Computers & Industrial Engineering 200 (2025) 110855

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

HGNP: A PCA-based heterogeneous graph neural network for a family distributed flexible job shop

Jiake Li [a], Junqing Li [a,b,*], Ying Xu [b]

[a] Department of Mathematics, Yunnan Normal University, Kunming, Yunnan, 650500, China
[b] School of Information Engineering, HengXing University, Qingdao, 266199, China

ARTICLE INFO

Keywords:
Distributed flexible job shop
Heterogeneous graph neural network
Principal component analysis
Family setup time
Enhanced local search

ABSTRACT

The distributed flexible job shop scheduling problem (DFJSP) has gained increasing attention in recent years. Meanwhile, the family setup time constraint exists in many realistic manufacturing systems, e.g., prefabricated components system. In this study, first, a mixed integer programming (MIP) model is formulated for the DFJSP with family setup time. To minimize the makespan, a hybrid heterogeneous graph neural network with a principal component analysis (PCA)-based transform mechanism (HGNP) is proposed. In the proposed algorithm, a novel state representation is designed, which combines the features of operation, machine and factory assignment. Then, a multilayer perceptron (MLP) mechanism is used for the operation embedding, and graph attention networks (GATs) are embedded for the machine and factory embeddings. Next, a PCA-based transform mechanism is developed to further fuse all the three embeddings. To improve the solution performance, a simple enhanced local search method is developed. Three different scale of instances are generated to test the performance of HGNP, including small instances to test the effectiveness of the mathematical model, medium and large instances to test the efficiency, and extended public instances to test the generalization abilities. Experimental results and comparisons with different types of state-of-the-art algorithms show the competitiveness and efficiency of the proposed algorithm, both in performance and generalization capabilities.

1. Introduction

Flexible job shop (FJSP) is a classical scheduling problem (Cao, Lin, & Zhou, 2021; Chen, Li, Wang, Li, & Gao, 2025; Fan, Zhang, Liu, Shen, & Gao, 2022), that has been applied in many realistic industries, such as semiconductor manufacturing systems (Jamrus, Chien, Gen, & Sethanan, 2018), and steelmaking systems (Li, Du, et al., 2022). In the classical FJSP, there are n jobs to be processed on m machines, where each job has a certain number of operations. A suitable machine is selected from a set of candidate machines for each operation, which increases the machine utilities and therefore decreases the production costs. The classical FJSP has been verified to be an NP-hard problem, which is competitive with other typical scheduling problems, such as the job shop scheduling problem (JSP), flow shop scheduling problem (FSP), and hybrid flow shop (HFS).

In recent years, the FJSP has been the subject of an increasing number of studies. Cao et al. (2021) developed a knowledge-based cuckoo search algorithm to solve FJSPs with sequencing flexibility. Fan et al. (2022) solved this problem considering reconfigurable machine tools with limited auxiliary modules by using a genetic algorithm (GA). Jamrus et al. (2018) considered a typical FJSP in a semiconductor manufacturing system by using hybrid particle swarm optimization (PSO), in which uncertain processing time is considered. Li, Du, et al. (2022) applied FJSPs in a steelmaking casting system and solved it by using a hybrid iterated greedy (IG) algorithm. Tutumlu and Saraç (2023) formulated FJSPs with a mixed integral programming (MIP) model and solved it by using a GA method.

To apply the FJSP in realistic manufacturing systems, an increasing number of realistic constraints should be considered. The setup time is one of the most commonly used realistic constraints (Shen, Dauzère-Pérès, & Neufeld, 2018; Zhang & Wang, 2018), which is used to calculate the setup time between consecutive operations on the same machine during preparatory processes, such as resource switching, and machine cleaning. Especially, in realistic manufacturing system, such as prefabricated component (PC) system, an additional mold switching setup time should be added for jobs in different families. Meanwhile, with the development of global economics, distributed FJSP (DFJSP) has become popular in modern industries (Wang & Wang, 2020).

* Corresponding author at: Department of Mathematics, Yunnan Normal University, Kunming, Yunnan, 650500, China.
E-mail address: lijunqing@ynnu.edu.cn (J. Li).

https://doi.org/10.1016/j.cie.2024.110855
Received 31 May 2024; Received in revised form 21 October 2024; Accepted 31 December 2024
Available online 8 January 2025

Meta-heuristics have been utilized to solve the FJSP with different constraints, such as the GA (Fan et al., 2022), PSO (Jamrus et al., 2018), and IG (Li, Du, et al., 2022). However, the greatest disadvantages of applying meta-heuristics to solve scheduling problems are: (1) knowledge is difficult to be utilized during the evolution stages; and (2) knowledge transfer is difficult to be implemented, and therefore, for different instances, the algorithm should test the instance from scratch. Meanwhile, deep learning (Li, Li, J., Yu, & Xu, 2025; Li, Wang, Wang, Han, & Wang, 2021; Song, Li, Du, Yu, & Xu, 2025) and neural network methods (Müller, Müller, Kress, & Pesch, 2022; Tremblet, Thevenin, & Dolgui, 2023) have also been applied to solve different types of optimization problems. The main advantage of these methods is that it is easy to utilize knowledge. However, their disadvantage is also obvious; it is difficult to reach optimal results, and the training process has high time complexity.

To address this issue, in this study, we propose a hybrid algorithm combining the deep reinforcement learning (DRL) method and meta-heuristics to solve FJSPs with family setup time constraints. The main contributions are as follows.

(1) To the best of our knowledge, this is the first study to solve the DFJSP with family setup time constraints (DFJSP-FS). The mathematical model is formulated by using mixed integer programming (MIP) and solved by using CPLEX.
(2) A special type of DRL method is proposed to solve the DFJSP-FS problems, where a novel state representation method is developed.
(3) A hybrid heterogeneous graph neural network-based architecture is embedded, where the multilayer perceptron (MLP) mechanism is used for the operation embedding, and graph attention networks (GATs) are embedded for the machine and factory embeddings.
(4) A principal component analysis (PCA)-based transform mechanism is developed to further fuse all three embeddings, which can significantly improve the learning abilities of the policy network.

The structure of the remainder of this study is as follows. Section 2 analysis the works related to the considered problem and algorithms. Then, the problem formulation is described in Section 3. Next, the framework of the proposed HGNP algorithm is illustrated in Section 4. Section 5 describes the comparisons and analysis after detailed experiments. Finally, the last section concludes the work and provides the future research focuses.

2. literature review

2.1. FJSP with realistic constraints

Many realistic constraints have been investigated in FJSPs. The setup time constraint is one of the typical realistic constraints, which considers the sequence-dependent (SDST) or sequence independent setup time out of the processing time. Zhang and Wang (2018) considered FJSPs with SDSTs and formulated them by a dynamic MIP model. Shen et al. (2018) addressed the SDST FJSP using a tabu search (TS) algorithm. Li and Lei (2021) developed an imperialist competitive algorithm (ICA) to solve the FJSP with transportation and SDST constraints. Sun, Zhang, Lu, and Zhang (2021) solved the FJSP with setup and transportation times to minimize multiple objectives, including the makespan, total workload, workload of the critical machine, and penalties of earliness/tardiness.

Except for the setup time constraint, other realistic constraints have also obtained an increasing number of focuses. An, Chen, Gao, Li, and Zhang (2023) investigated new job insertion and machine preventive maintenance in FJSPs and solved it by a non-dominated sorting genetic algorithm III (NSGA-III). Gao, Yang, Zhou, Pan, and Suganthan (2019) considered FJSP rescheduling with new job insertion by using discrete Jaya algorithm. Xu, Mei, Zhang, and Zhang (2023) and Zhang, Mei, Nguyen, Tan, and Zhang (2022) studied dynamic FJSPs by using genetic programming (GP), and multitask GP methods, respectively. Other constraints including finite transportation resources (Pan, Wang, Zheng, Chen, & Wang, 2022), mandatory outsourcing constraints (Su, Huang, Li, Li, & Hao, 2023), and fuzzy processing time (Sun, Lin, Gen, & Li, 2019), have been investigated in FJSPs.

2.2. FJSP with energy and distribution

With the development of global environments, distribution and low-carbon production modes have become increasingly popular.

Considering minimization of the energy objective in FJSPs, Gong et al. (2022) developed a two-stage memetic algorithm (MA). Li, Han, Gao, Xiao, and Duan (2023) utilized a bi-population balancing multi-objective algorithm to solve FJSPs with fuzzy processing time to minimize energy and makespan. Li, Gong, Lu, and Wang (2023) proposed a learning-based MA for solving energy-efficient FJSPs with Type-2 fuzzy processing time. Park and Ham (2022) considered time-of-use (TOU) pricing in solving FJSPs. Lv, Li, Tang, and Kou (2022) minimized energy objectives in solving FJSPs with dynamic events and alternative process plans.

The distributed production mode is another method to increase production capabilities and decrease energy consumption. Shao, Shao, and Pi (2023) developed a learning-based selection hyper-heuristic to solve distributed HFS with blocking constraints. Zhao, Di, and Wang (2023) solved distributed blocking FSP with a Q-learning driven multi-objective algorithm. Li, Gong, Wang, Lu, and Zhuang (2023) considered DFJSP and solved it by using an adaptive MA. Lin, Li, Wei, and Wu (2020) joined process planning and DFJSP in an integrated mode and solved it by using a GA. De Giovanni and Pezzella (2010) solved DFJSP with GA methods.

2.3. Deep learning methods

In recent years, deep learning methods have received increasing attention for solving scheduling problems. Brammer, Lutz, and Neumann (2022) solved the permutation flow shop problem (PFSP) with multiple lines and demand plans by using a reinforcement learning (RL) method. Li, Gao, Duan, Li, and Zhang (2023) developed a hybrid algorithm combining artificial bee colony (ABC) and Q-learning to solve PFSPs. Pan, Wang, Dong, and Chen (2023) addressed FSPs with a knowledge-guided RL method. Pan, Wang, Wang, and Lu (2023) developed a DRL method to solve PFSPs.

Recently, deep learning methods have also been applied to solve FJSPs. These methods can be categorized into two classes.

The first class is the classical RL method. Chen, Yang, Li, and Wang (2020) solved FJSPs with a self-learning GA driven by RL methods. Du, Li, Li, and Duan (2022) proposed a hybrid algorithm combined with RL and a local search heuristic to solve FJSPs with crane transportation and setup time constraints. Du, Li, Chen, Duan, and Pan (2023) developed a hybrid algorithm with RL and an estimation of distribution algorithm (EDA) to solve FJSPs. Li, Gong, and Lu (2022) developed a hybrid algorithm with RL and multi-objective evolutionary algorithm based on decomposition (MOEA/D) to solve fuzzy FJSPs.

The second class is the DRL method. Gui, Tang, Zhu, Zhang, and Zhang (2023) solved dynamic FJSPs with a DRL method. Lei et al. (2022) created a multi-action DRL framework for FJSPs. Liu, Piplani, and Toro (2022) solved dynamic FJSP with a DRL method. Li, Gu, Yuan, and Tang (2022) utilized a hybrid deep Q-network (DQN) to solve FJSP with real-time data and dynamic resources. Luo (2020) and Luo, Zhang, and Fan (2022) developed DRL methods to solve the dynamic FJSPs with new job insertions and partial-no-wait constraints, respectively. Song, Chen, Li, and Cao (2023) solved FJSPs via graph neural network (GNN) and DRL methods. Zhang, He, Chan, and Chow (2023) developed a hybrid algorithm with DRL and multi-agent graphs to solve FJSPs. Zhao, Fan, Zhang, Shen, and Zhuang (2023) considered

Table 1
Features of the operations.

Constraints	Objectives	Methods	Reference
FJSP, SDST, assemble	Makespan	Constraint programming, dispatching rules	Zhang and Wang (2018)
FJSP, SDST	Makespan	Tabu search	Shen et al. (2018)
FJSP, SDST, transportation	Makespan, total tardiness, energy	ICA with feedback	Li and Lei (2021)
FJSP, SDST, transportation	Makespan, workload, earliness/tardiness	Hybrid many-objective evolutionary algorithm	Sun et al. (2021)
FJSP, new job arrival, maintenance	Makespan, workload, stability, production cost	NSGA-III, local search	An et al. (2023)
FJSP, new job arrival	Makespan, instability, workload	Jaya	Gao et al. (2019)
FJSP, limited transportation	Makespan	Multi-population evolutionary algorithm based on learning	Pan et al. (2022)
FJSP, outsourcing, maintenance	Tardiness	Self-organizing neural scheduler	Su et al. (2023)
FJSP, fuzzy	Fuzzy Makespan	Cooperative coevolution algorithm	Sun et al. (2019)
FJSP, energy	Makespan, energy	Two-stage MA	Gong et al. (2022)
FJSP, fuzzy, transportation	Makespan, energy	Bi-population balancing multi-objective algorithm	Li, Han, et al. (2023)
FJSP, type-2 fuzzy	Makespan, energy	Learning-based MA	Li, Gong, et al. (2023)
FJSP, TOU	Makespan, energy	Constraint programming	Park and Ham (2022)
FJSP, new job arrivals, machine breakdowns	Makespan, energy	Heuristic	Lv et al. (2022)
DFJSP	Makespan, energy	Popular-based adaptive MA	Li, Gong, Wang, et al. (2023)
DFJSP	Makespan	GA	Lin et al. (2020)
DFJSP	Makespan	GA	De Giovanni and Pezzella (2010)
FJSP	Makespan	GA, Q-learning	Chen et al. (2020)
FJSP	Makespan, energy	EDA, deep Q-network	Du et al. (2022)
FJSP, fuzzy	Makespan, workload	MOEA/D, RL	Li, Gong, and Lu (2022)
FJSP, dynamic	Mean tardiness	Deep Q-network	Gui et al. (2023)
FJSP	Makespan	Multi-pointer graph networks	Lei et al. (2022)
FJSP, dynamic	Makespan	Deep Q-network	Liu et al. (2022)
FJSP, limited transportation	Makespan, energy	Deep Q-network	Li, Gu, et al. (2022)
FJSP, new job arrival	Total tardiness	Deep Q-network	Luo (2020)
FJSP	Makespan	GNN	Song et al. (2023)
FJSP	Makespan	Multi-agent graph based DRL	Zhang et al. (2023)
FJSP, new job arrival	Total tardiness	DRL-based reactive policy	Zhao, Fan, et al. (2023)
DFJSP, family setup time,	Makespan	PCA-based HGNN	ours

FJSPs with random job arrivals and solved them by using a DRL-based reactive policy.

In a nutshell, meta-heuristics and deep learning methods have different capabilities for solving scheduling problems. It is a challenging issue to combine the advantages of meta-heuristics and deep learning methods to solve DFJSP with realistic constraints.

2.4. literature analysis

Table 1 shows the related studies with the considered problems and methods in this study, which can be observed that: (1) there is an increasing number of works considering the FJSPs with many types of realistic constraints; however, less literature considered DFJSP with family setup time constraint, which can be found in many types of manufacturing systems; and (2) for the generalization capabilities, RL or DRL methods have gained focus in recent years for solving FJSPs; however, applying RL or DRL method to solve DFJSP is still a challenging issue.

In addition, many types of RL or DRL methods have been developed to solve different types of scheduling problems. Among these methods, graph neural network has been verified to be efficient to capture neighboring features, and therefore, the network training efficiency can be enhanced.

Based on the literature analysis, to address the DFJSP with family setup time constraint, we developed a PCA-based hybrid heterogeneous graph neural network method.

3. Problem formulation

A DFJSP is a typical scheduling problem, that has the following features. First, there are n jobs to be processed on m machines in h factories. Each job has a certain number of operations, and the processing routine of each job is independent of each other. There are parallel machines for each operation, and therefore, one task is to select a suitable machine for each operation. There are parallel factories that should be selected for each job as well. After assigning operations on machines in factories, we should decide the processing sequences.

3.1. Assumptions

In this study, the following assumptions are given.

(1) Each operation should be assigned to exactly one machine at a time; each machine should process exactly one operation at a time.
(1) All operations of the same job should be assigned to the same factory.
(2) All the parallel factories have the same machine layout and capabilities.
(3) Processing overlap is not permitted.
(4) Disruptions, e.g., machine breakdown and order insertion/cancellation are ignored.
(5) Setup time is considered between jobs in different groups.
(6) Resources are infinite, and there are enough buffers between machines.
(7) Transportation time is ignored, or included in the processing time.
(8) The processing time of each operation is known as a deterministic value.

3.2. Indices and parameters

- n: Total number of jobs.
- m: Total number of machines.
- h: Total number of factories.
- g: Total number of groups or families.

- θ_j: Total number of operations of job j.
- θ: Total number of operations, that is, $\theta = \sum_j \theta_j$.
- i, i': Index of jobs.
- k : Index of machines.
- j, j' : Index of operations.
- f: Index of factories.
- r: Index of processing positions.
- $O_{i,j}$: The jth operation of job i.
- $S_{i,j,f}, C_{i,j,f}$: Starting and completion time of $O_{i,j}$ in factory f.
- $S_{k,r,f}, C_{k,r,f}$: Starting and completion time related to the rth position on machine k in factory f.
- $P_{i,j,k}$: Processing time of the $O_{i,j}$ on machine k.
- $M_{i,j}$: The available machines for $O_{i,j}$.
- $ST_{i,i'}$: Setup time between jobs i and i'.
- G_i: Group of job i.
- U: The set of all the jobs.
- L: A very large number.

3.3. Decision variables

- $Y_{i,j,k,r,f}$: a binary value that is set to 1 if $O_{i,j}$ is assigned to the rth position on machine k in factory f; otherwise, it is set to 0.
- $Z_{i,j,f}$: a binary value that is set to 1 if $O_{i,j}$ is assigned to factory f; otherwise, it is set to 0.

3.4. Mathematical model

(1) Objective of the considered problem

The objective of this study is to minimize the maximum completion time of all operations. Eq. (2) lists the calculation of the objective.

$$\min \quad C_{\max} \tag{1}$$

$$C_{\max} = \max\{C_{i,j,f}\}, \forall j, q, f \tag{2}$$

(2) The constraint of the starting and completion times

Constraints (3) and (4) ensure the starting times should be positive values.

$$S_{k,r,f} \geq 0, \forall k, r, f \tag{3}$$

$$S_{i,j,f} \geq 0, \forall j, q \in \theta_j, f \tag{4}$$

Constraints (5)–(6) guarantee the overlap between two consecutive operations should not be permitted.

$$C_{i,j,f} \leq S_{i,j+1,f}, \forall i, j < \theta_i, f \tag{5}$$

$$C_{k,r,f} \leq S_{k,r+1,f}, \forall i, f, r \in \{2, \ldots, \theta\} \tag{6}$$

Constraint (7) ensures that the overlap should not be permitted for two consecutive operations on the same machine.

$$S_{k,r+1,f} \geq C_{k,r,f} + L \times (2 - Y_{i,j,k,r,f} - Y_{i',j',k,r+1,f}),$$
$$\forall i, j, k \in M_{i,j} \cap M_{i',j'}, f, r < \theta \tag{7}$$

In Eq. (8), the completion time $C_{k,r,f}$ is calculated, which equals the starting time plus the processing time of the corresponding operation.

$$C_{k,r,f} = S_{k,r,f} + \sum_{i=1}^{n} \sum_{j=1}^{\theta_i} P_{i,j,k} \times Y_{i,j,k,r,f}, \forall f, r \tag{8}$$

(3) Selection constraints

Constraint (9) restricts each operation to be processed at exactly one position on one machine in one factory. Constraint (10) ensures that only one factory can be selected for each operation. Constraint (11) ensures that all the operations belonging to the same job should select the same factory.

$$\sum_{k \in M_{i,j}} \sum_{f} \sum_{r} Y_{i,j,k,r,f} = 1, \forall i, j \tag{9}$$

$$\sum_{f} Z_{i,j,f} = 1, \forall i, j \tag{10}$$

$$Z_{i,j,f} = Z_{i,j+1,f}, \forall i, j < \theta_i, f \tag{11}$$

(4) Processing position-related constraints

The processing positions on each machine are designed for the setup time constraints. Constraint (12) ensures that each position on each machine should process no more than one operation at a time. Constraint (13) ensures the assignment of the machine processing positions should not be skipped, that is, blank positions are not permitted between two processing positions. Constraints (14) and (15) set the relations of the starting and completion times between the processing position and the corresponding operations processed on it. Constraint (16) ensures that if the operation is assigned to one factory, exactly one position should be selected.

$$\sum_{i} \sum_{j} Y_{i,j,k,r,f} \leq 1, \forall f, r \tag{12}$$

$$\sum_{i} \sum_{j} Y_{i,j,k,r,f} \geq \sum_{i'} \sum_{j'} Y_{i',j',k,r+1,f},$$
$$\forall k \in M_{i,j} \cap M_{i',j'}, f, \ r < \theta \tag{13}$$

$$S_{k,r,f} = S_{i,j,f} \times \sum_{k \in M_{i,j}} \sum_{r} Y_{i,j,k,r,f}, \forall f \tag{14}$$

$$C_{k,r,f} = C_{i,j,f} \times \sum_{k \in M_{i,j}} \sum_{r} Y_{i,j,k,r,f}, \forall f \tag{15}$$

$$Z_{i,j,f} = \sum_{k \in M_{i,j}} \sum_{r} Y_{i,j,k,r,f}, \forall f \tag{16}$$

(5) Family setup time constraints

The family/group setup time constraints illustrated in Constraints (17) and (18) ensure that the setup time between two jobs should be considered, if the following conditions are satisfied: (a) the two jobs are processed on consecutive positions of the same machine; and (b) the two jobs are not in the same family or group.

$$\sum_{i \neq i'} \sum_{j} ST_{i,j,i',j'} \times Y_{i,j,k,r,f} \leq S_{k,r+1,f} - C_{k,r,f} + L \times$$
$$(1 - Y_{i',j',k,r+1,f}), \forall i', k \in M_{i',j'}, j', f, r, G_i \neq G_i' \tag{17}$$

$$\sum_{i \neq i'} \sum_{j} ST_{i,j,i',j'} \times Y_{i,j,k,r,f} + L \times (1 - Y_{i',j',k,r+1,f}) \geq$$
$$S_{k,r+1,f} - C_{k,r,f}, \forall i', k \in M_{i',j'}, j', f, r, G_i \neq G_i' \tag{18}$$

4. Methodology

This section lists the proposed HGNP, where the framework is introduced first. Then, the components including state representation, actions, rewards, policy learning approaches, embedding mechanism, PCA-based transform method, policy network, and enhanced local search procedure are presented, respectively.

4.1. Framework of the proposed algorithm

Algorithm 1 illustrates the main framework of the proposed HGNP algorithm. Fig. 1 shows the main structure of HGNP.

4.2. State representation

(1) Feature representation

To fuse more features considering machines, operations, and factories, we develop a novel feature representation method, which embeds the state s_t with the features of the machines, operations, and factories at step t. The features of the operations contain six parts, i.e., (1) τ_{op}^1: the status of the current operation, which tells whether the current operation has been scheduled; (2) τ_{op}^2: number of neighboring machines

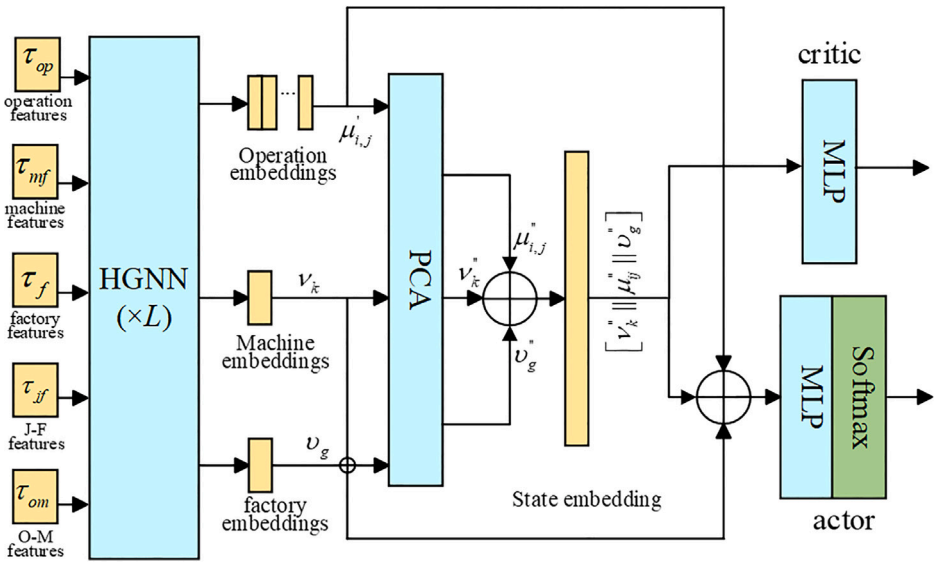

Fig. 1. Structure of the proposed HGNP.

$\left|N_t(O_{i,j})\right|$, which gives the number of candidate machines for the current operation; (3) τ^3_{op}: the processing time $\widetilde{P_{i,j,k}}$, which equals $P_{i,j,k}$ if $O_{i,j}$ is scheduled on machine k; otherwise, it is set to the average processing time $\overline{P_{i,j}}$ on all the candidate machines, wherein $\overline{P_{i,j}} = \sum_{k\in M_{i,j}} P_{i,j,k} \Big/ |M_{i,j}|$; (4) τ^4_{op}: starting time $\widetilde{S_{i,j}}$, which equals the actual starting time of $O_{i,j}$ if it is scheduled; otherwise, it is set to the expected starting time. The detailed steps to calculate the expected $\widetilde{S_{i,j}}$ are as follows: if $O_{i,j-1}$ has been scheduled, then $\widetilde{S_{i,j}} = \widetilde{S_{i,j-1}} + P_{i,j-1,k} + \overline{ST_{i,i'}}$, wherein $\overline{ST_{i,i'}} = \sum_{i'\notin G_i} ST_{i,i'} \Big/ \left|U \backslash G_i\right|$ is the average setup time between job i and all the other jobs; otherwise, $\widetilde{S_{i,j}} = \widetilde{S_{i,j-1}} + \overline{P_{i,j-1}} + \overline{ST_{i,i'}}$; (5) τ^5_{op}: number of unscheduled operations in the same job; (6) τ^6_{op}: completion time, which equals to the addition of the starting time, processing time, and setup time.

The features of the machines contain four parts: (1) τ^1_{mf}: number of neighboring operations, which indicates the number of operations that can be processed by the current machine. (2) τ^2_{mf}: machine available time. (3) τ^3_{mf}: machine utilization, which is used to calculate the ratio between the total processing times of the current machine, and the current time t. (4) τ^4_{mf}: current processing operation, which is used to calculate the setup time between two operations processed on the same machine.

The features of the factories contain two parts: (1) τ^1_f: number of assigned jobs at the current factory. (2) τ^2_f: factory utilization, which is used to calculate the ratio between the total processing time of the current factory and the current time t.

The features of the O-M arcs tell the relations between operations and machines, which contain only one part, i.e., τ^1_{om}, a binary value set to "1" if the operation is assigned to the corresponding machine. The features of the J-F arcs tell the relations between jobs and factories, which contain only one part, i.e., τ^1_{jf}, a binary value set to "1" if the job is assigned to the corresponding factory.

Altogether, the state representation s_t is given by

$$s_t = \{\tau^1_{op}, \ldots, \tau^6_{op}; \tau^1_{mf}, \ldots, \tau^4_{mf}; \tau^1_f, \tau^2_f; \tau^1_{om}; \tau^1_{jf}\} \tag{19}$$

(2) Representation Examples

We provide an example in Fig. 2 to illustrate the state representation. Suppose there are two factories, six machines, and six jobs in the production system. The first five jobs have three operations, while the last job has four operations. At time step 5, the state s_1 is combining when the operation $O_{1,1}$ has completed. The features of the first operation $O_{1,1}$ are set to $< 1, 2, 5, 0, 2, 5 >$, in which "1" indicates that the operation $O_{1,1}$ has been scheduled; "2" means $O_{1,1}$ has two candidate machines; "5" shows the processing time of $O_{1,1}$; "0" is the starting time of $O_{1,1}$; "2" means there are still two unscheduled operations in J_1; and "5" shows the completion time of $O_{1,1}$. Based on the analysis, the features of all 19 operations are given as follows.

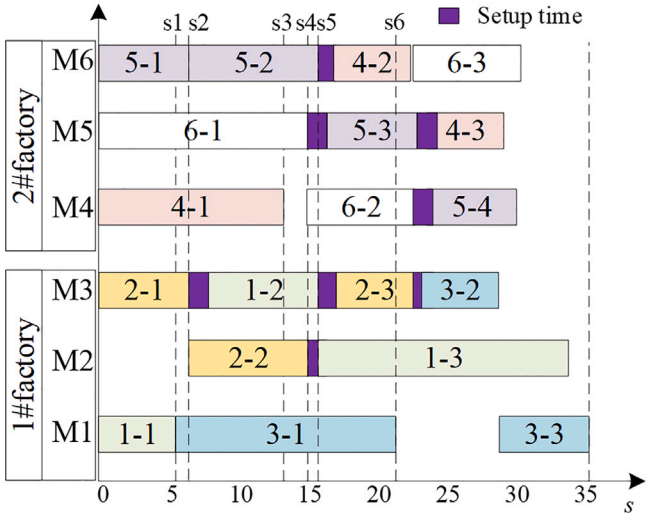

Fig. 2. Illustration of a complete solution with states.

At time 5, five operations are scheduled, i.e., $O_{1,1}$, $O_{2,1}$, $O_{4,1}$, $O_{5,1}$, and $O_{6,1}$. The following steps are used to calculate the estimated starting and completion times of the unscheduled operations.

Step 1. Calculate the mean processing time $\widetilde{P_{i,j}}$ for each operation $O_{i,j}$, i.e.,

$$\widetilde{P_{i,j}} = \left(\sum_{k\in N_t(O_{i,j})} P_{i,j,k}\right) \Big/ \left|N_t(O_{i,j})\right|. \tag{20}$$

Note that if an operation has been scheduled, $\widetilde{P_{i,j}}$ equals its real starting time.

Step 2. Let matrix $R_{\eta\times\eta}$ represent the subsequent association relationship between the operations of the same job, in which each operation occupies one line, and the positions marked with "1" mean the operations belonging to the same job with the current operation.

Algorithm 1: Framework of HGNP

Input: the training instances, the parameters
Output: trained models
for $iter = 1 \rightarrow \ell_t$ **do**
  for $b = 1 \rightarrow B_s$ **do**
    Let $\phi_J = \{\}$ and $\psi_J = \{1, 2, ..., n\}$ be the scheduled and unscheduled job set.
    Let time step $t = 1$.
    for $len(\psi_J) > 0$ **do**
      Update the five raw features, i.e., $\tau_{op}, \tau_{mf}, \tau_f, \tau_{jf}$, and τ_{om}, to construct the current state s_t (c.f. Section 4.2).
      Input the five raw features into the L-layer of HGNN to obtain the three embeddings, i.e., v_k, $\mu'_{i,j}$, and v_g, for machine, operation, and factory, respectively (c.f. Section 4.6).
      Apply the PCA-based transform method to v_k, $\mu'_{i,j}$, and v_g to obtain the resulting embeddings, v''_k, $\mu''_{i,j}$, and v''_g (c.f. Section 4.7).
      Apply a MLP as $\text{MLP}_\omega\left[v''_k||\mu''_{ij}||v''_g||v_k||\mu'_{ij}||v_g\right]$ to the actor in the policy network (c.f. Section 4.8).
      Apply a MLP as $\text{MLP}_\phi\left[v''_k||\mu''_{ij}||v''_g\right]$ to the critic in the policy network (c.f. Section 4.8).
      Obtain the selected operation, machine, and factory according to $\pi_\omega(a_t|s_t)$.
      Perform the PPO-based policy learning method (c.f. Section 4.5).
      Update ϕ_J and ψ_J.
    end
  end
  Compute the loss values and optimize the parameters.
  Update the network parameters.
  if $iter\%10 = 0$ **then**
    Perform validation.
  end
  if $iter\%20 = 0$ **then**
    Sample a new batch of B_s instances.
  end
end

Step 3. Let vector $Z_{i,j}$ represent whether the operation is an unscheduled operation, in which "1" represents an unscheduled operation.

Step 4. Calculate the estimated starting time for all the unscheduled operations by $E_{i,j} = \{\widetilde{P_{i,j}} \times R_{\eta\times\eta} \times Z_{i,j} | Z_{i,j} = 1\}$.

Table 2 shows the features of the operations. For example, the mean processing times for all 19 operations are {5, 6, 16 | 6, 8, 4 | 8, 9, 11 | 13, 10, 9 | 6, 7, 4 | 14, 9, 10, 9}, in which the number marked with underlined indicates the real starting time. The first line of $R_{\eta\times\eta}$ is {0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0}, which tells the following two operations at positions 2 and 3 belong to the same job as the current operation. The resulting estimated starting time for all operations is {0, 5, 11 | 0, 6, 14 | 0, 8, 17 | 0, 13, 23 | 0, 6, 13 | 0, 14, 23, 33}. The features of factories are {3, 2; 1.0, 0.67}, and the features of machines are given in Table 3.

4.3. Actions

Actions are used to interact with the environment under certain states. At time step t, a suitable action $a_t \in A$ is selected, in which a batch of operations, factories, and machines is fetched. Note that the selected operation becomes the scheduled operation and should be unmarked in the following steps. Therefore, A is a set of actions defined as follows: $A = \{\langle op_1, ma_1, fa_1\rangle, \ldots, \langle op_n, ma_n, fa_n\rangle\}$, wherein op_i, ma_i and fa_i $(i = 1, 2, \ldots, n)$ represent the assigned operation, machine, and factory at the ith step, respectively.

Table 2
Features of the operations.

Operations	No	Features					
		τ^1_{op}	τ^2_{op}	τ^3_{op}	τ^4_{op}	τ^5_{op}	τ^6_{op}
$O_{1,1}$	1	1	2	5	0	2	5
$O_{1,2}$	2	0	2	6	5	2	11
$O_{1,3}$	3	0	2	16	11	2	27
$O_{2,1}$	4	1	3	6	0	2	6
$O_{2,2}$	5	0	2	8	6	2	14
$O_{2,3}$	6	0	2	4	14	2	18
$O_{3,1}$	7	0	3	8	0	3	8
$O_{3,2}$	8	0	2	9	8	3	17
$O_{3,3}$	9	0	2	11	17	3	28
$O_{4,1}$	10	1	3	13	0	2	13
$O_{4,2}$	11	0	2	10	13	2	23
$O_{4,3}$	12	0	3	9	23	2	32
$O_{5,1}$	13	1	2	6	0	2	6
$O_{5,2}$	14	0	3	7	6	2	13
$O_{5,3}$	15	0	2	4	13	2	17
$O_{6,1}$	16	1	1	14	0	3	14
$O_{6,2}$	17	0	2	9	14	3	23
$O_{6,3}$	18	0	3	10	23	3	33
$O_{6,4}$	19	0	2	9	33	3	42

Table 3
Features of the machines.

Machines	Features			
	τ^1_{mf}	τ^2_{mf}	τ^3_{mf}	τ^4_{mf}
M_1	15	5	1.0	1
M_2	14	0	0.0	0
M_3	16	6	1.0	4
M_4	15	13	1.0	10
M_5	13	14	1.0	16
M_6	12	6	1.0	13

4.4. Rewards

The difference of the makespan between s_t and s_{t+1} is chosen as the reward, i.e., $r(s_t, a_t, s_{t+1}) = C_{\max}(s_t) - C_{\max}(s_{t+1})$. The calculation of the reward values comprises two stages:

(1) Calculation of the rewards $r_\alpha(\alpha = 1, 2, \ldots, N)$ during the partial scheduling generation. From the illustration in Fig. 2, it is clear that, the action at the ith step is used to select suitable operation, machine, and factory at that time point, and therefore, the next chosen operation can be assigned and scheduled. At that time point, $C_{\max}(s_{t+1})$ is calculated by considering the two completion times, i.e., the estimation completion time of the unscheduled operation, and the real completion time of the scheduled operation. Therefore, the reward can be a negative value if $C_{\max}(s_{t+1}) > C_{\max}(s_t)$ with the scheduling of the chosen operation op_t; the reward can also be zero if the estimated makespan of the partial schedule unchanged with the addition of op_t. We accept the selected action under negative or zero reward value until all operations have been scheduled.

(2) Calculation of the cumulative rewards during the policy learning procedure. The rewards $r_\alpha(\alpha = 1, 2, \ldots, N)$ are used to calculate the discounted cumulative rewards of s_t as R_t in Eq. (21). If a zero reward is obtained, the selected action is still accepted to avoid overfitting during the training procedure. Note that although the current reward value is zero, the cumulative reward calculated by formulation (21) can still evaluate the effectiveness of the chosen action, and therefore, the policy network can also be optimized.

4.5. Policy learning

Proximal policy optimization (PPO) is used as the policy learning method, in which the policy $\pi_\theta(a_t|s_t)$ is learned by using a neural network with s_t as the input, and the probability distribution over the actions a_t as the output. The learning parameters are denoted as θ, which is updated by maximizing $L(\theta)$ as follows.

Step 1. Generate N trajectories $(s_1,a_1,r_1),\ldots,(s_N,a_N,r_N)$ by using the current policy π_θ. Update $\pi_{\theta_{old}}=\pi_\theta$.

Step 2. For each trajectory t, calculate the advantage estimate values as follows.

$$R_t=\sum_{\alpha=t}^{N}\gamma^{\alpha-t}r_\alpha, \tag{21}$$

$$\widehat{A}_t=R_t-V^\theta(s_t) \tag{22}$$

where, $V^\theta(s_t)$ is the resulting value calculated by the Critic component, and γ is the discount parameter (set to 0.99 according to Song et al. (2023)).

Step 3. Update θ by maximizing $L(\theta)$ based on $(s_1,a_1,r_1,\widehat{A}_1),\ldots,(s_N,a_N,r_N,\widehat{A}_N)$ as follows.

$$\begin{aligned}&\max_\theta L(\theta)=\sum_{t=1}^{N}\min\left\{Y\widehat{A}_t, clip(Y,1-\varepsilon,1+\varepsilon)\widehat{A}_t\right\}-\\&c_1(R_t-V^\theta(s_t))^2-c_2\sum_{a_t\in A}\pi_\theta(a_t|s_t)\log_2\pi_\theta(a_t|s_t)\end{aligned} \tag{23}$$

where, $Y=\min\{\frac{\theta_1}{\theta_2},e^{\theta_1-\theta_2}\}$, $\theta_1=\pi_\theta(a_t|s_t)$, and $\theta_2=\pi_{\theta_{old}}(a_t|s_t)$.

The loss function $L(\theta)$ contains three parts. (1) The first part ensures that the higher $\widehat{A}_t$ will be assigned with higher probabilities. Note that we also use a clip function to make the ratio between the new and old probability in the range $[1-\varepsilon,1+\varepsilon]$, in which ε is set to 0.2. (2) The second part ensures the improvement of $V^\theta(s_t)$. (3) The third part ensures the randomness of the policy learning method, which can balance the exploitation and exploration abilities. The two parameters c_1 and c_2 are set to 0.50 and 0.01, respectively.

4.6. Embedding mechanism

(1) Machine node embedding

Similar to Song et al. (2023), the graph attention networks (GAT) method is used for the machine node embedding, which can capture different importance among the neighboring nodes using the attention mechanism. Let $\tau_{op}^{i,j}$ be the raw operation feature for $O_{i,j}$, and τ_{om}^{ijk} be the raw O-M arc feature for $O_{i,j}$ processing on M_k. The detailed machine node embedding can be described as follows.

Step 1. Given a machine M_k, for all the neighboring operations in $O_{i,j}\in N_t(M_k)$, perform the following two steps.

Step 2. Concatenate τ_{op}^{ij} and τ_{om}^{ijk} to construct the resulting feature $\mu_{ijk}=[\tau_{op}^{ij}\parallel\tau_{om}^{ijk}]\in R^7$.

Step 3. Calculate the attention coefficients on the edge between M_k and $O_{i,j}$ as follows.

$$e_{ijk}=\text{LeakyReLu}\left(a^{\mathrm{T}}[W^M\tau_{mf}^k\parallel W^O\mu_{ijk}]\right) \tag{24}$$

where, $W^M\in R^{d\times4}$, $W^O\in R^{d\times7}$, $\tau_{mf}^k\in R^{d\times4}$, and $a^T\in R^{2d}$.

Step 4. Calculate the attention coefficients on M_k itself as follow.

$$e_{kk}=\text{LeakyReLu}\left(a^{\mathrm{T}}[W^M\tau_{mf}^k\parallel W^M\tau_{mf}^k]\right) \tag{25}$$

Step 5. Calculate the normalized attention coefficients as follows.

$$\alpha_{ijk}=\sigma\big(Norm\left([e_{ijk}\parallel e_{kk}]\right)\big)_{1:N} \tag{26}$$

$$\alpha_{kk}=\sigma\big(Norm\left([e_{ijk}\parallel e_{kk}]\right)\big)_{N+1} \tag{27}$$

where, $\sigma(.)$ is the softmax function, $Norm(.)$ is the normalize function, $(.)_{1:N}$ is to obtain the first N lines, and $(.)_{N+1}$ is to obtain the $(N+1)^{th}$ line.

Step 6. The machine embedding v_k for M_k is calculated as follows.

$$v_k=\sigma\left(\alpha_{kk}W^M\tau_{mf}^k+\sum_{O_{i,j}\in N_t(M_k)}\alpha_{ijk}W^O\mu_{ijk}\right) \tag{28}$$

(2) Operation node embedding

The multilayer perceptron (MLP) mechanism is used for the operation node embedding $\mu'_{i,j}$, in which there are two d_h-dimensional hidden layers, d-dimensional output, and ELU as the activation function. Four MLP layers are used to capture four types of information, i.e., the neighboring machines of $O_{i,j}$ ($v_{ij}=\sum_{k\in N_t(O_{i,j})}\tau_{mf}^k$), $O_{i,j-1}$($\tau_{op}^{i,j-1}$), $O_{i,j}$, and $O_{i,j+1}$.

The following formulation is used to combine the four features to construct the operation node embedding.

$$\mu'_{i,j}=\text{MLP}_{\theta_0}\begin{pmatrix}\text{ELU}[\text{MLP}_{\theta_1}\left(v_{ij}\right)\parallel\text{MLP}_{\theta_2}\left(\tau_{op}^{i,j-1}\right)\\\parallel\text{MLP}_{\theta_3}\left(\tau_{op}^{i,j}\right)\parallel\text{MLP}_{\theta_4}\left(\tau_{op}^{i,j+1}\right)]\end{pmatrix} \tag{29}$$

(3) Factory node embedding

The GAT method is used for factory node embedding. The detailed factory node embedding can be described as follows.

Step 1. Calculate the attention coefficients on the edge between F_g and τ_{jf}^1 as follows.

$$e_{ig}=\text{LeakyReLu}\left(a^{\mathrm{T}}[W^F\tau_f^g\parallel W^J\tau_{jf}^i]\right) \tag{30}$$

where, $W^F\in R^{d\times2}$, $W^J\in R^{d\times1}$, $\tau_f^G\in R^2$, and $a^T\in R^{2d}$.

Step 2. Calculate the normalized attention coefficients as follows.

$$\alpha_{ig}=\sigma\left(Norm\left(e_{ig}\right)\right) \tag{31}$$

Step 3. The factory node embedding v_g for F_g is calculated as follows.

$$v_g=\sigma\left(\sum_i\alpha_{ig}W^J\tau_{jf}^i\right) \tag{32}$$

4.7. Stacking and PCA-based transform mechanism

The three embeddings for the machine, operation, and factory construct one HGNN layer. To enhance the fusion of the features, L HGNN layers with the same structure are used to obtain the three embeddings, i.e., v_k, $\mu'_{i,j}$, and v_g.

Different from Song et al. (2023), where a pooling stage is applied after the stacking procedure, in this study, we present a PCA-based transform mechanism to enhance the abilities to capture and fuse the embeddings, and therefore, the features can be utilized efficiently. The detailed steps of the PCA-based transform procedure are given in Algorithm 2, where $QR(.)$ represents the QR decomposition, and $SVD(.)$ is the singular value decomposition.

Algorithm 2: PCA-based transform procedure

Input: the three embeddings, i.e., v_k, $\mu'_{i,j}$, and v_g, the embedding length n

Output: the resulting embeddings, v''_k, $\mu''_{i,j}$, and v''_g.

Compute the mean values $\overline{v_k}$, $\overline{\mu_{i,j}}$, and $\overline{v_g}$, for the three embeddings, i.e., v_k, $\mu'_{i,j}$, and v_g, respectively.

Compute U,S,V for v_k, $\mu'_{i,j}$, and v_g as steps 4 to 13.

Generate a random matrix $R_{n\times q}$, $q=min(n,6)$.

$Q'_m=QR((v_k-\overline{v_k})\times R_{n\times q})$.

for $i=1\rightarrow\ell$ **do**

  $Q''_m=QR((v_k-\overline{v_k})^TQ'_m)$.

  $Q_m=QR((v_k-\overline{v_k})Q''_m)$.

end

$U_m,S_m,V_m=SVD((v_k-\overline{v_k})^TQ_m)$.

$v'_k=v_k\times V_m$.

$\mu_{i,j}=\mu'_{i,j}\times V_j$.

$v'_g=v_g\times V_g$

Let $v''_k=\text{mean}(v'_k)$, $\mu''_{i,j}=\text{mean}(\mu_{i,j})$, $v''_g=\text{mean}(v'_g)$

4.8. Policy network

(1) Actor component

The policy network contains two parts, i.e., the *actor* component which is an MLP containing two d-dimensional hidden layers and tanh activation; and the *critic* component, which has the same structure and dependent parameters as the *actor*. The probabilities of the activations under the current state s_t are calculated as follows.

$$P(a_t|s_t) = \mathrm{MLP}_\omega\left[v_k'' \parallel \mu_{ij}''\|v_g''\|v_k\|\mu_{ij}'\|v_g\right] \tag{33}$$

Then, a softmax is applied to all the $P(a_t|s_t)$ to obtain the selection probabilities as follows.

$$\pi_\omega(a_t|s_t) = \frac{\exp(P(a_t|s_t))}{\sum_{a_t'\in A_t}\exp(P(a_t'|s_t))}, \forall a_t \in A_t \tag{34}$$

(2) Critic component

A MLP layer is also used for the *critic* component, with a similar structure to the *actor*. The input of the *critic* is different from that of the *actor*, which has only three parts rather than six parts as in the *actor*. The calculation of the *critic* is as follows.

$$\psi(s_t) = \mathrm{MLP}_\phi\left[v_k''\|\mu_{ij}''\|v_g''\right] \tag{35}$$

4.9. Enhanced local search

During the testing procedures, after a complete solution is generated by the policy network, an enhanced local search heuristic is applied to the solution to further improve the performance. In detail, after generating a solution s_i by the policy network, randomly generate a set of P_s solutions and replace one with s_i. Then, perform the GA-based local search on the P_s solutions. The detailed steps are illustrated in Algorithm 3, in which ℓ_1 is the parameter for the local search iterations defined in Table 5.

Algorithm 3: Enhanced local search procedure

Input: a solution s_i generated by the policy network
. **Output:** the improved solution $\overline{s_i}$
Randomly generate P_s solutions, and replace one with s_i.
for $i = 1 \to \ell_1$ **do**
  Perform partially-matched crossover (PMX), three types of mutations (swap, insertion, inversion) for the population.
  Update the best solution found so far.
end
Record the best solution found so far as $\overline{s_i}$.

5. Numerical analysis

5.1. Experimental settings

(1) Evaluation instances

Three types of instances are selected to test the performance. The first type of instance is used to train and test the performance of each component and generalization capabilities of the proposed algorithm. The training instances are randomly generated during the training process (with 100 validation instances). For testing, 100 instances are sampled for each problem size. Similar to Song et al. (2023), the training dataset contains six different problem sizes, i.e., (10-jobs, 5-machines), (20-jobs, 5-machines), (15-jobs, 10-machines), (20-jobs, 10-machines), (30-jobs, 10-machines), (40-jobs, 10-machines). The number of operations is sampled from U(4, 6) for problems with 5-machines, and U(8, 12) for problems with 10-machines. The number of candidate machines for each operation is sampled from U(1, 5) for problems with 5-machines, and U(1, 10) for problems with 10-machines. The processing time for each operation is sampled from U(1, 20) for all instances. Different from Song et al. (2023), this study extends the instances with distributed factories, where the number of factories range in {2, 3, 5} and all the instances have the same machine layout and processing capabilities in the parallel factories. The number of families is set to $\lceil n/5 \rceil$. Table 4 lists the detailed features of the first type of training and testing instances, where θ_j is the number of operations for each job j; $\left\|M_{j,q}\right\|$ represents the number of candidate machines for each operation; $P_{j,q}$ is the processing time of each operation; $ST_{i,i'}$ represents the setup time between two jobs; F is the number of families.

Table 4
Problem scales.

Size $n \times m \times h$	θ_j	$\left\|M_{j,q}\right\|$	$P_{j,q}$	$ST_{i,i'}$	g
$10 \times 5 \times 2$	$U(4,6)$	$U(1,5)$	$U(1,20)$	$U(1,5)$	2
$20 \times 5 \times 2$	$U(4,6)$	$U(1,5)$	$U(1,20)$	$U(1,5)$	4
$15 \times 10 \times 2$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	3
$20 \times 10 \times 2$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	4
$30 \times 10 \times 2$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	6
$40 \times 10 \times 2$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	8
$10 \times 5 \times 3$	$U(4,6)$	$U(1,5)$	$U(1,20)$	$U(1,5)$	2
$20 \times 5 \times 3$	$U(4,6)$	$U(1,5)$	$U(1,20)$	$U(1,5)$	4
$15 \times 10 \times 3$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	3
$20 \times 10 \times 3$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	4
$30 \times 10 \times 3$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	6
$40 \times 10 \times 3$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	8
$10 \times 5 \times 5$	$U(4,6)$	$U(1,5)$	$U(1,20)$	$U(1,5)$	2
$20 \times 5 \times 5$	$U(4,6)$	$U(1,5)$	$U(1,20)$	$U(1,5)$	4
$15 \times 10 \times 5$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	3
$20 \times 10 \times 5$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	4
$30 \times 10 \times 5$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	6
$40 \times 10 \times 5$	$U(8,12)$	$U(1,10)$	$U(1,20)$	$U(1,5)$	8

The second type of instance is used to make comparisons with the MIP, in which a set of small-scale instances with two distributed factories are tested based on the above generation methods. The instances include seven different problem sizes, i.e., (4, 2), (4, 3), (4, 4), (6, 2), (6, 3), (6, 4), and (6, 5), where the two numbers represent the number of jobs and machines, respectively.

The third type of instance is extended from two well-known FJSP benchmarks, including the 10 mk instances (mk01-mk10) in Hurink, Jurisch, and Thole (1994); and the three groups of la instances in Behnke and Geiger (2012), Hurink et al. (1994) (rdata, edata and vdata), where each group has 40 instances. The dataset of the distributed factories, setup time, and family information is set based on the above generation methods discussed in the first type of instance.

(2) Configuration

All training and testing are performed with a 16 vCPU Intel(R) Xeon(R) Platinum 8350C CPU @ 2.60 GHz, one RTX 3090(24 GB), and ubuntu 20.04 OS. The code is implemented in PyTorch. The detailed parameters are given in Table 5. Note that the population size P_s for the enhanced local search is set to 10 and the local search iteration ℓ_1 is set to 10, the main reason is that the enhance local search should take a tradeoff between the searching strength and time complexity.

(3) Compared algorithms

The compared algorithms include three types. The first type includes heuristics such as FIFO (first-in-first-out), SPT (shortest processing time), LPT (longest processing time). The second type includes the deep learning method HGNN (Song et al., 2023), which is the basic framework extended by the proposed HGNP algorithm. The third type includes meta-heuristics, such as HGA (Tutumlu & Saraç, 2023), and HGA-VNS (Sun et al., 2023). The main reasons to select these compared algorithms are as follows: (1) FIFO, LPT, and SPT are typical heuristics with simple and fast speed, which are generally chosen as compared algorithms with RL or DRL methods; (2) HGA and HGA-VNS are two recently published method for solving different types of FJSPs; and (3) HGNN is a recently published DRL method for solving FJSPs.

The gap value is calculated as follows:

$$gap = (C_{\max} - C_B)/C_B \times 100\% \tag{36}$$

Table 5
Parameters.

Parameter	Value	Parameter	Value
HGNN layer L	2	PPO policy loss	1
Embedding dimension d	8	PPO clip ratio	0.2
Hidden dimensions d_h	128	PPO value loss	0.5
Hidden dimensions $d_\pi = d_\phi$	64	PPO entropy term	0.01
Training iterations ℓ_t	1000	PPO optimization	3
Batch size B_s	20	PPO discount factor	1
Network optimizer	Adam	Optimizer learning rate	2×10^{-4}
Local search population size P_s	10	Local search crossover probability P_c	0.5
Local search iterations ℓ_1	10	Local search mutation probability P_m	0.5

where, C_B is the best makespan obtained by all the compared algorithms.

The detailed descriptions and control parameters of the compared algorithms are given as follows.

- The hybrid genetic algorithm (HGA) (Tutumlu & Saraç, 2023) is selected because the algorithm is designed for the FJSP with job splitting, with a similar problem with different constraints. According to the literature, the parameters are used as follows: population size: 100, probability for crossover: 0.7, probability for mutation: 0.05, and 500 iterations as the stop criterion. To apply the HGA algorithm for solving the DFJSP with family setup time, the following modifications are made: (1) the solution representation is revised from 2-dimensional matrix to 3-dimensional matrix, in which the factory assignment is added; (2) the following components, i.e., the elitism selection, one-point crossover, job-based crossover (JBX) crossover, swap and inversion mutation operators, are utilized directly. Especially, we adapt one-point crossover and swap mutation operators for factory assignment part.
- Hybrid GA with variable neighborhood search (HGA-VNS) (Sun et al., 2023) is selected because the algorithm is also designed for the FJSP. As listed in the literature, the parameters are set as follows: mutation probability: 0.3, proportion of elite individuals: 0.1, crossover probability: 0.3, combined crossover probability: 0.3, combined mutation probability: 0.5, population size: 200, and iteration number set to 500 as the stop criterion. The following components are described as follows: (1) the decoding scheme is modified from a 2-dimensional to 3-dimensional matrix; (2) the initialization scheme is directly utilized as in the literature, except that the factory assignment is generated in a random way; (3) the elite retention strategy is embedded directly; (4) the hybrid selection operator including tournament selection and elite retention is used; and (5) crossover: 0–1 crossover, fixed position crossover (FPX), and the improved precedence operation crossover (IPOX); mutation operators: exchange mutation and swap mutation. One-point crossover and swap mutation operators are also used for the factory assignment part.
- Random heuristic, which generates a sequence of operations in a random way, and selects machines and factories for operations in a random way.
- The FIFO heuristic randomly generates a sequence of operations, and selects the first available factory and machine for each operation.
- The SPT heuristic sorts all the operations according to their mean processing time in a nondecreasing order, and selects the first available factory and machine for each operation.
- LPT heuristics, different from SPT heuristic, LPT sorts all the operations according to their mean processing time in a nonincreasing order.

Table 6
Comparisons with CPLEX.

Inst	Scale	Best	Makespan		Gap	
			CPLEX	HGNP	CPLEX	HGNP
1	4×2×2	22	22	22	0.00+	0.00+
2	4×2×2	22	22	22	0.00+	0.00+
3	4×2×2	28	28	28	0.00+	0.00+
4	4×3×2	17	18	17	5.88	0.00+
5	4×3×2	29	54	29	86.21	0.00+
6	4×3×2	18	18	18	0.00+	0.00+
7	4×4×2	36	37	36	2.78	0.00+
8	4×4×2	22	22	23	0.00+	4.55
9	4×4×2	32	32	34	0.00+	6.25
10	6×2×2	30	30	33	0.00+	10.00
11	6×2×2	25	25	25	0.00+	0.00+
12	6×2×2	30	30	32	0.00+	6.67
13	6×3×2	24	24	30	0.00+	25.00
14	6×3×2	29	29	34	0.00+	17.24
15	6×3×2	35	35	38	0.00+	8.57
16	6×4×2	43	44	43	2.33	0.00+
17	6×4×2	33	33	37	0.00+	12.12
18	6×4×2	42	50	42	19.05	0.00+
19	6×5×2	37	52	37	40.54	0.00+
20	6×5×2	22	–	49	–	0.00+
Avg		29.16	31.84	30.53	7.87	4.52+

- +means the better value.

5.2. Comparisons with CPLEX

The MIP is coded in CPLEX, in which the stop criterion is set to 5400 s for each instance. The proposed HGNP algorithm is trained individually for different size of instances, from 4-jobs-2-machines-2-factories to 6-jobs-5-machines-2-factories. For each group with different problem scales, 100 instances are randomly generated, in which 80 instances are randomly chosen for training and the other 20 instances are used for validation. To make fair comparisons with CPLEX, for each problem scale, two or three instances are randomly generated for testing. Totally, 20 small-sized instances are chosen for comparisons with CPLEX and the trained HGNP model.

The resulting makespan value is collected for comparisons, which are listed in Table 6. It can be concluded that: (1) for the given small-sized instances, the proposed algorithm obtained 12 optimal values out of the given 20 instances, while CPLEX obtained 14 better results; (2) the average gap values from the last line show that the proposed algorithm is better considered the average performance; and (3) the proposed algorithm found 7 better results compared with CPLEX within 5 min, which is significantly faster than 1.5 h in CPLEX.

5.3. Generalization performance on large-sized instances

Fig. 3 shows the training curve on 10 × 5 × 2 instances, which can tell us that, the proposed algorithm can learn a high-quality policy based on its own solving experiences.

One of the advantages of the DRL methods is the generalization abilities to solve unknown instances with different sizes. Most of the current literature aiming at testing the generalization in part of the problem

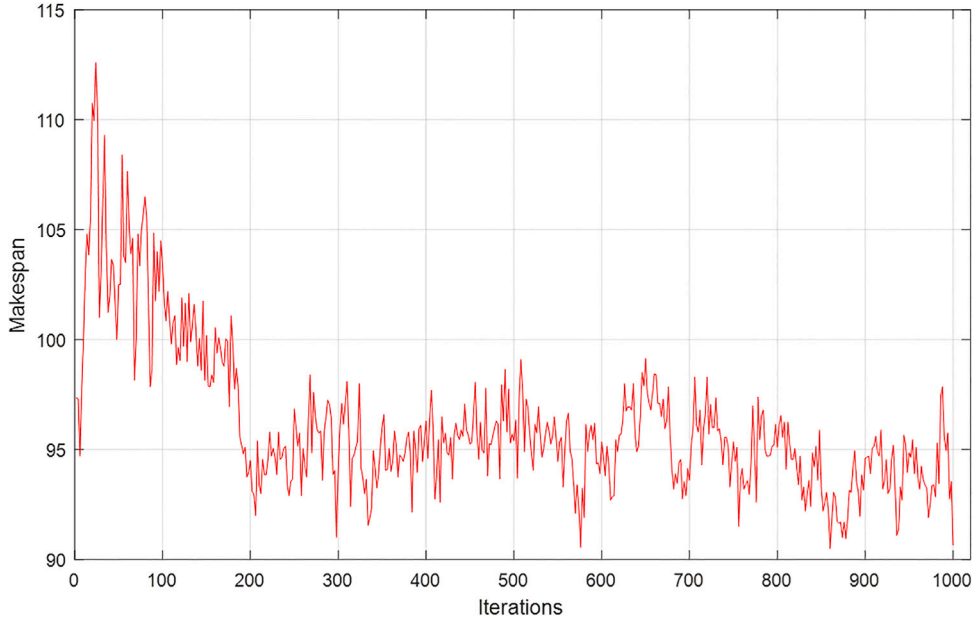

Fig. 3. Training curve on 10 × 5 × 2 instances.

scales. For example, Li, Li, Gao, and Duan (2024) developed a hybrid graph-based imitation learning method for a distributed hybrid flow shop scheduling problem, wherein the generalization testing focused on the number of jobs, whereas the number of stages and machines are fixed. Huang, Gao, and Li (2024) proposed an end-to-end DRL based on graph neural network for distributed job shop scheduling problem, wherein the generalization is tested for the unknown instances with different number of jobs and machines, however, the number of factories should be the same. In addition, relevant literature considering FJSP such as (Lei, Guo, Wang, Zhang, Meng, & Qian, 2023; Lei et al., 2022; Song et al., 2023) test the generalization with different number of jobs, whereas the number of machines of the unknown instances are the same with the trained models. Similar to Huang et al. (2024), we considered generalization testing for unknown DFJSP instances with different scales in number of jobs and machines.

To test the generalization performance of the proposed HGNP algorithm on instances with large problem scales, we trained the algorithm on three types of small-sized instances, namely, $10 \times 5 \times 2$, $10 \times 5 \times 3$, and $10 \times 5 \times 5$, hereafter called $M_{10\times5\times2}$, $M_{10\times5\times3}$, and $M_{10\times5\times5}$. Then, we randomly generated 21 groups of large-sized instances, from $50 \times 10 \times 2$ to $80 \times 30 \times 5$. Each group of instance contains 100 randomly-generated instances. Totally, 2100 instances are chosen to test the generalization capabilities of the proposed algorithm. Next, we applied the three trained models, i.e., $M_{10\times5\times2}$, $M_{10\times5\times3}$, and $M_{10\times5\times5}$, to test on these 2100 instances. Note that the trained model should apply for the testing instances with the same factory number, e.g., $M_{10\times5\times2}$ is used for 700 instances with two distributed factories, and $M_{10\times5\times3}$ and $M_{10\times5\times5}$ are used for 700 instances with three and five distributed factories, respectively. To make a fair comparison, we select HGNN (Song et al., 2023) as the compared algorithm, which tested on these 21 different-scale groups of instances after training on the same scale of instances.

Table 7 shows the comparison results, which can be observed that: (1) applying the $M_{10\times5\times2}$ model to solve the seven group of large-sized instances, the proposed algorithm can obtain all better results compared with HGNN; (2) applying the $M_{10\times5\times3}$ model to solve the instances with three distributed factories, HGNP obtained six better results. It is interesting that conducting the $M_{10\times5\times3}$ model, the proposed HGNP algorithm can find the best results for the three large-sized group of instances with 80 jobs; (3) applying the $M_{10\times5\times5}$ model, HGNP obtained five better results, which is obviously better than HGNN; and (4) the average performance displayed in the last line show that, the proposed HGNP algorithm is significantly better than HGNN by applying different type of models, which further verify the efficiency of the generalization performance.

Table 7
Comparisons of the generalization performance.

Model	Problem Scale	Best	Makespan		Gap	
			HGNN	HGNP	HGNN	HGNP
$M_{10\times5\times2}$	50 × 10 × 2	485.35	494.87	485.35	1.96	0.00+
	60 × 10 × 2	561.26	572.70	561.26	2.04	0.00+
	70 × 10 × 2	643.64	647.64	643.64	0.62	0.00+
	70 × 20 × 2	780.06	797.29	780.06	2.21	0.00+
	80 × 10 × 2	723.33	729.59	723.33	0.87	0.00+
	80 × 20 × 2	871.69	884.85	871.69	1.51	0.00+
	80 × 30 × 2	1054.20	1112.56	1054.20	5.54	0.00+
$M_{10\times5\times3}$	50 × 10 × 3	387.52	396.83	387.52	2.40	0.00+
	60 × 10 × 3	415.28	433.12	415.28	4.30	0.00+
	70 × 10 × 3	489.33	521.12	489.33	6.50	0.00+
	70 × 20 × 3	621.15	621.15	622.83	0.00+	0.27
	80 × 10 × 3	675.67	699.34	675.67	3.50	0.00+
	80 × 20 × 3	733.48	745.61	733.48	1.65	0.00+
	80 × 30 × 3	887.15	903.14	887.15	1.80	0.00+
$M_{10\times5\times5}$	50 × 10 × 5	289.19	321.18	289.19	11.06	0.00+
	60 × 10 × 5	356.47	377.49	356.47	5.90	0.00+
	70 × 10 × 5	392.91	411.69	392.91	4.78	0.00+
	70 × 20 × 5	479.87	501.65	479.87	4.54	0.00+
	80 × 10 × 5	598.79	598.79	601.12	0.00+	0.39
	80 × 20 × 5	633.73	633.73	651.34	0.00+	2.78
	80 × 30 × 5	769.12	785.73	769.12	2.16	0.00+
Avg		611.87	628.10	612.90	3.02	0.16+

- +means the better value.

5.4. Efficiency of the enhanced local search heuristic

To verify the efficiency of the enhanced local search heuristic, two types of HGNP are implemented, i.e., HGNP without the local search method (HGNP-NL), and HGNP with the local search procedure.

Table 8 shows that: (1) HGNP obtained all the better results, which are significantly better than HGNP-NL; (2) for small-sized instances, such as 15 × 10 and 20 × 5, the enhanced local search heuristic deeply improves the network performance, and the gap values are

Table 8
Comparisons of the enhanced local search heuristic.

Size	Best	Makespan		Gap	
		HGNP-NL	HGNP	HGNP-NL	HGNP
15 × 10 × 2	164.97	171.16	164.97	3.75	0.00+
20 × 5 × 2	148.95	154.21	148.95	3.53	0.00+
20 × 10 × 2	183.77	185.05	183.77	0.70	0.00+
30 × 10 × 2	242.87	245.57	242.87	1.11	0.00+
40 × 10 × 2	298.63	299.77	298.63	0.38	0.00+
15 × 10 × 3	119.13	125.65	119.13	5.47	0.00+
20 × 5 × 3	132.67	136.88	132.67	3.17	0.00+
20 × 10 × 3	147.19	149.27	147.19	1.41	0.00+
30 × 10 × 3	185.88	187.8	185.88	1.03	0.00+
40 × 10 × 3	224.22	225.64	224.22	0.63	0.00+
15 × 10 × 5	105.22	108.34	105.22	2.97	0.00+
20 × 5 × 5	109.12	115.23	109.12	5.60	0.00+
20 × 10 × 5	129.29	134.01	129.29	3.65	0.00+
30 × 10 × 5	153.11	155.92	153.11	1.84	0.00+
40 × 10 × 5	195.59	196.77	195.59	0.60	0.00+
Avg	169.37	170.15	169.37	2.40	0.00+

- +means the better value.

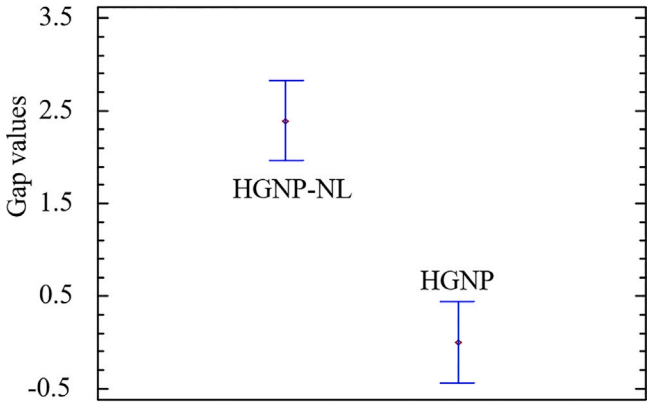

Fig. 4. ANOVA comparisons with HGNP and HGNP-NL.

largely contributed from these instances; (3) for the relatively large-sized instances, the two methods are slightly different from each other, which shows that the enhanced local search heuristic made slight contributions for large-sized instances. The main reason is that, the stop criterion is set to a limited time to adapt to the realistic production requirements; (4) we can find that, although the proposed HGNP is efficient for medium and large scales of instances, HGNP needs the help of local search heuristic to enhance the local search performance in solving small-sized instances; and (5) the average values listed in the last line show that the enhanced local search heuristic significantly improves the average performance.

Analysis of variance (ANOVA) is used to verify whether there is a significant difference between the two compared algorithms. Fig. 4 gives the ANOVA comparisons with HGNP-NL and HGNP, which shows that the difference between the two algorithms is significant.

5.5. Efficiency of the PCA-based transform mechanism

To test the efficiency of the proposed PCA-based transform mechanism, we encode two types of methods, i.e., the HGNN with the average pooling method, extended from Song et al. (2023); and the HGNP-NL, which replaces the pooling method with the PCA-based mechanism.

Table 9 shows the results obtained by the two compared algorithms and the gap values. It can be observed from the table that: (1) for the instances with two factories, the proposed algorithm obtained all the better values out of the given five different size instances; (2) for the

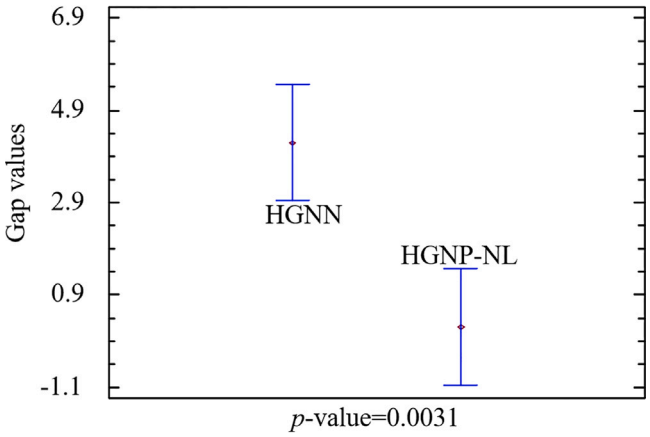

Fig. 5. ANOVA comparisons for HGNN and HGNP-NL.

Table 9
Comparisons of the PCA-based transform mechanism.

Size	Best	Makespan		Gap	
		HGNN	HGNP-NL	HGNN	HGNP-NL
15 × 10 × 2	171.16	171.20	171.16	0.02	+0.00
20 × 5 × 2	154.21	176.36	154.21	14.36	+0.00
20 × 10 × 2	185.05	189.54	185.05	2.43	+0.00
30 × 10 × 2	245.57	268.49	245.57	9.33	+0.00
40 × 10 × 2	299.77	341.90	299.77	14.05	+0.00
15 × 10 × 3	123.1	123.10	125.65	+0.00	2.07
20 × 5 × 3	136.88	139.93	136.88	2.23	+0.00
20 × 10 × 3	149.27	157.18	149.27	5.30	+0.00
30 × 10 × 3	187.8	197.22	187.8	5.02	+0.00
40 × 10 × 3	225.64	229.16	225.64	1.56	+0.00
15 × 10 × 5	107.25	107.25	108.34	+0.00	1.02
20 × 5 × 5	115.23	117.18	115.23	1.69	+0.00
20 × 10 × 5	134.01	138.39	134.01	3.27	+0.00
30 × 10 × 5	155.92	158.93	155.92	1.93	+0.00
40 × 10 × 5	196.77	200.16	196.77	1.72	+0.00
Avg	172.51	181.07	170.15	4.19	+0.21

- +means the better value.

five groups of instances with three and five distributed factories, HGNP-NL obtained four better results, whereas HGNN can only obtain one better result, respectively; and (2) the average performance shown in the last line verifies that the proposed HGNP-NL algorithm obtained a value of 0.21, which is approximately 20 times lower than the HGNN algorithm.

Fig. 5 shows the ANOVA comparisons for HGNN and HGNP-NL, wherein the p-value is 0.0031 and significantly less than the threshold value 0.05. It can be concluded from Fig. 5 that the proposed algorithm is significantly better than HGNN. Therefore, the PCA-based transform mechanism is efficient in improving the algorithm.

5.6. Comparisons with heuristics and meta-heuristics

Tables 10–12 show the comparison results of the gap values obtained by the seven compared algorithms, i.e., HGA, HGA-VNS, RANDOM, LPT, SPT, FIFO, and HGNP, for 2-factory, 3-factory, and 5-factory instances, respectively.

It can be observed from these tables that: (1) considering the 2-factory instances, the proposed HGNP algorithm can obtain four better values out of the given five different size instances, especially for the relatively large-sized instances, i.e., 20×10, 30×10, 40×10 instances. HGA is the second-best algorithm, which obtains the better results for the 20×5 instances. However, the HGA becomes worse in solving the large-sized instances. Therefore, the HGA is suitable for small-sized instances; (2) for the 3-factory instances, HGNP is the most

Table 10
Comparisons with heuristics and meta-heuristics for 2-factory DFJSP (Gap values).

Problem	Size					Avg
	15 × 10 × 2	20 × 5 × 2	20 × 10 × 2	30 × 10 × 2	40 × 10 × 2	
HGA	0.93	+0.00	12.65	9.81	11.63	7.00
HGA-VNS	2.51	1.66	13.53	10.61	11.85	8.03
RANDOM	63.17	54.48	87.56	87.53	90.76	76.70
LPT	14.07	12.00	24.80	17.65	17.82	17.27
SPT	16.53	14.49	27.60	20.36	19.18	19.63
FIFO	16.57	14.90	25.46	19.16	18.92	19.00
HGNP	+0.00	2.66	+0.00	+0.00	+0.00	+0.53

- +means the better value.

Table 11
Comparisons with heuristics and meta-heuristics for 3-factory DFJSP (Gap values).

Problem	Size					Avg
	15 × 10 × 3	20 × 5 × 3	20 × 10 × 3	30 × 10 × 3	40 × 10 × 3	
HGA	0.87	0.23	5.81	10.13	13.45	6.10
HGA-VNS	+0.00	0.13	4.92	8.12	10.42	4.72
RANDOM	76.23	69.34	90.43	95.76	100.34	86.42
LPT	16.71	15.87	30.83	25.3	18.88	21.52
SPT	18.48	16.62	29.92	23.5	20.18	21.74
FIFO	19.66	17.91	24.48	20.1	22.32	20.89
HGNP	0.53	+0.00	+0.00	+0.00	+0.00	+0.11

- +means the better value.

Table 12
Comparisons with heuristics and meta-heuristics for 5-factory DFJSP (Gap values).

Problem	Size					Avg
	15 × 10 × 5	20 × 5 × 5	20 × 10 × 5	30 × 10 × 5	40 × 10 × 5	
HGA	0.52	1.46	5.87	4.98	8.72	4.31
HGA-VNS	+0.00	+0.00	2.91	4.35	5.09	2.47
RANDOM	90.12	78.85	100.19	98.77	120.43	97.67
LPT	29.14	20.33	31.17	28.19	46.01	30.97
SPT	19.93	20.68	35.73	33.89	48.18	31.68
FIFO	19.45	28.81	33.17	35.57	29.76	29.35
HGNP	0.15	0.09	+0.00	+0.00	+0.00	+0.05

- +means the better value.

efficient algorithm with four better results for the relative large-size instances. HGV-VNS can obtain the better result for the small-size instances with 15-jobs and 10-machines; (3) considering the 5-factory instances, HGNP also shows better performance in solving the large-size instances; (4) the average values obtained by HGNP for the three types of instances with different number of factories are 0.53, 0.11, and 0.05, respectively, which are significantly better than the other compared algorithms. and (5) in summary, HGNP is the most efficient algorithm among the seven algorithms, the following better algorithms are HGA-VNS and HGA. Then, the heuristics, such as LPT, FIFO, SPT, show worse performance than these meta-heuristics, but better performance than the RANDOM method.

5.7. Comparisons of run time

To make detailed comparisons of the computation time, we select four algorithms, i.e., HGA, HGA-VNS, HGNP and HGNP-NL. Table 13 collects the comparison results, which can be concluded that: (1) HGNP-NL consumes the least computation times for all different size instances, which is approximately 10% of the other algorithms. Therefore, HGNP-NL can generate relatively satisfactory solutions for different size instances. For example, HGNP-NL took 1.7E+3 s for the given 300 40-jobs-10-machines instances with different number of factories, which is approximately 5 times lower than HGA and applicable in realistic production systems; (2) after embedding the local search, the time consumed increased slightly for small or medium-sized instances, but increased largely for large-sized instances. However, the proposed HGNP algorithm shows better performance while took significantly lower computational times compared with HGA and HGA-VNS.

Table 13
Comparisons of total consumed computational times (second).

Problem	Size				
	15 × 10	20 × 5	20 × 10	30 × 10	40 × 10
HGA	4.2E3	2.4E3	5.2E3	6.4 E3	8.5E3
HGA-VNS	5.9E3	2.9 E3	6.3E3	9.8E3	1.5E4
HGNP-NL	3.7E2+	2.5E2+	4.9E2+	8.2E2+	1.7E3+
HGNP	5.8E2	4.3E2	1.5E3	2.3E3	3.2E3

- +means the better value.

5.8. Comparisons of public instances

The public instances include four categories, i.e., the mk, rdata, edata and vdata, which range from 10 jobs to 30 jobs, and 5 machines to 15 machines. We extend the four groups of public instances and added distributed factories in them, wherein two factories, three factories, and five factories are considered. Eight methods are chosen to make detailed comparisons, including HGA, HGA-VNS, and six different models applied to the public instances, in which $HGNP_1$, $HGNP_2$, $HGNP_3$, $HGNP_4$, $HGNP_5$, and $HGNP_6$ represents the proposed HGNP algorithm with the training models $M_{10\times5}$, $M_{15\times10}$, $M_{20\times5}$, $M_{20\times10}$, $M_{30\times10}$, and $M_{40\times10}$, respectively. Note that we should train six models for 2-factory, 3-factory, and 5-factory instances, respectively. Therefore, totally 18 trained models are generated and used for the extended public instances. The average makespan results are collected for each compared algorithm after testing all these instances, which are shown in Table 14.

It can be summarized from the table that: (1) for the 30 extended mk instances, $HGNP_4$ obtained the best performance. Compared with the policy network, the HGA and HGA-VNS obtained relatively worse results, and the gap values were 5.7% and 8.5%, respectively. Note that the average computational time consumed by the proposed algorithm is approximately 20 times lower than that of the meta-heuristic; (2) for the 120 extended rdata instances, the HGA-VNS method is the best, while $HGNP_3$ follows the HGA-VNS with a small average time consumption; (3) for the 120 extended edata instances, the $HGNP_6$ performs best, which is significantly better than the two meta-heuristics within 1.6 seconds for each instance, which is more applicable in realistic production systems; (4) for the 120 extended vdata instances, $HGNP_6$ also obtained better results, especially within the smallest computational times. Note that, for the vdata instances, all the models obtained better results than HGA and HGA-VNS, and the time consumed by the policy network is approximately 20 times lower than the meta-heuristics; and (5) generally, the policy network proposed in this study shows efficiency both in the solution performance and the time consumption.

Fig. 6 shows the Gantt chart for one solution of the "e_la11" instance, where there are 2-factories, 10-jobs, and 5-machines. Each job is represented by a different color, and each operation is represented by a rectangle labeled with the job number and operation number. It can be seen clearly that all the operations have been scheduled without violating the constraints.

5.9. Comparisons analysis

Through the experimental comparisons for each component and the generalization capabilities, it can be concluded that the proposed algorithm is efficient for solving the considered problem with realistic constraints. The main advantages of the proposed algorithm are as follows: (1) a novel state representation is designed, which combines the features of operation, machine and factory assignment; (2) a multilayer perceptron (MLP) mechanism is used for the operation embedding, and graph attention networks (GATs) are embedded for the machine and factory embeddings; and (3) a PCA-based transform mechanism is developed to further fuse all three embeddings.

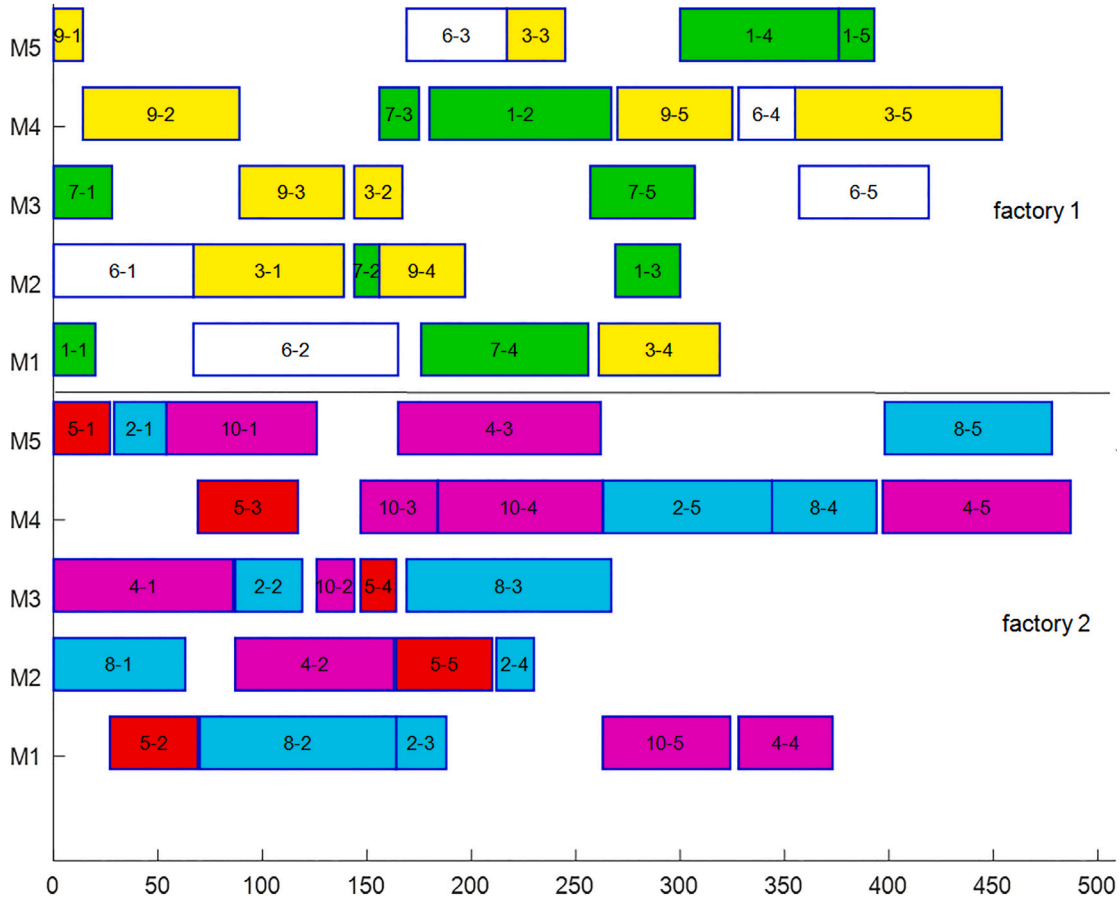

Fig. 6. Gantt chart of one solution for "e_la11" with two factories.

Table 14
Comparisons of the extended public instances.

Algorithm	Mk			rdata			edata			vdata		
	C_{max}	Gap	Time(s)	C_{max}	Gap	Time(s)	C_{max}	Gap	Time(s)	C_{max}	Gap	Time(s)
HGA	149.3	5.7%	35.1	721.3	1.1%	29.1	709.1	3.4%	27.6	695.8	8.3%	34.5
HGA-VNS	153.2	8.5%	40.2	713.2	+0.0%	38.9	713.4	4.1%	38.4	694.3	8.1%	42.3
$HGNP_1$	146.1	3.5%	1.8	720.7	1.1%	0.8	746.2	8.8%	1.7	683.7	6.4%	1.8
$HGNP_2$	145.6	3.1%	2.1	718.9	0.8%	1.2	701.9	2.4%	1.9	657.0	2.3%	1.6
$HGNP_3$	150.7	6.7%	1.9	713.7	0.1%	1.1	719.2	4.9%	1.8	656.2	2.1%	1.9
$HGNP_4$	141.2	+0.0%	1.9	717.9	0.7%	1.0	693.4	1.1%	1.9	649.7	1.1%	1.8
$HGNP_5$	145.8	3.3%	1.6	719.5	0.9%	1.0	697.0	1.7%	1.8	659.1	2.6%	1.7
$HGNP_6$	143.2	1.4%	1.8	717.3	0.6%	1.3	685.6	+0.0%	1.6	642.5	+0.0%	1.5

- +means the better value.

6. Conclusions

In this study, the DFJSP with family setup time was firstly considered. To solve the problem, a mathematical model was formulated. Then, a novel state representation was developed, which considers the features of operation, machine and factory assignment. Then, a multi-layer heterogeneous graph neural network was embedded to transform the raw features to embeddings with the same length. To fuse the embeddings efficiently, a principal component analysis (PCA)-based transform mechanism was conducted. Then, the embeddings were used as the input of the actor–critic policy network, which can select operations, machines and factory assignment. An enhanced local search heuristic is developed to further improve the solution performance obtained by the policy network.

Although the efficiency and effectiveness of the proposed HGNP algorithm has been verified, one main challenging issue is that, the trained model can only be generalized to unknown instances with different number of jobs and machines. To the best of our knowledge, there are very few literature considering generalization simultaneously in the number of distributed factories, jobs and machines, especially in the fields of DFJSP. From managerial and practical insights, future works will mainly focus on the following: (1) consider more realistic constraints in the DFJSP, such as fuzzy processing time, no-wait, and limited resource constraints; (2) combine efficient components into the heterogeneous graph neural network, and therefore, the embeddings of different raw features can be fused efficiently; (3) consider more useful objectives, such as energy consumption, working efficiency, and economic costs; (4) apply the proposed policy network to solve realistic optimization problems, such as steelmaking casting problems, and prefabricated scheduling problems; and (5) consider generalization simultaneously in the number of distributed factories, jobs and machines.

CRediT authorship contribution statement

Jiake Li: Writing – original draft, Methodology, Conceptualization. **Junqing Li:** Writing – review & editing, Supervision, Software, Methodology, Funding acquisition, Conceptualization. **Ying Xu:** Visualization, Investigation, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is partially supported by Key projects of Yunnan Province Basic Research Program, China (202401AS070036), Yunnan Key Laboratory of Modern Analytical Mathematics and Applications, China (No. 202302AN360007).

Data availability

Data will be made available on request.

References

An, Y., Chen, X., Gao, K., Li, Y., & Zhang, L. (2023). Multiobjective flexible job-shop rescheduling with new job insertion and machine preventive maintenance. *IEEE Transactions on Cybernetics*, *53*(5), 3101–3113.

Behnke, D., & Geiger, M. J. (2012). *Test instances for the flexible job shop scheduling problem with work centers*: *Tech. Rep.*.

Brammer, J., Lutz, B., & Neumann, D. (2022). Permutation flow shop scheduling with multiple lines and demand plans using reinforcement learning. *European Journal of Operational Research*, *299*(1), 75–86.

Cao, Z., Lin, C., & Zhou, M. (2021). A knowledge-based cuckoo search algorithm to schedule a flexible job shop with sequencing flexibility. *IEEE Transactions on Automation Science and Engineering*, *18*(1), 56–69.

Chen, X., Li, J., Wang, Z., Li, J., & Gao, K. (2025). A genetic programming based cooperative evolutionary algorithm for flexible job shop with crane transportation and setup times. *Applied Soft Computing*, *169*, http://dx.doi.org/10.1016/j.asoc.2024.112614.

Chen, R., Yang, B., Li, S., & Wang, S. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering*, *149*.

De Giovanni, L., & Pezzella, F. (2010). An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. *European Journal of Operational Research*, *200*(2), 395–408.

Du, Y., Li, J.-q., Chen, X.-l., Duan, P.-y., & Pan, Q.-k. (2023). Knowledge-based reinforcement learning and estimation of distribution algorithm for flexible job shop scheduling problem. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *7*(4), 1036–1050.

Du, Y., Li, J., Li, C., & Duan, P. (2022). A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, 1–15.

Fan, J., Zhang, C., Liu, Q., Shen, W., & Gao, L. (2022). An improved genetic algorithm for flexible job shop scheduling problem considering reconfigurable machine tools with limited auxiliary modules. *Journal of Manufacturing Systems*, *62*, 650–667.

Gao, K., Yang, F., Zhou, M., Pan, Q., & Suganthan, P. N. (2019). Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm. *IEEE Transactions on Cybernetics*, *49*(5), 1944–1955.

Gong, G., Chiong, R., Deng, Q., Gong, X., Lin, W., Han, W., et al. (2022). A two-stage memetic algorithm for energy-efficient flexible job shop scheduling by means of decreasing the total number of machine restarts. *Swarm and Evolutionary Computation*, *75*, http://dx.doi.org/10.1016/j.swevo.2022.101131.

Gui, Y., Tang, D., Zhu, H., Zhang, Y., & Zhang, Z. (2023). Dynamic scheduling for flexible job shop using a deep reinforcement learning approach. *Computers & Industrial Engineering*, *180*, http://dx.doi.org/10.1016/j.cie.2023.109255.

Huang, J.-P., Gao, L., & Li, X.-Y. (2024). An end-to-end deep reinforcement learning method based on graph neural network for distributed job-shop scheduling problem. *Expert Systems with Applications*, *238*, Article 121756.

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machines. *OR Spektrum*, *15*(4), 205–215.

Jamrus, T., Chien, C.-F., Gen, M., & Sethanan, K. (2018). Hybrid particle swarm optimization combined with genetic operators for flexible job-shop scheduling under uncertain processing time for semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, *31*(1), 32–41.

Lei, K., Guo, P., Wang, Y., Zhang, J., Meng, X., & Qian, L. (2023). Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning. *IEEE Transactions on Industrial Informatics*, *20*(1), 1007–1018.

Lei, K., Guo, P., Zhao, W., Wang, Y., Qian, L., Meng, X., et al. (2022). A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. *Expert Systems with Applications*, *205*, http://dx.doi.org/10.1016/j.eswa.2022.117796.

Li, J. Q., Du, Y., Gao, K. Z., Duan, P. Y., Gong, D. W., Pan, Q. K., et al. (2022). A hybrid iterated greedy algorithm for a crane transportation flexible job shop problem. *IEEE Transactions on Automation Science and Engineering*, *19*(3), 2153–2170.

Li, H., Gao, K., Duan, P.-Y., Li, J.-Q., & Zhang, L. (2023). An improved artificial bee colony algorithm with Q-learning for solving permutation flow-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *53*(5), 2684–2693.

Li, R., Gong, W., & Lu, C. (2022). A reinforcement learning based RMOEA/D for bi-objective fuzzy flexible job shop scheduling. *Expert Systems with Applications*, *203*.

Li, R., Gong, W., Lu, C., & Wang, L. (2023). A learning-based memetic algorithm for energy-efficient flexible job-shop scheduling with type-2 fuzzy processing time. *IEEE Transactions on Evolutionary Computation*, *27*(3), 610–620.

Li, R., Gong, W., Wang, L., Lu, C., & Zhuang, X. (2023). Surprisingly popular-based adaptive memetic algorithm for energy-efficient distributed flexible job shop scheduling. *IEEE Transactions on Cybernetics*, 1–11. http://dx.doi.org/10.1109/tcyb.2023.3280175.

Li, Y., Gu, W., Yuan, M., & Tang, Y. (2022). Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network. *Robotics and Computer-Integrated Manufacturing*, *74*, http://dx.doi.org/10.1016/j.rcim.2021.102283.

Li, J., Han, Y., Gao, K., Xiao, X., & Duan, P. (2023). Bi-population balancing multi-objective algorithm for fuzzy flexible job shop with energy and transportation. *IEEE Transactions on Automation Science and Engineering*, 1–17. http://dx.doi.org/10.1109/tase.2023.3300922.

Li, M., & Lei, D. (2021). An imperialist competitive algorithm with feedback for energy-efficient flexible job shop scheduling with transportation and sequence-dependent setup times. *Engineering Applications of Artificial Intelligence*, *103*.

Li, J., Li, J., Gao, K., & Duan, P. (2024). A hybrid graph-based imitation learning method for a realistic distributed hybrid flow shop with family setup time. *IEEE Transactions on Systems, Man and Cybernetics: Systems*, http://dx.doi.org/10.1109/TSMC.2024.3449413.

Li, J., Li, R., J., Li., Yu, X., & Xu, Y. (2025). A multi-dimensional co-evolutionary algorithm for multi-objective resource-constrained flexible flowshop with robotic transportation. *Applied Soft Computing*, *169*, http://dx.doi.org/10.1016/j.asoc.2024.112689.

Li, K. Z. T., Wang, R., Wang, Y., Han, Y., & Wang, L. (2021). Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, *52*(12), 13142–13155.

Lin, C.-S., Li, P.-Y., Wei, J.-M., & Wu, M.-C. (2020). Integration of process planning and scheduling for distributed flexible job shops. *Computers & Operations Research*, *124*.

Liu, R., Piplani, R., & Toro, C. (2022). Deep reinforcement learning for dynamic scheduling of a flexible job shop. *International Journal of Production Research*, *60*(13), 4049–4069.

Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing*, *91*, http://dx.doi.org/10.1016/j.asoc.2020.106208.

Luo, S., Zhang, L., & Fan, Y. (2022). Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning. *IEEE Transactions on Automation Science and Engineering*, *19*(4), 3020–3038.

Lv, Y., Li, C., Tang, Y., & Kou, Y. (2022). Toward energy-efficient rescheduling decision mechanisms for flexible job shop with dynamic events and alternative process plans. *IEEE Transactions on Automation Science and Engineering*, *19*(4), 3259–3275.

Müller, D., Müller, M. G., Kress, D., & Pesch, E. (2022). An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning. *European Journal of Operational Research*, *302*(3), 874–891.

Pan, Z., Wang, L., Dong, C., & f. Chen, J. (2023). A knowledge-guided end-to-end optimization framework based on reinforcement learning for flow shop scheduling. *IEEE Transactions on Industrial Informatics*, 1–9. http://dx.doi.org/10.1109/tii.2023.3282313.

Pan, Z., Wang, L., Wang, J., & Lu, J. (2023). Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *7*(4), 983–994.

Pan, Z., Wang, L., Zheng, J., Chen, J.-f., & Wang, X. (2022). A learning-based multi-population evolutionary optimization for flexible job shop scheduling problem with finite transportation resources. *IEEE Transactions on Evolutionary Computation*, http://dx.doi.org/10.1109/tevc.2022.3219238, 1–1.

Park, M.-J., & Ham, A. (2022). Energy-aware flexible job shop scheduling under time-of-use pricing. *International Journal of Production Economics*, *248*, http://dx.doi.org/10.1016/j.ijpe.2022.108507.

Shao, Z., Shao, W., & Pi, D. (2023). LS-HH: A learning-based selection hyper-heuristic for distributed heterogeneous hybrid blocking flow-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *7*(1), 111–127.

Shen, L., Dauzère-Pérès, S., & Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, *265*(2), 503–516.

Song, W., Chen, X., Li, Q., & Cao, Z. (2023). Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, *19*(2), 1600–1610.

Song, H., Li, J., Du, Z., Yu, X., & Xu, Y. (2025). A q-learning driven multi-objective evolutionary algorithm for worker fatigue dual-resource-constrained distributed hybrid flow shop. *Computers & Operations Research*, *175*, http://dx.doi.org/10.1016/j.cor.2024.106919.

Su, J., Huang, H., Li, G., Li, X., & Hao, Z. (2023). Self-organizing neural scheduler for the flexible job shop problem with periodic maintenance and mandatory outsourcing constraints. *IEEE Transactions on Cybernetics*, *53*(9), 5533–5544.

Sun, L., Lin, L., Gen, M., & Li, H. (2019). A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling. *IEEE Transactions on Fuzzy Systems*, *27*(5), 1008–1022.

Sun, J., Zhang, G., Lu, J., & Zhang, W. (2021). A hybrid many-objective evolutionary algorithm for flexible job-shop scheduling problem with transportation and setup times. *Computers & Operations Research*, *132*.

Sun, K., Zheng, D., Song, H., Cheng, Z., Lang, X., Yuan, W., et al. (2023). Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system. *Expert Systems with Applications*, *215*, Article 119359.

Tremblet, D., Thevenin, S., & Dolgui, A. (2023). Makespan estimation in a flexible job-shop scheduling environment using machine learning. *International Journal of Production Research*, 1–17. http://dx.doi.org/10.1080/00207543.2023.2245918.

Tutumlu, B., & Saraç, T. (2023). A MIP model and a hybrid genetic algorithm for flexible job-shop scheduling problem with job-splitting. *Computers & Operations Research*, *155*.

Wang, J.-J., & Wang, L. (2020). A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *50*(5), 1805–1819.

Xu, M., Mei, Y., Zhang, F., & Zhang, M. (2023). Genetic programming with lexicase selection for large-scale dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, http://dx.doi.org/10.1109/tevc.2023.3244607, 1–1.

Zhang, J.-D., He, Z., Chan, W.-H., & Chow, C.-Y. (2023). DeepMAG: Deep reinforcement learning with multi-agent graphs for flexible job shop scheduling. *Knowledge-Based Systems*, *259*, http://dx.doi.org/10.1016/j.knosys.2022.110083.

Zhang, F., Mei, Y., Nguyen, S., Tan, K. C., & Zhang, M. (2022). Task relatedness based multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, http://dx.doi.org/10.1109/tevc.2022.3199783, 1–1.

Zhang, S., & Wang, S. (2018). Flexible assembly job-shop scheduling with sequence-dependent setup times and part sharing in a dynamic environment: Constraint programming model, mixed-integer programming model, and dispatching rules. *IEEE Transactions on Engineering Management*, *65*(3), 487–504.

Zhao, F., Di, S., & Wang, L. (2023). A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem. *IEEE Transactions on Cybernetics*, *53*(5), 3337–3350.

Zhao, L., Fan, J., Zhang, C., Shen, W., & Zhuang, J. (2023). A DRL-based reactive scheduling policy for flexible job shops with random job arrivals. *IEEE Transactions on Automation Science and Engineering*, 1–12. http://dx.doi.org/10.1109/tase.2023.3271666.