Swarm and Evolutionary Computation 88 (2024) 101605

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Fast Pareto set approximation for multi-objective flexible job shop scheduling via parallel preference-conditioned graph reinforcement learning

Chupeng Su [a,1], Cong Zhang [b,1], Chuang Wang [a], Weihong Cen [a], Gang Chen [a,*], Longhan Xie [a]

[a] *Shien-Ming Wu School of Intelligent Engineering, South China University of Technology, Guangzhou, 511442, China*
[b] *School of Computer Science and Engineering, Nanyang Technological University, Singapore*

ARTICLE INFO

Keywords:
Deep reinforcement learning
Graph neural network
Multi-objective flexible job shop scheduling problem

ABSTRACT

The Multi-Objective Flexible Job Shop Scheduling Problem (MOFJSP) is a complex challenge in manufacturing, requiring balancing multiple, often conflicting objectives. Traditional methods, such as Multi-Objective Evolutionary Algorithms (MOEA), can be time-consuming and unsuitable for real-time applications. This paper introduces a novel Graph Reinforcement Learning (GRL) approach, named Preference-Conditioned GRL, which efficiently approximates the Pareto set for MOFJSP in a parallelized manner. By decomposing the MOFJSP into distinct sub-problems based on preferences and leveraging a parallel multi-objective training algorithm, our method efficiently produces high-quality Pareto sets, significantly outperforming MOEA methods in both solution quality and speed, especially for large-scale problems. Extensive experiments demonstrate the superiority of our approach, with remarkable results on large instances, showcasing its potential for real-time scheduling in dynamic manufacturing environments. Notably, for large instances (50×20), our approach outperforms MOEA baselines with remarkably shorter computation time (less than 1% of that of MOEA baselines). The robust generalization performance across various instances also highlights the practical value of our method for decision-makers seeking optimized production resource utilization.

1. Introduction

Job Shop Scheduling Problem (JSSP) is a fundamental Combinatorial Optimization Problem (COP), which is used to model a wide range of scheduling problems prevalent in various industries, including manufacturing systems, supply chain, and transport systems [1]. A significant variant of the JSSP is the Flexible Job Shop Scheduling Problem (FJSP), which allows operations to be processed on any machine within an eligible set. This flexibility makes the FJSP more suitable for scenarios where task-resource relationships are decentralized and adaptable [2]. While the single-objective FJSP has been extensively studied, in practice, Decision-Makers (DM) often need to consider trade-offs among multiple objectives to meet specific requirements. This makes the Multi-Objective FJSP (MOFJSP) more aligned with real-world scenarios and increasingly attractive for research attention [3].

Computing optimal solutions for FJSP and JSSP is known to be NP-hard [4]. Therefore, most research on the MOFJSP has focused on meta-heuristic approaches, especially Evolutionary Algorithms (EA), due to their ability to find high-quality solutions at reasonable costs [1]. Notable success has been achieved with classical Multi-Objective Evolutionary Algorithms (MOEA) like Non-Dominated Sorting Genetic Algorithm II (NSGA-II) and Decomposition-based Multi-Objective Evolutionary Algorithm (MOEA/D) [5,6]. However, these methods become increasingly time-consuming as scheduling problem sizes grow, posing challenges for real-time scenarios such as cloud manufacturing and online food delivery [2,7].

Deep reinforcement learning (DRL) is a sub-field of machine learning that integrates reinforcement learning (RL) [8,9] with deep learning techniques. It has found widespread application across various domains, including gaming [10], robotics [11], finance [12], and cooperation research [8,9,13]. DRL has emerged as a promising approach for solving production scheduling problems, offering a novel perspective on dynamic scheduling challenges with real-time demands and uncertainties [7]. For the MOFJSP, Luo et al. [14] first proposed a DRL-based method that employs a vector state representation and Priority Dispatching Rules (PDRs) as actions. Moreover, the Hierarchical Deep Reinforcement Learning (HDRL) technique effectively balanced conflicting objectives by leveraging a high-level DRL agent responsible for objective selection and a low-level agent handling scheduling action generation. Multiple subsequent studies [14–16] have referenced this

* Correspondence to: Shien-Ming Wu School of Intelligent Engineering, South China University of Technology, Guang Zhou, China.
E-mail addresses: chupengamos@gmail.com (C. Su), gangchen@scut.edu.cn (G. Chen), melhxie@scut.edu.cn (L. Xie).

[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.swevo.2024.101605
Received 23 December 2023; Received in revised form 9 May 2024; Accepted 12 May 2024
Available online 28 May 2024

HDRL-based approach to address other complex MOFJSP variants. Despite the good optimized result obtained by the above approaches, it exhibit certain limitations. On the one hand, the effectiveness of PDRs-based action space heavily relies on the quality of the predefined PDRs set, which may miss promising ones, resulting in limited performance. On the other hand, HDRL-based approaches provide solutions that balance multiple conflicting objectives but lack the ability to approximate a Pareto set comprising a range of Pareto optimal solutions.

Graph Reinforcement Learning (GRL), which combines Graph Neural Network (GNN) and DRL, shows promise for solving the FJSP [2]. Compared to a PDRs-based action setting, GRL-based techniques offer improved exploration capabilities by extracting spatial embeddings of FJSP and directly outputting operation/machine as actions. Recent studies by Song et al. [2], and Lei et al. [17] propose GRL-based methods that represent FJSP as a disjunctive graph (DG), achieving optimization results comparable to meta-heuristics. However, while GRL holds potential for solving FJSP and its variants, a clear research gap exists in adapting GRL to address the MOFJSP effectively.

To further address the MOFJSP, it seems natural to consider combining the HDRL approach from Luo et al. [14] with GRL. However, as mentioned earlier, HDRL-based methods face challenges in generating a Pareto set for fixed MOFJSP instances, indicating a need to revisit the idea of combining GRL and HDRL for MOFJSP optimization. Therefore, there is a pressing need to develop novel GRL-based techniques that can effectively generate the Pareto set for MOFJSP instances, bridging this research gap.

To this end, this paper presents a novel end-to-end Preference-Conditioned GRL (PGRL) approach that can quickly approximate the entire Pareto set for MOFJSP without further search procedures. Specifically, the MOFJSP is decomposed into sub-problems with various preferences, each corresponding to one possible combination of objectives. A preference-conditioned scheduling model with two encoder–decoder structures is proposed for operation selection and machine assignment. To incorporate discrepancies in preferences, the preference-conditioned scheduling model exploits a hypernetwork to generate the parameters for the preference-conditioned decoder. Furthermore, we design a multi-objective training algorithm based on Evolution Strategies (ES) to train the model and a preference parallel inference algorithm to accelerate inference. Experimental results on both synthetic instances and realistic benchmarks demonstrate the effectiveness of the proposed method in efficiently computing the Pareto set for MOFJSP instances.

In addition to the above methodological novelty, the proposed method holds significant practical value. Its neural architecture exhibits a size-agnostic nature, allowing the scheduling model to efficiently solve instances of varying sizes once trained. Moreover, the trained model efficiently solves large-scale MOFJSP instances, outperforming traditional MOEA and providing superior schedules. This makes it an excellent choice for DM seeking to optimize the utilization of production resources. Our main contributions are summarized as follows:

- We propose a novel end-to-end GRL-based method that can efficiently approximate the Pareto set for MOFJSP. The proposed method allows DM to obtain any preferred trade-off solution without any search effort.
- A PGRL scheduling model is designed to solve all sub-problems of MOFJSP. The proposed model comprises two encoder–decoder components, with the decoder being preference-conditioned to enable adjustments on sub-problem preferences.
- We develop an efficient parallel ES [10] training algorithm to train the single preference-conditioned model to solve sub-problems with different preferences, and a parallel inference algorithm to increase computational speed significantly.
- Extensive experiments are conducted on synthetic instances and public benchmarks. The results indicate that our approach achieves comparable solutions to MOEA methods for small instances and outperforms them for larger ones. Additionally, the proposed method exhibits significantly faster solution speed due to our proposed parallel inference algorithm.

The remainder of this article is organized as follows. Section 2 summarizes the related work. Section 3 describes the formulation of the MOFJSP and prerequisite backgrounds. Section 4 introduces the proposed method in detail. Section 5 provides experimental results and analysis. Finally, further discussions and conclusions are presented in Sections 6 and 7, respectively.

2. Related work

This section overviews two categories of methods relevant to the underlying study. The first aspect covers previous heuristic-based studies for MOFJSP, for which we refer the readers interested to [1,4] for a detailed review. The second aspect focuses on DRL-based approaches.

2.1. Meta-heuristic approach for MOFJSP

The FJSP is a highly complex problem and has been proven NP-hard; as a result, for both the FJSP and the MOFJSP, most studies have concentrated on utilizing heuristic methods to obtain high-quality solutions within a reasonable time instead of using exact methods to find optimal solutions [1]. The methods reported for tackling the MOFJSP can be categorized into two paradigms: priori and posteriori [3].

Earlier studies have utilized the priori paradigm to tackle the MOFJSP. This approach transforms multiple objectives into a single objective by applying a weighted sum approach. Then, it employs a meta-heuristic algorithm to optimize the above single weighted-sum objectives. For instance, Li et al. [18] also introduce a hybrid tabu search algorithm to address the MOFJSP with the weighted-sum objective. However, the priori approach is limited because it relies on the DM to pre-determine weighted factors for each objective [3]. In practice, it may be difficult for the DM to express their preferences on each objective before seeing any solutions, which leads to frequent queries from the DM to look for a desirable solution. Therefore, this limitation may result in intensive labor and considerable computation time [19].

For the posteriori paradigm, a group of Pareto optimal solutions (Pareto set) is generated and asks DM to select the most preferred one from the above group. In contrast to the priori approach, the posteriori approach does not require subjective inputs from the DM. Instead, it presents a set of optimal solutions that showcase the trade-off between conflicting objectives. These advantages have recently led to an increased recognition and adoption of multi-objective optimization among researchers [3]. Deng et al. [20] proposed a bee evolutionary guiding nondominated sorting genetic algorithm II with a two-stage optimization mechanism to solve MOFJSP. Huang et al. [21] presented a teaching-and-learning-based hybrid genetic-particle swarm optimization algorithm to address the MOFJP, which contains Genetic Algorithm (GA), bi-memory, and discrete Particle Swarm Optimization (PSO) three modules. The algorithm allows GA and discrete PSO modules to search for solutions and exchange information, leading to high-quality MOFJSP solutions. Dai et al. [22] developed an enhanced GA to minimize energy consumption and makespan for MOFJSP with transportation constraints. Yazdani et al. [23] proposed a hybrid algorithm for the MOFJSP with dual-resource constraints by combining NSGA-II and Non-Dominated Ranking Genetic Algorithm (NRGA). By incorporating controlled elitism, their approach aimed to enhance the diversity of solutions. An et al. [24] proposed an improved non-dominated sorting biogeography-based optimization algorithm to address MOFJSP. They also developed a hybrid neighborhood search structure, elite storage strategy, and improved mutation operators for further performance enhancement. Caldeira and Gnanavelbabu [3] designed a multi-objective discrete Jaya algorithm for the MOFJSP. Their proposed approach integrates a neighborhood search technique, where a dynamic mutation operator and an improved crowding distance measure are employed to enhance and balance the exploitation and exploration. Soto et al. [25] introduced an exact Parallel Branch and Bound (PBB) algorithm for solving the MOFJSP. Chen et al. [26] presented a multi-objective immune algorithm combined with a Q-learning algorithm for the MOFJSP with fuzzy processing time and variable processing speeds. The proposed algorithm demonstrated promising results in extensive experiments, showing the prominence of combining deep reinforcement learning and traditional heuristic methods. Meng et al. [27] proposed a method that combines a mixed integer linear programming model and a multi-objective hybrid shuffled frog-leaping algorithm to address the MOFJSP with controllable processing times. As mentioned above, in the posteriori paradigm, numerous research studies have focused on developing handcrafted and specialized meta-heuristic algorithms to obtain high-quality solutions for the MOFJSP. However, a drawback of methods based on the posteriori paradigm is that approximating the Pareto set necessitates extensive search iterations, resulting in time-consuming processes.

In recent years, there has been an increasing trend to use RL as a component of EA/MOEA, called Reinforcement Learning-Assisted Evolutionary Algorithm (RL-EA), to improve performance [28–31],. Han et al. [32] proposed a new multi-objective differential evolutionary algorithm. It applied an adaptive reference point activation mechanism based on RL to adjust reference points dynamically, leading to better solution quality and efficiency than fixed reference points. Hu et al. [33] introduced a co-evolutionary differential evolution algorithm assisted by DRL to solve constrained optimization problems, which employs DRL to evaluate the population state and select suitable parent populations and corresponding archives for mutation. It demonstrates competitiveness and robustness compared to other state-of-the-art algorithms. Zhao et al. [34] presented a hyperheuristic approach using Q-learning to select appropriate low-level heuristics. The experiment demonstrates that Q-learning-assisted hyperheuristic outperforms other algorithms regarding efficiency and quality. Even though the RL-EA can enhance the coverage speed of EA/MOEA, its essence of massive search iterations to obtain a good solution has stayed the same, resulting in inapplicable in solving scheduling problems with high real-time requirements.

2.2. DRL method for FJSP and MOFJSP

In the past few years, there have been several advancements in the application of DRL-based methods for FJSP/MOFJSP, allowing for the rapid generation of high-quality solutions end-to-end. Table 1 provides an overview of the state-of-the-art studies and their differences. The critical issue in DRL-based scheduling methods is state representation [2]. Consequently, we have categorized recent studies into four paradigms based on the state representation method and scheduling problem type, as summarized in Table 1: (1) ***Vector-based DRL for FJSP***, (2) ***Vector-based DRL for MOFJSP***, (3) ***Graph-based DRL for FJSP***, and our method, which is (4) ***Graph-based DRL for MOFJSP***.

2.2.1. Vector-based DRL for FJSP

This paradigm involves using vectors (manually engineered features) as states. These manually engineered features are then fed into Multilayer Perceptron (MLP), and finally, the PDRs are outputted as scheduling actions. Luo [35] proposed seven generic features to represent the state of dynamic FJSP with randomly job insertions and used a Double Deep Q Network (DDQN) agent to select the most suitable scheduling action from designed composite PDRs. Han and Yang [36] introduced a modified pointer network to encode manually designed features and select the action from the PDRs set, generating real-time scheduling solutions. Liu et al. [37] presented a hierarchical and distributed DRL method with specialized state and action representations to address the dynamic FJSP. Gui et al. [38] proposed a DRL-based method for dynamic FJSP, in which single PDRs aggregate the scheduling actions, and the model is trained by Deep Deterministic Policy Gradient (DDPG).

In general, the vector-based DRL for the FJSP paradigm utilizes manually designed features as states and PDRs as actions, leading to high interpretability by human experts [42]. However, the performance of scheduling agents heavily relies on the quality of the predefined PDRs set. Moreover, using PDRs as scheduling actions only covers a subset of operations space, limiting the exploration capability and resulting in relatively less competitiveness [42].

2.2.2. Vector-based DRL for MOFJSP

Based on the vector-based DRL for the FJSP paradigm, researchers introduced the HRL method to tackle MOFJSP further. Luo et al. [14] proposed a two-hierarchy DQN scheduling framework for solving the dynamic MOFJSP with two objectives. The proposed framework contains two scheduling agents: the higher-level agent acting as a controller to determine temporary optimization goals. A low-level agent selects a scheduling action from composite PDRs set to optimize the temporary optimization goals. Luo et al. [15] presented a hierarchical multi-agent deep reinforcement learning method to handle dynamic partial-no-wait MOFJSP with new job insertions and machine breakdowns. This approach comprises an objective agent, a job agent, and a machine agent. The objective agent serves as a higher controller that periodically sets temporary objectives, while the job and machine agents act as lower actuators by selecting job and machine assignment rules to achieve these objectives. Wu et al. [16] employed a dual-layer DRL-based method for dynamic scheduling of multi-objective flexible job shop, focusing on minimizing delay time sum and makespan.

In summary, this paradigm incorporates the status representation, action space, and neural network design from the Vector-based DRL for the FJSP paradigm. As a result, it inherits the benefits of high interpretability but also faces limitations in exploration capability. To solve MOFJSP, the HRL is further utilized to trade off multiple objectives. The HRL approach involves a high-level agent selecting temporary optimized objectives while a low-level agent solves the operation sequencing and machine allocation problems. However, it is essential to note that although this HRL-based approach can output a relatively good solution for multi-objective optimization, it cannot generate a Pareto set for fixed MOFJSP instances.

2.2.3. Graph-based DRL for FJSP

The well-known Disjunctive Graph (DG), which describes each operation's features and the problem's structural information, has been applied in recent studies to represent the state. These studies have shown promising results in leveraging GNN since it can capture the inherent structure of the FJSP and is size-agnostic. Because this paradigm combines GNN and DRL, it is also called GRL. Lei et al. [17] proposed a multi-action decision-making framework that combines DG and GNN for learning operation embeddings. In addition, an MLP is employed to facilitate the passing of machine messages. Song et al. [2] designed a heterogeneous graph to represent FJSP. They also introduced a heterogeneous GNN for learning embeddings of operations and machines for decision-making. Wang et al. [40] proposed a dual-attention network comprising several operation message blocks and machine massage attention blocks. Furthermore, their method achieves better optimization performance than traditional and DRL-based methods. Zhang et al. [41] used the Proximal Policy Optimization (PPO) algorithm containing an Actor-Critic network to solve the FJSP under machine processing time uncertainty. They designed a GNN-based network that inputs a processing information matrix and outputs PDRs as action.

Without loss of generality, the graph-based DRL paradigm directly uses the pair of operations and machines as scheduling action, which leads to more competitive exploration [42]. The trained model is also size-agnostic; it can generalize to varying sizes and unseen instances, showing strong generalization ability [2,17].

Table 1
Recent studies on solving JSSP/FJSP based on end-to-end DRL method.

Work	State representation	Problem	Neural network	Action space	Dynamic events	Pros and cons
Luo (2020) [35]				PDRs	New job insertions	
Han and Yang (2021) [36]	Vector	FJSP	MLP	PDRs	None	High interpretability; Theoretically limited exploration;
Liu et al. (2022) [37]				PDRs	New job insertions	
Gui et al. (2023) [38]				PDRs	New job insertions	
Luo et al. (2021) [14]				PDRs	New job insertions	High interpretability; Theoretically limited exploration; Unable to calculate Pareto set for fixed instance;
Luo et al. (2021) [15]	Vector	MOFJSP	MLP	PDRs	New job insertions; Machine breakdown	
Wu et al. (2023) [16]				PDRs	New job insertions	
Lei et al. (2022) [17]				Operations and machines	None	
Song et al. (2022) [2]	Graph	FJSP	GNN	Operations and machines	None	Theoretically more competitive exploration; Strong generalization ability; Poor interpretability
Lei et al. (2023) [39]				Operations and machines	New job insertions	
Wang et al. (2023) [40]				Operations and machines	None	
Zhang et al. (2023) [41]				PDRs	Variable processing times	
Our method	Graph	MOFJSP	GNN	Operations and machines	None	The pros and cons remain the same as the above GRL method, while also offering the advantage of fast Pareto set computation for MOFJSP instances.

2.2.4. *Graph-based DRL for MOFJSP*

According to the above-related works, the Graph-based paradigm has achieved an excellent result in solving FJSP. Therefore, extending the graph-based method to address MOFJSP is valuable since MOFJSP is more consistent with real-world applications.

Recently, DRL technologies have been increasingly applied to address Multi-Objective Combinatorial Optimization (MOCO) problems such as Multi-Objective Traveling Salesman Problem (MOTSP) [19,43–48], Multi-Objective Vehicle Routing Problem (MOVRP) [19,43,48], and Multi-Objective Knapsack Problem (MOKP) [43], yielding remarkable achievements in efficiently obtaining the Pareto set. However, there is a notable lack of research on applying DRL to solve multi-objective scheduling problems, particularly MOFJSP. Existing DRL methods for MOCO cannot be directly adapted to MOFJSP due to *incompatible model architectures* and *significant computational burdens*. The single encoder–decoder architecture used in current DRL methods for MOCO is insufficient to handle the composite nature of MOFJSP, which consists of operation sequencing and machine assignment problems. Furthermore, the dynamic nature of scheduling problems necessitates node embedding at each decision step, increasing the computational burden during training and inference. Consequently, the advantage of DRL in terms of solving speed is diminished when applied to MOFJSP.

To our knowledge, no studies exist to solve the MOFJSP via graph-based end-to-end DRL method that can efficiently output the Pareto set. This work proposes an end-to-end GRL-based method to quickly approximate the Pareto set of MOFJSP. The proposed PGRL scheduling model contains two encoder–decoder architectures to overcome the challenges posed by *incompatible model architecture*. Additionally, we proposed a parallel training and inference algorithm to alleviate the *significant computational burden*.

3. Preliminaries

This section introduces the necessary background knowledge of the proposed method, including the definition of the Multi-objective Combinatorial Optimization Problem (MOCO), the formulation for MOFJSP and its DG representations, and the decomposition strategy with preference-based scalarization.

3.1. *Multi-objective combinatorial optimization problem*

A MOCO problem can be defined as follows:

$$\min_{x\in\mathcal{X}} \boldsymbol{f}(x) = (f_1(x), f_2(x), \dots, f_m(x)) \tag{1}$$

where $\mathcal{X}$ is a discrete search space, and $\boldsymbol{f}(x)$ is an objective vector consisting of m objectives. MOCO is reduced to a Single-Objective Combinatorial Optimization (SOCO) when $m = 1$. Because the individual objectives conflict with each other, no single solution can optimize all objectives simultaneously. As a result, practitioners aim to identify Pareto optimal solutions (Pareto set) instead, which is defined as follows:

Definition 1 (*Pareto Dominance*)**.** A solution $x^1 \in \mathcal{X}$ dominates another solution $x^2 \in \mathcal{X}$ (denoted as $x^1 \prec x^2$) if and only if $f_i(x^1) \leq f_i(x^2)$ for all $i \in 1,\dots,m$ and $\exists j \in \{1,\dots,m\}$, $f_j(x^1) < f_j(x^2)$.

Definition 2 (*Pareto Optimality*)**.** A solution $x \in \mathcal{X}$ is Pareto-optimal if it is not dominated by any other solution, i.e., $\nexists \boldsymbol{x}' \in \mathcal{X}$ such that $x' \prec x^*$.

Definition 3 (*Pareto Set/front*)**.** Solving an MOCO problem aims to obtain a Pareto set consisting of all Pareto optimal solutions: $\mathcal{P} = \{x \in \mathcal{X} \mid \nexists x' \in \mathcal{X} : x' \prec x\}$. The Pareto front $\mathcal{PF} = \{\boldsymbol{f}(x) | x \in \mathcal{P}\}$ represents the objective values of the Pareto set, with each $\boldsymbol{f}(x)$ considered as a point in the objective space.

3.2. *MOFJSP formulation and disjunctive graph*

3.2.1. *MOFJSP formulation*

A standard (single-objective) FJSP involves n jobs and m machines, constituting two sets $\mathcal{J}$ and $\mathcal{M}$, respectively. Each job $J_i \in \mathcal{J}$ comprises n_i operations to be completed in a predefined order (the precedent constraint) $O_{i1} \to \cdots \to O_{ij} \to \cdots \to O_{in_i}$. Each operation $O_{ij} \in J_i$ can be executed on any machine M_k in a given compatible machine set $\mathcal{M}_{ij} \subseteq \mathcal{M}$ with a processing time of p_{ijk}. Additionally, any machine can only process one operation at a time. Solving FJSP is to determine each operation O_{ij}'s suitable machine and its start time to optimize objectives such as makespan or tardiness.

Expanding FJSP to MOFJSP involves optimizing multiple objectives. In this study, we focus on three objectives that are widely used by previous studies [1]:

$$\min_{x\in\mathcal{X}} \boldsymbol{f}(x) = (C_{max}, W_T, W_c), \tag{2}$$

$$\text{The makespan: } C_{max} = \max_{i,j} C_{ij}, \tag{3}$$

$$\text{The total workload: } W_T = \sum_{k=1}^{m} \mathrm{w}_k, \tag{4}$$

$$\text{The critical load: } W_c = \max_{1\le k\le m} \mathrm{w}_k, \tag{5}$$

where C_{ij} is the completion time of operation O_{ij}, w_k is the workload of machine M_k.

3.2.2. Disjunctive graph

An FJSP instance can be naturally represented by a DG $\mathcal{G} = (\mathcal{O}, \mathcal{C}, \mathcal{D})$. $\mathcal{O} = \{O_{ij}|\forall i,j\} \cup \{S,T\}$ is the set of nodes of operations, including two artificial nodes S and T (with zero processing time) denoting the start and terminal of a schedule. $\mathcal{C}$ is the set of directed conjunctive arcs indicating the precedent constraints. $\mathcal{D} = \bigcup_k \mathcal{D}_k$ represents the set of undirected disjunctive arcs. $\mathcal{D}_k$ is a clique that connects a series of operations that can be processed on machine M_k. To illustrate, Fig. 1(a) is a DG representation of an FJSP instance, and Fig. 1(c) displays the corresponding DG representation of some feasible solutions, where the direction of disjunctive arcs in each machine clique are determined.

3.3. Decomposition strategy and preference-based scalarization

3.3.1. Decomposition strategy

The decomposition strategy is a mainstream approach to solving multi-objective optimization problems [6]. It is simple, effective, and has promoted much MOEA-based research, such as MOEA/D [6] and its variants [49]. The decomposition strategy breaks down the multi-objective optimization problem into multiple single-objective sub-problems, where each sub-problem is a single-objective optimization problem. The Pareto set can thus be obtained by solving all possible combinations of sub-problems. In this paper, we employ the decomposition strategy [6] as the basic framework to solve the MOFJSP.

3.3.2. Preference-based scalarization

The preference-based scalarization is the widely adopted method for decomposing a multi-objective optimization problem into smaller ones [50]. Commonly used preference scalarization methods are the Weighted-Sum (WS) [51], the Weighted-Tchebycheff [52], and the Penalty-based Boundary Intersection approach [53]. Formally, for a MOCO with m objectives, the preference vector of the ith sub-problem is λ^i given $\lambda^i_j \ge 0$ and $\sum_{j=1}^{m} \lambda^i_j$. This study uses the WS, the simplest yet effective decomposition approach, [51] to decompose the MOFJSP. It uses the linear scalarization of m objectives, which defines the aggregation function to minimize the sub-problem associated with λ^i as:

$$\min_{x\in\mathcal{X}} g_{\mathrm{ws}}(x|\lambda^i_j) = \sum_{j=1}^{m} \lambda^i_j f_j(x) \tag{6}$$

Since g_{ws} is a scalar objective function, the decomposed sub-problem is the single-objective optimization problem. Using the WS-based Preference-based scalarization, we transit the MOCO (i.e., the objective function is multi-dimensional) into a set of SOCO (i.e., the objective function is one-dimensional) that can approximate the original problem.

4. Method

The overall framework of our method is depicted in Fig. 1. The core of the framework is the PGRL scheduling model. The input comprises MOFJSP instances represented as a DG, along with preferences λ traversing from the Preference set $\mathcal{F}$, and the output is the Pareto Set consisting of Pareto optimal solutions corresponding to several problems. The procedure for approximating the Pareto set using the PGRL model is outlined as follows:

***Step 1*:** The whole process begins by loading the trained PGRL model and initializing the preference set using the structural weight generate assignment method [54].

***Step 2*:** Select a preference λ^i from the preference set $\mathcal{F}$. The key to generating the Pareto set is to find the solution for different preferences. To this end, we make the PGRL model (with learnable parameters θ) aware of different preferences by employing a parameter generation module that generates the parameters of the PGRL model conditioned on the preference, i.e., $\theta(\lambda^i)$.

***Step 3*:** Exploit the model $\theta(\lambda^i)$ to infer the solution of the MOFJSP sub-problem. This process yields the Pareto optimal solution P_i for the ith sub-problem of MOFJSP with preference λ^i. Update the Pareto set: $\mathcal{P} \leftarrow \mathcal{P} \cup P_i$.

Step 4*:** Check if the process traverses all the preferences, which also means all the sub-problems are solved. If Yes, output the Pareto set $\mathcal{P}$, else return to ***step 2.

The preference set $\mathcal{F}$ is generated using the structural weight generate assignment method [54]. This method creates a set of preferences in the form of WS, decomposing the MOFJSP into various structural sub-problems. We kindly refer readers to Appendix A for detailed information. Fig. 2 also provides an example featuring 15 preferences.

The following subsection will present the Markov Decision Process (MDP) formulation (Section 4.1), policy parameterization (Section 4.2), parallel training (Section 4.3), and parallel inference algorithms (Section 4.4). The MDP formulation section outlines the input state, output action, and feedback reward of the PGRL model. Policy parameterization details the model composition, decision-making process, and how the model adapts to different sub-problems based on their preferences. The parallel training algorithm section explains the training procedure of the PGRL model based on OpenAI-ES. During the inference phase, the Parallel inference algorithm demonstrates how the PGRL model acquires the Pareto set in a parallel manner, serving as a parallelized version of the process for obtaining the Pareto set (***Step 1–4***).

4.1. MDP formulation of MOFJSP

As mentioned above, the proposed model must solve multiple MOFJSP sub-problems, a regular FJSP instance with difference preference vector λ^i. Solving the FJSP instance via DRL is to transform it into a sequential decision-making problem. Specifically, the DRL agent selects an operation at each decision step t and assigns it to one of its legal machines by considering the current state s_t of the FJSP environment. Then, the environment transits to the next decision step $t+1$ until all operations are assigned. The schedule can be computed given the complete assignment. Therefore, in our case, it involves making multiple actions (operation selection and machine assignment). The action space in this MDP formulation is two-dimensional. This type of MDP with a multi-dimensional action space is referred to as multiple MDPs [55]. We present the formal definition as follows:

4.1.1. State representation

At each decision step t, the state is represented as $s_t = (s^o_t, s^m_t)$, where s^o_t denotes the state of the operation selection model and s^m_t represents the state of the machine selection model. The state s^o_t is defined as a DG $\mathcal{G}(t) = (\mathcal{O}(t), \mathcal{C}, \mathcal{D}_u(t))$, where $\mathcal{D}_u(t)$ represents the assigned directions of disjunctive arcs that determine the processing sequence of operations on the same machine. Here, $\mathcal{O}(t)$ denotes the set of all operation nodes at decision step t. The raw features of each operation node are as follows:

- Estimated lower bound of completion time $LB_t(O_{ij})$.
- Scheduling status: binary value representing whether O_{ij} has scheduled (1) or not (0) till step t.

There is no graph structure in the machine's state information, which is applicable for solving the FJSP [17]. Consequently, s^t_m is a graph where each node encapsulates the primary features of a machine, and no node is connected to another through arcs. The raw features of each machine node are defined as follows:

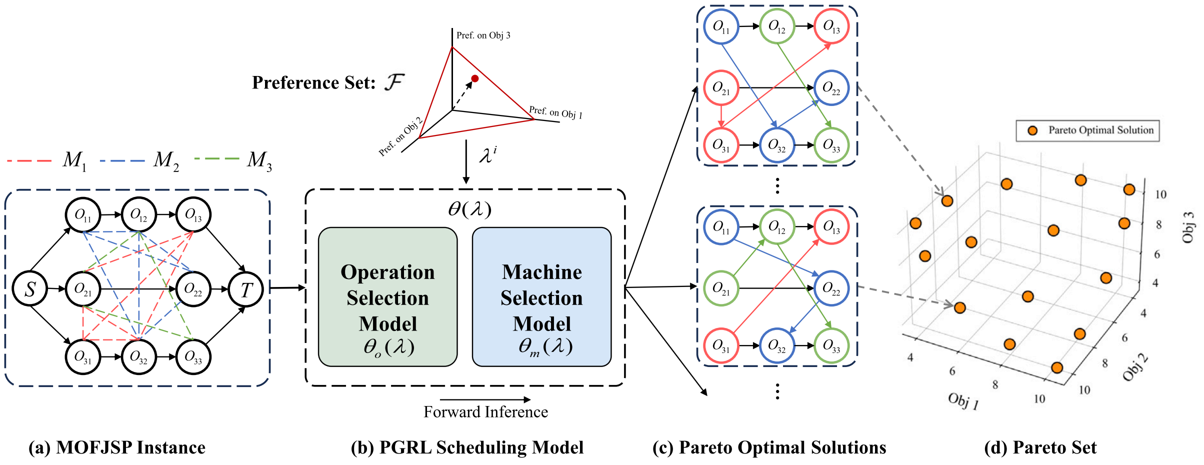


Fig. 1. The framework of the proposed method. Preferences are assigned to the PGRL model to facilitate its adaptation in solving various sub-problems of instances. Subsequently, the model ingests the problem instance as its input. It directly approximates Pareto optimal solutions through swift forward inferences, each offering distinct trade-offs.

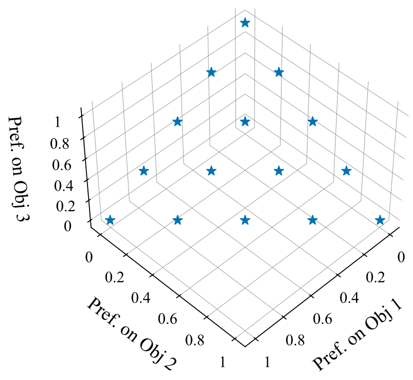


Fig. 2. An example of preferences set $\mathcal{F}$. Fifteen preferences generated by structural weight generate assignment method.

- The processing time p_{ijk} of selected operation O_{ij} for machine M_k
- Available time: the time when the machine M_k has completed all its assigned operations and is free.
- Received workload: the workload the machine M_k has been assigned.

4.1.2. Action

The action a_t consists of an operation select action, $a_o \in A_o$, and a machine assign action, $a_m \in A_m$. Thus, the action space can be expressed as $A = A_o \times A_m$. Here, A_o represents the set of eligible operations at decision step t, i.e., each job's current ready-to-process operation O_{ij}. Meanwhile, A_m refers to the compatible machines M_k for the selected operation a_o.

4.1.3. State transition

Once the agent passes action $a_t = [a_o^t, a_m^t]$ to the environment, it transits to the next state s_{t+1}. Specifically, the direction of disjunctive arcs for action/operation a_o^t is determined, and the features for all operation nodes O_{ij} and machine nodes M_k are updated accordingly. After all operations are scheduled, the environment transits to terminal state s_T, as shown in Fig. 1(c).

4.1.4. Reward

This study utilizes ES for model training. It approximates the gradient by applying a fitness function, which can be seen as the reward for an episode. In the training section (4.3), the fitness function (reward) will be introduced to facilitate describing the multi-objective ES training algorithm.

4.1.5. Policy

The DRL policy defines a probability distribution over the action set A for each state s_t. Our model $\theta(\lambda) = \{\theta_o(\lambda), \theta_m(\lambda)\}$ defines a preference-conditioned stochastic policy $\pi_{\theta(\lambda)}$, which consists of two policies: $\pi_{\theta_o(\lambda)}$ for operation selection and $\pi_{\theta_m(\lambda)}$ for machine selection. At each decision step s_t, policy $\pi_{\theta_o(\lambda)}$ selects a ready operation as action a_o, and then $\pi_{\theta_m(\lambda)}$ assign a compatible machine for above selected operation a_o as action a_m.

4.2. Parameterizing the policy

The overall architecture of our preference-conditioned scheduling model is based on a multi-pointer graph network [17], as shown in Fig. 3. Moreover, unlike study [17] work for single-objective FJSP, our model is preference-conditioned, which is suitable to solve different MOFJSP sub-problems. It consists of two encoder–decoder structures that parameterize the operation selection policy $\pi_{\theta_o(\lambda)}$ and the machine selection policy $\pi_{\theta_m(\lambda)}$, respectively. At each decision step t, the operation selection policy $\pi_{\theta_o(\lambda)}$ selects the operation first, followed by the machine selection policy $\pi_{\theta_m(\lambda)}$ assigning a compatible machine to the operation selected above.

For MOFJSP, the preference-agnostic encoders are capable of transferring instances into generalized embeddings (e.g., embeddings for all operations/machines), which the preference-conditioned decoder can use to solve all sub-problems. In our model, only the decoders are conditioned on the preference λ:

$$\theta_o(\lambda) = [\theta_{o\_enc}, \theta_{o\_dec}(\lambda)] \tag{7}$$

$$\theta_m(\lambda) = [\theta_{m\_enc}, \theta_{m\_dec}(\lambda)] \tag{8}$$

Next, we present the details of the encoder and preference-conditioned decoder.

4.2.1. Operation selection encoder

The state s_t^o input into the operation selection encoder θ_{o_enc} is a dynamic DG $\mathcal{G}(t)$ that evolves based on the environmental conditions at decision step t. The graph $\mathcal{G}(t) = (\mathcal{O}(t), \mathcal{C}, \mathcal{D}_u(t))$ captures both structural and dynamic information, including precedence constraints of FJSP and real-time messages of each operation node. Effectively extracting crucial information from this DG is crucial for scheduling decisions. This work applies the Graph Isomorphic Network (GIN) [56], which has

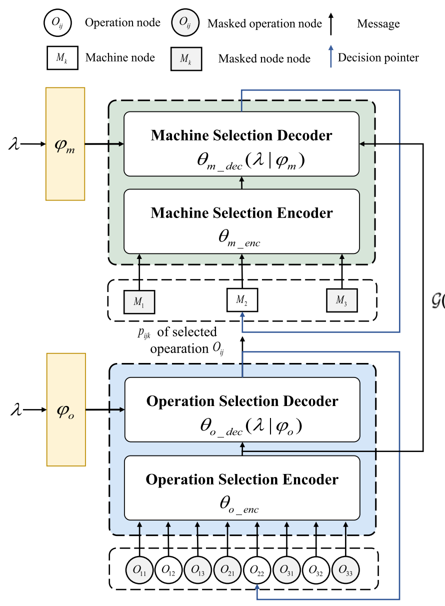

O_{ij} Operation node
O_{ij} Masked operation node
Message
M_k Machine node
M_k Masked node node
Decision pointer
λ
φ_m
Machine Selection Decoder
$\theta_{m\_dec}(\lambda \mid \varphi_m)$
Machine Selection Encoder
θ_{m_enc}
M_1
M_2
M_3
$\mathcal{G}(t)$
p_{ijk} of selected opearation O_{ij}
λ
φ_o
Operation Selection Decoder
$\theta_{o\_dec}(\lambda \mid \varphi_o)$
Operation Selection Encoder
θ_{o_enc}
O_{11} O_{12} O_{13} O_{21} O_{22} O_{31} O_{32} O_{33}

Fig. 3. The proposed PGRL model architecture. The model selects the next processing operation O_{ij} from the feasible operations first, then its processing machine M_k. The raw features in the machine selection encoder's input include the processing time p_{ijk} of selected operation O_{ij} for machine M_k. Note that the machine selection decoder takes as input the embeddings of the complete DG $h_{\mathcal{G}}(t)$, computed by the operation selection encoder.

shown solid discriminative power, to extract embeddings. GIN performs iterations of updates to generate p-dimensional embeddings for each node $v(v \in V)$ in a graph $\mathcal{G} = (V, E)$. The update at iteration l can be summarized as follows:

$$h_v^{(l)} = MLP_{\theta_l}^{(l)}\left(\left(1+\epsilon^{(l)}\right)\cdot h_v^{(l-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(l-1)}\right) \tag{9}$$

where $h_v^{(l)}$ indicates the embeddings of node v at iteration l, and h_v^0 denotes the raw features of each node v. $MLP_{\theta_l}^l$ is an Multilayer Perceptron (MLP) with parameter θ_l for iteration l. It follows by batch normalization [57]. Here, ϵ^l is a arbitrary number, both parameters θ_l and arbitrary number ϵ^l can be learned. Besides, $\mathcal{N}(v)$ refers to the set of neighbor nodes of node v.

After L rounds of updates, the operation selection encoder produces the graph pooling vector $h_{\mathcal{G}} \in \mathbb{R}^p$ as the representation for the entire graph $\mathcal{G}$. This vector is obtained by averaging the node embeddings: $h_{\mathcal{G}} = 1/|V| \sum_{v\in V} h_v^L$.

The aforementioned network, GIN, is utilized to extract embeddings of the DG of FJSP. At the decision step t, given the state s_t^o: $\mathcal{G}(t) = (\mathcal{O}(t), \mathcal{C}, \mathcal{D}_u(t))$, it outputs the embeddings of each operation node $h_{O_{ij}}^{(L)}(t)$ and the embeddings of the entire DG $h_{\mathcal{G}}(t)$.

4.2.2. Machine selection encoder

Since each machine node M_k encapsulates the primary features without any structural connections between them, so we utilize a simple fully connected layer for encoding the state s_t^m, resulting in the embedding vector $h_{M_k}(t)$ for each machine node and the mean pooling vector: $u(t) = 1/|\mathcal{M}| \sum_{k=1}^{|\mathcal{M}|} h_{M_k}(t)$.

4.2.3. Preference-based decoder

There are two key problems concerning the design of the preference-based decoder.

The first is how to generate the preference-conditioned parameters $\theta_{o\_dec}(\lambda)$ and $\theta_{m\_dec}(\lambda)$.

Inspired by studies [43,58], this work adopts hypernetwork [59] to generate the decoder parameters conditioned on the input λ. As shown in Fig. 3, two hypernetworks, φ_o and φ_m, are responsible for generating operation and machine selection decoder parameters, respectively. Additionally, the compression approach in the study [59] is applied to control the model size. The procedure for generating the preference-conditioned decoder based on the hypernetwork is as follows:

Initially, the MLP takes the input preference λ and produces hidden embeddings as output.

$$e_h(\lambda) = \mathbf{MLP}(\lambda|\varphi_h) \quad (h \in \{o, m\}) \tag{10}$$

$$e_h(\lambda) = [e_h^{x_1}, e_h^{y_1}, \dots, e_h^{x_l}, e_h^{y_l}, \dots, e_h^{x_L}, e_h^{y_L}] \tag{11}$$

where $e_h^{x_1}$, $e_h^{y_1}$ is the hidden embedding with the same dimension, which is used to generate the parameters for the decoder l layer. Next, the hidden embeddings are mapped to the decoder parameters via linear projection.

$$W_{\theta_{h\_dec}}^l = W^{x_l} e_h^{x_l} + b^{x_l} \quad (h \in \{o, m\}) \tag{12}$$

$$b_{\theta_{h\_dec}}^l = W^{y_l} e_h^{y_l} + b^{y_l} \quad (h \in \{o, m\}) \tag{13}$$

where $W_{\theta_{h\_dec}}^l$ and $b_{\theta_{h\_dec}}^l$ represent the weights and biases, respectively, of layer l in the preference-conditioned decoder. In other words, θ_{h_dec} is defined as the set $\{W_{\theta_{h\_dec}}^l, b_{\theta_{h\_dec}}^l |\forall l\}$. The hypernetwork φ_h consists of $\mathbf{MLP}(\lambda|\varphi_h)$ and the trainable parameters $\{W^{x_l}, b^{x_l}, W^{y_l}, W^{y_l} |\forall l\}$.

The remaining key problem is how preference-conditioned decoders work for decision-making.

Both the operation selection decoder $\theta_{o\_dec}(\lambda)$ and machine selection decoder $\theta_{m\_dec}(\lambda)$ employ the same structure, which is an MLP. They take the embeddings of operation or machine as input and output the scheduling selection action $a_t = \{a_t^o, a_o^m\}$. The decision-making process follows a similar approach as described in [17]:

First, the two decoders are applied to calculate the score for each operation/machine node as follows:

$$scr(O_{ij}(t)) = \mathbf{MLP}_{\theta_{o\_dec(\lambda)}}([h_{O_{ij}}^{(L)}(t), h_{\mathcal{G}}(t)]) \tag{14}$$

$$scr(M_k(t)) = \mathbf{MLP}_{\theta_{m\_dec(\lambda)}}([h_{M_k}(t), h_{\mathcal{G}}(t), u(t)]) \tag{15}$$

where $[\cdot, \cdot]$ is the vector concatenation operator. Then, they all use the softmax function to calculate the operation/machine selection probability distribution, respectively. Finally, selecting action a_t can be determined through sampling or a greedy approach. The proposed model is trained using ES, requiring a greedy action selection during training and testing. An action mask [17] is also applied to block illegal actions.

4.3. Multi-objective training via evolution strategies

We propose a multi-objective training algorithm for the proposed PGRL model based on OpenAI-ES [10], a highly parallelizable and efficient reinforcement learning approach. The ES-based training algorithm consists of two phases: (1) randomly perturbing the policy parameters and evaluating the perturbed policy, and (2) aggregating the evaluations (using the fitness function) from all episodes to compute stochastic gradient estimates and update the policy. The fitness function for evaluating the proposed model $\theta(\lambda^i)$, considering a specific preference λ^i, can be derived from (6) after solving an MOFJSP sub-problem:

$$F(\theta(\lambda^i)) = -(\lambda_1^i C_{max} + \lambda_2^i W_T + \lambda_3^i W_c). \tag{16}$$

Algorithm 1 presents the ES-based parallel multi-objective training algorithm (For clearance, we ignore λ during parameter updates and data transmissions between CPU workers). The CPU workers are responsible for evaluating the policy after perturbation by running episodes, where the master gathers empirical data from the CPU workers, updates the model, and communicates it back to the CPU workers.

During each training step, the master randomly samples preference weights λ^i from the Dirichlet distribution Λ. The master sends the model and λ^i to the CPU workers (line 5). Each CPU worker follows these steps: (1) generating a random FJSP instance s (line 8), (2) perturbing the parameters of the received model (lines 9–11), (3) assigning the preference weights to the perturbed model, i.e., generating decoder parameters conditioned on specific λ^i (line 12), and (4) applies the model to solve the instance according to Algorithm 2, and finally, evaluating the performance using (16) (lines 13–14). This process is repeated I times for all CPU workers, and the perturbed random seeds and evaluations (fitness function) are sent back to the master. Subsequently, the master updates the model based on the received data (line 19).

The training preferences sample from Dirichlet distribution [48] instead of a random sample can further enhance the learning of boundary solutions [48]. The detailed details of the Dirichlet distribution are provided in Fig. 7.

By sequentially training the PGRL model to optimize sub-problems with possible preferences, as outlined above, it can learn to accommodate all preferences effectively.

Algorithm 1: Training Procedure with Parallelized ES

```
Input: Learning rate $\alpha$, noise standard deviation $\sigma$, preference
       distribution $\Lambda$, FJSP instances distribution $\mathcal{S}$, number of
       training steps $U$, number of CPU workers $C$, number episodes
       of per CPU worker on one training step $I$
1  Initialize the master model parameters $\theta = \{\theta_o, \theta_m\}$
2  Initialize each CPU worker model parameters $\theta_c = \{\theta_{c,o}, \theta_{c,m}\}$ for
     $c = 1, \ldots, C$
3  for $i = 1$ to $U$ do
4      $\lambda^i \leftarrow$ SamplePreference($\Lambda$)
5      Send $\theta$ and $\lambda^i$ to each CPU worker
6      for each CPU worker $c = 1$ to $C$ do
7          for $j = 1$ to $I$ do
8              Sample a FJSP instance $s$ from $\mathcal{S}$
9              Randomly generate a random seed $\beta_j$
10             Sample $\epsilon_j \sim \mathcal{N}(0, 1)$ with random seed $\beta_j$
11             $\theta_c \leftarrow \theta + \sigma\epsilon_j$
12             Assign $\lambda^i$ to $\theta_c(\lambda)$ to get $\theta_c(\lambda^i)$
13             $P \leftarrow$ Solve-FJSP($\theta_c(\lambda^i)$, $s$) according to Algorithm 2
14             The fitness function $F_j(\theta_c(\lambda^i))$ is calculated according to
                 Eq. (16).
15         end
16         Send $F_j$ and $\beta_j$ for $j = 1, \ldots, I$ to the master
17     end
18     Reconstruct all perturbations $\epsilon_k$ using random seed $\beta_k$ for
         $k = 1, \ldots, C \times I$
19     Set $\theta \leftarrow \theta + \alpha \frac{1}{n\sigma} \sum_{k=1}^{C \times I} F_k \epsilon_k$
20 end
Output: The model parameter $\theta$
```

Algorithm 2: Solve-FJSP Algorithm via $\theta(\lambda^i)$

```
Input: preference conditioned model $\theta(\lambda^i)$ with specify $\lambda^i$ and FJSP
       instance $s$
1  Initialize $s_t$ based on sampled FJSP instance ($s_t = \{s_t^o, s_t^m\}$)
2  while $s_t$ is not terminal do
3      Greedily pick $a_t^o$ based on $\pi_{\theta_o(\lambda^i)}(\cdot|s_t^o)$
4      Greedily pick $a_t^m$ based on $\pi_{\theta_m(\lambda^i)}(\cdot|s_t^m)$
5      The environment receive $a_t = \{a_t^o, a_t^m\}$ transit to $s_{t+1}$
6      $s_t \leftarrow s_{t+1}$
7  end
8  Obtain Pareto optimal solution $P = (C_{max}, W_T, W_c)$
Output: $P$
```

4.4. Preference parallel inference

We observed that solving each MOFJSP sub-problem using the proposed model is independent of one another. This observation motivates using a parallel approach to accelerate the inference process. We proposed the PGRL preference parallel inference procedure (Algorithm 3), which exhibits high parallelism and negligible communication overhead. Initially, the master sends the FJSP instances and preference weights to the CPU workers (line 2), which then solve their respective MOFJSP sub-problems (lines 11–13) and transmit the solutions back to the master (line 14). The preference set $\mathcal{F}$ is generated using a structured weight assignment approach [54].

Algorithm 3: PGRL Preference Parallel Inference

```
Input: Trained model parameter $\theta$, FJSP instance $s$, preference set $\mathcal{F}$,
       number of CPU worker $C$
1  Initialize Pareto set $\mathcal{P} = \emptyset$
2  Send model parameter $\theta$ and FJSP instance $s$ to each CPU worker
3  while $\mathcal{F} \neq \emptyset$ do
4      if $|\mathcal{F}| \geq C$ then
5          Take $C$ $\lambda$ from the $\mathcal{D}$, send each of them to individual
             workers
6      end
7      else
8          Take remaining $\lambda$ from the $\mathcal{D}$, send each of them to
             individual workers
9      end
10     for each CPU worker do
11         Assign $\lambda^i$ to $\theta(\lambda)$ to get $\theta(\lambda^i)$
12         Initialize $s_t$ based on FJSP instance $s$ ($s_t = \{s_t^o, s_t^m\}$)
13         $P \leftarrow$ Solve-FJSP($\theta_c(\lambda^i)$, $s$) according to Algorithm 2
14         Send Pareto optimal solution $P$ to master
15     end
16     $\mathcal{P} = \mathcal{P} \cup \{P\}$ // Collect the Pareto optimal solutions sent by the CPU
         workers and put them into the Pareto set.
17 end
Output: Pareto set $\mathcal{P}$
```

5. Experiments

In this section, we conduct extensive experiments to evaluate the proposed PGRL method on multiple benchmarks. We consider several baselines, including various conventional, widely-used, efficient MOEA algorithms and state-of-the-art ones for MOFJSP. First, we introduce the experimental setup. Subsequently, we compare the in-distribution performance, generalization performance across different problem sizes and distributions, and computational efficiency of different approaches using synthetically generated instances and public FJSP benchmarks, respectively. Finally, we empirically analyze the time complexity of the proposed method.

5.1. Experiment settings

5.1.1. Training and testing instances

The proposed method is trained and tested on synthetic instances of various sizes. The instance generation method is similar to the approach employed in the study [17]. Specifically, the value of p_{ijk} are randomly sampled from a uniform distribution $U[1, 99]$, except when set to zero to indicate that machine M_k cannot process operation O_{ij}. Additionally, the number of operations for each job is set equal to the number of machines for simplicity. For example, consider a 3×3 FJSP/MOFJSP instance represented in Table 2. This instance involves three jobs, where each job consists of three operations. The processing times for each operation are provided, and a value of 0 indicates that a particular machine cannot perform a specific operation.

The PGRL model, trained on small instances, is tested on large-scale synthetic unseen instances to assess its generalization capability across various problem sizes. In addition to randomly generated instances, we evaluate the performance of the proposed method using various well-known published FJSP benchmarks. These benchmarks include three instances from Kacem et al. [60], ten mk instances from Brandimarte [61], ten sfjs instances from Fattahi et al. [62], two mfjs instances from Fattahi et al. [62], a dataset labeled as “la (rdata)” with 40 instances from Hurink et al. [63], and a dataset labeled as “la (vdata)” with 40 instances from Hurink et al. [63]. These benchmarks encompass problem instances of diverse sizes, ranging from 2×2 to

Table 2
An example of 3 × 3 FJSP/MOFJSP instance

p_{ijk}	J_1			J_2			J_3		
	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{31}	O_{32}	O_{33}
M_1	0	68	0	0	46	0	0	60	47
M_2	47	24	0	46	0	56	33	0	0
M_3	0	56	83	0	67	36	45	92	0

30 × 10, and are generated from distributions different from those used during training. Testing on these benchmarks further validates the proposed method's efficacy when confronted with out-of-distribution scenarios.

5.1.2. Parameter configuration

This study sets the number of GIN layers of the operation selection encoder θ_{o_enc} to 2. In each GIN layer, $MLP^{(l)}_{\theta_l}$ contains 2 hidden layers with hidden dimensions 128. Both the operation action selection decoder $\theta_{o\_dec}(\lambda)$ and the machine action selection decoder $\theta_{m\_dec}(\lambda)$ have two hidden layers with hidden dimensions 128. These decoders are preference-conditioned, and their weights and biases of each layer are generated according to Eq. (10) and Eq. (12), respectively. The MLP used to generate the hidden embeddings in the hypernetwork has two hidden layers with hidden dimension 256. The dimensions of the hidden embeddings $e_h^{x_1}$, $e_h^{y_1}$ are all set to 2.

For the training process, we have set the learning rate α to 1×10^{-3} and the noise standard deviation σ to 0.01. The number of training steps U is chosen as 200, while the number of CPU workers C is set to 10, and each CPU worker performs 10 episodes per training step $I = 10$. Detailed discussions on the setting of the above hyperparameters can be found in Appendix C. Additionally, we have employed a scaling parameter of 0.2 for the Dickey Distribution, which is used to generate sample training preference weights. This value has shown promising performance in various multi-objective optimizations [58].

During the inference phase, we employ 3 CPU workers, taking into account the GPU memory limitations of our experimental setup. We set the control parameter p to 4 for the structured weight assignment approach. This configuration leads to a preference set with 15 preferences, striking a balance between the solutions' quality and the inference time's efficiency. The proposed method is implemented in PyTorch, running on a workstation equipped with an Intel i9-7940X CPU and three Nvidia Titan Xp GPUs.

5.1.3. Baselines and performance evaluation metrics

Baselines. We consider four well-known MOEA-based algorithms, including NSGA-II [5], MOEA/D [6], NSGA-III [64] and RVEA [65]. For a fair comparison, the baseline MOEA algorithms are implemented in the same environment as the proposed method to verify that a learned PGRL model is superior to a hand-crafted MOEA competitor. We also compare with the recent state-of-the-art approaches for solving MOFJSP, namely, INSBBO [24], PBB [25] TL-HGAPSO [21], and BEG-NSGA-II [20]. The details of the above baselines can be found in Appendix E.

Performance evaluation metrics. In this paper, we adopt two indicators to assess the performance of each method.

(1) Hypervolume (HV) **metric**: HV [66] is a commonly used metric to evaluate the performance of multi-objective methods. Let $\mathcal{P}$ be the set of Pareto optimal solutions obtained by an algorithm, and the HV metric is defined as:

$$HV(\mathcal{P}) = \mathrm{VOL}(\bigcup_{x\in\mathcal{P}} [f_1(x), r_1^*] \times \cdots \times [f_m(x), r_m^*]) \tag{17}$$

where VOL(·) represents the Lebesgue measure, $\boldsymbol{f} = (f_1, \ldots, f_m)$ denotes an attained objective vector, $\boldsymbol{r}^* = (r_1^*, \ldots, r_m^*)$ and is a reference objective vector. In our experiments, $\boldsymbol{r}^*$ is set to a relatively large value compared to the objective vector obtained by all methods. The larger the hypervolume, the better the solution set tends to be, which means the algorithm achieves a better approximation to the Pareto front.
We also utilize Tukey's Honestly Significant Difference (HSD) Test with a 0.05 significance level to analyze the differences between the HV results produced by the algorithms under comparison. The symbols "+/−/=" indicate whether the results of the PGRL are superior to, inferior to, or equal to those of the compared algorithm, respectively.

(2) Gap **metric**: Gap measures the ratio of the HV difference relative to our method.

$$Gap = \frac{HV_c - HV_{ours}}{HV_{ours}} \tag{18}$$

A positive value of Gap indicates that the corresponding method is better than the proposed method, while the opposite value is worse.

5.2. Performance on synthetic instances

In this section, we report and analyze the performance of PGRL on synthetic datasets. The training curves (Fig. 9 in Appendix D) show that the proposed method is stable during training. It converges on all three training sizes. Remarkably, the training process demonstrates that the DRL agent can rely solely on its own experiences to learn to acquire a high-quality scheduling policy to solve MOFJSP without the need of human intervention. For testing, we first evaluate the trained policies on the synthetic instances of the same size as during training and large-scale unseen ones. Then, a run-time analysis will be provided. Finally, the trained model is compared with MOEA baselines under a reasonable time limit to simulate the time constraints for real manufacturing scenarios.

5.2.1. Performance on synthetic instances of training size

Table 3 gathers the statistical results on synthetic instances for each training size. Among the methods compared, NSGA-III and NSGA-II achieved the highest HV values for the 6 × 6 and 10 × 10 size instances. However, for the 15 × 15 instances, the proposed PGRL method exhibited the highest HV value, demonstrating its superior performance on large-size instances. Additionally, Tukey's HSD Test confirms that the HV values of PGRL are statistically significantly comparable to those of the MOEA algorithms across all training sizes.

5.2.2. Generalization performance on large-sized synthetic instances

This study further investigates the ability of the proposed size-agnostic policy to handle larger instances unseen in the training data. Specifically, the policy trained on 10 × 10 instances is directly applied to 20 × 20, 30 × 20, and 50 × 20 instances, and the results are summarized in Table 4. Tukey's HSD Test results indicate that the proposed method remains statistically comparable for large-size instances, indicating that the patterns learned from smaller and medium-sized instances are still effectively generalized to large-scale problems.

Across small-scale to large-scale instances, the statistical analysis indicates a comparable performance between the proposed PGRL method and MOEA-based approaches, with a performance difference of less than 2%. While the proposed model initially exhibits slightly lower HV values than MOEA baselines on small (6 × 6 and 10 × 10) instances, it outperforms them on larger instances. This difference may be attributed to the MOEA method's tendency to converge to local optima in the search space of large MOFJSP instances.

Table 3
Statistical results on synthetic instances of training size.

Method	6 × 6			10 × 10			15 × 15		
	HV	Gap	Time	HV	Gap	Time	HV	Gap	Time
NSGA-II	0.4510=	1.46%	5.07 s	0.2530=	0.99%	13.03 s	0.4013=	−0.16%	42.45 s
MOEA/D	0.4516=	1.53%	11.06 s	0.2529=	0.94%	23.95 s	0.4014=	−0.16%	65.17 s
NSGA-III	0.4520=	1.70%	4.73 s	0.2530=	0.95%	12.01 s	0.4014=	−0.15%	38.78 s
RVEA	0.4517=	1.62%	4.35 s	0.2523=	0.71%	11.67 s	0.4007=	−0.32%	38.21 s
PGRL	0.4448	0.00%	0.09 s	0.2506	0.00%	0.10 s	0.4020	0.00%	0.35 s

The best results are marked with a gray background. The Time is the average running time for solving 100 random test instances.

Table 4
Statistical generalization result on large-sized synthetic instances.

Method	20 × 20			30 × 20			50 × 20		
	HV	Gap	Time	HV	Gap	Time	HV	Gap	Time
NSGA-II	0.2112=	−0.10%	111.82 s	0.2555=	−0.38%	233.11 s	0.1888=	−0.15%	608.64 s
MOEA/D	0.2112=	−0.09%	159.61 s	0.2555=	−0.36%	327.47 s	0.1888=	−0.15%	743.53 s
NSGA-III	0.2112=	−0.10%	102.10 s	0.2556=	−0.35%	212.22 s	0.1887=	−0.19%	554.27 s
RVEA	0.2107=	−0.31%	101.42 s	0.2550=	−0.55%	211.57 s	0.1884=	−0.37%	553.86 s
PGRL(10 ×10)	0.2114	0.00%	0.99 s	0.2564	0.00%	1.62 s	0.1891	0.00%	5.56 s

The best results are marked with a gray background. The Time is the average running time for solving 100 random test instances. The PGRL(10 × 10) is the model trained on 10 × 10 instances mentioned in Table 3.

Table 5
Statistical results on the large-size instances in real-time.

Method	20 × 20			30 × 20			50 × 20		
	HV	Gap	Time	HV	Gap	Time	HV	Gap	Time
NSGA-II	0.0842+	−61.81%	22.95 s	0.0963+	−62.90%	23.18 s	0.0153+	−92.82%	20.50 s
MOEA/D	0.0986+	−55.02%	22.60 s	0.1156+	−55.41%	23.07 s	0.0180+	−91.34%	24.39 s
NSGA-III	0.0700+	−68.36%	20.82 s	0.0939+	−63.88%	22.22 s	0.0133+	−93.72%	24.58 s
RVEA	0.0782+	−64.38%	23.16 s	0.1157+	−55.31%	28.58 s	0.0401+	−80.11%	24.75 s
PGRL(10 ×10)	0.2114	0.00%	0.99 s	0.2564	0.00%	1.62 s	0.1891	0.00%	5.56 s

The best results are marked with a gray background. The Time is the average running time for solving 100 random test instances. The **PGRL(10 ×10)** is the model trained on 10 × 10 instances mentioned in Table 3.

5.2.3. *Run time analysis*

Tables 3 and 4 list the average run time of the proposed method and MOEA baselines. We can observe from the results that the PGRL is significantly faster than MOEA baselines. Furthermore, as the instance size increases, the running time of MOEA baselines is exponentially increased, while the proposed PGRL maintains a short computational time (only a few seconds for the 50 × 20 instances). The reason for the significant difference in running time is that MOEA obtains the Pareto set through an exhaustive search, while PGRL fastly approximates the Pareto set through the neural network, which is one of the main advantage of our method.

5.2.4. *Performance on large-sized instances in real-time*

In the previous experiment, we set a generation number of 100 for the MOEA method to obtain high-quality solutions, costing a significant amount of time. However, there is a growing need for online or near-line scheduling in real-world scenarios. To simulate this case, we evaluate the performance of different methods constrained on a reasonably limited running time.

Specifically, we select the best results of all the MOEA methods within a 20-second horizon. As shown in Table 5, the results indicate that our method significantly outperforms the MOEA approach in terms of quality and speed. Our method approximates the Pareto solution fast, whereas the MOEA method necessitates multiple iterations to achieve the solution with similar quality. The swift advantage of our proposed method makes it desirable in many real-world real-time scheduling applications.

5.3. *Performance on public benchmarks*

This subsection further evaluates the generalization performance of the trained policies on the public benchmarks often used in the community by directly applying the trained policies. It is worth noting that PGRL is trained on data generated from different distributions from that of the public benchmarks. That is, we are testing the zero-shot generalization performance of PGRL. We compare PGRL with MOEA baselines and state-of-the-art ones. Results are summarized in Tables 6 and 7. For the baselines, INSBBO [24], TL-HGAPSO [21], and BEG-NSGA-II [20] are handcraft meta-heuristic algorithms to solve MOFJSP. PBB [25] is a parallel exact method (based on branch and bound) for MOFJSP, which uses the NSGA-II to initialize its upper bound. For MOEA baselines, since their performance is comparative with each other, we removed the NSGA-II and MOEA/D for simplicity.

The first part of Tables 6 and 7 present the results of baselines, and the lower part showcases the results of our method. Table 6 highlights three key findings: (1) Tukey's HSD Test results indicate that the proposed method's performance is statistically comparable to the MOEA baselines and state-of-the-art studies, suggesting that the learned policies generalize effectively to out-of-distribution instances. (2) Our method retains the efficiency advantage, while search-based methods such as the MOEA baselines and state-of-the-art studies require a much longer time to achieve high-quality solutions. (3) Regarding the average HV value, the proposed PGRL method showcases inferior performance in small-sized problems (kacem and mk) compared to the comparison method. However, it matches or surpasses the comparison method in larger-sized problems (la-rdata). Specifically, on the kacem dataset, INSBBO and BEG-NSGA-II exhibited the best performance. On the mk dataset, INBBO outperformed others, with the proposed method and MOEA baselines surpassing TL-HGAPSO. On the larger la-rdata datasets, the performance of our proposed method closely aligns with that of the MOEA baselines, albeit slightly inferior to NSGA-III and superior to RVEA.

Table 6
Statistical results on the public benchmarks-Part I.

Method	kacem			mk			la (rdata)		
	HV	Gap	Time	HV	Gap	Time	HV	Gap	Time
BEG-NSGA-II	0.1728=	4.82%	42.30 s	0.2107=	1.32%	736.86 s	/	/	/
TL-HGAPSO	0.1701=	3.19%	7.00 s	0.2017=	−3.13%	264.90 s	/	/	/
INSBBO	0.1728=	4.82%	/	0.2142=	3.22%	/	/	/	/
NSGA-III	0.1694=	2.77%	4.85 s	0.2073=	0.03%	20.81 s	0.0120=	0.66%	23.11 s
RVEA	0.1696=	2.60%	4.41 s	0.2058=	−0.85%	20.32 s	0.0118=	−1.57%	26.35 s
PGRL(10 ×10)	0.1645	0.00%	2.78 s	0.2076	0.00%	2.26 s	0.0119	0.00%	2.41 s

The best results are marked with a gray background. '/' means not given by the state-of-the-art studies. The **PGRL(10 ×10)** is the model trained on 10 × 10 instances mentioned in Table 3.

Table 7
Statistical results on the public benchmarks-Part II.

Method	sfjs			mfjs			la (vdata)		
	HV	Gap	Time	HV	Gap	Time	HV	Gap	Time
PBB	0.1334=	2.66%	13.43 s	0.0922=	5.58%	63.96 h	/	/	/
NSGA-III	0.1328=	1.94%	4.50 s	0.0906=	3.81%	2.98 s	0.0121=	−0.85	23.05 s
RVEA	0.1332=	2.41%	4.83 s	0.0906=	3.67%	2.55 s	0.0117=	−3.65	23.23 s
PGRL(10 ×10)	0.1312	0.00%	0.21 s	0.0873	0.00%	0.31 s	0.0121	0.00%	2.40 s

The best results are marked with a gray background. '/' means not given by the state-of-the-art studies. The **PGRL(10 ×10)** is the model trained on 10 × 10 instances mentioned in Table 3.

Table 7 displays statistical results for public benchmarks different from Table 6. Several key findings stand out. Firstly, Tukey's HSD Test results show that our method is comparable to baselines. Secondly, while PBB demonstrates outstanding performance on the sfjs and mfjs instances, its efficiency is compromised due to the nature of the exact algorithm, leading to significantly longer computation times. Last but not least, our proposed method outperforms the MOEA baselines on large-size instances. This observed trend echoes the findings in Table 6, highlighting the method's scalability and effectiveness across varying problem sizes.

In summary, the statistical results presented in Tables 6 and 7 demonstrate that the optimization outcomes achieved by the proposed method are comparable to those of the MOEA baselines, as well as the current state-of-the-art method for the MOFJSP. Furthermore, the results underscore the generalization capabilities of the trained policy, even in scenarios involving out-of-distribution and more intricate MOFJSP instances.

5.4. *Time complexity analysis*

This study also analyzes the time complexity of the proposed method for MOFJSP instances of varying sizes. To quantify this, we perform a controlled variable experiment by manipulating the number of operations, which allows us to investigate the impact of MOFJSP instance size on the running time. Fig. 4 illustrates the increasing running time trend as the MOFJSP instance size grows. The running time of NSGA-III demonstrates a significantly faster increase than the proposed PGRL methods. We utilize a second-order polynomial to fit all growth curves in Fig. 4. The fitting results reveal that (1) the time complexity of all methods is $O(|\mathcal{O}|^2)$. (2) the second-order coefficients of NSGA-III are an order of magnitude higher than those of PGRL, contributing to its faster runtime increase. (3) The PGRL-3-CPU's runtime is three times faster than that of PGRL-1-CPU. This difference is minor for small instances but notable for larger ones, as evidenced by the comparison of runtime on 100 × 20 instances: PGRL-1-CPU took 107.14s, while PGRL-3-CPU took only 38.75 s.

The time complexity of PGRL is $O(|\mathcal{O}|^2)$ for two reasons. First, to solve an FJSP instance with $|\mathcal{O}|$ operation nodes, the PGRL model performs $|\mathcal{O}|$ forward inferences. Second, the GIN component's time complexity is $O(|\mathcal{O}|)$ for one forward inference [56], resulting in a total time complexity of $O(|\mathcal{O}|^2)$.

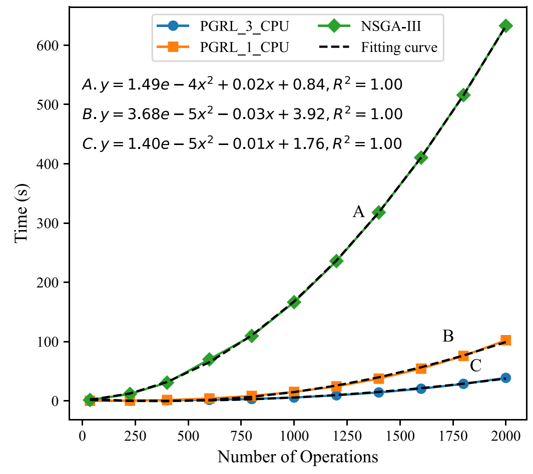


Fig. 4. The time complexity analysis on varying instance size. The result is the average running time of solving 100 random instances. To be consistent, The PGRL-3-CPU uses 3 CPU workers for parallel inference. Meanwhile, the PGRL-1-CPU only uses 1 CPU worker for parallel inference. The NSGA-III algorithm runs for 30 generations. A, B, and C represent the fitting curve functions of NSGA-III, PGRL-1-CPU, and PGRL-3-CPU, respectively.

This paper achieves superior solution speed by utilizing forward inference rather than exhaustive search for obtaining the Pareto set. Additionally, the proposed preference parallel inference algorithm reduces computation time by solving sub-problems in parallel. Due to hardware constraints, we only set the CPU workers to 3 for preference parallel inference, which means the solution time can be further reduced. Theoretically, setting the number of CPU workers equal to the size of the preference set $|\mathcal{F}|$, that is, the hyperparameter C in the parallel inference algorithm 3 is equal to $|\mathcal{F}|$, can make the runtime for solving a MOFJSP comparable to that of a MOFJSP sub-problem.

6. Discussion

The experimental results demonstrate exceptional performance of the proposed GRL-based method, particularly for larger instances of the MOFJSP, in terms of solution quality and speed. This preference-conditioned approach serves as a learning-based counterpart to

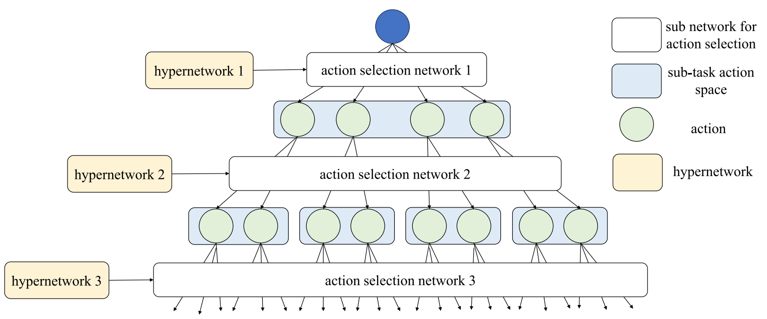

hypernetwork 1
action selection network 1
hypernetwork 2
action selection network 2
hypernetwork 3
action selection network 3
sub network for action selection
sub-task action space
action
hypernetwork

Fig. 5. A hierarchical action spaces structure for multi-action MOCO, which extends the PGRL model in this article to solve more complex MOCO with three or more decision actions.

decomposition-based algorithms like MOEA/D, which handle various trade-off preferences. Instead of relying on a finite set of solutions and conducting extensive iterative searches, this study develops a single learning-based model to simultaneously address sub-problems for all preferences via forward inference, resulting in remarkable solution speed.

The proposed method provides a single-model-for-all-preference solution, offering significant application value. Unlike previous DRL-based approaches, this method eliminates the need for instance-by-instance training and testing, thanks to its size-agnostic model. Once trained, the model can effectively solve unseen instances of varying sizes. In addition to its practical applicability, this method holds potential for extension to address other multi-objective scheduling problems or MOCO.

1. Our method demonstrates exceptional solution speed and low time complexity, making it suitable for handling more complex MOFJSP instances with dynamics and uncertainty.
2. The versatility of the proposed method allows for its extension to other intricate multi-objective scheduling problems, such as Open Shop Scheduling Problem and Flow Shop Scheduling Problem, as they can be represented as disjunctive graphs [67].
3. The method provides a multi-action framework that can be expanded to solve MOCO or MOFJSP problems with a multi-action space. Fig. 5 illustrates an example where three sub-tasks are present, constructing a hierarchical action space structure with three layers of action selection. The white areas represent the action selection network, the blue areas indicate the sub-task action space and the green nodes represent discrete actions. The yellow area represents the hypernetwork that adjusts each action-selection layer based on sub-problem preferences. By simply changing the fitness function, the parallel ES-based training algorithm can train this hierarchical structure model for various complex MOCO or MOFJSP with multi-action, like MOFJSP with AGV [68] and MOFJSP with crane transportation [69].

7. Conclusion

This paper introduces PGRL, a novel algorithm to effectively solve the MOFJSP via a graph-based reinforcement learning method. PGRL approximates the Pareto set directly, bypassing conventional search mechanisms. Moreover, PGRL integrates parallel training and inference algorithms, enhancing execution speed. Empirical evaluations show that PGRL yields superior performance on larger-scale problems due to its improved computational efficiency. Extending beyond the confines of MOFJSP, PGRL shows potential for a broader spectrum of complex, multi-objective scheduling and combinatorial challenges that involve multiple actions. However, deploying this DRL-based approach encounters difficulties in parameter tuning compared to MOEA systems, primarily because the training process of the PGRL model demands the accommodation of various sub-problems that exhibit NP-hard characteristics. This complexity is further elucidated in detailed experiments in Appendix C.

For future work, more robust DRL training algorithms can facilitate parameter tuning while leveraging powerful models like the graph transformer [70] can enhance MOFJSP solutions. Nevertheless, such enhancements may prolong computing time, necessitating enhanced parallel mechanisms implemented during the training and inference phases. Furthermore, the approach's low computational complexity suits real-time MOFJSP scenarios with dynamic factors, such as random job insertion and machine breakdown. Extending this method to real-world production settings shows promise. At the same time, designing reward functions for dynamic scenarios remains challenging.

CRediT authorship contribution statement

Chupeng Su: Writing – original draft, Software, Methodology, Conceptualization. **Cong Zhang:** Writing – review & editing, Software. **Chuang Wang:** Writing – review & editing, Software, Investigation. **Weihong Cen:** Software, Investigation. **Gang Chen:** Writing – review & editing, Supervision, Funding acquisition. **Longhan Xie:** Writing – review & editing, Supervision, Funding acquisition.

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.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No. 2021YFB3301400; National Natural Science Foundation of China under Grant No. 723B2002.

Appendix A. Structural weight generate assignment method

The study [54] proposed the structural weight generate assignment method. This method enables the generation of evenly distributed weights on the unit simplex, with a total of $n = C_p^{m+p-1}$ preferences. Here, m represents the number of objectives, and p is a parameter that controls the number of preferences. For instance, in a three-objective optimization problem, values of 4, 7, 10 and 13 are assigned to p, resulting in the generation of 15, 36, 66 and 105 preferences, respectively. Fig. 2 and Fig. 6 show the different preferences examples generated by the structural weight generate assignment method.

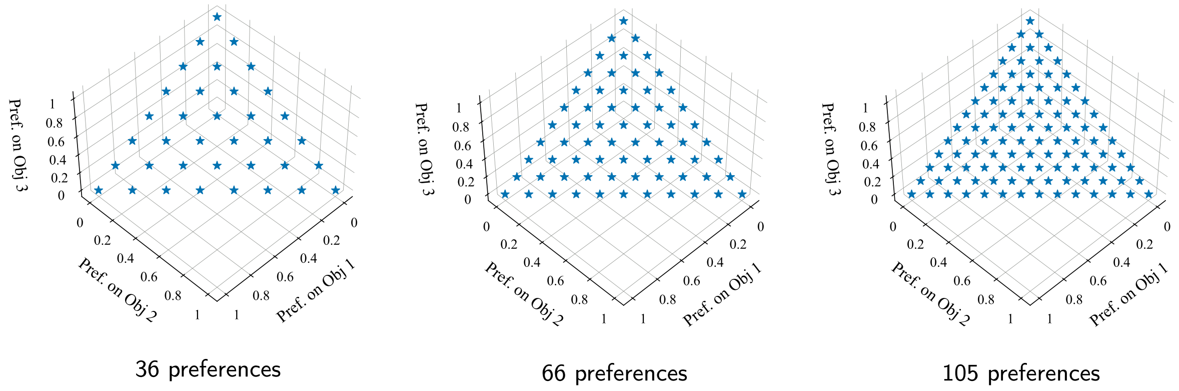


Fig. 6. Different number of uniform distributed preferences. The structural weight assignment method generates uniform distributed preferences, enabling the decomposition of the multi-objective optimization problem into structured sub-problems.

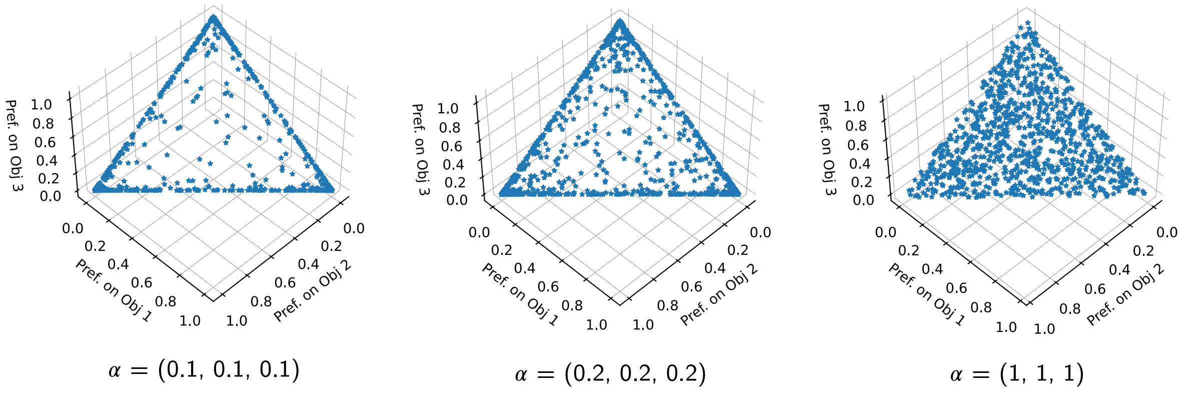


Fig. 7. The Weights generated by the Dirichlet Distribution. Controlling α in the Dirichlet distribution determines weight concentration: lower α concentrates weights on edges, while $\alpha = 1$ results in complete randomness.

Appendix B. Dirichlet distribution

The Dirichlet distribution can be seen as an extension of the beta distribution to higher dimensions [48]. It is characterized by its probability density function:

$$\mathrm{Dir}(X \mid \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{M} X_i^{\alpha_i - 1}$$

$$B(\alpha) = \frac{\prod_{i=1}^{M} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{M} \alpha_i\right)}$$

where X represents an M-dimensional variable that satisfies the condition $\sum_{i=1}^{M} X_i = 1$, with $X_i > 0$ for all i. Here, $\alpha = (\alpha_1, \dots, \alpha_M)$, where $\alpha_M > 0$. The function $B(\alpha)$ represents the multivariate Beta function, and $\Gamma(\cdot)$ denotes the Gamma function.

As an example of the Dirichlet distribution, Fig. 7 presents the results of generating 1000 weight vectors in a 3-dimensional space with varying values of the parameter α. When the α values are set to (0.1, 0.1, 0.1), the probability density is seen to congregate at the edges of the triangle. As α increases to (0.2, 0.2, 0.2), a small probability density is distributed within the triangle, while a more significant amount is still concentrated at the edges. Finally, as α reaches (1, 1, 1), the probability density becomes uniformly distributed over the triangular region.

Appendix C. Analysis and selection of hyperparameters in the training algorithm

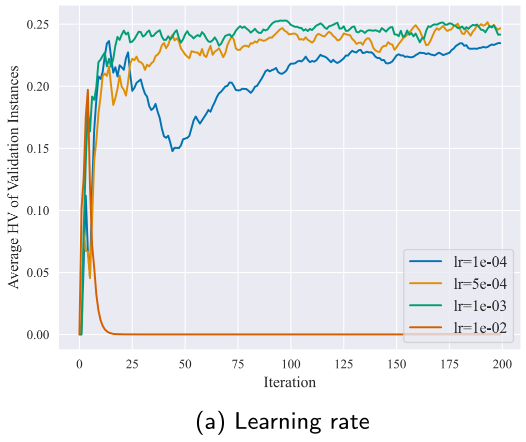
In the parallel OpenAI-ES training algorithm (Algorithm 1), various hyperparameters are involved, including the number of CPU workers C, the number of episodes per CPU worker I, the number of training steps U, the learning rate α, and the noise standard deviation σ.

Following the principles of OpenAI-ES [10], and considering the memory and CPU capacity limits, a larger number of CPU workers C and the number of episodes per CPU I, the larger population size for each training step in the parallel ES training algorithm and thus enhancing the performance. Therefore, we set the number of CPU workers C to 10 and the number of episodes per CPU I to 10 (the limit of our computational resources). Additionally, we have designated the number of training steps U as 200 to ensure that the ES can substantially explore for better performance.

Furthermore, we determined the optimal learning rate and noise standard deviation through ablation experiments. Fig. 8 depicts the impact of varying these hyperparameters on the model's training process for 10 × 10 MOFJSP instances. Specifically, we tested learning rates α of $[1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}]$, and explored noise standard deviations σ of $[1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 5 \times 10^{-2}]$.

Upon analyzing the learning rate, it is evident from Fig. 8(a) that settings of 5×10^{-4} and 1×10^{-3} lead to effective convergence of the model. Based on these observations, a learning rate of 1×10^{-3} was chosen for training the model across all problem sizes due to its ability to ensure rapid convergence and deliver satisfactory results. Conversely, an excessively high learning rate, such as 1×10^{-2}, can lead to unstable convergence, ultimately undermining performance to yield an average HV of zero. On the other hand, a lower learning rate (e.g., 1×10^{-4}) may slow down the convergence process or potentially trap the model in local optima, compromising its efficacy.

Turning to the noise standard deviation as illustrated in Fig. 8(b), values of 5×10^{-3} and 1×10^{-2} facilitate smooth convergence of the model. We opted for a noise standard deviation of 1×10^{-2} as it



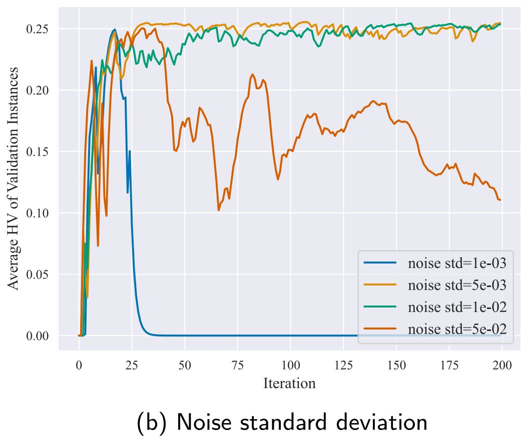


Fig. 8. Sensitivity analyses of hyperparameters in our training process.

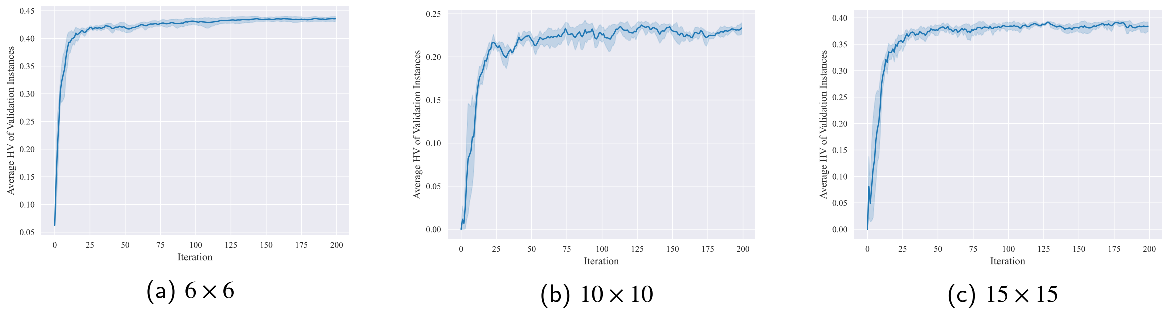


Fig. 9. The convergence curves of average HV values for each iteration during the training phase.

Table 8
The introduction and parameters setting of MOEA baselines.

Baselines	Year	Description	Parameters setting
NSGA-II [5]	2002	Classic multi-objective optimization algorithm based on non-dominated sorting and crowding.	pop_size=100, n_gen=100
MOEA/D [6]	2007	Well-known multi-objective optimization algorithm based on decomposition.	pop_size=100, n_gen=100 n_partitions=15
NSGA-III [64]	2014	An enhanced of NSGA-II developed for multi-objective optimization problems with more objectives.	pop_size=100, n_gen=100 n_partitions=12
RVEA [65]	2016	A reference direction based algorithm used utilizing angle-penalized metric.	pop_size=100, n_gen=100 n_partitions=12

leads to better performance. According to the principles of the OpenAI-ES algorithm [10], a relatively higher noise standard deviation can enhance parameter space exploration. However, this approach may sometimes result in unstable convergence, as evidenced by the curve for the 1×10^{-4} setting in Fig. 8(b). Conversely, a very low noise standard deviation, such as 5×10^{-3}, which leads to more stable convergence, might also confine the model within local optima, limiting its performance potential.

Appendix D. The convergence curves of average HV for problem of different sizes

This section shows the training curve of the average HV on the 100 validation MOFJSP instances of all MOFJSP instances sizes, including 6×6, 10×10 and 15×15. The average training time for each scale problem is 0.58 h, 1.68 h, and 4.65 h.

Appendix E. The details of the compare baselines

The Table 8 presents the MOEA baselines in this paper. We employed pymoo [71] to implement these algorithms and executed them on the same hardware as our proposed method. The crossover and mutation operators for all MOEA algorithms are implemented based on prior studies [72]. For all baselines, the population size (pop_size) is set at 100, and the generation number (n_gen) is 100. The crossover probability is 0.8, and the mutation probability is 0.05. In the case of algorithms requiring reference directions, such as MOEA/D, NSGA-III, and RVEA, we utilized the Das-Dennis method to generate reference directions systematically. The Das-Dennis method's partition number (n_partitions) in NSGA-III and RVEA is 12, while the MOEA/D is 15.

The state-of-the-art works for solving MOFJSP are shown in the Table 9. The hardware and software used by different studies, as well as the open-source public benchmark, vary. We list them in the Table 9 for the interests of potential readers.

Appendix F. Supplementary data

Our code is available at https://github.com/Chupeng24/PGRL.

The detailed results of the public benchmark instances, serving as supplementary data for this article, are available online on https://doi.org/10.1016/j.swevo.2024.101605.

Table 9
The Implementation Details and test instance set of recent studies on MOFJSP.

Baselines	Year	Hardware and software specifications	Test public benchmarks
BEG-NSGA-II [20]	2017	The algorithm was programmed by MATLAB R2014a and ran on a PC with a 2.67 GHz CPU and 4G RAM.	kacam, mk
TL-HGAPSO [21]	2018	The algorithm was programmed by MATLAB and ran on a PC with a 3.3 GHz CPU and 8G RAM.	kacam, mk
PBB [25]	2020	The algorithm was programmed by C++ and performed on Inter CPU with 12 physical cores at a 2.30 GHz frequency.	mfjs, sfjs
INSBBO [24]	2021	The algorithm was programmed by MATLAB R2017b and ran on a PC with a 3.9 GHz CPU.	kacam, mk

References

[1] A. Turkyilmaz, O. Senvar, I. Unal, S. Bulkan, A research survey: heuristic approaches for solving multi objective flexible job shop problems, J. Intell. Manuf. 31 (8) (2020) 1949–1983, http://dx.doi.org/10.1007/s10845-020-01547-4.

[2] W. Song, X. Chen, Q. Li, Z. Cao, Flexible job-shop scheduling via graph neural network and deep reinforcement learning, IEEE Trans. Ind. Inform. 19 (2) (2023) 1600–1610, http://dx.doi.org/10.1109/TII.2022.3189725.

[3] R.H. Caldeira, A. Gnanavelbabu, A Pareto based discrete jaya algorithm for multi-objective flexible job shop scheduling problem, Expert Syst. Appl. 170 (2021) http://dx.doi.org/10.1016/j.eswa.2021.114567.

[4] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, Q. Pan, A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems, IEEE/CAA J. Autom. Sin. 6 (4) (2019) 904–916, http://dx.doi.org/10.1109/JAS.2019.1911540.

[5] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197, http://dx.doi.org/10.1109/4235.996017.

[6] Q. Zhang, H. Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731, http://dx.doi.org/10.1109/TEVC.2007.892759.

[7] L. Wang, Z. Pan, J. Wang, A review of reinforcement learning based intelligent optimization for manufacturing scheduling, Complex Syst. Model. Simul. 1 (2021) 257–270, http://dx.doi.org/10.23919/CSMS.2021.0027.

[8] D. Jia, H. Guo, Z. Song, L. Shi, X. Deng, M. Perc, Z. Wang, Local and global stimuli in reinforcement learning, New J. Phys. 23 (8) (2021) http://dx.doi.org/10.1088/1367-2630/ac170a.

[9] Z. Song, H. Guo, D. Jia, M. Perc, X. Li, Z. Wang, Reinforcement learning facilitates an optimal interaction intensity for cooperation, Neurocomputing 513 (2022) 104–113, http://dx.doi.org/10.1016/j.neucom.2022.09.109.

[10] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution strategies as a scalable alternative to reinforcement learning, 2017, http://dx.doi.org/10.48550/arXiv.1703.03864, arXiv preprint.

[11] F.-Y. Liu, Z.-N. Li, C. Qian, Self-guided evolution strategies with historical estimated gradients, in: 29th International Joint Conference on Artificial Intelligence, 2020, pp. 1474–1480, http://dx.doi.org/10.24963/ijcai.2020/205.

[12] E. Sorensen, R. Ozzello, R. Rogan, E. Baker, N. Parks, W. Hu, Meta-learning of evolutionary strategy for stock trading, J. Data Anal. Inf. Process. 8 (2) (2020) 86–98, http://dx.doi.org/10.4236/jdaip.2020.82005.

[13] L. Wang, D. Jia, L. Zhang, P. Zhu, M. Perc, L. Shi, Z. Wang, Levy noise promotes cooperation in the prisoner's dilemma game with reinforcement learning, Nonlinear Dynam. 108 (2) (2022) 1837–1845, http://dx.doi.org/10.1007/s11071-022-07289-7.

[14] S. Luo, L. Zhang, Y. Fan, Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning, Comput. Ind. Eng. 159 (2021) http://dx.doi.org/10.1016/j.cie.2021.107489.

[15] S. Luo, L. Zhang, Y. Fan, Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning, IEEE Trans. Autom. Sci. Eng. 19 (4) (2022) 3020–3038, http://dx.doi.org/10.1109/TASE.2021.3104716.

[16] Z. Wu, H. Fan, Y. Sun, M. Peng, Efficient multi-objective optimization on dynamic flexible job shop scheduling using deep reinforcement learning approach, Processes 11 (7) (2023) http://dx.doi.org/10.3390/pr11072018.

[17] K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, L. Tang, A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem, Expert Syst. Appl. 205 (2022) http://dx.doi.org/10.1016/j.eswa.2022.117796.

[18] J.-Q. Li, Q.-K. Pan, Y.-C. Liang, An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, Comput. Ind. Eng. 59 (4) (2010) 647–662, http://dx.doi.org/10.1016/j.cie.2010.07.014.

[19] Z. Zhang, Z. Wu, H. Zhang, J. Wang, Meta-learning-based deep reinforcement learning for multiobjective optimization problems, IEEE Trans. Neural Netw. Learn. Syst. 34 (10) (2023) 7978–7991, http://dx.doi.org/10.1109/TNNLS.2022.3148435.

[20] Q. Deng, G. Gong, X. Gong, L. Zhang, W. Liu, Q. Ren, A bee evolutionary guiding nondominated sorting genetic algorithm II for multiobjective flexible job-shop scheduling, Comput. Intell. Neurosci. (2017) http://dx.doi.org/10.1155/2017/5232518.

[21] X. Huang, Z. Guan, L. Yang, An effective hybrid algorithm for multi-objective flexible job-shop scheduling problem, Adv. Mech. Eng. 10 (9) (2018) http://dx.doi.org/10.1177/1687814018801442.

[22] D. Min, T. Dunbing, G. Adriana, A. Salido Miguel, Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints, Robot. Comput.-Integr. Manuf. 59 (2019) 143–157, http://dx.doi.org/10.1016/j.rcim.2019.04.006.

[23] A. Alberto Garcia-Leon, S. Dauzere-Peres, Y. Mati, An efficient Pareto approach for solving the multi-objective flexible job-shop scheduling problem with regular criteria, Comput. Oper. Res. 108 (2019) 187–200, http://dx.doi.org/10.1016/j.cor.2019.04.012.

[24] Y. An, X. Chen, Y. Li, Y. Han, J. Zhang, H. Shi, An improved non-dominated sorting biogeography-based optimization algorithm for the (hybrid) multi-objective flexible job-shop scheduling problem, Appl. Soft Comput. 99 (2021) http://dx.doi.org/10.1016/j.asoc.2020.106869.

[25] C. Soto, B. Dorronsoro, H. Fraire, L. Cruz-Reyes, C. Gomez-Santillan, N. Rangel, Solving the multi-objective flexible job shop scheduling problem with a novel parallel branch and bound algorithm, Swarm Evol. Comput. 53 (2020) http://dx.doi.org/10.1016/j.swevo.2019.100632.

[26] X.-l. Chen, J.-q. Li, Y. Xu, Q-learning based multi-objective immune algorithm for fuzzy flexible job shop scheduling problem considering dynamic disruptions, Swarm Evol. Comput. 83 (2023) http://dx.doi.org/10.1016/j.swevo.2023.101414.

[27] L. Meng, C. Zhang, B. Zhang, K. Gao, Y. Ren, H. Sang, MILP modeling and optimization of multi-objective flexible job shop scheduling problem with controllable processing times, Swarm Evol. Comput. 82 (2023) http://dx.doi.org/10.1016/j.swevo.2023.101374.

[28] Y. Song, L. Wei, Q. Yang, J. Wu, L. Xing, Y. Chen, RL-GA: A reinforcement learning-based genetic algorithm for electromagnetic detection satellite scheduling problem, Swarm Evol. Comput. 77 (2023) http://dx.doi.org/10.1016/j.swevo.2023.101236.

[29] Y. Song, J. Ou, P.N. Suganthan, W. Pedrycz, Q. Yang, L. Xing, Learning adaptive genetic algorithm for earth electromagnetic satellite scheduling, IEEE Trans. Aerosp. Electron. Syst. 59 (6) (2023) 9010–9025, http://dx.doi.org/10.1109/TAES.2023.3312626.

[30] Y. Song, Y. Wu, Y. Guo, R. Yan, P.N. Suganthan, Y. Zhang, W. Pedrycz, S. Das, R. Mallipeddi, O.S. Ajani, Q. Feng, Reinforcement learning-assisted evolutionary algorithm: A survey and research opportunities, Swarm Evol. Comput. 86 (2024) http://dx.doi.org/10.1016/j.swevo.2024.101517.

[31] Y. Song, J. Ou, W. Pedrycz, P.N. Suganthan, X. Wang, L. Xing, Y. Zhang, Generalized model and deep reinforcement learning-based evolutionary method for multitype satellite observation scheduling, IEEE Trans. Syst. Man Cybern. Syst. (2024) http://dx.doi.org/10.1109/TSMC.2023.3345928.

[32] Y. Han, H. Peng, C. Mei, L. Cao, C. Deng, H. Wang, Z. Wu, Multi-strategy multi-objective differential evolutionary algorithm with reinforcement learning, Knowl.-Based Syst. 277 (2023) http://dx.doi.org/10.1016/j.knosys.2023.110801.

[33] Z. Hu, W. Gong, W. Pedrycz, Y. Li, Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization, Swarm Evol. Comput. 83 (2023) http://dx.doi.org/10.1016/j.swevo.2023.101387.

[34] F. Zhao, S. Di, L. Wang, A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem, IEEE Trans. Cybern. 53 (5) (2023) 3337–3350, http://dx.doi.org/10.1109/TCYB.2022.3192112.

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

[36] B.A. Han, J.J. Yang, A deep reinforcement learning based solution for flexible job shop scheduling problem, Int. J. Simul. Model. 20 (2) (2021) 375–386, http://dx.doi.org/10.2507/IJSIMM20-2-CO7.

[37] R. Liu, R. Piplani, C. Toro, Deep reinforcement learning for dynamic scheduling of a flexible job shop, Int. J. Prod. Res. 60 (13, SI) (2022) 4049–4069, http://dx.doi.org/10.1080/00207543.2022.2058432.

[38] Y. Gui, D. Tang, H. Zhu, Y. Zhang, Z. Zhang, Dynamic scheduling for flexible job shop using a deep reinforcement learning approach, Comput. Ind. Eng. 180 (2023) http://dx.doi.org/10.1016/j.cie.2023.109255.

[39] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, L. Qian, Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning, IEEE Trans. Ind. Inform. 20 (1) (2024) 1007–1018, http://dx.doi.org/10.1109/TII.2023.3272661.

[40] R. Wang, G. Wang, J. Sun, F. Deng, J. Chen, Flexible job shop scheduling via dual attention network-based reinforcement learning, IEEE Trans. Neural Netw. Learn. Syst. 35 (3) (2024) 3091–3102, http://dx.doi.org/10.1109/TNNLS.2023.3306421.

[41] L. Zhang, Y. Feng, Q. Xiao, Y. Xu, D. Li, D. Yang, Z. Yang, Deep reinforcement learning for dynamic flexible job shop scheduling problem considering variable processing times, J. Manuf. Syst. 71 (2023) 257–273, http://dx.doi.org/10.1016/j.jmsy.2023.09.009.

[42] C. Zhang, Y. Wu, Y. Ma, W. Song, Z. Le, Z. Cao, J. Zhang, A review on learning to solve combinatorial optimisation problems in manufacturing, IET Collab. Intell. Manuf. (2023) http://dx.doi.org/10.1049/cim2.12072, e12072 (24 pp.).

[43] X. Lin, Z. Yang, Q. Zhang, Pareto set learning for neural multi-objective combinatorial optimization, in: International Conference on Learning Representations, 2022, http://dx.doi.org/10.48550/arXiv.2203.15386.

[44] K. Li, T. Zhang, R. Wang, Deep reinforcement learning for multiobjective optimization, IEEE Trans. Cybern. 51 (6) (2021) 3103–3114, http://dx.doi.org/10.1109/TCYB.2020.2977661.

[45] H. Wu, J. Wang, Z. Zhang, MODRL/D-AM: multiobjective deep reinforcement learning algorithm using decomposition and attention model for multiobjective optimization, in: Artificial Intelligence Algorithms and Applications, Springer Singapore, 2020, pp. 575–589, http://dx.doi.org/10.48550/arXiv.2002.05484.

[46] Z. Wang, S. Yao, G. Li, Q. Zhang, Multiobjective combinatorial optimization using a single deep reinforcement learning model, IEEE Trans. Cybern. 54 (3) (2024) 1984–1996, http://dx.doi.org/10.1109/TCYB.2023.3312476.

[47] S. Li, F. Wang, Q. He, X. Wang, Deep reinforcement learning for multi-objective combinatorial optimization: A case study on multi-objective traveling salesman problem, Swarm Evol. Comput. 83 (2023) http://dx.doi.org/10.1016/j.swevo.2023.101398.

[48] T. Ye, Z. Zhang, J. Chen, J. Wang, Weight-specific-decoder attention model to solve multiobjective combinatorial optimization problems, in: 2022 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2022, pp. 2839–2844, http://dx.doi.org/10.1109/SMC53654.2022.9945568.

[49] A. Trivedi, D. Srinivasan, K. Sanyal, A. Ghosh, A survey of multiobjective evolutionary algorithms based on decomposition, IEEE Trans. Evol. Comput. 21 (3) (2017) 440–462, http://dx.doi.org/10.1109/TEVC.2016.2608507.

[50] M. Ehrgott, Multicriteria Optimization, Springer Science & Business Media, 2006, http://dx.doi.org/10.1007/3-540-27659-9.

[51] K. Miettinen, Nonlinear Multiobjective Optimization, vol. 12, Springer Science & Business Media, 2012, http://dx.doi.org/10.1007/978-1-4615-5563-6.

[52] R. Wang, Z. Zhou, H. Ishibuchi, T. Liao, T. Zhang, Localized weighted sum method for many-objective optimization, IEEE Trans. Evol. Comput. 22 (1, SI) (2018) 3–18, http://dx.doi.org/10.1109/TEVC.2016.2611642.

[53] R. Wang, Q. Zhang, T. Zhang, Decomposition-based algorithms using Pareto adaptive scalarizing methods, IEEE Trans. Evol. Comput. 20 (6) (2016) 821–837, http://dx.doi.org/10.1109/TEVC.2016.2521175.

[54] I. Das, J. Dennis, Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems, SIAM J. Optim. 8 (3) (1998) 631–657, http://dx.doi.org/10.1137/S1052623496307510.

[55] H. Wang, Y. Yu, Exploring multi-action relationship in reinforcement learning, in: Pacific Rim International Conference on Artificial Intelligence, Springer, 2016, pp. 574–587, http://dx.doi.org/10.1007/978-3-319-42911-3_48.

[56] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks? in: International Conference on Learning Representations, 2019, http://dx.doi.org/10.48550/arXiv.1810.00826.

[57] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: F. Bach, D. Blei (Eds.), International Conference on Machine Learning, in: Proceedings of Machine Learning Research, PMLR, Lille, France, 2015, http://dx.doi.org/10.48550/arXiv.1502.03167.

[58] A. Navon, A. Shamsian, G. Chechik, E. Fetaya, Learning the Pareto front with hypernetworks, in: International Conference on Learning Representations, 2021, http://dx.doi.org/10.48550/arXiv.2010.04104.

[59] D. Ha, A. Dai, Q.V. Le, Hypernetworks, in: International Conference on Learning Representations, 2017, http://dx.doi.org/10.48550/arXiv.1609.09106.

[60] I. Kacem, S. Hammadi, P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic, Math. Comput. Simul. 60 (3–5) (2002) 245–276, http://dx.doi.org/10.1016/S0378-4754(02)00019-8.

[61] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Ann. Oper. Res. 41 (1993) 157–183, http://dx.doi.org/10.1007/BF02023073.

[62] P. Fattahi, M.S. Mehrabad, F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, J. Intell. Manuf. 18 (3) (2007) 331–342, http://dx.doi.org/10.1007/s10845-007-0026-8.

[63] J. Hurink, B. Jurisch, M. Thole, Tabu search for the job-shop scheduling problem with multi-purpose machines, Oper.-Res.-Spektrum 15 (1994) 205–215, http://dx.doi.org/10.1007/BF01719451.

[64] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2014) 577–601, http://dx.doi.org/10.1109/TEVC.2013.2281535.

[65] R. Cheng, Y. Jin, M. Olhofer, B. Sendhoff, A reference vector guided evolutionary algorithm for many-objective optimization, IEEE Trans. Evol. Comput. 20 (5) (2016) 773–791, http://dx.doi.org/10.1109/TEVC.2016.2519378.

[66] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, V. da Fonseca, Performance assessment of multiobjective optimizers: An analysis and review, IEEE Trans. Evol. Comput. 7 (2) (2003) 117–132, http://dx.doi.org/10.1109/TEVC.2003.810758.

[67] C. Zhang, W. Song, Z. Cao, J. Zhang, P.S. Tan, X. Chi, Learning to dispatch for job shop scheduling via deep reinforcement learning, Adv. Neural Inf. Process. Syst. 33 (2020) 1621–1632, http://dx.doi.org/10.48550/arXiv.2010.12367.

[68] M. Zhang, L. Wang, F. Qiu, X. Liu, Dynamic scheduling for flexible job shop with insufficient transportation resources via graph neural network and deep reinforcement learning, Comput. Ind. Eng. 186 (2023) http://dx.doi.org/10.1016/j.cie.2023.109718.

[69] Y. Du, J. Li, C. Li, P. Duan, A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times, IEEE Trans. Neural Netw. Learn. Syst. (2022) http://dx.doi.org/10.1109/TNNLS.2022.3208942.

[70] L. Rampášek, M. Galkin, V.P. Dwivedi, A.T. Luu, G. Wolf, D. Beaini, Recipe for a general, powerful, scalable graph transformer, Adv. Neural Inf. Process. Syst. 35 (2022) 14501–14515, http://dx.doi.org/10.48550/arXiv.2205.12454.

[71] J. Blank, K. Deb, Pymoo: Multi-objective optimization in Python, IEEE Access 8 (2020) 89497–89509, http://dx.doi.org/10.1109/ACCESS.2020.2990567.

[72] X. Li, L. Gao, An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem, Int. J. Prod. Econ. 174 (2016) 93–110, http://dx.doi.org/10.1016/j.ijpe.2016.01.016.