Swarm and Evolutionary Computation 52 (2020) 100600

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Efficient multi-objective algorithm for the lot-streaming hybrid flowshop with variable sub-lots

Jun-qing Li [a,b,*], Xin-rui Tao [a], Bao-xian Jia [a], Yu-yan Han [a], Chuang Liu [a], Peng Duan [a], Zhi-xin Zheng [a], Hong-yan Sang [a]

[a] College of Computer Science, Liaocheng University, Liaocheng, 252059, PR China
[b] School of Information Science and Engineering, Shandong Normal University, Jinan 250014, China

ARTICLE INFO

Keywords:
Hybrid flowshop
Lot-streaming scheduling
Multi-objective optimization
Variable sub-lots

ABSTRACT

Recent years, the multi-objective evolutionary algorithm based on decomposition (MOEA/D) has been researched and applied for numerous optimization problems. In this study, we propose an improved version of MOEA/D with problem-specific heuristics, named PH-MOEAD, to solve the hybrid flowshop scheduling (HFS) lot-streaming problems, where the variable sub-lots constraint is considered to minimize four objectives, i.e., the penalty caused by the average sojourn time, the energy consumption in the last stage, as well as the earliness and the tardiness values. For solving this complex scheduling problem, each solution is coded by a two-vector-based solution representation, i.e., a sub-lot vector and a scheduling vector. Then, a novel mutation heuristic considering the permutations in the sub-lots is proposed, which can improve the exploitation abilities. Next, a problem-specific crossover heuristic is developed, which considered solutions with different sub-lot size, and therefore can make a solution feasible and enhance the exploration abilities of the algorithm as well. Moreover, several problem-specific lemmas are proposed and a right-shift heuristic based on them is subsequently developed, which can further improve the performance of the algorithm. Lastly, a population initialization mechanism is embedded that can assign a fit reference vector for each solution. Through comprehensive computational comparisons and statistical analysis, the highly effective performance of the proposed algorithm is favorably compared against several presented algorithms, both in solution quality and population diversity.

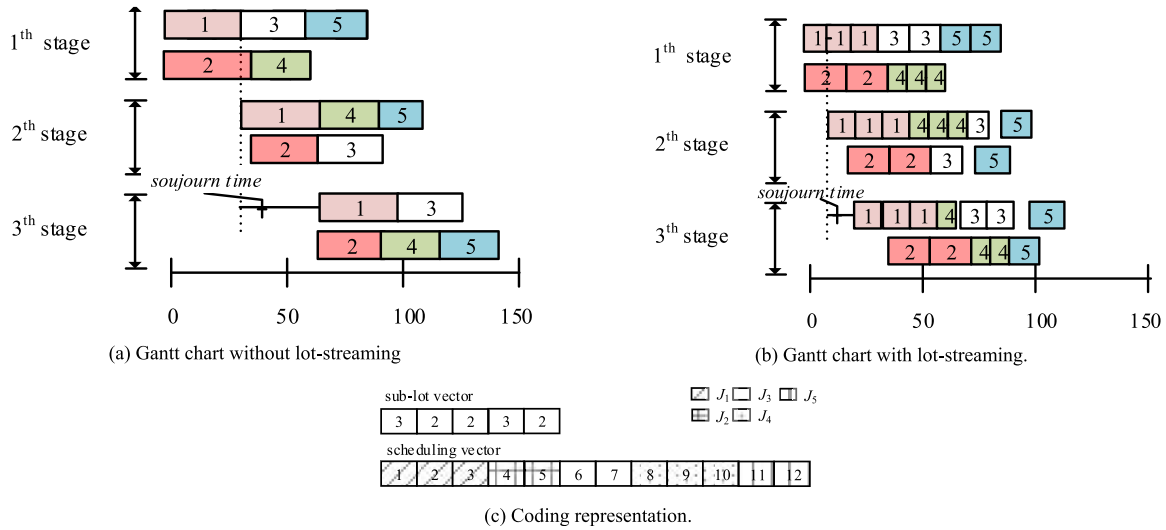
1. Introduction

In modern industrial production system, the scheduling problem has been widely applied and investigated [1–15], such as the hybrid flow shop scheduling (HFS) [1], parallel machine scheduling [2], flowshop scheduling [3–5], and flexible job shop scheduling problem [6]. The HFS is one of the typical scheduling problems, which is commonly considered as an NP-hard problem [1]. In HFS, there are commonly multiple stages, each stage contains parallel processing devices (which can be considered as a parallel machine scheduling problem), and each job should flow through each of the above stages according to the same sequence (which can be seen as a flowshop scheduling problem). In each stage, each job should select exactly one machine. Therefore, the performance of the system is mainly determined by the performance of each job in each stage. In recent years, how to schedule the jobs in the HFS system has gained more and more research focus. To solve a similar realistic problem, Pan et al. investigated a multi-objective hot-rolling scheduling problem in the compact strip production [7], and the distributed permutation flowshop scheduling problems [8,9]. Li et al. developed a hybrid fruit fly optimization algorithm (FOA) to solve the rescheduling problem in SCC systems [10]. Further, Yu et al. developed a heuristic to solve the SCC problem considering a job start-time delay event [11]. Li et al. investigated the HFS problem with operation skipping by utilizing an improved ABC algorithm [12]. Peng et al. solved the rescheduling in steelmaking-refining-continuous casting process by using an ABC algorithm [13]. Liu et al. solve a path planning problem for crowd evacuation in buildings [14]. It should be noted that, the above works related to the HFS problems considered each job as a whole part, which makes the downstream machine idle and lowers the efficiency of the system. Therefore, in the HFS systems, splitting a lot is both possible and desirable.

Lot streaming is the process to split a job into sub-lots and then all these sub-lots should be scheduled separately. Therefore, in the lot-streaming scheduling problems, there are generally three tasks to be

* Corresponding author. College of Computer Science, Liaocheng University, Liaocheng, 252059, PR China.
E-mail address: lijunqing@lcu-cs.com (J.-q. Li).

https://doi.org/10.1016/j.swevo.2019.100600
Received 16 January 2019; Received in revised form 25 August 2019; Accepted 22 October 2019
Available online 6 November 2019

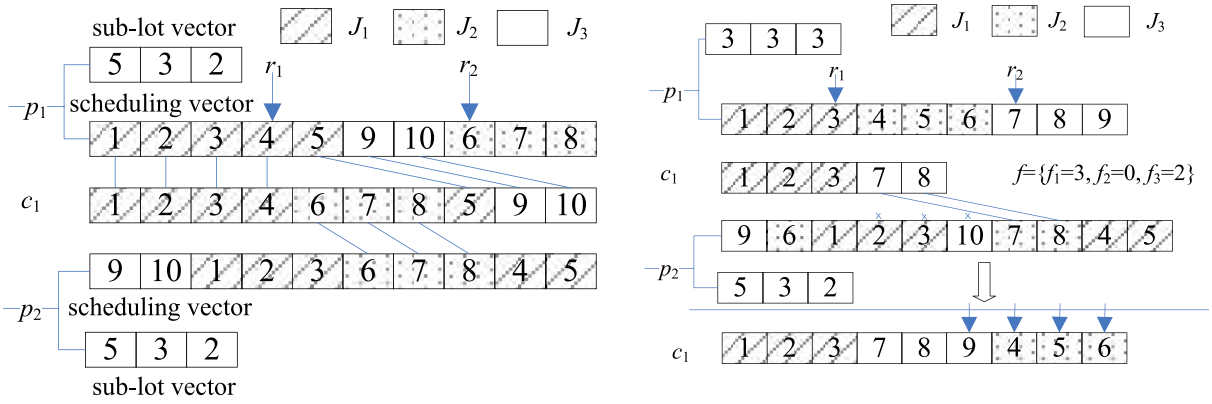


(a) Gantt chart without lot-streaming

(b) Gantt chart with lot-streaming.

(c) Coding representation.

Fig. 1. An example of solution representation.



(a) Crossover with equal sub-lot vector

(b) Crossover with different sub-lot vector

Fig. 2. The procedure of the V-Crossover heuristic.

completed, i.e., to decide the number of sub-lots, to decide the machine assignment for each sub-lot, and to schedule each sub-lot one each assigned machine. The lot-streaming technique has received increased attention [15] during recent years. An extensive survey of lot-streaming in flowshop systems can be found in Ref. [16]. In recent years, swarm intelligence algorithms have been applied to solve the lot-streaming flow shop problem. Tseng and Liao solved the problem with the minimization of the total weighted earliness and tardiness values by using a particle swarm optimization (PSO) algorithm [17]. Marimuthu et al. addressed the setup time problem by using evolutionary algorithms [18]. In Ref. [19], the same authors solved the same problem to minimize makespan by using an ant colony optimization (ACO) algorithm. Yoon and Ventura proposed a genetic algorithm (GA) to solve this problem [20]. Pan et al. studied the problem to minimize makespan by using an estimation of distribution algorithm (EDA) [21], a self-organizing migrating algorithm [22], an iterated local search algorithm (ILS) [23], and a discrete invasive weed optimization algorithm (IWO) [24] have also been applied to solve this problem. Recently, a hybrid algorithm has also been investigated to solve the problem, such as the combination of differential evolutionary (DE) and the PSO algorithms [25] and the combination of the sheep flock heredity algorithm and the ABC algorithm [26]. Meng et al. solved the integrated lot-streaming flow shop scheduling by using a migrating birds optimization (MBO) [27]. To solve this problem with multiple objectives, Han et al. proposed a multi-objective MBO algorithm [28]. Masmoudi et al. discussed a multi-item capacitated lot-sizing and scheduling problem in a flowshop system and developed a fix-and-relax heuristic [29].

Most of the papers we reviewed in the literature have studied lot streaming in flowshop systems. However, HFS is more realistic in the future of the industry; therefore, it is important to study lot streaming in HFS systems. Tsubone et al. studied the lot streaming problem in a two-stage HFS system [30]. Zhang et al. addressed lot streaming in an m-1 HFS system [31]. Then, Zhang et al. studied multi-job lot streaming to minimize the mean completion time in this same problem [32]. Liu addressed single-job lot streaming in an m+1 two-stage HFS system [33]. Naderi and Yazdani utilized an imperialist competitive algorithm to solve the HFS lot streaming problem with setup times [34]. Recently, Cheng et al. studied a single-lot HFS lot streaming problem in a 1 + 2 HFS system [35]. Very recently, Zhang et al. developed a modified MBO algorithm to solve the HFS lot-streaming problem [36]. Nejati et al. considered a two-stage assembly hybrid flow shop scheduling problem with a work shift constraint and investigated a hybrid algorithm by combining a GA with a simulated annealing (SA) algorithm [37]. Zohali et al. studied an integrated economic lot-sizing and sequencing problem (ELSP) in the HFS problem, which is one of the few literatures considering the variable lot-sizing problem [38]. However, the ELSP is different with the considered problem in this study, where the former is a cost-orient inventory model while the latter is a scheduling problem.

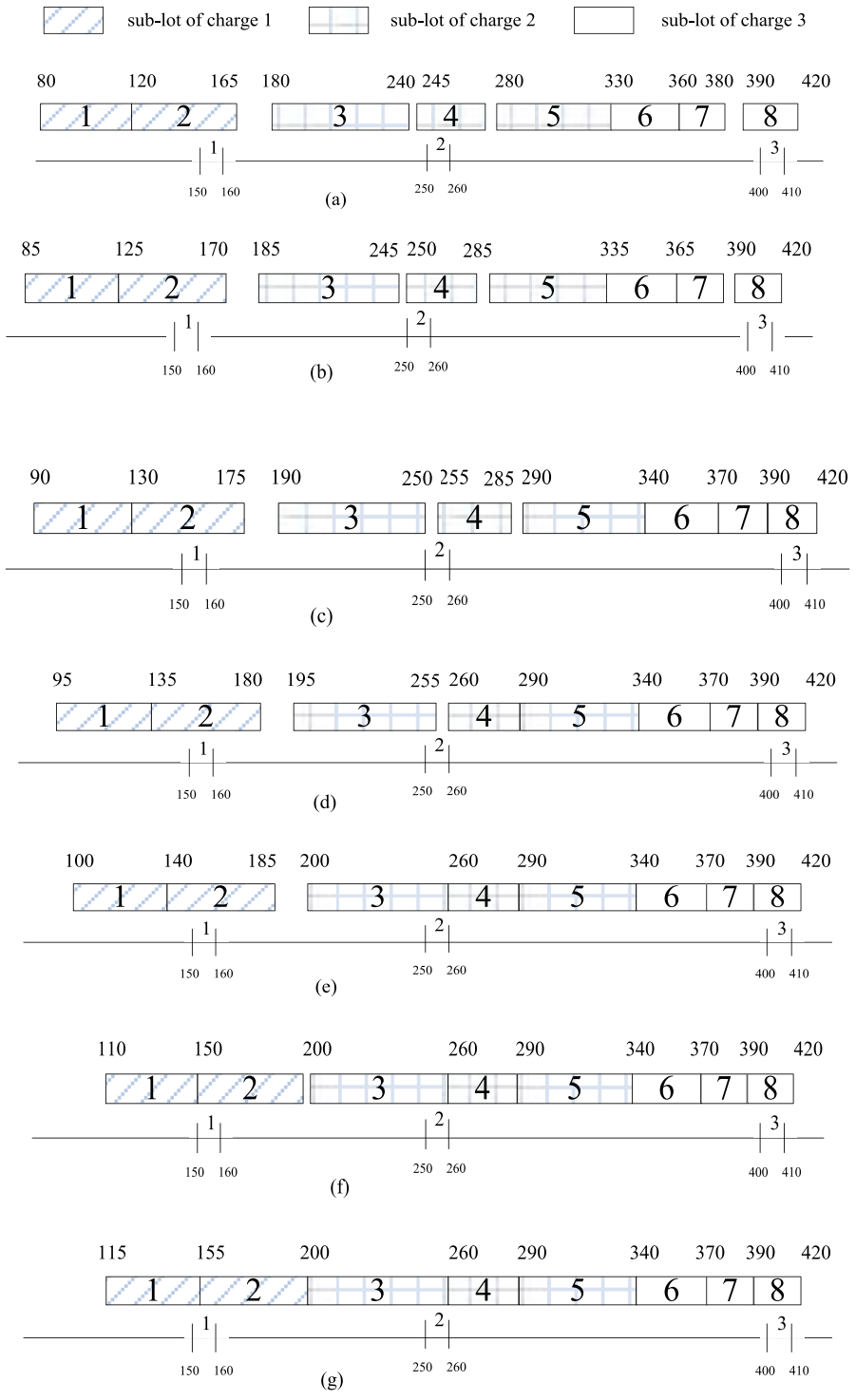


Fig. 3. Example for Algorithm 3.

Chen et al. solved the energy-efficient lot-streaming HFS problem by using a multi-objective optimization algorithm, where the problem is similar with this study except the due time window and the objectives [39]. It can be concluded from the literatures about the lot-streaming that, most of the literature assumes that the number of sub-lots is pre-defined and unchangeable, but in realistic systems, the number of sub-lots can always change according to the production performance.

In addition, most of the research work considers a single objective problem or weighted sum objectives, but the decision process is often multi-objective [40–45]. Meanwhile, during recent years, the multi-objective evolutionary algorithm based on decomposition (MOEA/D) has been researched and applied for numerous optimization problems [46–50]. Therefore, in this study, we consider multi-objective lot streaming in HFS systems with a flexible number of sub-lots.

The main challenges of the lot-streaming HFS are as follows: (1) how to define the optimal number of sub-lots for the considered problem, and therefore to minimize the considering multiple objectives; (2) how to apply the crossover operator to the solutions with different sub-lot size,

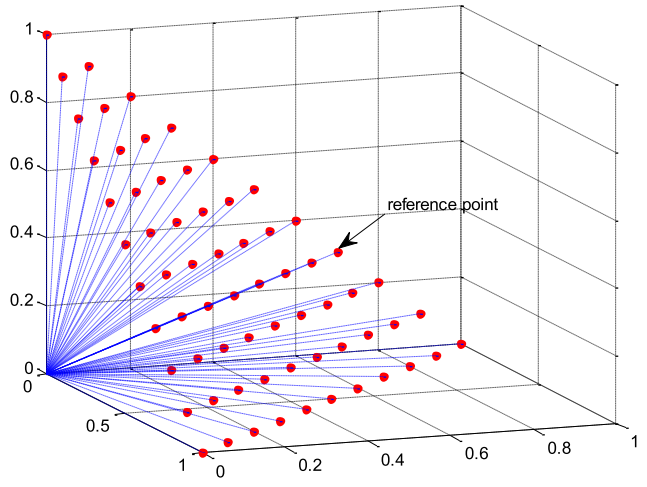


Fig. 4. Example of the reference points generalization.

and thus make the newly-generated solution feasible; and (3) how to design efficient and effective problem-specific heuristics and thus improve the performance of the algorithm. To solve the aforementioned challenges, we proposed a MOEA/D with problem-specific heuristics (PH-MOEAD), and the main contributions of this study are as follows:

- The lot-streaming HFS problem with a variable number of sub-lots is investigated, which has rarely been considered in the literature and which is common in realistic production systems;
- A variation crossover that considers the problem features is proposed to tackle two parent solutions with different sub-lot vectors;
- A right-shift heuristic considering the problem structure and objective characteristics is investigated to improve the solution quality;
- A novel mutation heuristic considering the permutations in the sub-lots is proposed, which can improve the exploitation abilities.

The rest of this paper is organized as follows. Section 2 briefly describes the problem. Next, the proposed algorithm, which embeds the problem-specific heuristics, is presented in Section 3. Section 4 illustrates the experimental results and makes comparisons to the performance results of algorithms from the literature to demonstrate the superiority of the proposed algorithm. Finally, the last section presents the concluding remarks and future research directions.

2. Problem description

In this study, we consider a lot-streaming HFS problem, where there are n jobs to be processed through h stages, and each stage contains several parallel machines. To make the production process more efficient, the lot streaming technique is desirable. In a lot streaming HFS system, each job can be split into several sub-lots, of which each sub-lot can be simultaneously processed on different machines in an independent and parallel manner.

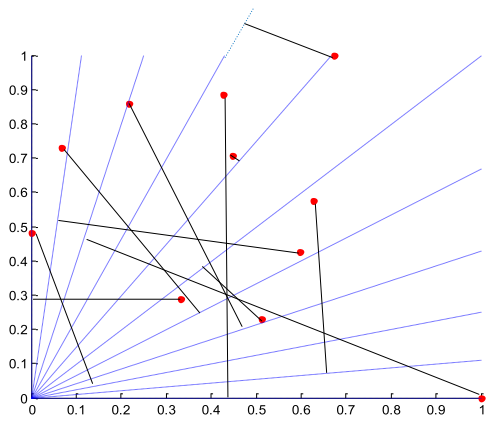
The problem is to determine the optimal: (i) number of sub-lots; (ii) sub-lot sizes, (iii) allocation of machines in each stage for each sub-lot; and (iv) the processing sequence of each sub-lot on each machine to minimize the four objectives simultaneously, i.e., the penalties caused by the average sojourn time, the energy consumption in the last stage, as well as the earliness and the tardiness values. It should be noted that the energy consumed in the last stage is very important for some industrial horizon, such as steelmaking casting system, and therefore the second objective function only considers the energy consumption in the last stage. Similar as most of the published literatures about the lot-streaming problems, in this study, we only consider the off-line features, that is, the real-time features are not considered.

(1) Assumptions

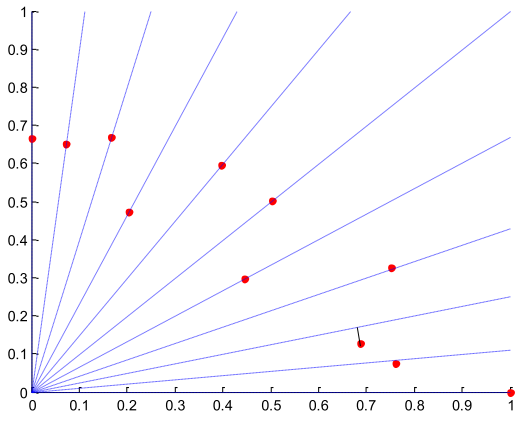
- The processing time is a positive integer for each sub-lot, which is pre-defined and deterministic.
- All the machines are available at time zero and remain continuously available during the entire production horizon.
- In each stage, each sub-lot of any job can select exactly one machine, and each machine can process only one sub-lot at any processing time.
- The processing of a sub-lot in the following stage can begin only after its completion in the previous stage.

Table 1
ANOVA table on scaled IGD values for the parameters.

Source	ss	DF	MS	F	Prob > F
v_n	0.00048	3	0.00016	5.4	0.0048
p_m	0.00112	3	0.00037	12.63	0.0000
p_c	0.00002	3	0.00001	0.17	0.913
$v_n * p_m$	0.00046	9	0.00005	1.72	0.1318
$v_n * p_c$	0.00021	9	0.00002	0.79	0.6299
$p_m * p_c$	0.00025	9	0.00003	0.95	0.5012
Error	0.0008	27	0.00003		
total	0.00333	63			

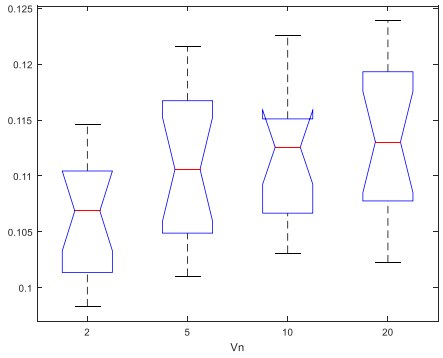


(a) Population without applying the initialization heuristic

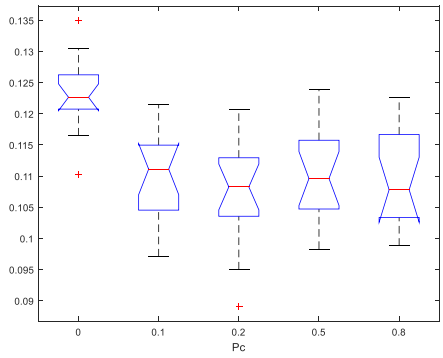


(b) Population after applying the initialization heuristic

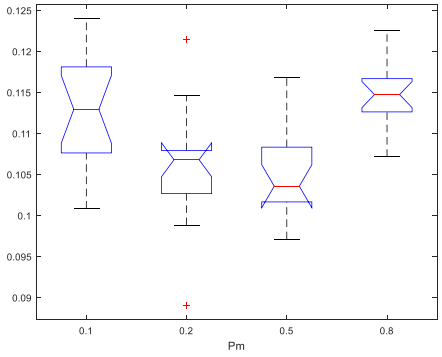
Fig. 5. Population initialization heuristic.



(a) 95% Confidence interval for v_n



(c) 95% Confidence interval for p_c



(b) 95% Confidence interval for p_m

Fig. 6. 95% Confidence interval for different selections of v_n, p_m, and p_c.

Table 2
Comparison results for the HFS lot-streaming with single-objective.

Instance	best	RPI values						dev					
		PH-MOEAD	EMBO	GA	GAR	DPSO	DABC	PH-MOEAD	EMBO	GA	GAR	DPSO	DABC
20 × 5	0.29	**0.29**	**0.29**	1.33	1.21	0.39	0.80	**0.00**	0.00	358.62	317.24	34.48	175.86
20 × 10	0.37	**0.37**	0.40	1.96	1.66	0.54	1.41	**0.00**	8.11	429.73	348.65	45.95	281.08
40 × 5	0.80	0.82	**0.80**	2.39	2.36	1.99	4.12	2.50	**0.00**	198.75	195.00	148.75	415.00
40 × 10	0.59	**0.59**	0.65	3.10	3.56	3.24	4.12	**0.00**	10.17	425.42	503.39	449.15	598.31
60 × 5	0.40	**0.40**	0.46	3.29	4.25	5.27	3.92	**0.00**	15.00	722.50	962.50	1217.50	880.00
60 × 10	0.78	**0.78**	0.82	2.40	3.32	4.22	7.62	**0.00**	5.13	207.69	325.64	441.03	876.92
80 × 5	0.33	**0.33**	0.43	2.36	2.11	4.40	6.66	**0.00**	30.30	615.15	539.39	1233.33	1918.18
80 × 10	0.81	**0.81**	0.86	2.72	3.61	6.80	4.47	**0.00**	6.17	235.80	345.68	739.51	451.85
100 × 5	0.28	**0.28**	0.39	2.36	4.97	4.05	3.74	**0.00**	39.29	742.86	1675.00	1346.43	1235.71
100 × 10	0.71	**0.71**	0.79	3.07	5.01	4.58	4.72	**0.00**	11.27	332.39	605.63	545.07	564.79
mean	0.54	**0.54**	0.59	2.50	3.21	3.55	4.16	**0.25**	12.54	426.89	581.81	620.12	739.77

- There are enough buffers between any two continuous stages, such that after the completion of the current stage, any sub-lot can be passed to the next stage immediately.
- Each job can be split into several sub-lots with the same lot sizes.

(2) Notation

Index:

- i: Index of jobs;
- j: Index of machines;
- k: Index of stages;
- l: Index of sub-lots;
- n: Total number of jobs;
- m: Total number of machines;
- h: Total number of stages;
- m_k: Number of machines in stage k;
- $p_{i,j,k,l}$: The processing time of the sub-lot l of job i in stage k on machine j;
- SD_i, ED_i: the starting and completion of the due time window of the ith job in the horizon, $i = 1,2, \ldots,n$;
- $T_{k,k+1}$: transfer time from stage k to stage $k+1$;
- $PW_{j,k}$: the working energy consumption of the jth machine in stage k;
- ω_1, ω_2, ω_3, and ω_4: the weight coefficients for the four objectives, respectively.

Decision variables:

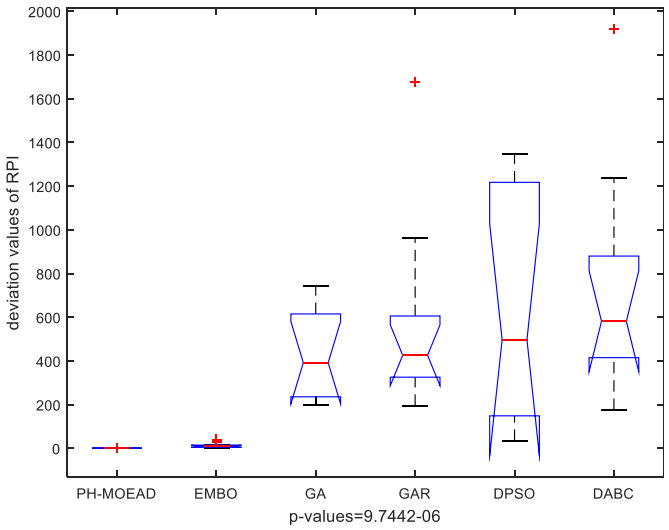


Fig. 7. Means and 95% LSD interval for PH-MOEAD, EMBO, GA, GAR, DPSO and DABC (CPU = 30).

- N: total number of sub-lots in the system;
- $n_{i,k}$: number of sub-lots of job i in stage k;
- $s_{i,k,l}$: start time of sub-lot l of job i in stage k;
- $c_{i,k,l}$: completion time of sub-lot l of job i in stage k;
- c_{max}: the completion time of the last sub-lot on the last machine in the last stage.
- $x_{i,l,k,j}$: a binary value that is set to 1 if sub-lot l of job i in stage k is assigned to machine j; otherwise, $x_{i,l,k,j}$ is set to 0;
- $z^t_{j,k}$: a binary value that is set to 1 if the jth machine in stage k is working at time t; otherwise, $z^t_{j,k}$ is set to 0;
- $\pi_{i,k,j,r,l}$: a binary value that is set to 1 if sub-lot l of job i in stage k is assigned to machine j in priority r; otherwise, $\pi_{i,k,j,r,l}$ is set to 0;
- $s_{k,j,r}$: start time of the sub-lot processing in machine j in priority r in stage k;
- $q_{k,j}$: number of sub-lots assigned to machine j in stage k

The main mathematical model for the considered problem is given as follows:min $f = \{F_1, F_2, F_3, F_4\}$

$$F_1 = \left(\sum_{i=1}^{n}\sum_{l=1}^{n_{i,h}}(s_{i,h,l} - c_{i,1,l})\right)\Big/N \tag{1}$$

$$F_2 = \sum_{t=1}^{C_{max}}\sum_{j=1}^{m_h}\left(PW_{j,h}\cdot z^t_{j,h}\right) \tag{2}$$

$$F_3 = \sum_{i=1}^{n}\sum_{l=1}^{n_{i,h}}\max(SD_i - s_{i,h,l}, 0) \tag{3}$$

$$F_4 = \sum_{i=1}^{n}\sum_{l=1}^{n_{i,h}}\max(s_{i,h,l} - ED_i, 0) \tag{4}$$

$$\sum_{j=1}^{m_k} x_{i,l,k,j} = 1,\ \forall i, l, k \tag{5}$$

$$s_{i,k+1,l} \geq \left(s_{i,k,l} + p_{i,j,k,l} + T_{k,k+1}\right).\ \forall i, j,\ l,\ k = 1, ..., h-1 \tag{6}$$

$$s_{k,j,r} + p_{i,j,k,l}\cdot\pi_{i,k,j,r,l} \leq s_{k,j,r+1},\ \forall i, j, l, k, r = 1, ..., q_{kj} - 1 \tag{7}$$

$$\sum_{r=1}^{q_{k,j}}\pi_{i,k,j,r,l} = x_{i,k,j,l},\ \forall i, j, l \in \{1, \ldots, n_{i,h} - 1\} \tag{8}$$

Table 3
Comparisons of the HV values for the mutation heuristic.

Instance	best	PH-MOEAD			PH-MOEAD-NM			dev$^+$	
		max	avg	min	max	avg	min	PH-MOEAD	PH-MOEAD-NC
20 × 5	0.53	0.53	0.51	0.46	0.41	0.39	0.38	**0.00**	23.82
20 × 10	0.54	0.54	0.50	0.48	0.53	0.52	0.51	**0.00**	1.88
40 × 5	0.42	0.42	0.39	0.38	0.40	0.40	0.39	**0.00**	5.56
40 × 10	0.47	0.47	0.47	0.47	0.47	0.46	0.45	**0.00**	1.17
60 × 5	0.39	0.39	0.38	0.37	0.39	0.37	0.32	**0.00**	1.63
60 × 10	0.47	0.47	0.46	0.45	0.47	0.46	0.42	0.26	**0.00**
80 × 5	0.48	0.48	0.43	0.42	0.44	0.41	0.39	**0.00**	7.69
80 × 10	0.50	0.49	0.49	0.49	0.49	0.49	0.49	**0.00**	0.10
100 × 5	0.43	0.43	0.41	0.37	0.40	0.38	0.36	**0.00**	7.11
100 × 10	0.54	0.54	0.53	0.53	0.54	0.53	0.53	0.08	**0.00**
mean	0.48	0.48	0.46	0.44	0.45	0.44	0.42	**0.03**	4.90

Table 4
Comparisons of the HV values for the right-shift heuristic.

Instance	best	PH-MOEAD			PH-MOEAD-NR			dev$^+$	
		max	avg	min	max	avg	min	PH-MOEAD	PH-MOEAD-NR
20–5	0.58	0.58	0.58	0.55	0.56	0.56	0.55	0.00	3.01
20–10	0.69	0.69	0.69	0.68	0.67	0.66	0.66	0.00	3.00
40–5	0.60	0.60	0.58	0.57	0.53	0.53	0.52	0.00	11.27
40–10	0.43	0.43	0.43	0.42	0.37	0.37	0.37	0.00	13.60
60–5	0.48	0.44	0.43	0.42	0.48	0.47	0.46	7.90	0.00
60–10	0.68	0.67	0.67	0.66	0.68	0.67	0.67	0.94	0.00
80–5	0.66	0.66	0.64	0.62	0.55	0.51	0.51	0.00	17.15
80–10	0.68	0.68	0.67	0.67	0.53	0.52	0.51	0.00	22.02
100–5	0.51	0.51	0.51	0.50	0.44	0.43	0.43	0.00	14.42
100–10	0.53	0.53	0.53	0.52	0.53	0.52	0.52	0.00	1.30
mean	0.58	0.58	0.57	0.56	0.53	0.52	0.52	0.88	8.58

Table 5
Comparisons of the HV values for the crossover heuristic.

Instance	Best	PH-MOEAD			PH-MOEAD-NC			dev^+	
		max	avg	min	max	avg	min	PH-MOEAD	PH-MOEAD-NC
20–5	0.56	0.56	0.53	0.52	0.53	0.50	0.49	0.00	5.27
20–10	0.53	0.53	0.52	0.48	0.51	0.47	0.32	0.00	4.66
40–5	0.58	0.58	0.53	0.52	0.58	0.56	0.51	0.00	0.23
40–10	0.19	0.19	0.18	0.15	0.12	0.11	0.07	0.00	35.52
60–5	0.32	0.32	0.26	0.24	0.30	0.24	0.22	0.00	7.04
60–10	0.77	0.68	0.68	0.68	0.77	0.77	0.76	11.01	0.00
80–5	0.68	0.68	0.67	0.64	0.65	0.64	0.60	0.00	4.32
80–10	0.19	0.19	0.16	0.14	0.17	0.15	0.13	0.00	11.23
100–5	0.43	0.43	0.41	0.10	0.41	0.37	0.28	0.00	4.26
100–10	0.56	0.56	0.54	0.53	0.18	0.17	0.13	0.00	68.03
mean	0.48	0.47	0.45	0.40	0.42	0.40	0.35	1.10	14.06

Table 6
Comparisons of the HV values between MOEA/D, NSGA-II, DBEA, EMBO and PH-MOEAD ($u = 30$).

Instance	MOEA/D			NSGA-II			DBEA			EMBO			PH-MOEAD		
	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min
Case1	0.495	0.486	0.481	0.443	0.438	0.436	0.583	0.573	0.559	0.521	0.514	0.51	**0.602**	**0.59**	**0.587**
Case2	0.517	0.508	0.504	0.135	0.135	0.132	0.443	0.431	0.41	0.437	0.433	0.424	**0.893**	**0.89**	**0.885**
Case3	0.435	0.421	0.415	0.328	0.312	0.306	0.45	0.432	0.421	0.4	0.385	0.378	**0.543**	**0.522**	**0.504**
Case4	0.3	0.298	0.297	0.055	0.055	0.054	**0.508**	**0.499**	**0.491**	0.192	0.186	0.18	0.423	0.418	0.403
Case5	0.329	0.327	0.325	0.071	0.071	0.071	0.366	0.363	0.361	0.276	0.272	0.266	**0.48**	**0.476**	**0.47**
Case6	0.286	0.282	0.279	0.187	0.186	0.185	0.256	0.247	0.245	0.231	0.23	0.229	**0.991**	**0.991**	**0.99**
Case7	0.47	0.463	0.458	0.34	0.339	0.338	0.508	0.505	0.503	0.467	0.465	0.462	**0.869**	**0.868**	**0.866**
Case8	0.193	0.191	0.19	0.061	0.061	0.06	0.317	0.306	0.3	0.108	0.107	0.106	**0.434**	**0.432**	**0.429**
Case9	0.421	0.397	0.389	0.064	0.061	0.06	0.353	0.323	0.308	0.131	0.121	0.118	**0.483**	**0.451**	**0.439**
Case10	0.123	0.122	0.121	0.057	0.057	0.057	0.534	0.53	0.523	0.19	0.188	0.186	**0.472**	**0.467**	**0.463**
Mean	0.357	0.349	0.346	0.174	0.171	0.17	0.432	0.421	0.412	0.295	0.29	0.286	**0.619**	**0.611**	**0.604**

Table 7
Comparisons of the HV values between MOEA/D, NSGA-II, DBEA, EMBO and PH-MOEAD ($u = 50$).

Instance	MOEA/D			NSGA-II			DBEA			EMBO			PH-MOEAD		
	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min
Case1	0.595	0.572	0.553	0.453	0.449	0.443	**0.613**	**0.601**	**0.598**	0.528	0.526	0.521	0.596	0.593	0.586
Case2	0.513	0.499	0.49	0.135	0.135	0.134	0.427	0.423	0.42	0.447	0.445	0.443	**0.888**	**0.886**	**0.884**
Case3	0.402	0.399	0.396	0.309	0.304	0.303	0.41	0.403	0.389	0.374	0.372	0.371	**0.524**	**0.516**	**0.506**
Case4	0.262	0.248	0.242	0.048	0.046	0.044	0.429	0.399	0.384	0.164	0.148	0.141	**0.555**	**0.532**	**0.519**
Case5	0.308	0.285	0.269	0.069	0.064	0.063	0.342	0.317	0.303	0.281	0.266	0.244	**0.515**	**0.505**	**0.501**
Case6	0.283	0.28	0.276	0.189	0.188	0.187	0.262	0.256	0.251	0.238	0.227	0.225	**0.992**	**0.991**	**0.991**
Case7	0.502	0.48	0.448	0.369	0.365	0.342	0.514	0.511	0.508	0.468	0.465	0.457	**0.879**	**0.877**	**0.873**
Case8	0.17	0.164	0.162	0.055	0.052	0.052	0.289	0.279	0.274	0.112	0.106	0.093	**0.434**	**0.422**	**0.411**
Case9	0.392	0.379	0.365	0.059	0.057	0.056	0.418	0.385	0.361	0.106	0.105	0.104	**0.52**	**0.505**	**0.494**
Case10	0.103	0.102	0.101	0.045	0.045	0.044	0.478	0.471	0.464	0.177	0.175	0.173	**0.566**	**0.562**	**0.55**
Mean	0.353	0.341	0.33	0.173	0.17	0.167	0.418	0.405	0.395	0.29	0.284	0.277	**0.647**	**0.639**	**0.632**

Table 8
Comparisons of the HV values between MOEA/D, NSGA-II, DBEA, EMBO and PH-MOEAD ($u = 100$).

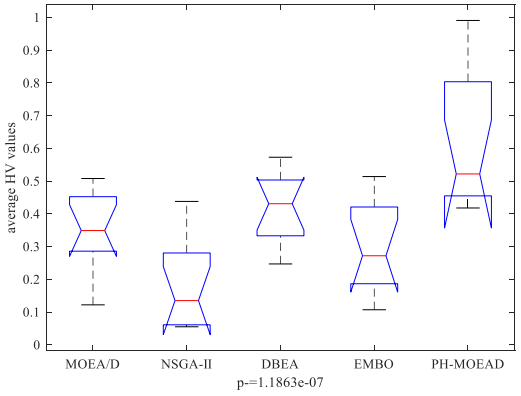
Instance	MOEA/D			NSGA-II			DBEA			EMBO			PH-MOEAD		
	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min
Case1	0.589	0.583	0.551	0.457	0.455	0.448	0.598	0.592	0.589	0.518	0.515	0.51	**0.629**	**0.622**	**0.614**
Case2	0.51	0.494	0.489	0.135	0.134	0.133	0.423	0.421	0.419	0.443	0.442	0.441	**0.887**	**0.885**	**0.883**
Case3	0.483	0.466	0.449	0.366	0.358	0.35	0.521	0.492	0.462	0.45	0.434	0.417	**0.595**	**0.565**	**0.532**
Case4	0.283	0.273	0.258	0.052	0.05	0.048	0.461	0.445	0.402	0.174	0.166	0.145	**0.526**	**0.514**	**0.479**
Case5	0.402	0.4	0.398	0.102	0.1	0.096	0.446	0.44	0.433	0.376	0.372	0.369	**0.454**	**0.45**	**0.447**
Case6	0.3	0.297	0.293	0.201	0.2	0.199	0.275	0.271	0.266	0.253	0.242	0.238	**0.996**	**0.995**	**0.994**
Case7	0.516	0.481	0.465	0.377	0.376	0.375	0.529	0.523	0.518	0.478	0.476	0.469	**0.864**	**0.862**	**0.859**
Case8	0.195	0.194	0.192	0.06	0.06	0.059	0.335	0.333	0.331	0.13	0.12	0.111	**0.414**	**0.371**	**0.344**
Case9	0.357	0.342	0.334	0.049	0.049	0.048	0.351	0.346	0.337	0.099	0.098	0.094	**0.455**	**0.449**	**0.439**
Case10	0.103	0.1	0.099	0.049	0.048	0.048	0.459	0.452	0.445	0.167	0.159	0.157	**0.512**	**0.502**	**0.472**
Mean	0.374	0.363	0.353	0.185	0.183	0.181	0.44	0.431	0.42	0.309	0.302	0.295	**0.633**	**0.622**	**0.606**

$$x_{i,l,k,j} \in \{0,1\},\ \forall i,l,k \tag{9}$$

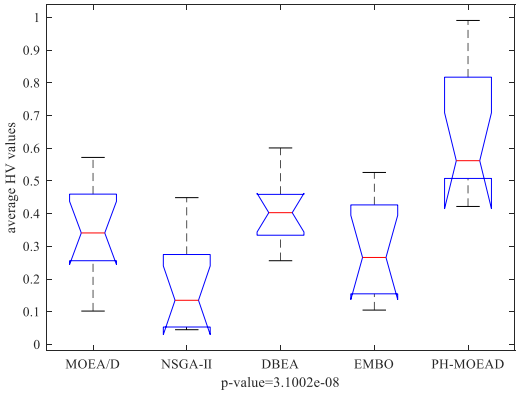
$$z^t_{j,k} \in \{0,1\},\ \forall j,k \tag{10}$$

Table 9
Comparisons of the IGD values between MOEA/D, NSGA-II, DBEA, EMBO and PH-MOEAD ($u = 100$).

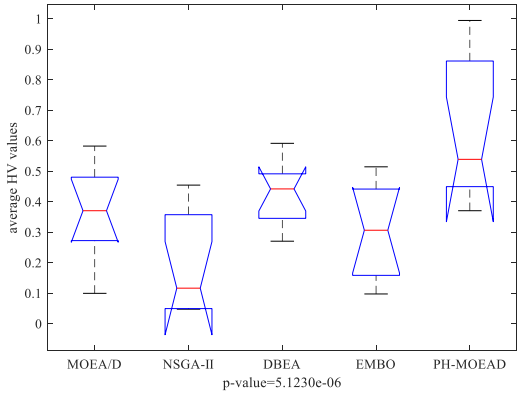
Instance	MOEA/D			NSGA-II			DBEA			EMBO			PH-MOEAD		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max	min	avg	max
20–5	0.17	**0.19**	0.22	0.17	0.25	0.36	0.21	0.23	0.24	0.20	0.21	**0.21**	**0.16**	**0.19**	**0.21**
20–10	0.38	0.45	0.57	0.05	**0.17**	0.56	0.13	0.68	0.73	0.58	0.58	0.58	**0.04**	0.21	**0.25**
40–5	0.16	0.18	0.20	0.26	0.38	0.45	**0.14**	**0.17**	**0.19**	0.16	**0.17**	0.20	0.16	0.19	0.20
40–10	**0.14**	**0.16**	**0.16**	0.25	0.28	0.35	0.15	0.18	0.23	0.21	0.22	0.22	0.15	0.23	0.42
60–5	0.27	0.28	0.29	0.23	0.34	0.45	0.21	0.24	0.27	0.31	0.31	0.31	**0.18**	**0.19**	**0.19**
60–10	0.07	0.44	0.69	0.07	0.16	0.38	0.36	0.54	0.88	0.55	0.61	0.95	**0.04**	**0.06**	**0.07**
80–5	**0.19**	**0.20**	**0.21**	0.31	0.37	0.44	0.30	0.33	0.37	0.32	0.35	0.36	0.28	0.29	0.39
80–10	**0.16**	**0.17**	**0.18**	0.24	0.33	0.36	0.23	0.23	0.24	0.41	0.43	0.44	0.24	0.25	0.26
100–5	0.34	0.41	0.44	0.27	0.29	0.36	0.21	0.23	**0.26**	0.40	0.40	0.40	**0.16**	**0.20**	0.28
100–10	0.25	0.28	0.29	0.17	0.21	0.25	0.17	0.21	**0.23**	0.31	0.32	0.33	**0.10**	**0.14**	0.30
20–5	0.21	0.28	0.33	0.20	0.28	0.40	0.21	0.30	0.36	0.35	0.36	0.40	**0.15**	**0.20**	**0.26**



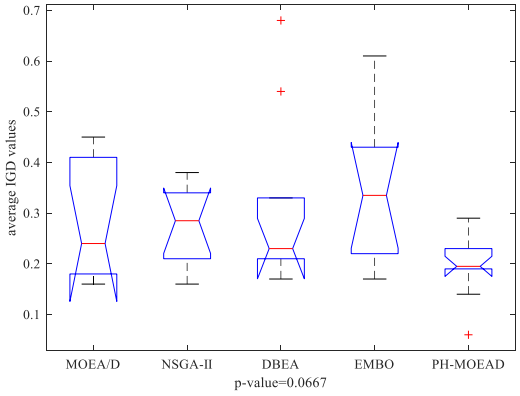
(a) ANOVA results for average values of HV comparisons (u=30).



(b) ANOVA results for average values of HV comparisons (u=50).



(c) ANOVA results for average values of HV comparisons (u=100).



(d) ANOVA results for average values of IGD comparisons (u=100).

Fig. 8. Means and 95% LSD interval for MOEA/D, NSGA-II, DBEA, EMBO, and PH-MOEAD.

$$\pi_{i,k,j,r,l} \in \{0,1\}, \forall i,l,j,k,r = 1,\ldots,q_{k,j} \tag{11}$$

The objective functions (1)–(4) minimize the penalty caused by the average sojourn time, the energy consumption in the last stage, as well as the earliness and the tardiness values. Constraint (5) ensures that a job must select only one machine in each stage. Constraint (6) guarantees that, for two consecutive sub-lots of each job, it should be started after the completion time of its predecessor stage plus the transfer times between the two stages. Constraints (7) and (8) are used to avoid an overlap of the processing times of two sub-lots on a machine. Constraints (9)–(11) define the value ranges for the decision variables.

3. The proposed algorithm

In this section, we describe the detailed implementation of the proposed algorithm, i.e., PH-MOEAD, to solve the hybrid flowshop scheduling (HFS) problem, in which lot-streaming constraints are considered. First, we describe the framework of the proposed algorithm. Then, the problem-specific heuristics including coding, mutation, crossover, and right-shift heuristic are presented. Finally, the multi-objective heuristics including reference regions generalization, population initialization, and neighborhood adaption heuristics are illustrated.

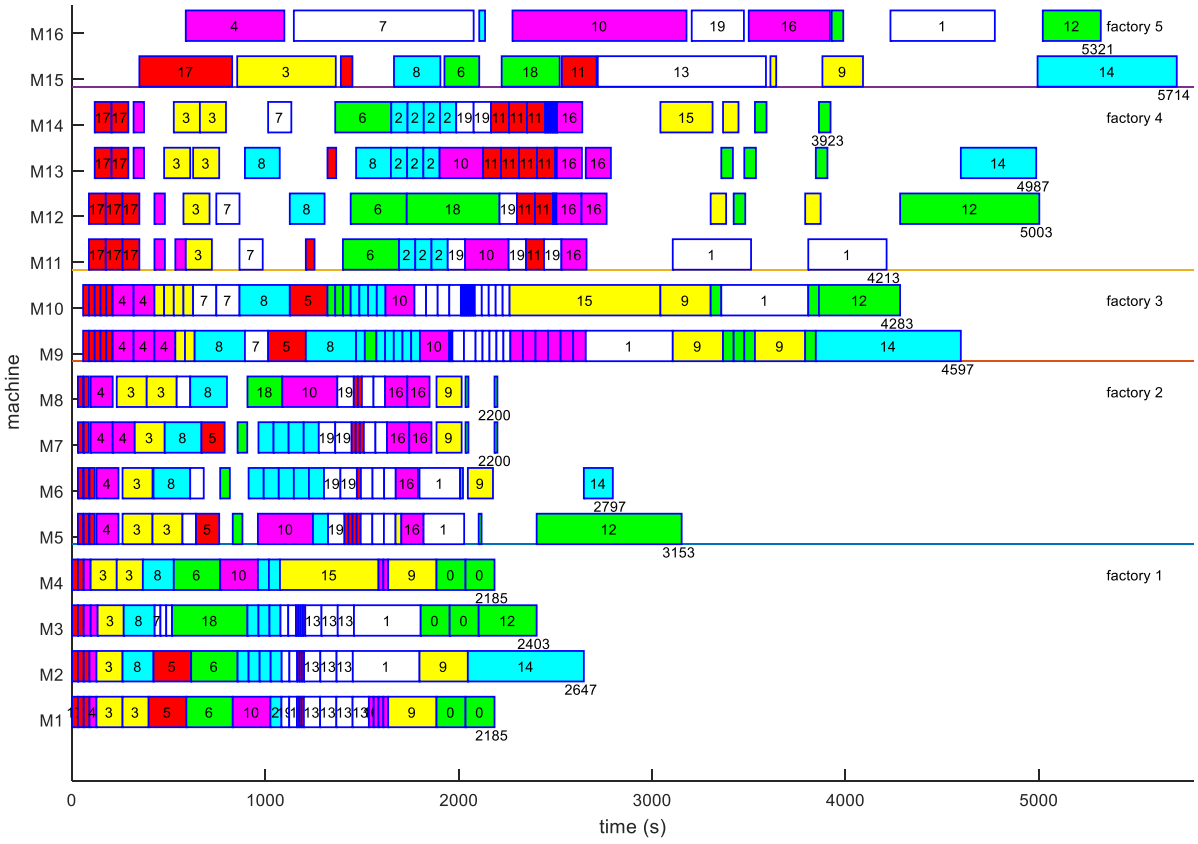


Fig. 9. Gantt chart for a solution for one of the 20-5 instance.

3.1. Framework of the proposed algorithm

Algorithm 1
General Framework of PH-MOEAD.

Algorithm 1: General Framework of PH-MOEAD
input: system parameters, including the population size P_{size} and the stop criterion,
output: the final Pareto archive set PA
1. $[P, W, E] \leftarrow$Initialization() as listed in sub-section 3.7; // P is the parent population, W is the weight vector set or reference lines, and E is the neighborhood index set
2. Perform following steps v_n times
3. **while** stop criterion is not satisfied **do**
4. **for** $i \leftarrow N$ **do**
5. $P' \leftarrow$Mating_selection($E(i)$, P); //select the parent solutions from the neighborhood region
6. $\boldsymbol{S} \leftarrow$GA_operator($P'$) //as illustrated in sub-section 3.3 and 3.4
7. $\boldsymbol{S} \leftarrow$right-shift($S$) //as illustrated in sub-section 3.5
8. **foreach** $x_c \in S$ **do** //x_c is an offspring
9. $P \leftarrow$Update_population(P', x_c) //as illustrated in the canonical MOEA/D
10. $PA \leftarrow$Update_PAS(PA, x_c) //as illustrated in the canonical MOEA/D
11. **end**
12. perform the neighborhood adaption heuristic illustrated in sub-section 3.8
13. **end**

Algorithm 1 presents the general framework of the proposed PH-MOEAD. First, the initialization procedure discussed in sub-section 3.7 generates P_{size} initial solutions with encoding as in sub-section 3.2, and each solution is assigned to the most suitable weighted coefficient or reference region generated as in sub-section 3.6. Following the initial procedures, the main while procedure begins. During the main while, we randomly select two parent solutions in the neighborhood region to generate an offspring by using the operators discussed in sub-section 3.3 and sub-section 3.4, and use the newly generated offspring solution to update the Pareto level and the parent population. In the following sub-sections, the implementation details of each component in the proposed PH-MOEAD will be explained step by step.

3.2. Encoding

To represent a solution, we introduce a simple encoding that contains two vectors. The first vector, named the sub-lot vector, lists the number of sub-lots for each job. The second vector, named the scheduling vector, gives the processing sequence for each sub-lot. Fig. 1 (a) shows a Gantt chart without lot-streaming, and Fig. 1 (b) gives a Gantt chart with lot-streaming. In this example given in Fig. 1 (b), the number of sub-lots for each job is {3,2,2,3,2}, which means that the first job is split into

three sub-lots, and the second job is split into two parts and so on. The second vector displays the processing sequence of each sub-lot in each stage. One example vector is {1,2,3,4,5,6,7,8,9,10,11,12}, which means that, in the first stage, each sub-lot of the first job is processed immediately on the assigned machine, and then the sub-lots of the second job are scheduled. The last job to be considered is the fifth job. It should be noted that, in the following stages, each sub-lot of the same job can be transmitted and processed immediately when the assigned machine is idle without considering other sub-lots. That is, each sub-lot of the same job is processed independently. Fig. 1 (c) illustrates the coding representation of the example solution.

3.3. Mutation heuristic

To generate a different solution from the initial solution, a mutation operator has commonly been used in the literature. In this study, two types of mutation operators are used.

The first type of mutation operator generates a different vector for the sub-lot vector. For this type of operator, we use a simple method, that is, we randomly select an element in this vector and then randomly assign another available value for this selected element. It should be noted that, this type of mutation is special for the lot-streaming with variable sub-lots, and therefore can complete the exploitation task for the considered problem.

The second type of mutation operator generates a different vector for the scheduling vector. For this type of operator, we use the two commonly used mutation operators, i.e., swap and insertion. The detailed procedures of these mutation operators are as follows: (1) for the swap operator, we randomly select two elements in this vector and then swap these two elements; and (2) for the insertion operator, we randomly select two elements in this vector and then insert the successor operation into the position before the previous operation.

3.4. Crossover heuristic

In the proposed algorithm, we adopt a two-point crossover operator for both vectors. The main difficulty in designing the crossover operator is that the two parent solutions may have different sub-lot vectors, that is, their first vectors are different. To tackle this issue, we propose a variation crossover operator named V-Crossover (see Fig. 2). The detailed implementation of the V-Crossover is given in Algorithm 2.

Algorithm 2
V-Crossover.

Algorithm 2: V-Crossover
input: two parent solutions p_1 and p_2
output: the offspring solution c_1

```
h1←the length of the sub-lot vector of p1, h2←the length of the sub-lot vector of p2
if h1= h2 then //(as illustrated in Fig. 2(a)
    randomly select two positions r1 and r2 in the sub-lot vector
    copy the elements in [0, r1] from p1 to c1
    copy the elements in [r1, r2] from p2 to c1
    copy the elements in [r2,|h1|] from p1 to c1
    delete the duplicate element in c1 and insert the remaining elements that have not been included in c1
    according to their sequence in p1.
otherwise //(as illustrated in Fig. 2(b)
    len←min{|h1|,|h2|}
    randomly select two positions r1 and r2 in the range of [0, len] in the sub-lot vector
    copy the elements in [0, r1] from p1 to c1
    for j←r1 to r2 do
        if p2[j]< len then // p2[j] is the j^th element of p2
            copy the elements in position j from p2 to c1
        end
    end
    copy the elements in [r2,|h1|] from p1 to c1
    let f={f1=0, f2=0,…, fn=0}, where fi represents the occurrence number of job i and n is the total number
    of jobs.
    delete the duplicate element in c1 and count the occurrence number fi for each job.
    for j←1 to |c1| do
        compute the job number i containing the j^th sub-lot
        if h1(i)- fi >0 then
            insert the remaining h1(i)- fi sub-lot number of job i immediately after the last sub-lot of
            job i.
        end
    end
end
```

3.5. Right-shift heuristic

In this study, we consider four objectives simultaneously. Based on the decomposition-based MOEA/D approach, each solution has an assigned weight vector or reference line used to compute a determined value. Therefore, in computing the fitness value for each solution, the process is similar to the weighted sum method. Considering the four problem-specific objectives, we propose a novel right-shift heuristic, which can clearly improve the solution quality.

Let S_E denotes the sub-lots which can be considered to be right shift and its start time $s_{i,h,l} < SD_i$, S_D denotes the sub-lots that can be considered to be right shift and its start time $SD_i \le s_{i,h,l} \le ED_i$, and S_T denotes the sub-lots which can be considered to be right shift and its start time $s_{i,h,l} > ED_i$. Let c_1, c_2, c_3, and c_4 represents the changes in the four objectives after right shift one time unit. We have following three lemmas:

Lemma 1. Given a sub-lot l, let $c_1=\frac{|S_E|+|S_D|+|S_T|}{N}\times\ \omega_1$, $c_2=SW_{j,h}\times\omega_2$, $c_3=|S_E|\times\omega_3$, $c_4=|S_T|\times\omega_4$, if $c_2+c_3>c_1+c_4$, then we can right shift all sub-lots on the same machine with and being processed before l.

Proof. In this case, we should consider the increasing penalties caused by the average sojourn time ($c_1=\frac{|S_E|+|S_D|+|S_T|}{N}\times\ \omega_1$) and the tardiness penalty ($c_4=\ |S_T|\times\ \omega_4$), and the decreasing penalties caused by the

earliness penalty ($c_3=|S_E|\times\omega_3$) and the standby energy consumption ($c_2=SW_{j,h}\times\omega_2$). Because $c_2+c_3>c_1+c_4$, this means that the decreasing penalties are bigger than the increasing penalties, and we can right shift the considered sub-lots.

Lemma 2. Given a sub-lot l, let $c_1=\frac{|S_E|+|S_D|}{N}\times\omega_1$, $c_2=SW_{j,h}\times\omega_2$, $c_3=|S_E|\times\omega_3$, if $c_2+c_3>c_1$, then we can shift each sub-lot processing before l and in $\{S_E,S_D\}$ to the right.

Proof. In this case, the sub-lots being in S_E or S_D sets and on the same machine with the given sub-lot l can be right shifted simultaneously. The right shift condition is $c_2+c_3>c_1$.

Lemma 3. Given a sub-lot l, let $c_1=\frac{|S_E|}{N}\times\omega_1$, $c_2=SW_{j,h}\times\omega_2$, $c_3=|S_E|\times\omega_3$, if $c_2+c_3>c_1$, then we can shift each sub-lot processing before l and in $\{S_E\}$ to the right.

Proof. In this case, the sub-lots being in S_E and on the same machine with the given sub-lot l can be right shifted simultaneously. The right shift condition is $c_2+c_3>c_1$ and $|S_E|>0$.

The time complexity of the right-shift heuristic is $O(n^2m)$, and the detailed implementation of the right-shift heuristic is illustrated in Algorithms 3 and 4.

Here, we give an example to illustrate the right shift procedures. Let $\omega_1=0.4$, $\omega_2=0.1$, $\omega_3=0.1$, $\omega_4=0.3$, and $SW_{j,h}=1.2$, In Fig. 3(a), given that the currently considered sub-lot is l_8, and that the other sub-lots processed before l_8 can be divided into three sets, i.e., $S_E=\{1,2,3,4,6,7\}$, $S_D=\{\}$, $S_T=\{5\}$. It should be noted that the sub-lots belonging to the same job have the same due data window. The earliness values of the sub-lots in S_E are 70, 30, 70, 5, 70, and 40, respectively.

First, we consider the while procedure listed in Algorithm 4 from line 10 to 18, where $c_1=(|S_E|+|S_D|+|S_T|)/N\times\omega_1=(7/8)\times 0.4=0.35$, $c_2=SW_{j,h}\times\omega_2=0.12$, $c_3=|S_E|\times\omega_3=0.6$, $c_4=|S_T|\times\omega_4=0.3$. Therefore, $c_2+c_3>c_1+c_4$, and the first loop procedure will be performed. Here, $\theta_1=\min\{70,30,70,5,70,40\}=5$, $\theta_2=\min\{260-250\}=10$, $\gamma=390-380=10$, and $\theta=\min\{\theta_1,\theta_2,\gamma\}=5$. Therefore, we can right shift all sub-lots processing before l_8 by 5 time units. The resulting Gantt chart is given in Fig. 3(b). Then, the resulting three set $S_E=\{1,2,3,6,7\}$, $S_D=\{4\}$, $S_T=\{5\}$, and the updated values are $c_1=0.35$, $c_2=0.12$, $c_3=0.5$, and $c_4=0.1$. Therefore, the first loop procedure will be performed again by right shifting 5 time units. The resulting Gantt chart is given in Fig. 3(c), where $\gamma=0$ means that sub-lots l_5, l_6, and l_7 cannot be right shifted.

Algorithm 3
Right shift according to the processing sequence.

Algorithm 3: Right shift according to the processing sequence

input: the processing blocks in the last stage
output: the final processing blocks

```
1.  for j←1 to m_s do   //m_s is the number of machines in the last stage
2.      if |T_j| >1 then   //|T_j| is the number of processing sub-lots on machine j
3.          for l←|T_j| to 1 do
4.              put the l^th sub-lot into a vector named V_p
5.          end
6.          perform Algorithm 4 with (j, V_p)
7.      end
8.  end
```

Algorithm 4
Execute the right shift.

Algorithm 4: execute the right shift
input: the machine number j and the processing blocks V_p
output: the final processing blocks

1. **for** l←1 to $|V_p|$ do
2. c_1←c_2←c_3←c_4←θ ←0, S_E←S_D←S_T←∅
3. compute the starting time for each sub-lot in the last stage , and for each sub-lot p before l perform following three steps.
4. **if** $s_{i,h,p} < SD_i$ **then** store the sub-lot p in S_E
5. **if** $SD_i \le s_{i,h,p} \le ED_i$ **then** store the sub-lot p in S_D
6. **if** $s_{i,h,p} > ED_i$ **then** store the sub-lot p in S_T
7. $c_1 \leftarrow \frac{|S_E|+|S_D|+|S_T|}{N} \times \omega_1$, $c_2 \leftarrow SW_{j,h} \times \omega_2$, c_3←$|S_E| \times \omega_3$, c_4←$|S_T| \times \omega_4$
8. γ←time idle between the last sub-lot that can be right shifted and its successor sub-lot.
   Perform one of the following loop procedures if the corresponding condition satisfies.
9.
10. **while** c_2+c_3>c_1+c_4 **do** //Case-I
11. θ_1←the minimum earliness value in S_E
12. θ_2←the minimum $\{ED_i$-$s_{i,h,l}\}$ value in S_D
13. $\theta = \min\{\theta_1, \theta_2, \gamma\}$
14. **if** θ>0 **then**
15. perform the right shift for the jobs being processed before j
16. c_1←$\frac{|S_E|+|S_D|+|S_T|}{N} \times \omega_1$, c_2←$SW_{j,h} \times \omega_2$, c_3←$|S_E| \times \omega_3$, c_4←$|S_T| \times \omega_4$
17.
18. **end**
19. **while** c_2+c_3>c_1 **do** //Case-II
20. θ_1←the minimum earliness value in S_E
21. θ_2←the minimum $\{ED_i$-$s_{i,h,l}\}$ value in S_D
22. $\theta = \min\{\theta_1, \theta_2, \gamma\}$
23. **if** θ>0 **then**
24. shift each sub-lot processing before l and in $\{S_E, S_D\}$ to the right by θ time units
25. c_1←$\frac{|S_E|+|S_D|}{N} \times \omega_1$, c_2←$SW_{j,h} \times \omega_2$,
26. c_3←$|S_E| \times \omega_3$
27. **end**
28. **while** c_2+c_3>c_1 and $|S_E|$>0 **do** //Case-III
29. θ_1←the minimum earliness value in S_E
30. $\theta = \min\{\theta_1, \gamma\}$
31. **if** θ>0 **then**
32. perform the right-shift procedure for the sub-lots in S_E
33. c_1←$\frac{|S_E|}{N} \times \omega_1$, c_2←$SW_{j,h} \times \omega_2$,
34. c_3←$|S_E| \times \omega_3$
35. **end**
36. **end**

Next, we can start the second loop procedure listed in Algorithm 4 from line 19 to 27. In the second loop, S_E={1,2,3} and S_D={4}, and the updated values are c_1=($|S_E| + |S_D|$)/$N \times \omega_1$=0.2, c_2=0.12, and c_3=0.3. Therefore, c_2+c_3>c_1, and the second loop starts. In this case, the last sub-lot that can be right shifted is l_4. Here, θ_1=min{60,20,60}=20, θ_2=260-255=5, γ=290-285=5, and $\theta = \min\{\theta_1, \theta_2, \gamma\}$=5. Therefore, we can right shift all sub-lots processing before l_5 by 5 time units. The resulting Gantt chart is given in Fig. 3(d).

Because the sub-lots after l_3 cannot be right shifted, we consider l_1, l_2, and l_3, where S_E={1,2,3}, and the updated values are c_1=$|S_E|$/ $N \times \omega_1$=0.15, c_2=0.12, c_3=0.3. Therefore, c_2+c_3>c_1 and $|S_E|$>0 and the third loop starts. In this case, the last sub-lot which can be right shifted is l_3. Here, θ_1=min{55,15,55}=15, γ=290-285=5, and $\theta = \min\{\theta_1, \gamma\}$=5. Therefore, we can right shift all sub-lots processing before l_4 by 5 time units. The resulting Gantt chart is given in Fig. 3(e).

Next, we perform the third loop procedure again by considering right shifting l_1 and l_2, and obtain the resulting Gantt chart given in Fig. 3(f), in which only l_1 is remaining in S_E and both l_1 and l_2 can be considered to be right shifted. The last Gantt chart after right shifting is given in Fig. 3(g).

3.6. Reference regions generalization

A set of weight vectors $W = \{w_1, w_2, \ldots w_N\}$ is generated by a systematic approach, which is illustrated in detail from Das and Dennis [51]. In this approach, weight vectors are sampled with a uniform spacing $\sigma = 1/s$; therefore, $P_{size} = C^s_{m+s-1}$ reference points are generated. Fig. 4 presents an example with $s = 10$, $m = 3$ and $P_{size} = C^{10}_{12} = 66$.

3.7. Population initialization

In the canonical MOEA/D, the initial population is typically generated in a random way, that is, all the solutions of the population are initialized without any future knowledge. After generating the initial population, each solution of the population should be assigned a weight vector or reference line in a random way as well. However, as illustrated in Fig. 5(a), in the initial population, each solution is not assigned to the nearest reference line. For example, the solution in the bottom right has been assigned to the second reference line, which is far away from it. Therefore, in the following evolutionary process, the random assignment method will affect the search performance. Algorithm 5 gives the initialization steps.

Algorithm 5
P=Initialization(P_{size}).

Algorithm 5: P=Initialization(P_{size})
input: population size: P_{size}
output: initialized population P with size equal to P_{size}
1. randomly generate 5* P_{size} solutions and store them in set S
2. evaluate each newly generated solution
3. create a vector named RL with size equal to the number of reference lines
4. **for** each solution S_i in the current population S **do**
5. for each reference line, compute the acute angle with the solution S_i and select the reference line RL_m with the minimum acute angle.
store the solution S_i in the set corresponding to RL_m.
6.
7. **end**
8. **for** each reference line RL_i in RL **do**
9. compute the number of solutions N_i combined with it.
10. sequence each solution in RL_i according to its acute angle A_j in non-decreasing order.
11. compute the last value L_{vj} for the j^{th} solution S_j in RL_i as follows:
12. $L_{vj}=j^* N_i+ A_j$
end
13. sequence each solution in the initial population according to their L_{vj} in non-decreasing order.
14. select the first P_{size} solutions and store them in a new set P

To tackle the problem discussed above, in this study, we propose a novel population initialization mechanism. The main idea is as follows. First, we compute the acute angle for each solution with the reference lines and select the minimum acute angle for each solution. Then, we compute a weighted sum value for each solution and select P_{size} solutions with lower relative values. The detailed steps are described in Algorithm 5. Fig. 5(b) gives the initial population with each solution combined with the nearest reference line.

3.8. Neighborhood adaption heuristic

Similar to other literatures about the multi-objective optimization algorithms, such as [49], in the proposed algorithm, we also embed the neighborhood adaption heuristic. If the neighborhood size is too small, the selection procedure cannot find enough solutions with different performances, while if the neighborhood is too large, the selection can seem to be a random selection from the entire population. Therefore, both small and large neighborhood sizes have advantages and disadvantages considering the convergence and diversity performance values. To balance their convergence and diversity capabilities, we adapt the adaption heuristic as follows:

$$NS_{t+1} = \begin{cases} NS_t + (P_{size} - NS_0) \times t/t_{\max}, & otherwise \\ NS_t, & if \quad \left(t/t_{\max} \; mod \; f_r\right) = 0 \end{cases} \tag{12}$$

where, t is the current iteration; NS_t represents the neighborhood size for the tth iteration; P_{size} is the population size of the current iteration, NS_0 is the initial neighborhood size, f_r is the adaption frequency index.

4. Experimental results

This section discusses the computational experiments used to evaluate the performance of the proposed algorithm. Our algorithm was implemented in C++ on an Intel Core *i*7 3.4-GHz PC with 16 GB of memory. To take a fair comparison with the existing heuristics, two types of problem are tested. The first type is the lot-streaming HFS problems given in Ref. [36]. The following components of the proposed algorithm are used to solve these single-objective problems, i.e., the encoding and decoding heuristic except the sub-lot vector, the exploitation heuristic discussed in sub-section 3.3, the crossover operator list in sub-section 3.4, and the problem-specific heuristic described in sub-section 3.5. It should be noted that, all the components should not consider the variable sub-lot size constraint because the number of sub-lot in Ref. [36] is of a constant number for each job.

The second type is the lot-streaming HFS problems with variable sub-lot size, and the instances are generated based on the lot-streaming HFS problems given in Ref. [36], where the total numbers of jobs, stages, and machines are all collected from the published literature. Then, we extend the instances by adding the due time window constraint, and the energy consumptions. Meanwhile, four objectives are minimized simultaneously. To the best of our knowledge there are none literature considering the HFS lot-streaming with due time window and variable sub-lots. To conduct a fair comparison between the proposed algorithm and the other efficient algorithms, we implement MOEA/D [46], NSGA-II (Chen et al., 2018) [39], DBEA [52], and EMBO [36]. The reasons for selecting the above four algorithms are as follows: (1) MOEAD is the canonical multi-objective algorithm that is based on the decomposition method and has been verified to be efficient for solving many types of multi-objective optimization problems; (2) NSGA-II (Chen et al., 2018) is applied for solving the lot-streaming HFS problems, which have similar features with the considered problems in this study; (3) DBEA is one of the recently published algorithms that are also based on decomposition and are efficient in solving many continuous and discrete multi-objective optimization problems; and (4) EMBO is recently published for the lot-streaming HFS problems with large scales.

4.1. Experimental instances

The first type of problems is taken from Ref [36] for the lot-streaming HFS problems, where there are 100 problems which are grouped into ten types according to the problem scales. The second type of problem is extended based on the first type of problems, where the number of sub-lost for each job is not a constant value as in Ref. [36]. The technological constraints of the second type of problems are given as follows.

- Each job can be divided into 1, 2, 3, 5, 6, or 10 sub-lots and are to be processed in each stage, where the number of machines is a random integer number in Refs. [1,5];
- The processing times for each job are set to 30*p_i, where p_i is the processing time for each job in Ref. [36];
- For the due time window, the start and end time points for all sub-lots are set to $[0.3 \times \sum_{i=1}^{n} p_i, 0.5 \times \sum_{i=1}^{n} p_i]$.
- For each machine, the release time is not considered as a technical capability;

- The transfer times for each of two consecutive stages are generated randomly in the range of [10,15];
- The setup time for each job is not considered to be a technical capability.
- The energy consumption is set to one Watt for one time unit.

4.2. Experimental metric

To verify the effectiveness and efficiency of the proposed algorithm, after 30 independent runs of all compared algorithms, the resulting Pareto archive sets which contain all the non-dominated solutions obtained by all compared algorithms were collected for performance comparisons. The average HV (Hypervolume [53]) and IGD (Inverted Generational Distance [54]) values are used as the performance evaluating indicator.

Let A be a Pareto set obtained by a compared algorithm, the reference point Ref=(r_1, r_2, …, r_k), for each solution $x \in A$, a hypercube v_x is constructed with the reference point Ref and the solution x as the diagonal corners of the hypercube. The HV value of A is a union of all hypercubes, which is calculated as follows [54]:

$$\mathrm{HV}(\mathrm{A}) = \mathrm{volume}\left(\cup_{x=1}^{|A|} v_x \right) \tag{13}$$

As the calculation of the hypervolume needs the true Pareto front, which is not available for the realistic production problems considered in this study. To solve this problem, we perform following steps: first, in each run, each compared algorithm obtained a Pareto archive set; next, with these Pareto archive sets and by using the non-dominated sorting heuristic [56], we can obtain a set of non-dominated solutions as the "true Pareto front" to calculate the hypervolume and IGD values for each compared algorithm. Meanwhile, the four objective values of each compared algorithm are also normalized because different objectives have different scales.

The indicator IGD is used to measure the average distance from the "true Pareto front" to A and calculated:

$$IGD(A, P^*) = \frac{\sum_{x \in P^*} d(x, A)}{|P^*|} \tag{14}$$

where $d(x, A)$ is the minimum Euclidean distance between x and the points in A.

Note that the larger the hypervolume value is, the better the performance of the algorithm. By contrast, a smaller IGD value indicates better performance of the MOEA. In addition, since the exact calculation of hypervolume is computationally extremely intensive, the Monte Carlo method used in HypE [55] is adopted for estimating the hypervolume, where 10000 sampling points are used. Moreover, the scheduling problems with realistic constraints are hard to find the true Pareto front, and therefore, similar to other literatures, we collect all the results from the compared algorithms and construct the approximate Pareto front as the compared values to calculated IGD values.

4.3. Experimental parameters

The stop criterion for each instance is set to be different with the instance scale, which is $u \times n \times h$ seconds, where u is the time coefficient value, n is the total number of jobs, and h is the total number of stages. Four key parameters are considered, the probability for crossover (p_c), the probability for mutation (p_m), the exploitation strength (v_n), and the uniform spacing value (σ) which is used to compute the weighted vectors and the population size. According to our preliminary experiments, the values of the three parameters are set as follows: $v_n \in \{5, 10, 15, 20\}$, $p_m \in \{0.1, 0.2, 0.5, 0.8\}$, and $p_c \in \{0.1, 0.2, 0.5, 0.8\}$. According to published literatures and realistic production requirements, u is generally set to 30, 50, and 100, respectively.

Similar to Refs. [57–59], we have utilized the design of experiments (DOE) method for the parameter tuning of the proposed algorithm. However, rather than using the DOE Taguchi method, we adopted a full factorial design in which the three parameters were used as factors. The full factorial design yields a total of 64 distinct combinations of the three parameters. We performed 10 runs per instance to test the parameter tuning experiments, which were carried out on a cluster of computers with Intel Core *i*7 3.4-GHz PCs and 16 GB of memory. The scaled IGD values were computed and used as the response variable of the experiments.

The results were analyzed via a multiway analysis of variance (ANOVA), where the three parameters are used as the controllable factors. To evaluate the ANOVA model hypothesis, namely, normality, homoscedasticity and independence, the standardized residuals were analyzed. A major advantage of the ANOVA technique is that it calculates the magnitude of the F-ratio, where a large F-ratio indicates that the analyzed factor has a substantial effect on the response variable.

Table 1 lists the results of the analysis. The results demonstrate that two of the three parameters are all of the significant factors (p-value<0.05), i.e., v_n and p_m. The parameter p_m, with an F-ratio value 12.63, has a significant effect on the performance of the proposed algorithm. The parameter p_c, with an F-ratio value 0.17, has a non-significant effect compared with the other two parameters. Meanwhile, the factor interactions between the three parameters are non-significant, with p-values of greater than 0.05. Plots of the 95% confidence intervals for the scaled IGD values under selections of parameter v_n and p_m are provided in Fig. 6. According to Fig. 6 (a), $v_n = 5$ yields a much better scaled IGD value than the others. According to Fig. 6 (b), $p_m = 0.5$ yields a much better scaled IGD value than the others. In addition, considering the non-significant factor p_c, we select the factor level with the minimum average value, i.e., $p_c = 0.2$. Fig. 6 (c) also shows that $p_c = 0.2$ yields a much better performance compared to the zero value, which further verify the efficiency of the crossover heuristic.

In this study, four objectives are considered simultaneously. As discussed in Section 3.6 and after our detailed experiments, we set $\sigma = 0.2$; therefore, we obtain $P_{size} = C_8^5 = 56$.

4.4. Comparisons of the single-objective lot-streaming HFS problems

To take a fair comparison with the existing heuristics, we also test the proposed algorithm for solving the lot-streaming HFS problems given in Ref. [36]. The following components of the proposed algorithm are used to solve these single-objective problems, i.e., the encoding and decoding heuristic except the sub-lot vector, the exploitation heuristic discussed in sub-section 3.3, the crossover operator list in sub-section 3.4, and the problem-specific heuristic described in sub-section 3.5. It should be noted that, all the components should not consider the variable sub-lot size constraint because the number of sub-lot in Ref. [36] is of a constant number for each job.

There are 100 instances, which are grouped into ten types of problems range from 20 jobs to 100 jobs. For each of the 100 instances with single objective, the average RPI values are gained across 20 independent replications similar as Ref [36]. To collect the results and make a detailed comparison, all the instances are grouped into ten types. The results of CPU time of $\mu = 30$ is selected to test the performance of the proposed algorithm. The detailed comparisons are list in Table 2, where the first column tells the instance with two numbers representing the problem scale, i.e., the first is the number of jobs and the second is the number of stages. The second column lists the best fitness value for each type of instance collected by all the compared algorithms. The next six columns represent the fitness values collected by each of the compared algorithm, respectively, i.e., PH-MOEAD, EMBO, GA, GAR, DPSO, and DABC. To make a fair comparison, the experimental results of the six compared algorithms are collected directly from their literatures. Then, the last six columns tell the *dev* values for all of the compared algorithms. The calculation of the *dev* values are as follows.

$$dev = \frac{C_{\text{comp}} - C_{best}}{C_{best}} \times 100\% \quad (15)$$

where, C_{comp} is the fitness value obtained by the compared algorithm, C_{best} is the best fitness value among the compared algorithms for each instance. It can be seen from Table 2 that (1) the proposed algorithm obtained nine better values out of the given ten types of instances, which is significant better than the second best EMBO algorithm; (2) from the average performance given in the last line, it can be concluded that the proposed algorithm is better than the other compared algorithms. For example, on average, the proposed algorithm obtained an average value of 0.54, which is obviously better than the second best algorithm EMBO; (3) from the last six columns we can conclude that, on average PH-MOEAD obtained a significant better value for the ten types of single-objective problems. From the comparison results, we can see that the main component of the proposed algorithm can solve the single-objective lot-streaming HFS problems efficiently.

To evaluate whether the difference of the two methods is significant, we perform a multifactor analysis of variance (ANOVA) in which the compared methods are considered as factors. Fig. 7 shows the means and the 95% LSD (Least-Significant Difference) intervals for the RPI values of the six compared methods. It is obviously that there are significant differences between the compared methods considering the RPI values.

4.5. Efficiency of the mutation heuristic

To verify the effectiveness of the mutation heuristic discussed in sub-section 3.3, we code the two algorithms, i.e., PH-MOEAD and PH-MOEAD-NM, where PH-MOEAD-NM represents the proposed algorithm with all of the components discussed in section 3 except the mutation heuristic discussed in sub-section 3.3. The parameters for the two compared algorithms are set equal. The two compared algorithms are tested on the same PC and with the same test instances. After 30 independent runs, the minimum, average and maximum HV results for each instance are collected for comparison, which are given in Table 3. In Table 3, the first column provides the problem scale size, where the two numbers represent the total number of jobs and stages, respectively. The following three columns report the maximum, average and minimum HV values obtained by PH-MOEAD. The next three columns tell the results obtained by PH-MOEAD-NM. The last two columns give the dev^+ values due to the two compared algorithms, where the calculation of the dev^+ value is as follows:

$$dev^+ = \left| \frac{HV_{\text{comp}} - HV_{\max}}{HV_{\max}} \times 100\% \right| \quad (16)$$

where HV_{comp} is the HV values obtained by the compared algorithm, and HV_{best} is the best value for the corresponding instance.

It can be observed from Table 3 that (1) PH-MOEAD obtained eight better values out of the given ten instances, whereas PH-MOEAD-NM can only obtain two better values; (2) from the last row in the table, we can conclude that PH-MOEAD obviously performs better than PH-MOEAD-NM; and (3) in a nutshell, we can obtain better results after applying the proposed mutation heuristic. It can be concluded from the comparisons that, the proposed mutation heuristic can enhance the exploitation capabilities of the algorithm.

4.6. Efficiency of the right-shift heuristic

To verify the effectiveness of the right-shift heuristic discussed in sub-section 3.5, we code the two algorithms, i.e., PH-MOEAD and PH-MOEAD-NR, where PH-MOEAD-NR represents the proposed algorithm with all of the components discussed in section 3 except the right-shift heuristic discussed in sub-section 3.5. The parameters for the two compared algorithms are set equal, and the results after 30 independent runs are collected in Table 4.

It can be observed from Table 4 that (1) PH-MOEAD obtained 8 optimal values out of the given 10 types of instances, whereas the PH-MOEAD-NR obtains two optimal values; (2) from the last row in the table, we can see that PH-MOEAD obviously performs better than PH-MOEAD-NR; and (3) in a nutshell, we can obtain better results after applying the proposed right-shift heuristic.

4.7. Efficiency of the crossover heuristic

To investigate the effectiveness of the crossover heuristic, we also code the two algorithms, i.e., PH-MOEAD with all components, and PH-MOEAD-NC with all components except the proposed crossover heuristic discussed in sub-section 3.4. Table 5 lists the comparison results after performing 30 independent tests.

It can be observed from Table 5 that (1) PH-MOEAD obtained nine optimal values out of the given ten groups of instances; (2) from the last row in the table, we can see that PH-MOEAD obviously performs better than PH-MOEAD-NC; and (3) from the dev^+ values list in the last two columns, we can also conclude that the proposed crossover heuristic can enhance the performance of the algorithm.

4.8. Comparisons with the presented efficient algorithms

To further verify the performance of the framework of the proposed algorithm, we select four relevant efficient algorithms to make detailed comparisons. It should be noted that, the considered HFS lot-streaming with the due time window and variable sub-lots are of the first time to be solved, and therefore, all the compared algorithms are recoded to adapt to solve the considered problems. The main components of all the compared algorithms described in their literatures are embedded, and necessary mechanisms are modified for the HFS lot-streaming problems, such as the encoding and decoding methods.

The results for the comparisons of HV values with MOEA/D, NSGA-II (Chen et al., 2018), DBEA, and EMBO under $u = 30$ and $u = 50$ are reported in Tables 6 and 7, respectively. It can be observed from the two tables that (1) under $u = 30$, the proposed PH-MOEAD obtained 9 out of the given 10 groups of instances, which is obviously better than the second best DBEA algorithm which can only find one optimal value; (2) under $u = 50$, the proposed PH-MOEAD also obtained 9 out of the given 10 groups of instances, while the second best DBEA algorithm can only find one better value; and (3) from the average performance of the two circumstances, we can conclude that the proposed algorithm shows competitive performances.

Tables 8 and 9 describe the comparison results for HV and IGD values under $u = 100$, respectively. From the comparison results for long performing period, we can observe that (1) considering the HV values, the proposed algorithm obtain all optimal values for all of the ten groups of instances, i.e., from 20 jobs to 100 jobs, which show the convergence and diversity performances of the proposed algorithm; (2) comparing the IGD values, the proposed algorithm obtained six optimal values out of the given ten groups of instances, which is obviously better that the other compared algorithms; (3) from the last line in Table 8, we can find that, on average the proposed algorithm obtained an HV value of 0.622, which is obviously better than the second-best performer DBEA with a 0.431 overall average HV value; (4) from the last line in Table 9, we can find that, on average the proposed algorithm obtained an IGD value of 0.20, which is obviously better than the second-best performer NSGA-II with a 0.28 overall average IGD value; and (5) in a nutshell, the proposed PH-MOEAD also shows competitive performances for solving the extended HFS lot-streaming problems considering long running period.

Fig. 8 (a) illustrates the means and the 95% LSD interval for the best HV values of the five compared algorithms under $u = 30$, while Fig. 8 (b) shows the ANOVA results for the best HV values under $u = 50$. The ANOVA results for the HV and IGD values under $u = 100$ are given in Fig. 8 (c) and (d). It can be concluded that the proposed algorithm has a statistically significant competitive performance compared with the

other efficient algorithms. Fig. 9 gives the Gantt chart for one of the obtained solution by the proposed algorithm for one of the 20-5 instance. It can be concluded from Fig. 9, that the proposed algorithm can solve the considered problem effectively.

4.9. Comparison analysis

From the experimental comparisons, it can be concluded that the proposed algorithm is efficient for solving the lot-streaming HFS with variable sub-lots. The main advantages of the proposed algorithm are as follows: (1) the proposed crossover which considered solutions with different sub-lot size can make a solution feasible and enhance the exploration abilities of the algorithm as well; (2) the proposed mutation heuristic considering the permutations in the sub-lot size can improve the performance of the proposed algorithm, and thus enhance the exploitation abilities; and (3) the proposed right-shift heuristics can further improve the performance of the algorithm.

5. Conclusions

In this study, a problem-specific heuristic based MOEA/D was proposed to solve the hybrid flowshop lot-streaming scheduling problem. The primary contributions of the proposed algorithm are as follows: (1) a novel crossover operator is proposed to tackle the case of parent solutions with different sub-lot vectors; (2) a right-shift heuristic that considers the problem structure and objective features is advanced to increase the performance of the proposed algorithm; (3) a population initialization heuristic is proposed to assign each solution to the nearest reference vector; (4) a novel mutation heuristic considering the permutations in the sub-lots is proposed, which can improve the exploitation abilities; and (5) the experimental results demonstrate the efficiency and effectiveness of the proposed PH-MOEAD algorithm.

Future works mainly focus on following aspects: (1) to apply the proposed algorithm to solve the hybrid flowshop rescheduling problem with other types of constraints, objectives and disruptions; (2) to embed the data-driven heuristics in the proposed optimization algorithm, and therefore improve the exploration performance; (3) considering problem-specific knowledge, apply the proposed algorithm in solving more realistic optimization problem; (4) to solve the distributed hybrid flowshop with lot-streaming constraints by using the proposed multi-objective optimization algorithm; (5) to apply the proposed algorithm directly to the actual system; (6) to find the features of the scheduling problem with multi-objective and realistic constraints [60], and therefore to calculate the approximate Pareto front boundaries; and (7) the delay of algorithm allowed by the system is very limited to consider the time-consuming features of the evolutionary algorithm in solving the real-time optimization problems, and therefore, the future work will consider the lot-streaming HFS in the real-time environments [61] and analyze the influence of algorithm delay on system performance.

Acknowledgements

This research is partially supported by National Science Foundation of China under Grant 61773192, 61803192, and 61773246, Shandong Province Higher Educational Science and Technology Program (J17KZ005), the Open Project of Henan Key Laboratory of Intelligent Manufacturing of Mechanical Equipment, Zhengzhou University of Light Industry (IM201906), and major Program of Shandong Province Natural Science Foundation (ZR2018ZB0419).

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.swevo.2019.100600.

References

[1] R. Ruiz, J.A. Vázquez-Rodríguez, The hybrid flow shop scheduling problem, Eur. J. Oper. Res. 205 (1) (2010) 1–18.
[2] X.L. Zheng, L. Wang, A collaborative multiobjective fruit fly optimization algorithm for the resource constrained unrelated parallel machine green scheduling problem, IEEE Trans. Syst. Man. Cy. A. 48 (5) (2018) 790–800.
[3] J.Q. Li, M.X. Song, L. Wang, P.Y. Duan, Y.Y. Han, H.Y. Sang, Q.K. Pan, Hybrid artificial bee colony algorithm for a parallel batching distributed flow shop problem with deteriorating jobs, IEEE Trans. Cybernetics (2019), https://doi.org/10.1109/TCYB.2019.2943606.
[4] H.Y. Sang, Q.K. Pan, J.Q. Li, P. Wang, Y.Y. Han, K.Z. Gao, P. Duan, Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion, Swarm. Evol. Comput. 44 (2) (2019) 64–73.
[5] J.Q. Li, S.C. Bai, P.Y. Duan, H.Y. Sang, Y.Y. Han, Z.X. Zheng, An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system, Int. J. Prod. Res. 57 (2019) 6922–6942.
[6] D. Lei, M. Li, L. Wang, A two-phase meta-heuristic for multiobjective flexible job shop scheduling problem with total energy consumption threshold, IEEE Trans. Cybernetics. (2018), https://doi.org/10.1109/TCYB.2018.2796119.
[7] Q.K. Pan, L. Gao, L. Wang, A multi-objective hot-rolling scheduling problem in the compact strip production, Appl. Math. Model. 73 (2019) 327–348.
[8] Q.K. Pan, L. Gao, X.Y. Li, M. Framinan, Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem, Appl. Soft Comput. 81 (2019) 105492.
[9] Q.K. Pan, L. Gao, L. Wang, J. Liang, X.Y. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, Expert Syst. Appl. 124 (2019) 309–324.
[10] J.Q. Li, Q.K. Pan, K. Mao, A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems, IEEE Trans. Autom. Sci. Eng. 13 (2016) 932–949.
[11] S.P. Yu, T.Y. Chai, Y. Tang, An effective heuristic rescheduling method for steelmaking and continuous casting production process with multirefining modes, IEEE Trans. Syst. Man. Cy. A. 46 (2016) 1675–1688.
[12] J.Q. Li, Q.K. Pan, P.Y. Duan, An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping, IEEE Trans. Cybernetics. 46 (2016) 1311–1324.
[13] K.K. Peng, Q.K. Pan, L. Gao, B. Zhang, X.F. Pang, An Improved Artificial Bee Colony algorithm for real-world hybrid flowshop rescheduling in Steelmaking-refining-Continuous Casting process, Comput. Ind. Eng. 122 (2018) 235–250.
[14] H. Liu, B. Xu, D. Lu, G. Zhang, A path planning approach for crowd evacuation in buildings based on improved artificial bee colony algorithm, Appl. Soft Comput. 68 (2018) 360–376.
[15] A.A. Kalir, S.C. Sari, Evaluation of the potential benefits of lot streaming in flow-shop systems, Int. J. Prod. Econ. 66 (2000) 131–142.
[16] J.H. Chang, H.N. Chiu, A comprehensive review of lot streaming, Int. J. Prod. Res. 43 (2005) 1515–1536.
[17] C.T. Tseng, C.J. Liao, A discrete particle swarm optimization for lot-streaming flowshop scheduling problem, Eur. J. Oper. Res. 191 (2008) 360–373.
[18] S. Marimuthu, S.G. Ponnambalam, N. Jawahar, Evolutionary algorithms for scheduling m-machine flow shop with lot streaming, Robot. Comput. Integr. Manuf. 24 (2008) 125–139.
[19] S. Marimuthu, S.G. Ponnambalam, N. Jawahar, Threshold accepting and Ant-colony optimization algorithms for scheduling m-machine flow shops with lot streaming, J. Mater. Process. Technol. 209 (2009) 1026–1041.
[20] S.H. Yoon, J.A. Ventura, An application of genetic algorithms to lot streaming flow shop scheduling, IIE Trans. 34 (2002) 779–787.
[21] Q.K. Pan, R. Ruiz, An estimation of distribution algorithm for lot-streaming flow shop problems with setup times, Omega 40 (2012) 166–180.
[22] D. Davendra, R. Senkerik, I. Zelinka, M. Pluhacek, M. Bialic-Davendra, Utilising the chaos-induced discrete self-organizing migrating algorithm to solve the lot-streaming flowshop scheduling problem with setup time, Soft. Comput. 18 (2014) 669–681.
[23] H. Sang, L. Gao, X. Li, An iterated local search algorithm for the lot-streaming flow shop scheduling problem, Asia, Pac. J. Oper. Res. 31 (2014) 1450045.
[24] H. Sang, Q.K. Pan, P.Y. Duan, J.Q. Li, An effective discrete invasive weed optimization algorithm for lot-streaming flowshop scheduling problems, J. Intell. Manuf. 29 (6) (2018) 1337–1349.
[25] G. Vijay Chakaravarthy, S. Marimuthu, A.N. Sait, Performance evaluation of proposed differential evolution and particle swarm optimization algorithms for scheduling m-machine flow shops with lot streaming, J. Intell. Manuf. 24 (2013) 175–191.
[26] G.V. Chakaravarthy, S. Marimuthu, S.G. Ponnambalam, G. Kanagaraj, Improved sheep flock heredity algorithm and artificial bee colony algorithm for scheduling m-machine flow shops lot streaming with equal size sub-lot problems, Int. J. Prod. Res. 52 (2014) 1509–1527.
[27] T. Meng, Q.K. Pan, J.Q. Li, H.Y. Sang, An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem, Swarm Evolut. Comput. 38 (2018) 64–78.
[28] Y.Y. Han, J.Q. Li, D.W. Gong, H.Y. Sang, Multi-objective migrating birds optimization algorithm for stochastic lot-streaming flow shop scheduling with blocking, IEEE Access 7 (2018) 5946–5962.
[29] O. Masmoudi, A. Yalaoui, Y. Ouazene, H. Chehade, Multi-item capacitated lot-sizing problem in a flow-shop system with energy consideration, IFAC-PapersOnLine 49 (2016) 301–306.

[30] H. Tsubone, M. Ohba, T. Uetake, The impact of lot sizing and sequencing on manufacturing performance in a two-stage hybrid flow shop, Int. J. Prod. Res. 34 (1996) 3037–3053.
[31] W. Zhang, J. Liu, R.J. Linn, Model and heuristics for lot streaming of one job in m-1 hybrid flowshops, Int. J. Oper. Quant. Manag. 9 (2003) 49–64.
[32] W. Zhang, C. Yin, J. Liu, R.J. Linn, Multi-job lot streaming to minimize the mean completion time in m-1 hybrid flowshops, Int. J. Prod. Econ. 96 (2005) 189–200.
[33] J. Liu, Single-job lot streaming in m+1 two-stage hybrid flowshops, Eur. J. Oper. Res. 187 (2008) 1171–1183.
[34] B. Naderi, M. Yazdani, A model and imperialist competitive algorithm for hybrid flow shops with sublots and setup times, J. Manuf. Syst. 33 (2014) 647–653.
[35] M. Cheng, S. C Sarin, S. Singh, Two-stage, single-lot, lot streaming problem for a 1+ 2 hybrid flow shop, J. Glob. Optim. 66 (2016) 263–290.
[36] B. Zhang, Q.K. Pan, L. Gao, X.L. Zhang, H.Y. Sang, J.Q. Li, An effective modified migrating birds optimization for hybrid flowshop scheduling problem with lot streaming, Appl. Soft Comput. 52 (2017) 14–27.
[37] M. Nejati, I. Mahdavi, R. Hassanzadeh, N. Mahdavi-Amiri, Lot streaming in a two-stage assembly hybrid flow shop scheduling problem with a work shift constraint, Journal of Industrial and Production Engineering 33 (2016) 459–471.
[38] H. Zohali, B. Naderi, M. Mohammadi, V. Roshanaei, Reformulation, linearization, and a hybrid iterated local search algorithm for economic lot-sizing and sequencing in hybrid flow shop problems, Comput. Oper. Res. 104 (2019) 127–138.
[39] T.L. Chen, C.Y. Cheng, Y.H. Chou, Multi-objective genetic algorithm for energy-efficient hybrid flow shop scheduling with lot streaming, Ann. Oper. Res. (2018), https://doi.org/10.1007/s10479-018-2969-x.
[40] R. Wang, S. Lai, G. Wu, L. Xing, L. Wang, H. Ishibuchi, Multi-clustering via evolutionary multi-objective optimization, Inf. Sci. 450 (2018) 128–140.
[41] J.Q. Li, Q.K. Pan, M. Tasgetiren, A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities, Appl. Math. Model. 38 (2014) 1111–1132.
[42] R. Wang, Z. Zhou, H. Ishibuchi, T. Liao, T. Zhang, Localized weighted sum method for many-objective optimization, IEEE Trans. Evol. Comput. 22 (1) (2018) 3–18.
[43] J.Q. Li, Q.K. Pan, S.X. Xie, An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems, Appl. Math. Comput. 218 (2012) 9353–9371.
[44] S.Y. Wang, L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, IEEE Trans. Syst. Man. Cy. A. 46 (2016) 139–149.
[45] J. Sun, Z. Miao, D. Gong, X. Zeng, J. Li, G. Wang, Interval multiobjective optimization with memetic algorithms, IEEE Trans. Cybernetics. (2019), https://doi.org/10.1109/TCYB.2019.2908485.
[46] Q. Zhang, H. Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (2007) 712–731.
[47] Z. Wang, Y.S. Ong, J. Sun, A. Gupta, Q. Zhang, A generator for multiobjective test problems with difficult-to-approximate Pareto front boundaries, IEEE Trans. Evol. Comput. (2018), https://doi.org/10.1109/tevc.2018.2872453.
[48] Z. Wang, Y.S. Ong, H. Ishibuchi, On scalable multiobjective test problems with hardly dominated boundaries, IEEE Trans. Evol. Comput. 23 (2) (2019) 217–231.
[49] Z. Wang, Q. Zhang, A. Zhou, M. Gong, L. Jiao, Adaptive replacement strategies for MOEA/D, IEEE Trans. Cybernetics 46 (2) (2016) 474–486.
[50] K. Li, K. Deb, Q. Zhang, S. Kwong, An evolutionary many-objective optimization algorithm based on dominance and decomposition, IEEE Trans. Evol. Comput. 19 (2015) 694–716.
[51] I. Das, J.E. Dennis, Normal-boundary intersection: a new method for generating the Pareto surface in nonlinear multicriteria optimization problems, SIAM J. Optim. 8 (1998) 631–657.
[52] M. Asafuddoula, T. Ray, R. Sarker, A decomposition-based evolutionary algorithm for many objective optimization, IEEE Trans. Evol. Comput. 19 (2015) 445–460.
[53] E. Zitzler L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Trans. Evol. Comput. 3 (1999) 257–271.
[54] P. Bosman D. Thierens, The balance between proximity and diversity in multiobjective evolutionary algorithms, IEEE Trans. Evol. Comput. 7 (2003) 174–188.
[55] J. Bader, E. Zitzler, HypE: an algorithm for fast hypervolume-based many-objective optimization, Evol. Comput. 19 (2011) 45–76.
[56] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2014) 577–601.
[57] R. Ruiz, T. Stutzle, A simple and effective iterated greedy algorithm for the permutation flowshop with scheduling problem, Eur. J. Oper. Res. 177 (3) (2007) 2033–2049.
[58] I. Ribas, R. Companys, X. Tort-Martorell, An iterated greedy algorithm for the flowshop scheduling problem with blocking, Omega 39 (3) (2011) 293–301.
[59] R. Rui, Q.K. Pan, B. Naderi, Iterated Greedy methods for the distributed permutation flowshop scheduling problem, Omega 83 (2019) 213–222.
[60] Y. Han, D. Gong, X. Sun, A discrete artificial bee colony algorithm incorporating differential evolution for flow shop scheduling problem with blocking, Eng. Optim. 47 (2015) 927–946.
[61] H. Luo, J. Fang, G. Huang, Real-time scheduling for hybrid flowshop in ubiquitous manufacturing environment, Comput. Ind. Eng. 84 (2015) 12–23.