A Hyperheuristic With Q-Learning for the Multiobjective Energy-Efficient Distributed Blocking Flow Shop Scheduling Problem

Fuqing Zhao, Shilu Di, and Ling Wang

***Abstract*—Carbon peaking and carbon neutrality, which are the significant national strategy for sustainable development, have attracted considerable attention from production enterprises. In this study, the energy consumption is considered in the distributed blocking flow shop scheduling problem (DBFSP). A hyperheuristic with *Q*-learning (HHQL) is presented to address the energy-efficient DBFSP (EEDBFSP). *Q*-learning is employed to select an appropriate low-level heuristic (LLH) from a predesigned LLH set according to historical information fed back by LLH. An initialization method, which considers both total tardiness (TTD) and total energy consumption (TEC), is proposed to construct the initial population. The ε-greedy strategy is introduced to utilize the learned knowledge while retaining a certain degree of exploration in the process of selecting LLH. The acceleration operation of the job on the critical path is designed to optimize TTD. The deceleration operation of the job on the noncritical path is designed to optimize TEC. The statistical and computational experimentation in an extensive benchmark testified that the HHQL outperforms the other comparison algorithm regarding efficiency and significance in solving EEDBFSP.**

***Index Terms*—Distributed blocking flow shop scheduling, hyperheuristic, *Q*-learning, total energy consumption (TEC), total tardiness (TTD).**

I. Introduction

CARBON peaking and carbon neutrality, which are conducive to guiding green technological innovation and enhancing the global competitiveness of industry and economy, are the key strategies for green sustainable development [1]. Green manufacturing is a modern production mode as the embodiment of carbon peaking and carbon neutrality [2]. While pursuing economic benefits, manufacturing enterprises should pay more attention to reducing energy consumption. As a typical scenario in intelligent manufacturing [3], the energy-efficient distributed blocking flow shop scheduling problem (EEDBFSP) has attracted a substantial amount of attention from researchers and practitioners.

The distributed blocking flow shop scheduling problem (DBFSP) has a widespread industrial background, such as steel manufacturing, chemical production, and nonferrous metallurgy [4]. DBFSP considers the job assignment for factories and the processing sequence of jobs. The solution space of DBFSP demonstrates an exponential growth trend as the number of jobs and factories increases. In addition, one or two production objectives, such as $C_{\max}$, total tardiness (TTD), and total flowtime, are primarily optimized in DBFSP [4]. However, if energy consumption is taken into account in DBFSP, the complexity of solving the problem increases even further. For instance, EEDBFSP must consider not only the job allocation for factories and the sequence of jobs within the factory but also the speed of jobs. In this study, the energy consumption of machines is regarded as an important environmental indicator. The energy consumption of machines consists of three components: 1) the energy consumption of machines for processing jobs; 2) the energy consumption of machines in standby mode; and 3) the energy consumption of machines in blocking mode. It has been demonstrated that the permutation flow shop scheduling problem (PFSP) is an NP-hard problem [5]. The distributed PFSP (DPFSP) is similar to the well-known PFSP in the case where $F = 1$ [6]. DBFSP considers blocking constraints based on DPFSP. Consequently, EEDBFSP is likewise an NP-hard problem.

The large-scale problems are difficult to solve by exact methods due to high computational complexity, such as mathematical methods, branch, and bound [7]. Heuristic methods construct a scheduling solution quickly, but they lack solution accuracy. In contrast, it is difficult to design an effective heuristic method for complex scheduling problems when lacking the characteristics of the problem [8]. Intelligent optimization algorithms have achieved satisfactory performance in various scheduling problems, but the iterative search process is usually time consuming. Moreover, historical information is rarely utilized to adjust the search behavior [9]. A hyperheuristic algorithm is a framework for addressing complex problems

Manuscript received 18 March 2022; revised 29 May 2022; accepted 14 July 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62063021 and Grant 61873328; in part by the Key Talent Project of Gansu Province under Grant ZZ2021G50700016; in part by the Key Research Programs of Science and Technology Commission Foundation of Gansu Province under Grant 21YF5WA086; in part by the Lanzhou Science Bureau Project under Grant 2018-rc-98; and in part by the Project of Gansu Natural Science Foundation under Grant 21JR7RA204. This article was recommended by Associate Editor C. Catal. *(Corresponding authors: Fuqing Zhao; Ling Wang.)*

Fuqing Zhao and Shilu Di are with the School of Computer and Communication, Lanzhou University of Technology, Lanzhou 730050, China (e-mail: fzhao2000@hotmail.com; 1373590969@qq.com).

Ling Wang is with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: wangling@tsinghua.edu.cn).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TCYB.2022.3192112.

Digital Object Identifier 10.1109/TCYB.2022.3192112

that adaptively selects the optimizer. In addition, the hyperheuristic algorithm features simple operations and a small number of parameters.

Reinforcement learning (RL) aims to interact with the environment via an intelligent agent. The returned feedback information is used to guide the intelligent agent to select the optimal action in subsequent processes [10], [11]. RL consists of five principles: 1) the input and output system; 2) the reward; 3) the artificial intelligence environment; 4) the Markov decision process; and 5) training and inference. As a well-known and fundamental RL algorithm, Q-learning enables intelligent agents to determine an optimal behavior based on trial and error [12], [13]. The learning process of Q-learning is comparable to temporal difference learning [14]. An agent tries an action a_t according to the state s_t at time t. Then, the immediate reward or penalty r_t is received and a Q-value of a state–action pair (s_t and a_t) is calculated by a utility function known as a Q-function. By repeatedly trying all actions in all states, the agent learns which are best overall, judged by long-term discounted reward. A Q-value of a state–action pair (s_t and a_t) represents the expected utility of taking action a_t in state s_t and following a fixed policy thereafter. Q table is usually used to record the learning experience of the agent.

The main contributions are summarized as follows.

1) The processing of jobs, the idle of machines, and the blocking of jobs all contribute to energy consumption. A mixed-integer linear programming (MILP) model of EEDBFSP is developed to meet manufacturing requirements. Proposed is an initialization method that considers both TTD and total energy consumption (TEC) when constructing the initial population.
2) Q-learning is utilized to select a suitable low-level heuristic (LLH) from a predesigned set of LLHs based on the historical information fed back by LLH. The ε-greedy strategy is introduced to make use of acquired knowledge while retaining a degree of exploration during the selection of LLH. As executable actions, eight simple heuristics that are introduced to construct the LLH set are defined.
3) The critical path of EEDBFSP is defined. The acceleration operation of the job on the critical path is designed to optimize TTD. The deceleration operation of the job on the noncritical path is designed to optimize TEC.

The remainder of this article is organized as follows. The related literature is reviewed in Section II. Section III describes the EEDBFSP and explains the MILP model of the EEDBFSP. The description of the hyperheuristic with Q-learning (HHQL) algorithm is provided in Section IV. The experiment and analysis are implemented in Section V. Finally, the conclusion and future work are summarized in Section VI.

II. Literature Review

The traditional centralized production model does not meet the enterprise and market requirements for distributed manufacturing systems. Progressively, multiregional and multifactory cooperative production has become the dominant mode of production [15]. Naderi and Ruiz [16] first extended the multifactory environment in the PFSP. Six distinct MILP models were introduced and carefully analyzed in this article. In addition, two rules for factory allocation and 14 heuristics involving various dispatching rules were presented. Based on this pioneering work, several scheduling solutions for distributed flop shops were presented. To address DPFSP, an iterated greedy (IG) [17] was presented. This article presented a destruction operator with a dynamic damage degree to improve the search capability of the algorithm across the entire search space. Combining the parallel machine scheduling and the property of distributed flow shop scheduling problem (DFSP), a distributed hybrid flop shop scheduling problem (DHFSP) to minimize makespan was studied [18]. A cooperative coevolution algorithm, which cooperated IG and estimation of distribution, was proposed to solve multiobjective fuzzy DHFSP [19].

In contrast to the production conditions described previously, there are no intermediate buffers between two adjacent machines for the blocking flow shop scheduling problem (BFSP) [20]. Specifically, a job is blocked on the current machine due to lacking intermediate buffers until the next machine becomes available. The phenomenon of blocking is prevalent in actual production scenarios, such as steel manufacturing [21]. Under the blocking constraint, job completion time is extended, and resources are wasted. Consequently, the DFSP with blocking constraints has considerable practical importance. DBFSP was proposed for the first time in the literature [22]. In addition, heuristic and metaheuristic methods were introduced. Based on this mathematical model, various methods for DBFSP have been presented. To address the DBFSP, for example, a discrete differential evolution (DDE) algorithm was presented [23]. Meanwhile, a local search method and an elitist retain strategy are incorporated into the DDE framework to maintain the balance between local exploitation and global exploration. A discrete fly fruit optimization was proposed to address DBFSP [11]. In addition to the methods mentioned above, DBFSP can also be solved by employing the distributed gray wolf optimizer [24] and discrete Jaya algorithm [25].

Moreover, energy conservation in the manufacturing sector is consistent with the notion of sustainable development [26], [27]. Certain energy-efficient methods have been presented in flow shop scheduling problems. An effective hybrid collaborative algorithm for energy-efficient distributed permutation flow-shop inverse scheduling was presented [27]. A double-population cooperative search link based on a learning mechanism was used to balance the global exploration and local development ability of the algorithm. A competitive memetic algorithm (AMC) for solving multiobjective DPFSP with makespan and total carbon emissions was presented [28]. A discrete evolutionary multiobjective optimization (DEMO) was presented to solve the energy-efficient BFSP [29]. Self-adaptive methods were proposed in DRMO to minimize the makespan and energy consumption simultaneously. Nevertheless, only the energy consumption of machine blocking and idle were considered. The processing energy consumption of the job was not accounted for. The turn-off strategy

was utilized to shut down nonbottleneck machines after a predetermined amount of time in standby mode. However, frequent shutdowns would diminish the service life of the machine to some extent [30]. The speed scaling strategy, which assumes that the energy consumption of a machine is proportional to its operating speed, is widely used in flow shop scheduling problems. Utilizing a knowledge-based cooperative algorithm (KCA), the energy-efficient DPFSP was addressed. The acceleration of critical jobs and the deceleration of noncritical jobs were employed to optimize $C_{\max}$ and TEC [31]. Energy-efficient multiobjective distributed no idle permutation flow-shop scheduling problem (EEDNIPFSP) was addressed by a collaborative optimization algorithm (COA) [32]. A knowledge-based two-population optimization (KTPO) algorithm was introduced to simultaneously optimize TEC and TTD [15]. The energy-efficient blocking hybrid flow shop problem (EEBHFSP) was modeled, and an improved iterative greedy algorithm (IGA) based on the swap strategy was developed to optimize the proposed model [33]. A greedy cooperative co-evolutionary algorithm (GCCEA) was presented to address the flow shop sequence-dependent group scheduling problem (FSDGSP) with an energy efficiency indicator [34]. In the FSDGSP, the makespan, total flowtime, and TEC are simultaneously optimized.

Hyperheuristic is a general solution framework that selects the optimizer adaptively to address complex optimization problems [35]. In addition, the hyperheuristic algorithm has the characteristics of simple operation and few parameters. The primary distinction between a hyperheuristic algorithm and a heuristic algorithm is that the hyperheuristic algorithm operates on a pile of operations as opposed to acting directly on particular problems [36]. Heuristic algorithms operate on the problem domain directly. The hyperheuristic algorithm is separated from the specific problem situation. Because the high-level strategy is well separated from the problem domain, the algorithm designer will pay more attention to the design of the high-level strategy. Similar to an integrated learning algorithm, hyperheuristic combines the benefits of multiple single learning algorithms organically. The application of the hyperheuristic algorithm to the flow shop scheduling problem was successful. To address distributed assembly permutation flow-shop scheduling problem (DAPFSP), a backtracking search hyperheuristic (BS-HH) algorithm was presented [37]. Certain operators in BS-HH were designed to be LLHs. In addition, a backtracking search was considered a high-level heuristic (HLH) for manipulating these LLHs. A genetic programming hyperheuristic (GPHH) was applied to address DAPFSP with SDST [38]. In GPHH, genetic programming was considered an HLH for generating LLH sequences. Therefore, utilizing the hyperheuristic algorithms to solve the flow shop scheduling problem is feasible and effective.

RL, which does not require the time-consuming metaoptimization procedure, has unique benefits for intelligently guiding the selection of LLHs [39]. Q-learning and hyperheuristic were utilized to select the optimal bioinspired algorithm for the well-known 0–1 knapsack problem [40]. Four bio-inspired algorithms were considered as LLHs, allowing the hyperheuristic and Q-learning to automatically select the heuristic for each optimization cycle. A Q-learning-based hyperheuristic (QHH) algorithm was presented for solving the semiconductor final testing scheduling problem (SFTSP). Q-learning was used to select a heuristic from the LLH set as a high-level strategy [41]. These insights motivate the design of a QHH to address the EEDBFSP.

III. Description of EEDBFSP

In this section, the EEDBFSP is explained through an MILP. Some necessary notations are given as follows to simplify the expression of the formula.

Indices

i	Index for machines, where $i \in \{1, 2, \ldots, m\}$.
j	Index for jobs, where $j \in \{1, 2, \ldots, n\}$.
f	Index for factories, where $f \in \{1, 2, \ldots, F\}$.
k	Index of the job position in a given sequence, where $k \in \{1, 2, \ldots, n^f\}$.
v	Index for speeds, where $v \in \{1, 2, \ldots, s\}$.

Parameters

n	Number of jobs.
m	Number of machines in each factory.
F	Number of factories.
s	Number of speeds.
L	Infinite positive integer.
$o_{j,i}$	Operation of J_j on the M_i.
$p_{j,i}$	Standard processing time of $o_{j,i}$.
$t_{j,i}$	Actual processing time of $o_{j,i}$.
V_v	vth processing speed.
d_j	Due date of the job J_j.

Decision Variables

$D_{f,k,i}$	Departure time of the job in position k on M_i in factory f.
D_j	Departure time of the J_j on the M_m.
$X_{f,k,j}$	Binary variable that takes value 1 if the J_j occupies position k in factory f, otherwise 0.
$Y_{j,i,v}$	Binary variable that takes value 1 if J_j is processed on the M_i at speed V_v, otherwise 0.

The EEDBFSP is described as follows. A set of n jobs $\{J_1, J_2, \ldots, J_j, \ldots, J_n\}$ is allocated to F factories for processing. Each factory is a blocking flow shop that contains m machines $\{M_1, M_2, \ldots, M_i, \ldots, M_m\}$. There are m operations $\{o_{j,1}, o_{j,2}, \ldots, o_{j,i}, \ldots, o_{j,m}\}$ are needed to be processed in turn for each job. Once a job is assigned to a certain factory, it is not allowed to be transferred to another factory for processing. To be specific, all operations belonging to the job J_j must be finished in the order $\{o_{j,1}, o_{j,2}, \ldots, o_{j,m}\}$. No intermediate buffer exists between any two machines. To be more specific, a job J_j is blocked on the machine M_i, while the preceding job is being processed by the machine M_{i-1}. Furthermore, the following constraints must be satisfied. Each job must be processed on one machine at a certain time. Each machine only processes one job at a certain time. Once an operation has begun, it cannot be interrupted until it has been completed.

In addition, the time required for transportation and setup is included in the processing time. The energy consumption of the machines is considered in the DBFSP. There are a series of different speeds $\{V_1, V_2, \ldots, V_s\}$ for each machine. Once a job has been completed on a particular machine, the speed of that machine cannot be altered. The standard processing time of a job J_j, which is processed on a machine M_i, is $p_{j,i}$. The actual processing time of a job J_j is $t_{j,i} = p_{j,i}/V_v$ when the operation $o_{j,i}$ is processed at the speed V_v. In addition, the energy consumption of the machine in blocking mode and the standby mode is considered. Suppose that the energy consumption of the machine M_i at blocking mode is $bp_{f,i}$ per unit time and the energy consumption of the machine M_i at standby mode is $sp_{f,i}$ per unit time.

The MILP model of EEDBFSP is formulated as follows.

$$\text{Objective}: \ \min\{\text{TTD}, \text{TEC}\}$$

$$\text{subject to} \sum_{f=1}^{F}\sum_{k=1}^{n} X_{f,k,j} = 1, j \in \{1, 2, \ldots, n\} \tag{1}$$

$$\sum_{f=1}^{F}\sum_{j=1}^{n} X_{f,k,j} = 1, k \in \{1, 2, \ldots, n\} \tag{2}$$

$$\sum_{v=1}^{s} Y_{j,i,v} = 1 \tag{3}$$

$$j \in \{1, 2, \ldots, n\}, i \in \{1, 2, \ldots, m\}$$

$$t_{j,i} = p_{j,i} \cdot \sum_{v=1}^{s} \frac{Y_{j,i,v}}{V_v}$$

$$j \in \{1, 2, \ldots, n\}, i \in \{1, 2, \ldots, m\} \tag{4}$$

$$D_{f,0,i} \geq 0, i \in \{1, 2, \ldots, m\}, \ f \in \{1, 2, \ldots, F\} \tag{5}$$

$$D_{f,k,0} = D_{f,k-1,1}, k \in \{1, 2, \ldots, n\}, \ f \in \{1, 2, \ldots, F\} \tag{6}$$

$$D_{f,k,i} \geq D_{f,k,i-1} + \sum_{j=1}^{n} X_{f,k,j} \cdot t_{j,i}$$

$$k \in \{1, 2, \ldots, n\}, \ i \in \{1, 2, \ldots, m\}, \ f \in \{1, 2, \ldots, F\}$$

$$D_{f,k,i} \geq D_{f,k-1,i+1}, \ k \in \{2, 3, \ldots, n\} \tag{7}$$

$$i \in \{1, 2, \ldots, m-1\}, \ f \in \{1, 2, \ldots, F\} \tag{8}$$

$$D_j \geq D_{f,k,m} - L \cdot \left(1 - X_{f,k,j}\right)$$

$$k \in \{1, 2, \ldots, n\}, \ j \in \{1, 2, \ldots, n\}, \ f \in \{1, 2, \ldots, F\} \tag{9}$$

$$\text{TTD} \ \geq \sum_{j=1}^{n} \max(D_j - d_j, 0) \tag{10}$$

$$\text{TEC} \ \geq \ \sum_{f=1}^{F}\sum_{k=1}^{n}\sum_{j=1}^{n}\sum_{i=1}^{m} 4V_V^2 \cdot t_{j,i} + \sum_{f=1}^{F}\sum_{k=1}^{n-1}\sum_{i=1}^{m} \left(D_{f,k+1,i-1} - D_{f,k,i}\right) \cdot sp_{f,i} + \sum_{f=1}^{F}\sum_{k=2}^{n}\sum_{i=1}^{m-1} \left(D_{f,k,i} - D_{f,k,i} - \sum_{j=1}^{n} X_{f,k,j} \cdot t_{j,i}\right) \cdot bp_{f,i} \tag{11}$$

$$X_{f,k,j} \in \{0, 1\}$$

$$j \in \{1, 2, \ldots, n\}, \ k \in \{1, 2, \ldots, n\}, f \in \{1, 2, \ldots, F\} \tag{12}$$

$$Y_{j,i,v} \in \{0, 1\}$$

$$j \in \{1, 2, \ldots, n\}, i \in \{1, 2, \ldots, m\}, v \in \{1, 2, \ldots, s\}. \tag{13}$$

The objective of the EEDBFSP is to minimize TTD and TEC. Constraint (1) guarantees that each job is allocated to only one factory and only one position at that factory. Constraint (2) ensures that each position in a certain factory is occupied by only one job. Constraint (3) indicates that each $o_{j,i}$ is processed at only one speed. The actual processing time of $o_{j,i}$ is calculated by (4). Constraint (5) defines the initial condition. $D_{f,0,i}$ indicates the starting time of the first processing job on each machine. Constraint (6) indicates the starting time of a job on the first machine in each factory. Constraint (7) indicates the relationship between the departure time of two adjacent operations for the same job in each factory. Constraint (8) represents the relationship between the departure time of two adjacent jobs in each factory, which indicates that the blocking constraint is satisfied. Constraint (9) indicates the completion time of each job. Constraint (10) and constraint (11) define the objective functions TTD and TEC, respectively. Constraints (12) and (13) define the decision variables. The variable $X_{f,k,j}$ provides the assignment of each job for factories and the job sequence for each factory. The variable $Y_{j,i,v}$ provides the speed selection for each operation.

In order to express the calculation process of the objective function clearly, some additional variables are defined, where n^f represents the number of jobs in the factory f. π_k^f represents the job at position k in factory f. TTD^f represents the TTD in factory f. PEC^f represents the TEC for processing n^f jobs in the factory f. SEC^f represents the TEC of machines at standby mode in factory f. BEC^f represents the TEC of machines at blocking mode in factory f. TTD and TEC of EEDBFSP are calculated as follows. First, the departure time of jobs in each factory is calculated by (14)–(18). Then, TTD is calculated by (19) and(20). Finally, TEC is calculated by (21)–(24)

$$D_{f,1,0} = 0, f \in \{1, 2, \ldots, F\} \tag{14}$$

$$D_{f,1,i} = D_{f,1,i-1} + t_{\pi_k^f,i}$$

$$i \in \{1, 2, \ldots, m\}, \ f \in \{1, 2, \ldots, F\} \tag{15}$$

$$D_{f,k,0} = D_{f,k-1,1}$$

$$k \in \left\{2, 3, \ldots, n^f\right\}, \ f \in \{1, 2, \ldots, F\} \tag{16}$$

$$D_{f,k,i} = \max\left\{D_{f,k,i-1} + t_{\pi_k^f,i}, D_{f,k-1,i+1}\right\}$$

$$k \in \left\{2, 3, \ldots, n^f\right\}, i \in \{1, 2, \ldots, m-1\}, \ f \in \{1, 2, \ldots, F\} \tag{17}$$

$$D_{f,k,m} = D_{f,k,m-1} + t_{\pi_k^f,i}$$

$$k \in \left\{2, 3, \ldots, n^f\right\}, \ f \in \{1, 2, \ldots, F\} \tag{18}$$

$$\text{TTD}^f = \sum_{j=1}^{n^f} \max\left(D_j - d_j, \ 0\right), f \in \{1, 2, \ldots, F\} \tag{19}$$

$$\text{TTD} = \sum_{f=1}^{F} \text{TTD}^f \tag{20}$$

$$\text{PEC}^f = \sum_{i=1}^{m}\sum_{k=1}^{n^f} \left(4 \times \left(V_{\pi_k^f,i}\right)^2 \cdot t_{\pi_k^f,i}\right) \tag{21}$$

$$f \in \{1, 2, \ldots, F\}$$
$$\mathrm{SEC}^f = \sum_{k=1}^{n^f-1} \sum_{i=2}^{m} \left(D_{f,k+1,i-1} - D_{f,k,i}\right) \cdot sp_{f,i} \tag{22}$$
$$f \in \{1, 2, \ldots, F\}$$
$$\mathrm{BEC}^f = \sum_{k=2}^{n^f} \sum_{i=1}^{m-1} \left(D_{f,k,i} - \left(D_{f,k,i-1} + t_{\pi_k^f,i}\right)\right) \cdot bp_{f,i} \tag{23}$$
$$f \in \{1, 2, \ldots, F\}$$
$$\mathrm{TEC} = \sum_{f=1}^{F} \mathrm{PEC}^f + \mathrm{SEC}^f + \mathrm{BEC}^f. \tag{24}$$

Suppose that a solution is encoded as $\Pi = (\pi,\ V)$, where $\pi = \{\pi^1, \pi^2, \ldots, \pi^F\}$ denotes the job processing sequence for each factory, where π^f is the processing sequence of jobs in factory f. $V = (V_{j,i})_{n\times m}$ denotes the speed matrix and $V_{j,i}$ is the processing speed of $o_{j,i}$. Suppose there are eight jobs, three machines, and two factories, where $\pi = \{5, 2, 1, 7; 3, 4, 6, 8\}$, which means that the job processing sequence of factory 1 is ordered as 5→2→1→7, the job processing sequence of factory 2 is ordered as 3→4→6→8. The speed matrix is shown as follows:

$$(V_{j,i})_{n\times m} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{bmatrix}.$$

The indicators, which are standard processing time, processing speed, and actual processing time of each job, are provided in Table XII (supplementary material). The Gantt chart of the example for the EEDBFSP is provided in Fig. 9 (supplementary material). Herein, the processing speed on each machine is randomly initialized to $V_v \in \{1,\ 2\}$. In addition, assume that the variables of $sp_{f,i}$ and $bp_{f,i}$ take values of 1 and 1.5, respectively. The calculating process of objective functions is shown in the supplementary material.

IV. HHQL FOR EEDBFSP

A. Initialization

In the HHQL algorithm, the initial speed of $o_{j,i}$ is generated randomly. Using an extended NEH (ENEH) method, the job sequence for each factory is constructed. First, the first F jobs of the initial jobs sequence σ are allocated to F factories, respectively. Second, the unscheduled jobs $\sigma(i \sim n)$ are allocated to the suitable position with the best TTD. Specifically, the selected job *jobA* is tentatively inserted into all possible factory positions, and the position with the minimum TTD is identified. The selected *jobA* is then inserted into the position *bestPos*.

In addition, an energy-efficient strategy (ESS) is implemented to balance TTD and TEC in the objective function optimization process. Repetition of the procedure mentioned above until the unscheduled jobs set is empty produces a complete solution. The TTD and TEC are then calculated. All individuals of the population are generated in the same manner. Note that, the initial jobs sequence σ of each individual is generated randomly to maintain the diversity of solutions in the population. The procedure of ENEH is shown in Algorithm 1.

Algorithm 1 ENEH

Input: n, $P_{j,i}$, V
Output: π, *TTD*, *TEC*
Initialize the matrix of speed for $o_{j,i}$ and actual processing time of $o_{j,i}$ is computed.
Sort jobs in a descending order according to the actual processing time of each job and an initial job sequence σ is finished.
for $i = 1$ **to** F **do**
  $\pi_1^i \leftarrow \sigma(i)$;
end
count ← **0;**
for $i = F + 1$ **to** n **do**
  $jobA \leftarrow \sigma(i)$, $valueA \leftarrow MAX$;
  for $f = 1$ **to** F **do**
    for $k = 1$ **to** $n^f + 1$ **do**
      $valueB \leftarrow$ Compute *TTD* of the partial solution when inserting the selected *jobA* to the kth position of factory f.
      if $valueB > ValueA$ **then**
        $valueB \leftarrow valueA$, $bestFac \leftarrow f$, $bestPos \leftarrow k$;
      end if
    end for
  end for
  Insert *job* to the best position *bestPos* of the factory *bestFac*;
  count ← count + 1;
  if count % 2 == 1 **do**
  Decrease the processing speed of the job on the non-critical path (ESS).
  end if
end for
Compute the *TTD* and *TEC* of the initial solution π;

B. HHQL

A canonical framework of hyperheuristic has two levels, including a high-level strategy and a set of LLHs [42]. The heuristic selection mechanism and acceptance criterion are two crucial components of high-level strategies. The heuristic selection mechanism determines which LLH to call. The acceptance criterion is then used to decide whether or not to accept the returned solution. The LLHs are a set of problem-specific heuristics. In this study, HHQL seeks the most appropriate LLH by Q-learning-based high-level strategy according to the problem state during the evolution. The framework of the HHQL algorithm is shown in Fig. 1.

Q-learning is used in the heuristic selection mechanism to select the appropriate LLH. As the crucial component in RL, reasonable designs of action and state can accurately describe the scheduling environment and enhance the efficiency of the learning process [43]. In this study, each state corresponds to

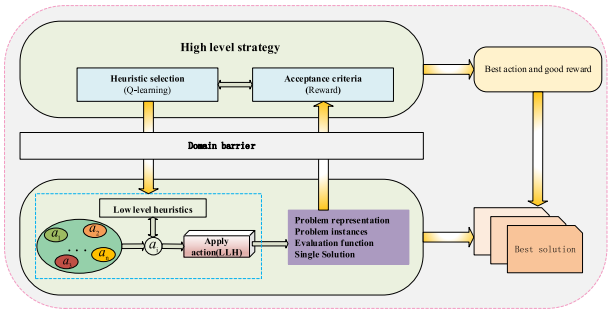


Fig. 1. Framework of the HHQL algorithm.

an individual in the population. Each action is regarded as an LLH. During the evolution of the population, the initial state is determined by selecting the first individual of the initial population. The initial values of the Q-table are set as zero and the initial values of the R-table are set as zero (Algorithm 2, line 7).

An improved ϵ-greedy strategy is applied to select the action (Algorithm 2, line 11). The ϵ-greedy strategy is shown as follows:

$$\epsilon = \frac{0.5}{1 + e^{\frac{10\times(t-0.6\times t_{\max})}{t_{\max}}}} \tag{25}$$

$$\mu(a_t|s_t) = \begin{cases} \text{random } A(s_t), & \text{rand} > 1-\epsilon \\ a^*, & \text{other} \end{cases} \tag{26}$$

where ϵ represents a sample value subject to standard normal distribution at time t. $t_{\max}$ denotes stopping criteria, which equals $0.5 \times n$ seconds. $A(s_t)$ represents the set of all actions. a^* represents the action with maximum q-value at state s_t. Equation (25) indicates that ϵ gradually decreases until it approaches 0, while $1-\epsilon$ gradually increases and approaches 1 with the growth of time t. Equation (26) indicates that the agent has a nearly 50% probability of exploring new actions at the beginning of training. While the possibility that the condition rand $> 1-\epsilon$ holds gradually decreases as time t increases. Agent prefers to select the action with a maximum q-value. In other words, the agent retains a certain degree of exploration in the early stage of action selection (i.e., the selection of LLH) and prefers to use the learned knowledge to guide the selection of action with the increase of training times.

Applying the selected action (LLH) to the current solution yields a candidate solution. Then, acceptance of the candidate solution is determined based on the acceptance criterion (Algorithm 2, lines 13–21). In three cases, the candidate solution is considered acceptable. If the TTD and TEC of the candidate solution are smaller than those of the current solution, respectively, the candidate solution is accepted. The candidate solution is accepted if its TTD is less than the TTD of the current solution and the TEC remains unchanged. The candidate solution is accepted if its TEC is less than the TEC of the current solution and the TTD is not altered. Note that as long as the quality of the candidate solution is not enhanced, the candidate solution is not considered acceptable. The reward is intended to obtain a preference for action through the interaction of agent and environment. Consequently, the rewards and punishments (Algorithm 2, lines 13–21) are distributed based on the acceptance criterion.

Algorithm 2 HHQL

1: **Input:** NP, α, γ, t_{max}
2: **Output:** population
3: **for** $i = 1$ **to** NP **do**
4: Generate the *ith* individual according to **Algorithm 1**;
5: **end**
6: Push the best individual of the initial population into the archive;
7: Initialize Q-table, R-table for each state-action pair.
8: **while** not satisfied (stopping termination condition) **do**
9: $S_{current} \leftarrow$ *ith* individual of the current population;
10: Identify the current state s_t;
11: Select an action a_t (LLH) using the ϵ-greedy strategy;
12: $S_{candidate} \leftarrow$ Apply the selected LLH on the $S_{current}$;
13: **if** $S_{candidate}.TTD < S_{current}.TTD$ **and** $S_{candidate}.TEC < S_{current}.TEC$ **do**
14: $r_t \leftarrow r_t + 2$, $S_{current} \leftarrow S_{candidate}$;
15: **else if** $S_{candidate}.TTD < S_{current}.TTD$ **and** $S_{candidate}.TEC = S_{current}.TEC$ **do**
16: $r_t \leftarrow r_t + 1$, $S_{current} \leftarrow S_{candidate}$;
17: **else if** $S_{candidate}.TEC < S_{current}.TEC$ **and** $S_{candidate}.TTD = S_{current}.TTD$ **do**
18: $r_t \leftarrow r_t + 1$, $S_{current} \leftarrow S_{candidate}$;
19: **else**
20: $r_t \leftarrow r_t - 1$;
21: **end if**
22: $s_t \leftarrow s_t + 1$, $i \leftarrow i + 1$;
23: Update the Q-table according to Eq. (27);
24: **if** $s_t > NP$ **do**
25: $s_t \leftarrow 1$;
26: **end if**
27: **if** the population is updated once **do**
28: Push the best individual of the current population into the archive;
29: Replace the worst individual of the current population with the previous best individual of the archive;
30: **end if**
31: **end while**

If the quality of the solution is improved, this action (LLH) will be rewarded. This action will be punished if the quality of its solution is not improved. The procedure of HHQL is shown in Algorithm 2.

Q-table is utilized to record the learning experience of the agent from the environment. Each row in the Q-table represents a state and each column represents an action. The Q-value of each action a_t at state s_t is calculated by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right) \tag{27}$$

where $Q(s_t, a_t)$ is the q-value of the state s_t when taking action a_t. $\alpha \in [0, 1]$ represents the learning rate. $\gamma \in [0, 1]$ represents the discount rate. r_{t+1} is the real-time reward obtained by the agent after taking action a_t at state s_t. $\max Q(s_{t+1}, a_{t+1})$ is

the maximum Q-value of the Q-table at the state s_{t+1} when taking action a_{t+1}.

C. Low-Level Heuristics

Since TTD and TEC are two conflicting objectives in the optimization process, two critical factories are defined, that is, factory F_t with the maximum TTD^f and the factory F_e with the maximum TEC^f. The reduction of idle and blocking time in EEDBFSP contributes to the overall factory's energy savings. Therefore, adjusting the job allocation for factories and the job sequence at the same processing speed will simultaneously reduce TTD and TEC. Moreover, the speed adjustment can decrease the TTD and TEC. To our best knowledge, LLHs influence the performance of hyperheuristic. A pool of LLHs is constructed using eight simple heuristics. The details are shown as follows.

LLH1: Each job is removed from the factory and inserted into all possible positions from the original factory. The position with the best TTD is selected.

LLH2: Each job is removed from the factory and swapped with all possible positions from that factory. The position with the best TTD is selected.

LLH3: Each job is selected from F_t and inserted into all possible positions from all other factories. The position with the best TTD is selected.

LLH4: Each job is selected from F_t and swapped with all possible positions from all other factories. The position with the best TTD is selected.

LLH5: Each job is selected from F_e and inserted into all possible positions from all other factories. The position with the best TTD is selected.

LLH6: Each job is selected from F_e and swapped with all possible positions from all other factories. The position with the best TTD is selected.

TTD^f is reduced by improving the processing speed of the job on the critical path. As shown in Fig. 2, red arrows represent the critical path. The critical path begins at the last machine of the last processing job. The $o_{j-1,i}$ is the next critical path if no idle time or blocking time exists before the current $o_{j,i}$. Otherwise, two cases are considered. On the one hand, $o_{j,i-1}$ is the next critical path if idle time exists before the current $o_{j,i}$ belonging to the critical path. On the other hand, $o_{j-1,i+1}$ is the next critical path if the blocking time exists before the current $o_{j,i}$ belonging to the critical path. In addition, TTD^f is reduced by improving the processing speed of the job on the last machine without violating constraints. The acceleration operation of the job on the critical path and the deceleration operation on the noncritical path are designed to optimize TTD and TEC. The acceleration operation of the job on the critical path is effective in reducing TTD^f, but TEC^f is increased. TEC^f is reduced by slowing down the speed of jobs on noncritical paths without violating constraints.

LLH7: A job is selected randomly. The speed of $o_{j,i}$ on the critical path is improved a level without violating constraints. Moreover, the speed of $o_{j,i}$ on the noncritical path is slowed down a level when J_j is blocked on M_i without violating constraints.

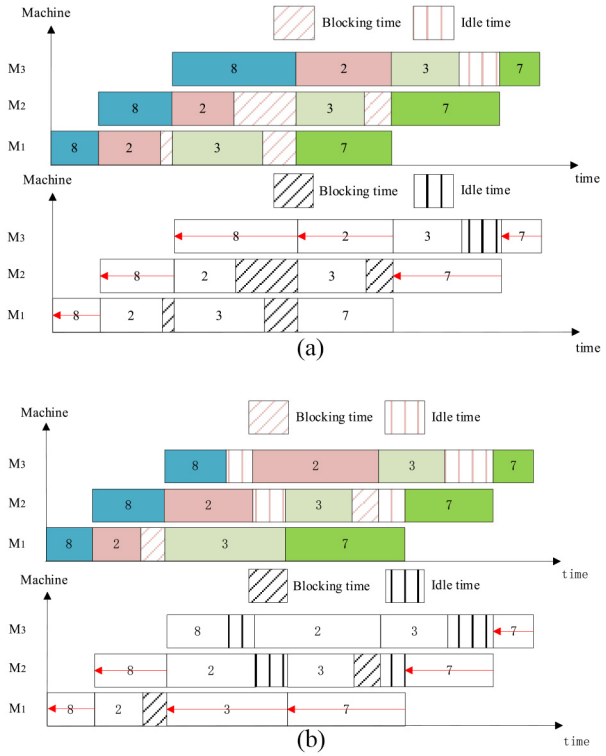


Fig. 2. Illustration of the critical path.

LLH8: For the noncritical path in F_c, all the operations on the noncritical path are reduced by a level without violating constraints.

V. Experiment and Analysis

In this section, Taillard's benchmark set is adopted to test the performance of the proposed algorithm [44], [45]. This benchmark set contains 120 problem instances ranging in size from 20 jobs and 5 machines to 500 jobs and 20 machines. There are ten instances of each size. Regarding the distributed environment, the factory set is $\{2, 3, 4, 5, 6, 7\}$ [3]. The information on the Taillard's benchmark set can be downloaded from http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/flowshop2.txt. The processing speed of each machine is divided into five levels, namely, $V = \{1.00, 1.30, 1.55, 1.75, 2.10\}$. $sp_{f,i}$ is 1.00 kW, and $bp_{f,i}$ is 1.50 kW. The relevant parameters of the energy consumption are shown in Table I. The due date of each job is defined as $d_j = \sum_{i=1}^{m} p_{j,i}$. Each instance is independently executed five times. In addition, all algorithms are coded using the programming language Java to guarantee the fairness of the experiment. The experiment strictly adheres to the parameters outlined in the original literature. The simulation experiments are conducted on a single server with Intel Xeon gold 5218 CPU @ 2.30 GHz 2.29-GHz processors and 64 GB of RAM running Windows Server 2019. Due to the fact that the EEDBFSP is a multiobjective optimization problem, two comparison measurements, the overall nondominated vector generation (ONVG) and C metric, are used together to assess the quality of the obtained Pareto set.

TABLE I
PARAMETERS OF ENERGY CONSUMPTION

v	V_v	$pp_{f,i,v}$	$sp_{f,i}$	$bp_{f,i}$
1	1.00			
2	1.30			
3	1.55	$4 \times V_v^2$	1.00	1.50
4	1.75			
5	2.10			

ONVG: The number of nondominated solutions obtained by the final Pareto set.

C Metric: The dominance relationship between the solutions in two Pareto set U_1 and U_2. $C(U_1, U_2)$ is calculated as follows:

$$C(U_1,\ U_2) = |\{b \in U_2 | \exists\ a \in U_1,\ a \succ b \text{ or } a = b\}|/|U_2|. \tag{28}$$

The greater $C(U_1, U_2)$ is, the more solutions in the set U_2 are dominated by solutions in the set U_1, indicating that the algorithm for generating set U_1 is superior to that of U_2. Generally speaking, the larger the ONVG and C Metric are, the better the Pareto solution set is. However, if the ONVG of the set U_1 is larger than that of the set U_2, but the solutions in the set U_1 is more dominated by the solutions in the set U_2, it means that the set U_2 is better and the set U_1 is simply dominant in ONVG. The algorithm name for generating the Pareto set is consistent with the set name to facilitate description.

Three critical parameters are involved in HHQL, including the population size PS, the learning rate α, and the discount rate γ. PS is set to five levels, which are PS $= \{20, 30, 40, 50, 60\}$. α is the amount that weighs the previous learning result against this learning result. When α is set too low, the agent only emphasizes prior knowledge, and no new reward is accumulated. The impact of past performance on the Q-value of state–action pairs decreases as α increases. In general, 0.5 is taken to balance the previous knowledge and the new reward [46]. γ is a factor that considers the impact of future rewards on the present. Generally, the external reward is fully considered by taking a larger γ. Therefore, γ is set to three levels, which are $\gamma = \{0.7, 0.8, 0.9\}$. Sixty instances are randomly selected from the instances with $f = 2$ to calibrate the parameters of HHQL. For each instance, the HHQL with each parameter runs five times independently to obtain a nondominant set $E_i (i = 1, 2, \ldots, 15)$. The nondominant solutions among $E_1 - E_{15}$ form the final set FE. The contribution of E_i is defined as follows:

$$\text{con}(i) = \left|E_i'\right|/|FE| \tag{29}$$

where $E_i' = E_i \cap FE$. After testing all the instances, it calculates the average contribution of each parameter combination as the response variable value (RV). The RV value of each set of parameter combinations is shown in Table II.

The results of ANOVA for parameters calibration are shown in Table III and the main effects plot of the control parameters is shown in Fig. 3. From Table III and Fig. 3, PS is the main parameter affecting the performance of the HHQL. From the above experiments, parameters are set as PS $= 60$, and $\gamma = 0.8$ for the following tests and comparisons.

TABLE II
ORTHOGONAL TABLE AND RV VALUES

No.	Parameter combinations		RV(%)
	PS	γ	
1	20	0.7	5.84
2	20	0.8	6.09
3	20	0.9	5.86
4	30	0.7	4.98
5	30	0.8	4.67
6	30	0.9	4.62
7	40	0.7	6.19
8	40	0.8	7.25
9	40	0.9	6.73
10	50	0.7	6.05
11	50	0.8	7.59
12	50	0.9	8.43
13	60	0.7	9.06
14	60	0.8	8.36
15	60	0.9	6.27

TABLE III
RESULTS OF ANOVA FOR THE PARAMETERS CALIBRATION

Source	Sum Sq.	d. f.	Mean Sq.	F	Prob>F
PS	18.2796	4	4.56991	5.01	0.0256
γ	0.5088	2	0.25441	0.28	0.7636
Error	7.2957	8	0.91196		
Total	26.0841	14			

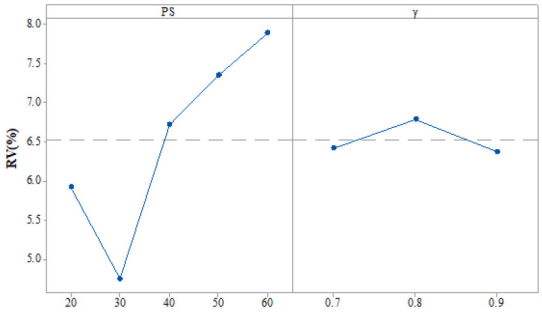


Fig. 3. Main effects plot of the control parameters.

The comparative results of HHQL and CPLEX-12.9 optimizer on the small-scale instance are shown in Table IV. From the table, several groups of experiments with $n = \{20,\ 50\}$, $m = \{2,\ 3, 4\}$, $f = 2$, are utilized to compare the performance of HHQL and CPLEX from ONVG and C Metric. From the two indicators, the ONVG of HHQL is greater than that of CPLEX, and the C Metric of HHQL is greater than that of CPLEX. In addition, the average time consumed by the HHQL algorithm to solve these instances is lower than that of CPLEX. Therefore, the HHQL algorithm is effective for solving EEDBFSP.

TABLE IV
COMPARATIVE RESULTS OF HHQL AND CPLEX-12.9 OPTIMIZER ON THE SMALL-SCALE INSTANCE

Instances	$ONVG$		C Metric		CPU time (s)	
$n \times m \times f$	$ONVG(HHQL)$	$ONVG(CPLEX)$	$C(HHQL, CPLEX)$	$C(CPLEX, HHQL)$	HHQL	CPLEX
$20 \times 5 \times 2$	**17.2**	14.2	**0.89**	0.09	**11.53**	16.21
$20 \times 10 \times 2$	**16.0**	12.8	**0.82**	0.10	**12.36**	18.14
$20 \times 20 \times 2$	**14.6**	14.2	**0.85**	0.12	**14.07**	19.37
$50 \times 5 \times 2$	**17.0**	13.4	**0.69**	0.18	**25.45**	34.65
$50 \times 10 \times 2$	**13.2**	11.2	**0.66**	0.13	**27.24**	37.06
$50 \times 20 \times 2$	**14.4**	11.0	**0.73**	0.11	**28.91**	41.32

TABLE V
ONVG OF TWO INITIALIZATION POPULATION METHODS

Instance	$f = 2$		$f = 3$		$f = 4$	
	$ONVG(M_1)$	$ONVG(M_2)$	$ONVG(M_1)$	$ONVG(M_2)$	$ONVG(M_1)$	$ONVG(M_2)$
20×5	15.0	**19.2**	17.8	**30.0**	15.2	**30.2**
20× 10	**14.0**	10.8	**13.8**	10.2	55.2	**55.8**
20× 20	**13.2**	9.2	**58.4**	55.2	273.8	**278.2**
50×5	**17.0**	10.4	**13.4**	9.0	**14.8**	11.2
50× 10	**11.2**	10.4	**14.0**	9.8	**14.0**	10.4
50× 20	**12.2**	9.0	**10.8**	7.6	**9.8**	7.8
100×5	**10.2**	5.4	**12.8**	5.4	**12.0**	8.6
100× 10	**9.2**	5.8	**11.2**	10.0	**11.0**	10.8
100× 20	**11.6**	10.4	**12.6**	9.8	**10.8**	8.0
200× 10	10.4	**11.0**	8.6	**10.2**	**11.2**	7.4
200× 20	9.6	**16.6**	12.0	**15.2**	**11.6**	**11.6**
500× 20	8.8	**16.0**	7.8	**19.0**	7.2	**30.2**

TABLE VI
C MATRIC OF TWO INITIALIZATION POPULATION METHODS

Instance	$f = 2$		$f = 3$		$f = 4$	
	$C(M_1, M_2)$	$C(M_2, M_1)$	$C(M_1, M_2)$	$C(M_2, M_1)$	$C(M_1, M_2)$	$C(M_2, M_1)$
20×5	**0.90**	0.09	**0.79**	0.06	**0.77**	0.14
20× 10	**0.80**	0.10	**0.80**	0.09	**0.15**	0.00
20× 20	**0.75**	0.09	**0.00**	**0.00**	**0.00**	**0.00**
50×5	**0.55**	0.16	**0.32**	0.19	**0.62**	0.16
50× 10	**0.69**	0.02	**0.71**	0.15	**0.53**	0.10
50× 20	**0.63**	0.14	**0.68**	0.15	**0.42**	0.27
100×5	**0.40**	0.12	**0.65**	0.19	**0.67**	0.08
100× 10	**0.33**	0.23	**0.80**	0.09	**0.75**	0.15
100× 20	**0.60**	0.13	**0.61**	0.20	**0.59**	0.10
200× 10	**0.81**	0.07	**0.97**	0.03	**0.88**	0.07
200× 20	**0.89**	0.02	**0.95**	0.00	**0.91**	0.08
500× 20	**0.87**	0.02	**0.80**	0.00	**0.96**	0.00
Average	**0.69**	0.10	**0.67**	0.10	**0.60**	0.10

Two methods are compared to demonstrate the effect of the initialization population method. The first one (abbreviated as $M1$) is that each individual of the population initializes the speed matrix randomly in the process of generating the initial solution. The second one (abbreviated as $M2$) is that the speed matrix is initialized randomly and all the individuals employ this speed metric to generate the initial solution. Three groups of experiments with $f = \{2, 3, 4\}$ are utilized to compare the performance of $M1$ and $M2$ from ONVG and C Metric. Although ONVG of $M2$ is larger than that of $M1$ under the instances 20×5, 200×10, 200×20, and 500×20 from Table V, the most of nondominated solutions in $M2$ is dominated by the nondominated solutions in $M1$ from Table VI. In addition, the C Metric of instance 20×20 under $f2$ and $f3$ is zero, which means that there exists no nondominated relationship between two nondominated solution sets generated by $M1$ and $M2$ separately. The C Metric for the two algorithms in each of the six factories is plotted as shown in Fig. 4. $M1$ outperforms $M2$ at $f = \{2, 3, 4\}$ with 95% confidence level intervals, which indicates the Pareto set obtained by $M1$ is superior to the Pareto set obtained by $M2$. In general, the performance of $M1$ is superior to that of $M2$.

TABLE VII
COMPARISON OF ONVG BETWEEN HHQL AND R ALGORITHM

Instance	$f=2$		$f=3$		$f=4$	
	$ONVG(HHQL)$	$ONVG(R)$	$ONVG(HHQL)$	$ONVG(R)$	$ONVG(HHQL)$	$ONVG(R)$
20×5	15.0	**19.2**	17.8	**19.0**	**15.2**	11.4
20× 10	**14.0**	13.6	13.8	**14.0**	55.2	**61.8**
20× 20	**13.2**	12.0	58.4	**63.0**	273.8	**287.6**
50×5	**17.0**	16.8	13.4	**14.4**	**14.8**	13.0
50× 10	11.2	**12.8**	**14.0**	11.4	14.0	**14.2**
50× 20	12.2	**15.6**	10.8	**12.4**	**9.8**	9.2
100×5	10.2	**11.8**	**12.8**	12.2	**12.0**	10.6
100× 10	9.2	**10.8**	**11.2**	10.6	**11.0**	9.8
100× 20	11.6	**13.2**	**12.6**	11.8	**10.8**	**10.8**
200× 10	10.4	9.0	8.6	**9.0**	**11.2**	8.4
200× 20	9.6	7.6	**12.0**	10.2	**11.6**	9.4
500× 20	8.8	**9.0**	7.8	**8.8**	7.2	**9.2**

TABLE VIII
COMPARISON OF C METRIC BETWEEN HHQL AND R ALGORITHM

Instance	$f=2$		$f=3$		$f=4$	
	$C(HHQL,R)$	$C(R,HHQL)$	$C(HHQL,R)$	$C(R,HHQL)$	$C(HHQL,R)$	$C(R,HHQL)$
20×5	**0.40**	0.36	**0.48**	0.47	**0.50**	0.37
20× 10	**0.41**	0.34	**0.44**	0.23	**0.00**	**0.00**
20× 20	**0.50**	0.45	**0.00**	**0.00**	**0.00**	**0.00**
50×5	**0.55**	0.40	**0.38**	0.37	**0.52**	0.32
50× 10	**0.37**	0.35	**0.48**	0.27	**0.43**	0.31
50× 20	**0.60**	0.34	**0.51**	0.39	**0.44**	0.35
100×5	**0.57**	0.37	**0.58**	0.20	**0.49**	0.36
100× 10	**0.70**	0.17	**0.38**	0.36	**0.45**	0.38
100× 20	**0.56**	0.47	**0.49**	0.43	**0.43**	0.41
200× 10	**0.42**	0.41	**0.44**	0.36	**0.48**	0.26
200× 20	**0.41**	0.28	**0.68**	0.27	**0.45**	0.37
500× 20	**0.54**	0.45	**0.43**	0.29	**0.39**	0.37
Average	**0.50**	0.37	**0.44**	0.30	**0.38**	0.29

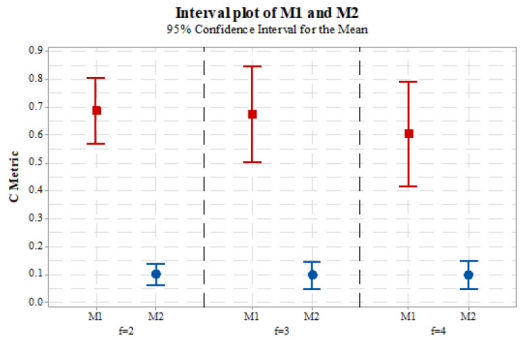


Fig. 4. Interval plot of $M1$ and $M2$.

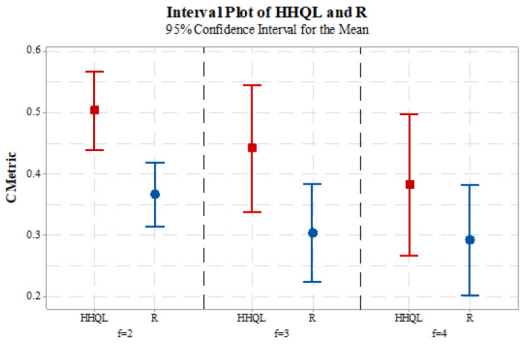


Fig. 5. Interval plot of HHQL and R.

To analyze the effectiveness of the ϵ-greedy strategy, the ϵ-greedy strategy is removed from the HHQL (abbreviated as R). R selects the action with the maximum q-value in the corresponding state. ONVG and C *metric* are used to evaluate the effectiveness of the ϵ-greedy strategy. From Table VII, there is no significant difference between the ONVG of HHQL and R. Nevertheless, the C *metric* of HHQL is larger than that of R from Table VIII. This means that when the number of nondominated solutions is almost the same, the number of nondominated solutions of R dominated by HHQL is relatively large. The C metric for the two algorithms in each of the six factories is plotted as shown in Fig. 5. The HHQL outperforms R at $f = \{2, 3, 4\}$ with 95% confidence level intervals, which indicates the Pareto set obtained by the HHQL is superior to the Pareto set obtained by R. To validate the performance of the HHQL algorithm for solving EEDBFSP, the HHQL algorithm was compared with the mainstream algorithm for solving similar energy-efficient DFSPs. KCA is the comparison algorithm [31]. The KCA algorithm is utilized to optimize the energy consumption of DPFSP.

TABLE IX
COMPARISON OF ONVG BETWEEN HHQL AND KCA

Instances	$f=2$		$f=3$		$f=4$		$f=5$		$f=6$		$f=7$	
	A	A'	A	A'	A	A'	A	A'	A	A'	A	A'
20×5	**16.8**	10.6	**18.1**	10.7	**16.2**	11.2	**21.2**	8.8	**96.0**	6.7	**219.8**	6.4
20× 10	**16.3**	8.0	**13.4**	8.7	**43.0**	6.6	**226.2**	6.8	**284.9**	6.8	**290.9**	7.2
20× 20	**11.4**	7.1	**71.3**	7.2	**284.9**	9.9	**294.9**	28.8	**293.9**	39.3	**295.1**	39.7
50×5	**12.9**	6.2	**13.8**	7.7	**15.1**	9.1	**15.8**	8.9	**17.3**	8.6	**15.2**	8.9
50× 10	**12.0**	7.5	**12.2**	6.8	**13.5**	7.3	**13.2**	6.6	**13.0**	7.8	**10.3**	7.8
50× 20	**12.6**	7.0	**11.5**	5.8	**11.1**	5.4	**10.5**	6.9	**9.3**	5.5	**12.2**	6.6
100×5	**11.4**	5.5	**11.2**	6.1	**11.7**	6.4	**12.3**	7.5	**11.9**	7.2	**13.8**	9.1
100× 10	**11.5**	6.3	**10.2**	4.9	**11.0**	4.3	**11.6**	6.3	**11.5**	6.6	**10.8**	7.9
100× 20	**11.2**	6.0	**12.3**	6.1	**11.9**	5.6	**11.5**	5.4	**10.9**	6.5	**11.3**	7.4
200× 10	**10.1**	5.0	**9.4**	5.1	**12.2**	4.3	**9.4**	4.6	**10.3**	5.9	**10.2**	5.0
200× 20	**9.7**	5.2	**11.1**	4.9	**11.6**	4.9	**10.7**	5.4	**11.8**	5.2	**11.1**	5.9
500× 20	**8.0**	4.7	**9.8**	4.2	**8.3**	4.6	**8.8**	5.0	**9.2**	4.6	**7.1**	4.2

TABLE X
COMPARISON OF C METRIC BETWEEN HHQL AND KCA

Instance	$f=2$		$f=3$		$f=4$		$f=5$		$f=6$		$f=7$	
	B	B'	B	B'	B	B'	B	B'	B	B'	B	B'
20×5	**0.69**	0.11	**0.70**	0.08	**0.64**	0.06	**0.54**	0.02	**0.55**	0.02	**0.43**	0.00
20× 10	**0.74**	0.11	**0.62**	0.04	**0.43**	0.04	**0.29**	0.00	**0.11**	0.00	**0.00**	0.00
20× 20	**0.83**	0.06	**0.41**	0.00	**0.07**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
50×5	**0.94**	0.00	**0.97**	0.00	**0.94**	0.00	**0.93**	0.00	**0.89**	0.00	**0.83**	0.00
50× 10	**1.00**	0.00	**0.95**	0.00	**0.96**	0.00	**0.82**	0.00	**0.74**	0.00	**0.51**	0.00
50× 20	**0.98**	0.00	**0.98**	0.00	**0.94**	0.01	**0.75**	0.00	**0.40**	0.00	**0.30**	0.00
100×5	**1.00**	0.00	**1.00**	0.00	**1.00**	0.00	**1.00**	0.00	**0.99**	0.00	**1.00**	0.00
100× 10	**0.97**	0.00	**0.96**	0.00	**1.00**	0.01	**1.00**	0.00	**1.00**	0.00	**0.99**	0.00
100× 20	**0.79**	0.06	**0.91**	0.03	**0.91**	0.08	**0.95**	0.02	**0.92**	0.02	**0.80**	0.00
200× 10	**0.42**	0.04	**0.63**	0.01	**0.67**	0.04	**0.66**	0.00	**0.98**	0.00	**0.94**	0.01
200× 20	**0.18**	0.07	**0.30**	0.06	**0.39**	0.01	**0.57**	0.11	**0.73**	0.06	**0.77**	0.06
500× 20	**0.07**	0.00	**0.02**	0.00	**0.03**	0.00	**0.06**	0.00	**0.04**	0.00	**0.03**	0.01
Average	**0.72**	0.04	**0.70**	0.02	**0.67**	0.02	**0.63**	0.01	**0.61**	0.01	**0.55**	0.01

The performance of the KCA is evaluated on 120 Ta problems using the ONVG and C Metric by the size of the test problem. The comparison of ONVG and C Metric between HHQL and KCA are shown in Tables IX and X, respectively. To simplify the expression, A = ONVG(HHQL), A' = ONVG(KCA), B = C(HHQL, KCA), and B' = C(KCA, HHQL). The HHQL algorithm generates a greater number of nondominated solutions than does the KCA algorithm. In particular, the ONVG of the HHQL algorithm increases as the number of factories for the 20×10 and 20×20 instances increases. As we all know, as the number of factories increases, TTD will tend toward zero, and when TTD is zero, the current solution is not dominated by any other solution. Nevertheless, it is insufficient to evaluate the performance of the algorithm-based solely on the number of nondominated solutions. The dominance relationship between the solutions in two Pareto sets of the compared algorithm is shown in Table X. To intuitively observe the dominance relationship between the solutions in two Pareto sets, the comparison of the C Metric at 95% confidence intervals is plotted in Fig. 6. Based on Table X and Fig. 6, the nondominated solutions obtained by the KCA are nearly always dominated by the nondominated solutions obtained by the HHQL algorithm, indicating that the HHQL algorithm outperforms the KCA algorithm.

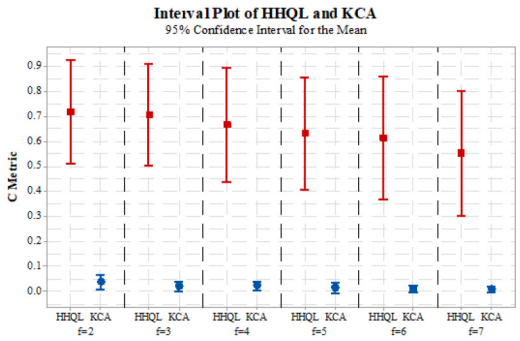


Fig. 6. Interval plot of HHQL and KCA.

The Pareto front is a curve that represents the two optimization objectives. The algorithm performs better the closer the generated curve is to the origin. Six examples from various factories are provided to plot the Pareto fronts curve. As shown in Fig. 7, the Pareto front curve obtained by the HHQL algorithm is closer to the origin than that of the

TABLE XI
WILCOXON'S TEST OF C METRIC AND ONVG FOR $\alpha = 0.05$ AND $\alpha = 0.10$ CONFIDENCE LEVEL

HHQL vs	Metrics	R^+	R^-	Z	p-value	$\alpha = 0.05$	$\alpha = 0.10$
KCA	C	2.54E+03	1.20E+01	-7.25E+00	0.00E+00	Yes	Yes
	$ONVG$	7.80E+01	0.00E+00	-3.06E+00	2.00E-03	Yes	Yes

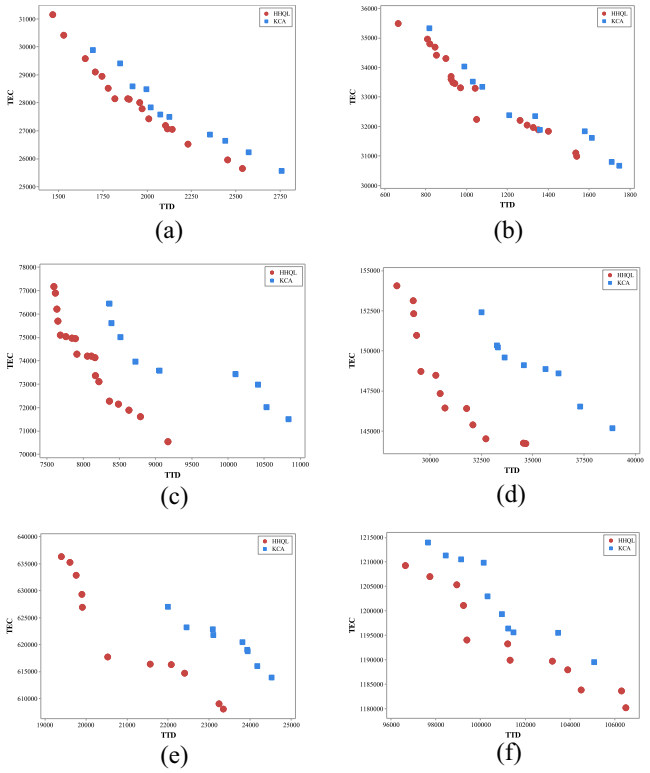


Fig. 7. Pareto front for the HHQL and KCA. (a) $f = 2$ and Ta $= 10$. (b) $f = 3$ and Ta $= 4$. (c) $f = 4$ and Ta $= 40$. (d) $f = 5$ and Ta $= 70$. (e) $f = 6$ and Ta $= 90$. (f) $f = 7$ and Ta $= 106$.

KCA algorithm. Therefore, the accuracy of the nondominated solution is superior to that of the KCA algorithm. In addition, the HHQL algorithm outperforms the KCA in terms of distribution and convergence of nondominated solutions. Based on Fig. 7, the distribution of the nondominated solution is more uniform and solution convergence is more compact for the HHQL algorithm than for the KCA algorithm.

The results of the Wilcoxon test for the compared algorithm are given in Table XI. The HHQL algorithm outperformed the KCA algorithm for both the C Metric and ONVG with 95% and 90% confidence intervals, indicating that the HHQL algorithm is capable of producing more uniformly distributed nondominated solutions than the compared algorithm. In solving the EEDBFSP, the HHQL algorithm is more effective than the current algorithm according to the analysis mentioned above.

VI. CONCLUSION AND FUTURE WORK

This article investigates the EEDBFSP with the objective of minimizing total energy and TTD. In terms of solution quality, numerical experimental results demonstrate that the HHQL algorithm has better performance than the well-established algorithms. Therefore, the proposed HHQL is confirmed to be an effective method for increasing factory efficiency by optimizing the goals determined by the decision maker. The proposed high-level strategy based on Q-learning provides a good idea for solving complex problems. However, the proposed algorithm may not be applied directly to other scheduling problems. Since decision makers need to design LLHs related to the problem. In addition, the proposed algorithm is not completely autonomous and requires some special experience. The decision makers need to design relevant reward mechanisms and return functions according to the specific problem, which are urgent problems to be solved.

Distributed production scheduling in a heterogeneous environment, distributed production scheduling in an uncertain environment, and distributed flexible hybrid flow shop scheduling is more in line with the actual production scenario. when considering future work. In DBFSP, it is crucial to consider constraint conditions, such as multitask allocation and sequence-dependent setup time. In addition, problem-specific knowledge and historical information hidden in the algorithm are extracted for guiding the algorithm to perform a search for a promising region. Therefore, the essence of the problem and intelligent algorithm is the key to DBFSP. Furthermore, more indicators affecting energy consumption will be explored.

REFERENCES

[1] J.-Y. Ding, S. Song, and C. Wu, "Carbon-efficient scheduling of flow shops by multi-objective optimization," *Eur. J. Oper. Res.*, vol. 248, no. 3, pp. 758–771, Feb. 2016, doi: 10.1016/J.EJOR.2015.05.019.

[2] Y.-Z. Li, Q.-K. Pan, K.-Z. Gao, M.-F. Tasgetiren, B. Zhang, and J.-Q. Li, "A green scheduling algorithm for the distributed flowshop problem," *Appl. Soft Comput.*, vol. 109, Sep. 2021, Art. no. 107526, doi: 10.1016/J.ASOC.2021.107526.

[3] F. Zhao, R. Ma, and L. Wang, "A self-learning discrete Jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system," *IEEE Trans. Cybern.*, early access, Aug. 20, 2021, doi: 10.1109/TCYB.2021.3086181.

[4] F. Zhao, L. Zhao, L. Wang, and H. Song, "An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion," *Expert Syst. Appl.*, vol. 160, Dec. 2020, Art. no. 113678, doi: 10.1016/J.ESWA.2020.113678.

[5] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976, doi: 10.1287/MOOR.1.2.117.

[6] R. Ruiz, Q.-K. Pan, and B. Naderi, "Iterated greedy methods for the distributed permutation flowshop scheduling problem," *Omega*, vol. 83, pp. 213–222, Mar. 2019, doi: 10.1016/j.omega.2018.03.004.

[7] W. Luo, Y. Qiao, X. Lin, P. Xu, and M. Preuss, "Hybridizing niching, particle swarm optimization, and evolution strategy for multimodal optimization," *IEEE Trans. Cybern.*, vol. 52, no. 7, pp. 6707–6720, Jul. 2022, doi: 10.1109/TCYB.2020.3032995.

[8] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling," *IEEE Trans. Cybern.*, vol. 51, no. 4, pp. 1797–1811, Apr. 2021, doi: 10.1109/TCYB.2020.3024849.

[9] K. Gao, F. Yang, M. Zhou, Q. Pan, and P. N. Suganthan, "Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm," *IEEE Trans. Cybern.*, vol. 49, no. 5, pp. 1944–1955, May 2019, doi: 10.1109/TCYB.2018.2817240.

[10] R. Chen, B. Yang, S. Li, and S. Wang, "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem," *Comput. Ind. Eng.*, vol. 149, Nov. 2020, Art. no. 106778, doi: 10.1016/J.CIE.2020.106778.

[11] Z. Shao, D. Pi, and W. Shao, "Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment," *Expert Syst. Appl.*, vol. 145, May 2020, Art. no. 113147, doi: 10.1016/J.ESWA.2019.113147.

[12] B. H. Abed-Alguni and M. A. Ottom, "Double delayed Q-learning," *Int. J. Artif. Intell.*, vol. 16, no. 2, pp. 41–59, 2018.

[13] B. Abedalguni, "Bat Q-learning algorithm," *Jordanian J. Comput. Inf. Technol.*, vol. 3, no. 1, pp. 51–70, 2017, doi: 10.5455/jjcit.71-1480540385.

[14] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, May 1992, doi: 10.1007/BF00992698.

[15] Z. Pan, D. Lei, and L. Wang, "A knowledge-based two-population optimization algorithm for distributed energy-efficient parallel machines scheduling," *IEEE Trans. Cybern.*, vol. 52, no. 6, pp. 5051–5063, Jun. 2022, doi: 10.1109/TCYB.2020.3026571.

[16] B. Naderi and R. Ruiz, "The distributed permutation flowshop scheduling problem," *Comput. Oper. Res.*, vol. 37, no. 4, pp. 754–768, Apr. 2010, doi: 10.1016/J.COR.2009.06.019.

[17] X.-L. Jing, Q.-K. Pan, L. Gao, and Y.-L. Wang, "An effective Iterated Greedy algorithm for the distributed permutation flowshop scheduling with due windows," *Appl. Soft Comput.*, vol. 96, Nov. 2020, Art. no. 106629, doi: 10.1016/J.ASOC.2020.106629.

[18] W. Shao, Z. Shao, and D. Pi, "Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem," *Knowl. Based Syst.*, vol. 194, Apr. 2020, Art. no. 105527, doi: 10.1016/J.KNOSYS.2020.105527.

[19] J. Zheng, L. Wang, and J.-J. Wang, "A cooperative coevolution algorithm for multi-objective fuzzy distributed hybrid flow shop," *Knowl. Based Syst.*, vol. 194, Apr. 2020, Art. no. 105536, doi: 10.1016/J.KNOSYS.2020.105536.

[20] Z. Shao, W. Shao, and D. Pi, "Effective constructive heuristic and metaheuristic for the distributed assembly blocking flow-shop scheduling problem," *Appl. Intell.*, vol. 50, no. 12, pp. 4647–4669, Jul. 2020, doi: 10.1007/S10489-020-01809-X.

[21] S. Aqil and K. Allali, "Two efficient nature inspired meta-heuristics solving blocking hybrid flow shop manufacturing problem," *Eng. Appl. Artif. Intell.*, vol. 100, Apr. 2021, Art. no. 104196, doi: 10.1016/J.ENGAPPAI.2021.104196.

[22] I. Ribas, R. Companys, and X. Tort-Martorell, "Efficient heuristics for the parallel blocking flow shop scheduling problem," *Expert Syst. Appl.*, vol. 74, pp. 41–54, May 2017, doi: 10.1016/J.ESWA.2017.01.006.

[23] G. Zhang, K. Xing, and F. Cao, "Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion," *Eng. Appl. Artif. Intell.*, vol. 76, pp. 96–107, Nov. 2018, doi: 10.1016/J.ENGAPPAI.2018.09.005.

[24] B. H. Abed-Alguni and N. A. Alawad, "Distributed Grey Wolf Optimizer for scheduling of workflow applications in cloud environments," *Appl. Soft Comput.*, vol. 102, Apr. 2021, Art. no. 107113, doi: 10.1016/j.asoc.2021.107113.

[25] N. A. Alawad and B. H. Abed-Alguni, "Discrete Jaya with refraction learning and three mutation methods for the permutation flow shop scheduling problem," *J. Supercomput.*, vol. 78, no. 3, pp. 3517–3538, Feb. 2022, doi: 10.1007/s11227-021-03998-9.

[26] G. Wang, L. Gao, X. Li, P. Li, and M. F. Tasgetiren, "Energy-efficient distributed permutation flow shop scheduling problem using a nobjective whale swarm algorithm," *Swarm Evol. Comput.*, vol. 57, Sep. 2020, Art. no. 100716, doi: 10.1016/J.SWEVO.2020.100716.

[27] J. Mou, P. Duan, L. Gao, X. Liu, and J. Li, "An effective hybrid collaborative algorithm for energy-efficient distributed permutation flow-shop inverse scheduling," *Future Gener. Comput. Syst.*, vol. 128, pp. 521–537, Mar. 2022, doi: 10.1016/J.FUTURE.2021.10.003.

[28] J. Deng, L. Wang, C. Wu, J. Wang, and X. Zheng, "A competitive memetic algorithm for carbon-efficient scheduling of distributed flow-shop," in *Intelligent Computing Theories and Application* (Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, 9771)). Cham, Switzerland: Springer, 2016, pp. 476–488, doi: 10.1007/978-3-319-42291-6_48.

[29] Y. Han, J. Li, H. Sang, Y. Liu, K. Gao, and Q. Pan, "Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time," *Appl. Soft Comput.*, vol. 93, Aug. 2020, Art. no. 106343, doi: 10.1016/J.ASOC.2020.106343.

[30] A. Che, X. Wu, J. Peng, and P. Yan, "Energy-efficient bi-objective single-machine scheduling with power-down mechanism," *Comput. Oper. Res.*, vol. 85, pp. 172–183, Sep. 2017, doi: 10.1016/J.COR.2017.04.004.

[31] J.-J. Wang and L. Wang, "A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 5, pp. 1805–1819, May 2020, doi: 10.1109/TSMC.2017.2788879.

[32] J.-F. Chen, L. Wang, and Z.-P. Peng, "A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling," *Swarm Evol. Comput.*, vol. 50, Nov. 2019, Art. no. 100557, doi: 10.1016/J.SWEVO.2019.100557.

[33] H.-X. Qin *et al.*, "An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem," *Swarm Evol. Comput.*, vol. 69, Mar. 2022, Art. no. 100992, doi: 10.1016/J.SWEVO.2021.100992.

[34] X. He, Q.-K. Pan, L. Gao, L. Wang, and P. N. Suganthan, "A greedy cooperative co-evolution ary algorithm with problem-specific knowledge for multi-objective flowshop group scheduling problems," *IEEE Trans. Evol. Comput.*, early access, Sep. 27, 2021, doi: 10.1109/TEVC.2021.3115795.

[35] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *Eur. J. Oper. Res.*, vol. 285, no. 2, pp. 405–428, Sep. 2020, doi: 10.1016/J.EJOR.2019.07.073.

[36] B. Dong, L. Jiao, and J. Wu, "A two-phase knowledge based hyper-heuristic scheduling algorithm in cellular system," *Knowl. Based Syst.*, vol. 88, pp. 244–252, Nov. 2015, doi: 10.1016/J.KNOSYS.2015.07.028.

[37] J. Lin, Z.-J. Wang, and X. Li, "A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem," *Swarm Evol. Comput.*, vol. 36, pp. 124–135, Oct. 2017, doi: 10.1016/J.SWEVO.2017.04.007.

[38] H.-B. Song and J. Lin, "A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times," *Swarm Evol. Comput.*, vol. 60, Feb. 2021, Art. no. 100807, doi: 10.1016/J.SWEVO.2020.100807.

[39] S. S. Choong, L.-P. Wong, and C. P. Lim, "Automatic design of hyper-heuristic based on reinforcement learning," *Inf. Sci.*, vols. 436–437, pp. 89–107, Apr. 2018, doi: 10.1016/J.INS.2018.01.005.

[40] İ. Gölcük and F. B. Ozsoydan, "Q-learning and hyper-heuristic based algorithm recommendation for changing environments," *Eng. Appl. Artif. Intell.*, vol. 102, Jun. 2021, Art. no. 104284, doi: 10.1016/J.ENGAPPAI.2021.104284.

[41] J. Lin, Y.-Y. Li, and H.-B. Song, "Semiconductor final testing scheduling using Q-learning based hyper-heuristic," *Expert Syst. Appl.*, vol. 187, Jan. 2022, Art. no. 115978, doi: 10.1016/J.ESWA.2021.115978.

[42] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 217–228, Feb. 2015, doi: 10.1109/TCYB.2014.2323936.

[43] L. Wang, Z. Pan, and J. Wang, "A review of reinforcement learning based intelligent optimization for manufacturing scheduling," *Complex Syst. Model. Simul.*, vol. 1, no. 4, pp. 257–270, Dec. 2021, doi: 10.23919/CSMS.2021.0027.

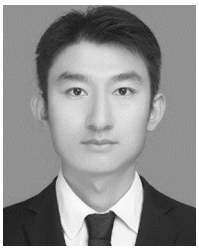
[44] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, Jan. 1993, doi: 10.1016/0377-2217(93)90182-M.

[45] J.-Y. Ding, S. Song, J. N. D. Gupta, R. Zhang, R. Chiong, and C. Wu, "An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem," *Appl. Soft Comput.*, vol. 30, pp. 604–613, May 2015, doi: 10.1016/J.ASOC.2015.02.006.

[46] J.-J. Ji, Y.-N. Guo, X.-Z. Gao, D.-W. Gong, and Y.-P. Wang, "Q-learning-based hyperheuristic evolutionary algorithm for dynamic task allocation of crowdsensing," *IEEE Trans. Cybern.*, early access, Oct. 4, 2021, doi: 10.1109/TCYB.2021.3112675.

Fuqing Zhao received the B.Sc. and Ph.D. degrees from the Lanzhou University of Technology, Lanzhou, China, in 1994 and 2006, respectively.

Since 1998, he has been with the School of Computer Science Department, Lanzhou University of Technology, where he became a Full Professor in 2012. He has been the Postdoctoral Fellow with the State Key Laboratory of Manufacturing System Engineering, Xi'an Jiaotong University, Xi'an, China, in 2009. He has been a Visiting Scholar with the Exeter Manufacturing Enterprise Center, Exeter University, Exeter, U.K., and Georgia Tech Manufacturing Institute, Georgia Institute of Technology, Atlanta, GA, USA, from 2008 to 2019 and from 2014 to 2015, respectively. He has authored two academic book and over 50 refereed papers. His current research interests include intelligent optimization and scheduling.



Shilu Di received the B.S. and M.S. degrees from the School of Computer and Communication Technology, Lanzhou University of Technology, Lanzhou, China, in 2018 and 2022, respectively.

His current research interests include intelligent optimization and scheduling algorithms.

Ling Wang received the B.Sc. degree in automation and the Ph.D. degree in control theory and control engineering from Tsinghua University, Beijing, China, in 1995 and 1999, respectively.

Since 1999, he has been with the Department of Automation, Tsinghua University, where he became a Full Professor in 2008. He has authored five academic books and more than 300 refereed papers. His current research interests include intelligent optimization and production scheduling.

Prof. Wang is a recipient of the National Natural Science Fund for Distinguished Young Scholars of China, the National Natural Science Award (Second Place) in 2014, the Science and Technology Award of Beijing City in 2008, and the Natural Science Award (First Place in 2003, and Second Place in 2007) nominated by the Ministry of Education of China. He is currently the Editor-in-Chief of *International Journal of Automation and Control*, and an Associate Editor of IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and SWARM AND EVOLUTIONARY COMPUTATION.