Table 4 Computational results of ASFLA, SFLA, IGSA, IPSOA, and HEA on MAX.

(continued)

Instance ($s \times H$)	MAX				
	ASFLA	SFLA	IGSA	IPSOA	HEA
120×4	7875	8127	8019	7917	**7837**
120×6	6508	6842	**6506**	6621	6719
120×8	**8306**	8590	8448	8405	8503
130×2	**4501**	4843	4566	4532	4591
130×4	**8431**	8688	8501	8596	8669
130×6	**7133**	7311	7236	7181	7259
130×8	**8964**	9199	9084	9010	9045
140×2	**4989**	5179	5056	5003	5094
140×4	**9473**	9642	9585	9542	9498
140×6	**7714**	7959	7767	7785	7850
140×8	**9751**	10027	9971	9871	9995
150×2	**5252**	5424	5280	5292	5343
150×4	**9927**	10090	10119	9985	10206
150×6	**8193**	8418	8345	8271	8332
150×8	**10332**	10533	10601	10478	10657

Table 5 Computational results of ASFLA, SFLA, IGSA, IPSOA, and HEA on AVG.

Instance ($s \times H$)	AVG				
	ASFLA	SFLA	IGSA	IPSOA	HEA
10×2	**361**	366	363	364	364
10×4	586	624	**579**	587	597
10×6	**740**	768	749	**740**	756
10×8	**872**	894	892	887	889
20×2	**689**	702	699	701	692
20×4	**975**	1077	986	994	990
20×6	**1148**	1246	1259	1291	1277
20×8	**1282**	1528	1359	1411	1485
30×2	**1060**	1243	1161	1109	1203
30×4	**1403**	1463	1449	1484	1490
30×6	**1484**	1664	1573	1564	1594
30×8	**1760**	1897	1790	1763	1856
40×2	**1360**	1566	1417	1395	1483
40×4	1926	2034	1996	1968	**1917**
40×6	**1889**	2082	1947	1929	1980
40×8	**2103**	2296	2281	2214	2234
50×2	1793	1894	1804	**1774**	1837
50×4	**2528**	2604	2642	2716	2687
50×6	**2441**	2528	2457	2477	2493
50×8	**2860**	2941	2978	2970	2930
60×2	**2052**	2186	2103	2076	2160
60×4	**3218**	3401	3305	3379	3268
60×6	2977	3146	**2870**	3001	2993
60×8	**3528**	3743	3590	3648	3693

(to be continued)

Table 5 Computational results of ASFLA, SFLA, IGSA, IPSOA, and HEA on AVG.

(continued)

Instance ($s \times H$)	AVG				
	ASFLA	SFLA	IGSA	IPSOA	HEA
70×2	**2397**	2494	2415	2423	2502
70×4	**3782**	3952	3820	3894	3908
70×6	3626	3766	3686	**3532**	3733
70×8	**4246**	4330	4247	4278	4315
80×2	**2719**	2857	2767	2784	2864
80×4	**4621**	4807	4692	4687	4742
80×6	**4087**	4275	4209	4165	4248
80×8	**4911**	5152	5006	5018	4971
90×2	**3161**	3308	3206	3209	3255
90×4	**5162**	5382	5239	5175	5376
90×6	4757	4931	**4716**	4842	4778
90×8	5746	5938	**5684**	5724	5934
100×2	**3548**	3767	3608	3627	3685
100×4	5963	5990	6081	6011	**5932**
100×6	5311	5453	5339	**5303**	5349
100×8	**6473**	6703	6587	6592	6632
110×2	**3916**	4163	3967	3987	3956
110×4	**6693**	6869	6726	6731	6719
110×6	5754	5872	**5576**	5651	5780
110×8	7475	7557	**7350**	7419	7355
120×2	4146	4264	**4139**	4180	4217
120×4	7656	7896	7756	**7531**	7602
120×6	6452	6641	6457	**6440**	6645
120×8	**7877**	8064	7967	7945	8087
130×2	**4467**	4679	4523	4483	4558
130×4	**8172**	8380	8296	8387	8355
130×6	**7046**	7249	7099	7049	7173
130×8	**8768**	8958	8796	8824	8867
140×2	**4951**	5149	5007	4983	5076
140×4	9045	9223	9073	**9043**	9121
140×6	**7585**	7742	7695	7650	7702
140×8	**9397**	9546	9496	9466	9565
150×2	**5221**	5376	5256	5258	5324
150×4	**9686**	9847	9697	9724	9808
150×6	**8051**	8243	8118	8195	8239
150×8	**10037**	10186	10085	10135	10117

moreover, ASFLA obtains worse MIN than HEA just on instance 120×2, gets bigger MIN than IGSA only on instances 40×4, 70×4, and 90×2, and generates worse MIN than IPSOA just on 8 instances. Table 6 and Figs. 4 and 5 also reveal the significant advantage of ASFLA on MIN.

With respect to MAX in Table 4, ASFLA performs better than IGSA, IPSOA, and HEA on 49 instances.

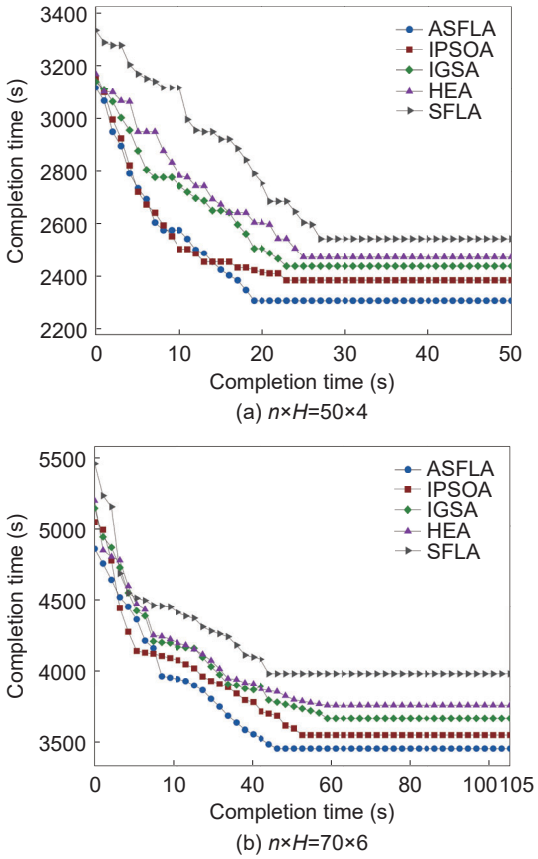

Fig. 4 Convergence curves on 50 × 4 and 70 × 6 .

Moreover, ASFLA gets worse MAX than HEA just on instance 20×2 and 120×4, obtains bigger MAX than IGSA only on 5 instances, and produces worse MAX than IPSOA just on 3 instances. Figure 5 and Table 6 also reveal the notable advantages of ASFLA on MAX. Similarly, Tables 5 and 6 and Fig. 5 also show the significant advantages of ASFLA on AVG.

In ASFLA, more solutions become optimization objects of memeplexes, and different memeplexes have different search strategies. These features lead to strong exploration ability. Moreover, good solution structure in some memeplexes with $\text{meq}_i \geqslant \eta$ are used fully, thus, ASFLA is a competitive method in solving HFSP with no precedence between some stages.

5 Conclusion and Future Topic

In this study, HFSP with no precedence between stages σ and $\sigma+1$ is solved by using ASFLA. In ASFLA, a new coding and decoding are used, an adaptive memeplex search and dynamical population shuffling are implemented together. The effectiveness of new strategies in ASFLA is first tested by comparing ASFLA and SFLA. Then the search advantages of ASFLA on HFSP with no precedence are tested by comparing ASFLA with IGSA, IPSOA, and HEA.

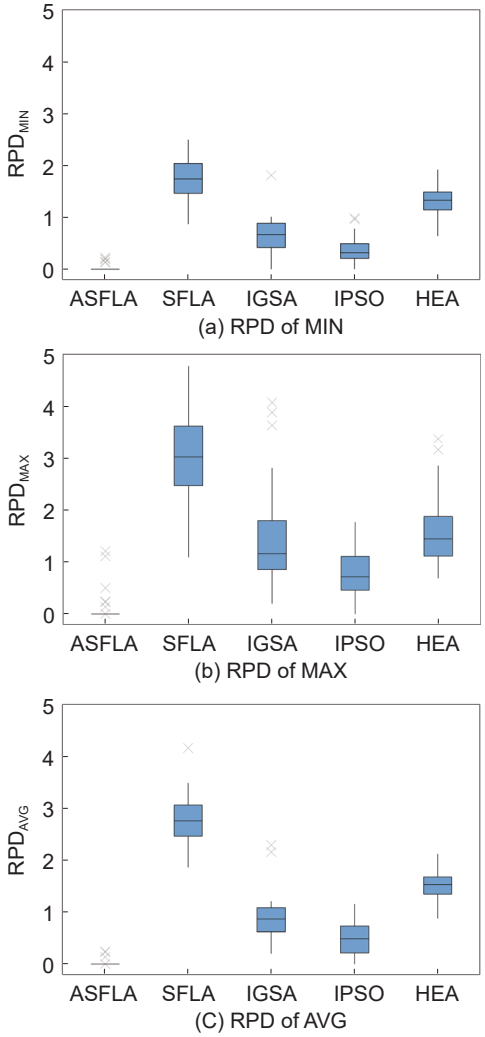

Fig. 5 Box plots of five algorithms.

Table 6 Paired sample *t*-test on ASFLA, SFLA, IGSA, IPSOA, and HEA.

Paired sample *t*-test	MIN	MAX	AVG
t-test (ASFLA, SFLA)	0	0	0
t-test (ASFLA, IGSA)	0	0	0
t-test (ASFLA, IPSOA)	0	0	0
t-test (ASFLA, HEA)	0	0	0

In the future, we will focus on HFSP with batch processing machine in casting process by applying some new optimization strategies such as learning in ABC and SFLA. Adding optimization mechanisms such as cooperation or feedback into metaheuristics is one of our future research topics.

References

[1] M. Rabiee, F. Jolai, H. Asefi, P. Fattahi, and S. Lim, A biogeography-based optimisation algorithm for a realistic

no-wait hybrid flow shop with unrelated parallel machines to minimise mean tardiness, *Int. J. Comput. Integr. Manuf.*, vol. 29, no. 9, pp. 1007−1024, 2016.

[2] X. Li, X. Guo, H. Tang, R. Wu, and J. Liu, An improved cuckoo search algorithm for the hybrid flow-shop scheduling problem in sand casting enterprises considering batch processing, *Comput. Ind. Eng.*, vol. 176, p. 108921, 2023.

[3] H. Lu and F. Qiao, An efficient adaptive genetic algorithm for energy saving in the hybrid flow shop scheduling with batch production at last stage, *Expert Syst.*, vol. 39, no. 2, p. e12678, 2022.

[4] M. Li, G. G. Wang, and H. Yu, Sorting-based discrete artificial bee colony algorithm for solving fuzzy hybrid flow shop green scheduling problem, *Mathematics*, vol. 9, no. 18, p. 2250, 2021.

[5] J. Q. Li, H. Y. Sang, Y. Y. Han, C. G. Wang, and K. Z. Gao, Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions, *J. Clean. Prod.*, vol. 181, pp. 584−598, 2018.

[6] K. Wang, L. M. Xu, and C. Lv, A novel SFLA for reentrant hybrid flow shop scheduling, presented at the 3rd Int. Conf. Electrical, Mechanical and Computer Engineering, Guizhou, China, 2019.

[7] Y. Guan, Y. Chen, Z. Gan, Z. Zou, W. Ding, H. Zhang, Y. Liu, and C. Ouyang, Hybrid flow-shop scheduling in collaborative manufacturing with a multi-crossover-operator genetic algorithm, *J. Ind. Inf. Integr.*, vol. 36, p. 100514, 2023.

[8] X. Wu, X. Yan, and L. Wang, Optimizing job release and scheduling jointly in a reentrant hybrid flow shop, *Expert Syst. Appl.*, vol. 209, p. 118278, 2022.

[9] S. Wang and M. Liu, Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method, *Int. J. Prod. Res.*, vol. 52, no. 5, pp. 1495−1508, 2014.

[10] Y. Wang, Y. Wang, Y. Han, J. Li, K. Gao, and Y. Nojima, Intelligent optimization under multiple factories: Hybrid flow shop scheduling problem with blocking constraints using an advanced iterated greedy algorithm, *Complex System Modeling and Simulation*, vol. 3, no. 4, pp. 282−306, 2023.

[11] M. Li, D. Lei, and J. Cai, Two-level imperialist competitive algorithm for energy-efficient hybrid flow shop scheduling problem with relative importance of objectives, *Swarm Evol. Comput.*, vol. 49, pp. 34−43, 2019.

[12] H. X. Qin, Y. Y. Han, B. Zhang, L. L. Meng, Y. P. Liu, Q. K. Pan, and D. W. Gong, An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem, *Swarm Evol. Comput.*, vol. 69, p. 100992, 2022.

[13] D. Lei and X. Guo, A shuffled frog-leaping algorithm for hybrid flow shop scheduling with two agents, *Expert Syst. Appl.*, vol. 42, no. 23, pp. 9333−9339, 2015.

[14] W. Wang, B. Zhang, and B. Jia, A multiobjective optimization approach for multiobjective hybrid flowshop green scheduling with consistent sublots, *Sustainability*, vol. 15, no. 3, p. 2622, 2023.

[15] S. Wu and L. Liu, Green hybrid flow shop scheduling problem considering sequence dependent setup times and transportation times, *IEEE Access*, vol. 11, pp. 39726−39737, 2023.

[16] C. Zhang, J. Tan, K. Peng, L. Gao, W. Shen, and K. Lian, A discrete whale swarm algorithm for hybrid flow-shop scheduling problem with limited buffers, *Robot. Comput. Integr. Manuf.*, vol. 68, p. 102081, 2021.

[17] Y. Wang, S. Wang, D. Li, C. Shen, and B. Yang, An improved multi-objective whale optimization algorithm for the hybrid flow shop scheduling problem considering device dynamic reconfiguration processes, *Expert Syst. Appl.*, vol. 174, p. 114793, 2021.

[18] M. K. Marichelvam, M. Geetha, and Ö. Tosun, An improved particle swarm optimization algorithm to solve hybrid flowshop scheduling problems with the effect of human factors—A case study, *Comput. Oper. Res.*, vol. 114, p. 104812, 2020.

[19] M. M. Eusuff and K. E. Lansey, Optimization of water distribution network design using the shuffled frog leaping algorithm, *J. Water Resour. Plann. Manage.*, vol. 129, no. 3, pp. 210−225, 2003.

[20] J. Cai and D. Lei, A cooperated shuffled frog-leaping algorithm for distributed energy-efficient hybrid flow shop scheduling with fuzzy processing time, *Complex Intell. Syst.*, vol. 7, no. 5, pp. 2235−2253, 2021.

[21] L. Wang and D. Li, Fuzzy distributed hybrid flow shop scheduling problem with heterogeneous factory and unrelated parallel machine: A shuffled frog leaping algorithm with collaboration of multiple search strategies, *IEEE Access*, vol. 8, pp. 214209−214223, 2020.

[22] M. Kong, X. Liu, J. Pei, P. M. Pardalos, and N. Mladenovic, Parallel-batching scheduling with nonlinear processing times on a single and unrelated parallel machines, *J. Glob. Optim.*, vol. 78, no. 4, pp. 693−715, 2020.

[23] H. Zhang, S. Liu, S. Moraca, and R. Ojstersek, An effective use of hybrid metaheuristics algorithm for job shop scheduling problem, *Int. J. Simul. Model.*, vol. 16, no. 4, pp. 644−657, 2016.

[24] Q. K. Pan, L. Wang, L. Gao, and J. Li, An effective shuffled frog-leaping algorithm for lot-streaming flow shop scheduling problem, *Int. J. Adv. Manuf. Technol.*, vol. 52, no. 5, pp. 699−713, 2011.

[25] Y. Xu, L. Wang, G. Zhou, and S. Wang, An effective shuffled frog leaping algorithm for solving hybrid flow-shop scheduling problem, in *Advanced Intelligent Computing*, D. Huang, Y. Gan, V. Bevilacqua, and J. C. Figueroa, eds. Berlin, Germany: Springer, 2011, pp. 560−567.

[26] Y. Xu, L. Wang, S. Wang, and M. Liu, An effective shuffled frog-leaping algorithm for solving the hybrid flow-shop scheduling problem with identical parallel machines, *Eng. Optim.*, vol. 45, no. 12, pp. 1409−1430, 2013.

[27] L. Kong, T. Li, K. Wang, H. Zhu, T. Makoto, and B. Yu, An improved shuffled frog-leaping algorithm for flexible job shop scheduling problem, *Algorithms*, vol. 8, no. 1, pp.

19−31, 2015.
[28] Z. J. Gao, J. Y. Peng, Z. H. Han and M. Q. Jia, Flow shop scheduling with variable processing times based on differential shuffled frog leaping algorithm, *Int. J. Model. Identif. Contr.*, vol. 33, no. 2, pp. 179−187, 2019.
[29] J. Cai, R. Zhou, and D. Lei, Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks, *Eng. Appl. Artif. Intell.*, vol. 90, p. 103540, 2020.
[30] D. Lei, B. Su, and M. Li, Cooperated teaching-learning-based optimisation for distributed two-stage assembly flow shop scheduling, *Int. J. Prod. Res.*, vol. 59, no. 23, pp. 7232−7245, 2021.
[31] C. Cao, Y. Zhang, X. Gu, D. Li, and J. Li, An improved gravitational search algorithm to the hybrid flowshop with unrelated parallel machines scheduling problem, *Int. J. Prod. Res.*, vol. 59, no. 18, pp. 5592−5608, 2021.
[32] J. Fan, Y. Li, J. Xie, C. Zhang, W. Shen, and L. Gao, A hybrid evolutionary algorithm using two solution representations for hybrid flow-shop scheduling problem, *IEEE Trans. Cybern.*, vol. 53, no. 3, pp. 1752−1764, 2023.
[33] G. Taguchi, *Introduction to Quality Engineering: Designing Quality into Products and Processes*. Tokyo, Japan: Asian Productivity Organization, 1986.

Zhenghui Yin is currently pursuing the master degree at the School of Automation, Wuhan University of Technology, Wuhan, China. His current research interest includes manufacturing systems intelligent optimization and scheduling.

Bo Yang received the bachelor and PhD degrees from Huazhong University of Science and Technology, China in 2003 and 2008, respectively. His primary research areas include machine learning, smart grids, complex systems modeling and control, and traffic information processing and control. As the first author/corresponding author, he has published more than 20 papers in top and significant academic journals both domestically and internationally, including *IEEE Transactions* and *Automatica*.

Deming Lei received the MS degree in applied mathematics from Xi'an Jiaotong University, Xi'an, China in 1996, and the PhD degree in automation science and engineering from Shanghai Jiao Tong University, Shanghai, China in 2005. He is currently a professor at the School of Automation, Wuhan University of Technology, Wuhan, China. He has authored or coauthored more than 50 journal papers. His research interests include intelligent system optimization and control, production scheduling, etc.