Preference-Guided Multi-Policy Optimization for Flexible Job Shop Scheduling

Inguk Choi , **Woo-Jin Shin** , **Sang-Hyun Cho** and **Hyun-Jung Kim***
Department of Industrial and Systems Engineering, KAIST
{inguk0826, wjshin, ie02002, hyunjungkim}@kaist.ac.kr

Abstract

Under the shift toward Industry 4.0, mass-customized manufacturing systems have introduced complex scheduling problems, such as the Flexible Job Shop Scheduling Problem (FJSSP). Recent Deep Reinforcement Learning (DRL)-based heuristics have shown promise, yet existing methods often suffer from two key limitations: they typically rely on single-policy optimization, which limits exploration, and on imprecise reward functions, which fail to accurately reflect decision quality. To address these challenges simultaneously, we propose PGMPO (**P**reference-**G**uided **M**ulti-**P**olicy **O**ptimization), a novel learning framework consisting of (1) a simple but effective multi-policy modeling approach that allows a single network to represent multiple decision-makers, and (2) a preference-driven model optimization method that effectively guides policies to learn diverse and specialized problem-solving strategies without the need for explicit reward functions. Experimental results demonstrate that PGMPO substantially boosts the performance of existing neural solvers across several benchmarks.

1 Introduction

The rise of the mass-customization paradigm has significantly increased the complexity of modern production systems. Within this paradigm, the Flexible Job Shop Scheduling Problem (FJSSP) has attracted considerable attention, as it can model contemporary production environments with diverse product types, flexible processing routes, and heterogeneous machine capabilities [Song *et al.*, 2022]. The goal of this problem is generally to minimize the makespan, defined as the completion time of all jobs. Solving the FJSSP not only improves operational performance but also supports higher-level decisions, including manufacturing equipment selection and production planning [Dauzère-Pérès *et al.*, 2024]. However, as one of the most complex scheduling tasks, the FJSSP is particularly difficult to solve. Therefore, developing high-quality scheduling algorithms solely through human expertise has become increasingly challenging, increasing the need for automated algorithm design frameworks.

Recently, Deep Reinforcement Learning (DRL) has emerged as a promising avenue for automating the design of algorithms for the FJSSP [Han and Yang, 2021; Feng *et al.*, 2021; Du *et al.*, 2022; Zeng *et al.*, 2022; Zhao *et al.*, 2022; Chen *et al.*, 2022; Zhao and Deng, 2024; Yuan *et al.*, 2024; Zhao *et al.*, 2025; Huang *et al.*, 2025a; Huang *et al.*, 2025b]. These approaches typically model the sequential decision process of dispatching rules as a Markov Decision Process (MDP) and train a parameterized policy that maps states to actions using rewards obtained through repeated interactions with the environment, thereby avoiding the need for costly labeled data for supervised learning. The learned policy represents a novel dispatching rule that effectively captures the overall factory state and produces high-quality solutions within short computation times.

However, despite substantial progress in *policy network design*, comparatively little attention has been paid to *policy optimization methodologies*, which still typically rely on single-policy optimization with standard Proximal Policy Optimization (PPO) [Schulman *et al.*, 2017] or the REINFORCE [Williams, 1992]. In fact, current DRL-based methods for the FJSSP face two key challenges. **First**, parameterized policies struggle to sufficiently explore the solution space. Due to the combinatorially large solution space of the FJSSP, effective exploration is essential for finding high-quality solutions. Intuitively, a more diverse set of solutions increases the chance of discovering better ones. Nevertheless, existing research typically trains a single policy, which often generates less diverse solutions due to a relatively deterministic policy distribution [Xin *et al.*, 2021]. **Second**, standard reward functions often fail to accurately capture the quality of policy decisions. Many DRL-based FJSSP methods use the change in the makespan lower bound before and after each decision to evaluate actions [Zhang *et al.*, 2020; Wang *et al.*, 2023]. However, this lower bound considers only partial information, such as precedence constraints, and overlooks complex machine conflicts among operations, leading to inaccurate estimates. Moreover, as shown in Figure 1, we find that this reward function yields zero rewards in approximately 84% of the steps in each trajectory, making it difficult to provide informative feedback during training.

This paper introduces PGMPO (**P**reference-**G**uided **M**ulti-

*Corresponding author

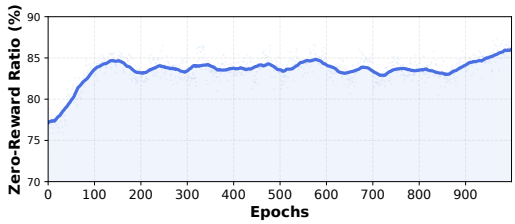


Figure 1: Zero-reward ratio during training of DAN with PPO on the 10-job $\times$ 5-machine SD_1 dataset [Wang *et al.*, 2023].

Policy **O**ptimization), a novel learning framework for the FJSSP. Specifically, we aim to train multiple decision-makers rather than relying on a single policy. For this purpose, we propose a simple yet effective population-modeling approach, called bit-conditioned policies, that enables a single network to represent multiple policies using bit (multi-hot) vectors. We also introduce a multi-policy optimization algorithm that leverages preference relations among policies, derived from the objective values of their solutions, as learning signals. PGMPO effectively addresses the limitations of existing DRL-based methods in two ways. **First**, it encourages multiple policies to explore different promising solution subspaces, enabling them to learn diverse and specialized solution patterns and thereby improving search capability. **Second**, it trains the model using natural preference relations among solutions generated by different policies, avoiding reliance on reward functions that are difficult to design properly. Consequently, PGMPO enables multiple decision-makers to rapidly generate diverse and high-quality schedules for the FJSSP.

Experimental results show that PGMPO substantially improves solution diversity without any explicit diversity-enforcing mechanism, while consistently enhancing model performance. Ablation studies further confirm its superiority over existing multi-policy and preference-based approaches.

2 Related Work

Learning-Based Approaches for FJSSP

Early DRL-based methods rely on simple state vectors or matrices, which compress information excessively and fail to reflect the structural nature of the FJSSP [Waschneck *et al.*, 2018; Park *et al.*, 2019]. In contrast, recent approaches [Han and Yang, 2021; Lei *et al.*, 2022; Song *et al.*, 2022; Wang *et al.*, 2023; Ho *et al.*, 2024; Zhao *et al.*, 2025] represent states as disjunctive or heterogeneous graphs and apply Graph Neural Networks (GNNs) to effectively capture structural information, enabling richer state representations and improving model performance.

Despite these advances in network architectures, the model training methodology has received little attention. Specifically, existing methods train only a single policy, thereby limiting exploration and leading to suboptimal performance. Although entropy regularization [Kim *et al.*, 2021] and softmax temperature strategies [Berto *et al.*, 2025] can introduce a certain degree of stochasticity, computing entropy over the entire trajectory is computationally intractable, and additional hyperparameters require careful calibration to balance exploration and exploitation. In addition, most DRL-based methods use rewards based on loose makespan lower bounds, which often fail to provide effective decision guidance. Although DOAGNN [Zhao *et al.*, 2025] redesigns the reward function to mitigate the impact of estimated makespan on decision evaluation, designing accurate evaluators requires substantial expert knowledge, which contradicts the goal of learning-based algorithm design: reducing dependence on human knowledge. To eliminate dense action evaluation, some studies have employed the REINFORCE algorithm, which directly uses the makespan as the reward for policy gradient [Iklassov *et al.*, 2023; Ho *et al.*, 2024]. However, the REINFORCE algorithm suffers from high variance, leading to unstable training and slow convergence.

Multi-Policy Optimization

Similar to our work, some studies have introduced a multi-policy paradigm to improve exploration capability for Combinatorial Optimization (CO) problems. MDAM [Xin *et al.*, 2021] utilizes multiple decoders to learn diverse policies for vehicle routing problems, encouraging distinct strategies by maximizing the Kullback–Leibler (KL) divergence among initial action distributions. Poppy [Grinsztajn *et al.*, 2023] introduces a "winner-takes-all" strategy, where only the top-performing policy (decoder) is updated during each iteration. However, these methods require a separate decoder for each policy, which is computationally expensive.

COMPASS [Chalumeau *et al.*, 2023], PolyNet [Hottung *et al.*, 2025], and MP-ASIL [Choi *et al.*, 2025] leverage latent variables to represent multiple policies in a single model. However, latent variable conditioning is applied only at the decision-making layer, which makes it challenging to disentangle problem representations across policies. In addition, these methods update the model only with the best solution, resulting in low sample efficiency.

Preference Optimization

Preference Optimization (PO) is a prevalent strategy for aligning large language models with human preferences [Rafailov *et al.*, 2023; Gheshlaghi Azar *et al.*, 2024; Meng *et al.*, 2024; Song *et al.*, 2024; Hong *et al.*, 2024]. Instead of relying on explicit reward signals, PO trains models using preference information, which is particularly useful when rewards are sparse or difficult to specify [Wirth *et al.*, 2017]. This advantage of PO has recently motivated research aimed at addressing CO tasks using PO [Liao *et al.*, 2025; Pan *et al.*, 2025], as many CO problems, including the FJSSP, involve sparse-reward characteristics. However, existing studies focus on single-policy optimization, which fundamentally differs from PGMPO. PGMPO defines preference relations among policies, rather than among solutions sampled from the same policy, thereby encouraging distinct behavioral patterns across policies.

3 Preliminaries

3.1 Flexible Job Shop Scheduling Problem

An FJSSP instance g of size $n \times m$ consists of a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and a set of machines $\mathcal{M} =$

$\{M_1, \ldots, M_m\}$. Each job $J_j \in \mathcal{J}$ has n_j operations $\mathcal{O}_j = \{O_{j1}, \ldots, O_{ji}, \ldots, O_{jn_j}\}$ that must be processed in a predefined order. Each operation O_{ji} can be assigned to exactly one machine from its set of compatible machines, $\mathcal{M}_{ji} \subseteq \mathcal{M}$, with a processing time of $p_{jik} \in \mathbb{R}_{>0}$ on machine $M_k \in \mathcal{M}_{ji}$. Each machine can only process one operation at a time. Solving the FJSSP requires assigning each O_{ji} to M_k and determining its start time S_{ji}. The goal of the FJSSP is to minimize the makespan $C_{max} = \max_{j,i} C_{ji}$, where $C_{ji} = S_{ji} + p_{jik}$ is the completion time of O_{ji}.

3.2 Learning Neural Schedulers for FJSSP

We formulate the FJSSP as a sequential decision-making problem, where a policy autoregressively selects scheduling actions until all operations are scheduled. Each action is defined as selecting an operation–machine (O_{ji}, M_k) pair, assigning O_{ji} to M_k at the earliest feasible start time S_{ji}. The policy π_θ is modeled by a neural network parameterized by θ. At each step t, given g and state s_t, the policy outputs a distribution $\pi_\theta(\cdot|s_t, g)$ over candidate actions. An action a_t is then sampled from this distribution. Feasibility is guaranteed by a masking mechanism that eliminates infeasible actions (e.g., precedence or eligibility violations) during solution construction. The overall policy $\pi_\theta(\boldsymbol{\tau} \mid g)$ that generates a solution $\boldsymbol{\tau} = (a_1, \ldots, a_{|\mathcal{O}|})$ for g is formulated as follows, where $\mathcal{O} = \bigcup_{j=1}^{n} \mathcal{O}_j$ is the set of all operations:

$$\pi_\theta(\boldsymbol{\tau} \mid g) = \prod_{t=1}^{|\mathcal{O}|} \pi_\theta(a_t \mid s_t, g). \tag{1}$$

4 PGMPO: Preference-Guided Multi-Policy Optimization

This section introduces a novel learning framework, named PGMPO. We begin by explaining the multi-policy representation, followed by the training method.

4.1 Bit-Conditioned Policies

Multi-Policy Representation

To address the limitations of a single decision-maker, we aim to train C different decision-makers while sharing a single network. This is realized by conditioning the policy network on an auxiliary input $\mathcal{Z} = \{z^c | c = 1, 2, \ldots C\}$, which indexes different policies as follows:

$$\Phi = \{\pi_\theta(\cdot \mid g, z^1), \pi_\theta(\cdot \mid g, z^2), \ldots, \pi_\theta(\cdot \mid g, z^C)\}, \tag{2}$$

where Φ is a set of policies. In this study, we define $z^1, z^2, \ldots, z^C$ using a set of unique bit vectors. This design enables a single model to efficiently represent multiple schedulers without requiring separate networks for each policy.

Multi-Policy Implementation

We describe the implementation of multiple policies using DAN [Wang *et al.*, 2023], a state-of-the-art neural FJSSP solver, as the backbone. The proposed method is not tied to DAN and can be readily plugged into other neural architectures; see Appendix A. DAN consists of L operation message attention blocks (OMBs) and L machine message attention blocks (MMBs). Initially, the raw operation and machine features ($x_{O_{ji}} \in \mathbb{R}^{d_O}$ and $x_{M_k} \in \mathbb{R}^{d_M}$) are projected into a d_h-dimensional embedding space through linear layers:

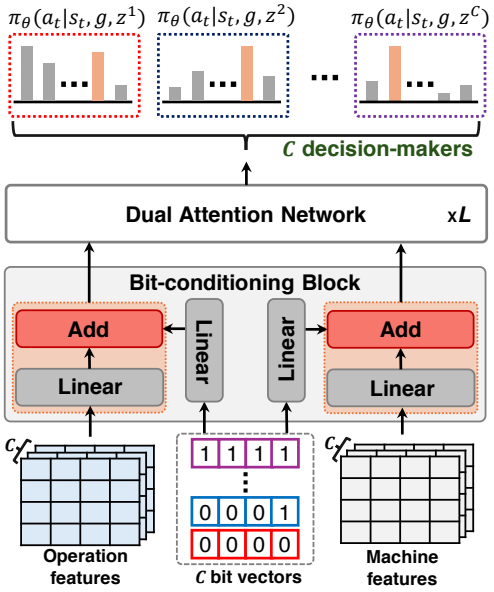


Figure 2: Illustration of the bit-conditioned policies.

$$h^{(0)}_{O_{ji}} = \mathbf{W}^1 x_{O_{ji}}, h^{(0)}_{M_k} = \mathbf{W}^2 x_{M_k}, \tag{3}$$

where $\mathbf{W}^1 \in \mathbb{R}^{d_h \times d_O}$ and $\mathbf{W}^2 \in \mathbb{R}^{d_h \times d_M}$ are learnable model parameters. The initial embeddings are then updated through the OMB and MMB modules.

Each OMB layer $l \geq 1$ computes operation hidden embeddings $h^{(l)}_{O_{ji}}$ for operations $O_{ji} \in \mathcal{O}^t_u$, where $\mathcal{O}^t_u$ is a set of unassigned operations at step t, through a multi-head self-attention as follows (for readability, time index t is omitted):

$$h^{(l)}_{O_{ji}} = \text{OMB}^{(l)}\left(\{h^{(l-1)}_{O_{ji}} \mid O_{ji} \in \mathcal{O}_u\}\right). \tag{4}$$

Similarly, each MMB layer $l \geq 1$ outputs machine hidden embeddings $h^{(l)}_{M_k}$ for machines $M_k \in \mathcal{M}_u$, where $\mathcal{M}_u$ is the set of assignable machines. The embeddings are computed using a multi-head self-attention mechanism, conditioned on the operation hidden embeddings, as follows:

$$h^{(l)}_{M_k} = \text{MMB}^{(l)}\Big(\{h^{(l-1)}_{M_k} \mid M_k \in \mathcal{M}_u\}, \{h^{(l-1)}_{O_{ji}} \mid O_{ji} \in \mathcal{O}_u\}\Big). \tag{5}$$

Then, the network calculates a logit $\mu_{(O_{ji},M_k)}$ for each candidate action (operation-machine pair) using a Multi-Layer Perceptron (MLP):

$$\mu_{(O_{ji},M_k)} = \text{MLP}(\Big[h^{(L)}_{O_{ji}} \,\|\, h^{(L)}_{M_k} \,\|\, h^{(L)}_g \,\|\, h_{(O_{ji},M_k)}\Big]), \tag{6}$$

where $\|$ is a horizontal concatenation operator, $h^{(L)}_g = (\Big[\frac{1}{|\mathcal{O}_u|}\sum_{O_{ji}\in\mathcal{O}_u} h^{(L)}_{O_{ji}} \| \frac{1}{|\mathcal{M}_u|}\sum_{M_k\in\mathcal{M}_u} h^{(L)}_{M_k}\Big])$ is a graph feature, and $h_{(O_{ji},M_k)}$ denotes operation-machine pair features. Finally, we apply the softmax function to the logits to

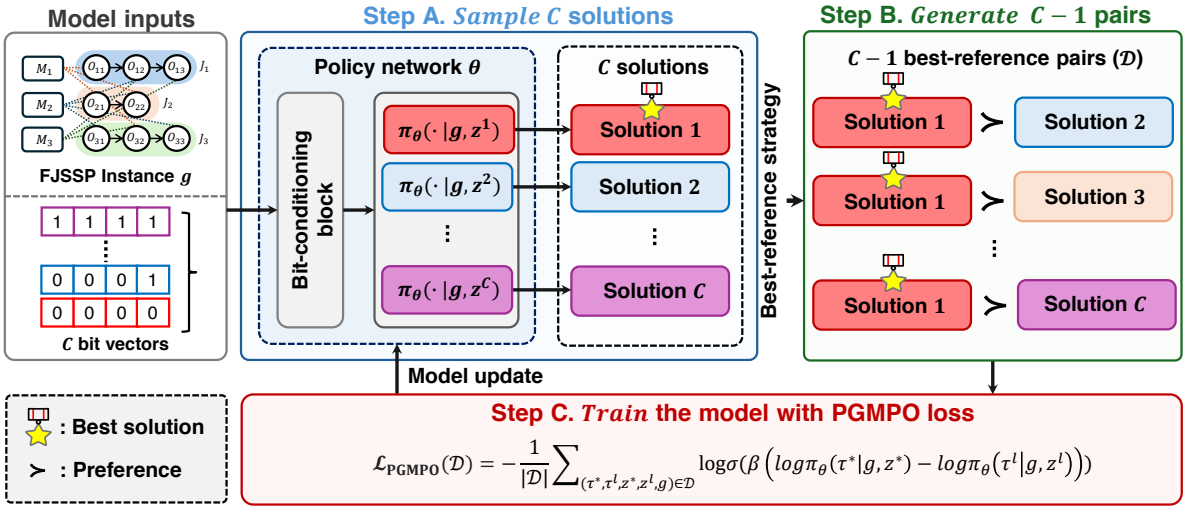


Figure 3: Overview of the PGMPO framework.

obtain an action distribution π_θ, and an action is then sampled from this distribution. We refer readers to [Wang *et al.*, 2023] for a more detailed procedure and feature definition.

We realize multiple policies by inserting lightweight additional layers into the backbone. The initial embeddings in Eq. (3) are used as inputs to the two blocks of the model. In this process, bit vectors are directly added to the initial hidden embeddings, as illustrated in Figure 2. The detailed procedure is as follows, while the other parts of the model remain unchanged:

$$h_{O_{ji}}^{(0)} = \mathbf{W}^3 z^c + \mathbf{W}^1 x_{O_{ji}}, h_{M_k}^{(0)} = \mathbf{W}^4 z^c + \mathbf{W}^2 x_{M_k}, \quad (7)$$

where $\mathbf{W}^3 \in \mathbb{R}^{d_h \times d_z}$ and $\mathbf{W}^4 \in \mathbb{R}^{d_h \times d_z}$ are learnable model weights. The updated hidden embeddings in Eq. (7) are subsequently fed into the OMB and MMB to output the actions. Through this straightforward approach, we can easily obtain C sets of bit-conditioned hidden embeddings, each representing a bit-specific policy as defined in Eq. (2). We refer to this shallow network (Eq. (7)) as the bit-conditioning block.

Rather than injecting latent vectors only at the decision-making layer, we fuse bit vectors into the initial embedding step, enabling the model to effectively learn policy-specific latent representations throughout training. This simple design choice yields substantial performance improvements, as demonstrated in Section 5.2.

4.2 Multi-Policy Optimization Procedure

This section details a multi-policy optimization method that leverages policy preference signals. An overview of the training pipeline is shown in Figure 3.

Step A: Solution Sampling

For an instance g, the C conditioned policies (defined in Eq. (2)) can perform scheduling in parallel, generating C solutions $\mathcal{S} = \{\boldsymbol{\tau}^1, \ldots, \boldsymbol{\tau}^C\}$. At this point, we can define preference relations among policies according to the objective value of their solutions as follows:

$$\boldsymbol{\tau}^1 \succ \cdots \succ \boldsymbol{\tau}^C \triangleq f(\boldsymbol{\tau}^1, g) > \cdots > f(\boldsymbol{\tau}^C, g), \quad (8)$$

where $f(\boldsymbol{\tau}^c, g) = -C_{\max}(\boldsymbol{\tau}^c, g)$ denotes the negative makespan of solution $\boldsymbol{\tau}^c$ for notational convenience. These preference relations are then used as training signals for model optimization.

Step B: Generating Preference Pairs

In PO, model performance is highly affected by how preference pairs are defined [Zhao *et al.*, 2016; Rafailov *et al.*, 2023; Song *et al.*, 2024]. One simple approach is to generate all $\binom{C}{2}$ pairwise comparisons among the C solutions. However, using all possible pairs for training makes it difficult to identify the best bit vector for each instance and increases training complexity due to multiple winners. Therefore, inspired by BOPO [Liao *et al.*, 2025], we construct $C-1$ pairs to effectively emphasize the best-performing policy z^* for each instance g:

$$\mathcal{D} = \left\{(\boldsymbol{\tau}^*, \boldsymbol{\tau}^l, z^*, z^l, g) \mid l = 1, \ldots, C,\ z^l \neq z^*\right\}. \quad (9)$$

Each pair compares the best solution $\boldsymbol{\tau}^*$ from z^* with one of the remaining solutions $\boldsymbol{\tau}^l$ from z^l.

Step C: Model Training Using PGMPO Loss

The generated preference pairs are used to guide each policy toward learning increasingly preferred solutions. However, since $f(\boldsymbol{\tau}, g)$ is non-differentiable, it cannot be directly used as a learning signal for model optimization via gradient descent. Thus, we need to transform the current preference relations into a differentiable preference objective. Starting from entropy-regularized reinforcement learning [Peters and Schaal, 2007; Rafailov *et al.*, 2023; Meng *et al.*, 2024; Pan *et al.*, 2025], it is straightforward to reparameterize $f(\boldsymbol{\tau}^c, g)$ using the log-likelihood of a solution:

$$f_\theta(\boldsymbol{\tau}^c, g) = \beta \log \pi_\theta(\boldsymbol{\tau}^c \mid g, z^c), \quad (10)$$

where β denotes a scaling parameter. We then leverage the probabilistic preference model to express the preference distribution $p_\theta(\boldsymbol{\tau}^* \succ \boldsymbol{\tau}^l \mid g)$. This pairwise preference framework employs a mapping function σ that transforms objective

Algorithm 1 PGMPO training

Input: Model parameters θ, training dataset $\mathcal{G}$, number of policies C, number of training steps E, and batch size B.
for $step = 1$ to E **do**
  $g_i \leftarrow \texttt{SampleInstance}(\mathcal{G}), \forall i \in \{1, \dots, B\}$
  $\Phi_i = \{\pi_\theta(\cdot \mid g_i, z_i^j)\}_{j=1}^{C}, \forall i \in \{1, \dots, B\}$
  $\mathcal{S}_i \leftarrow \texttt{SolutionRollout}(\Phi_i), \forall i \in \{1, \dots, B\}$
  $\mathcal{D}_i \leftarrow \texttt{GeneratePair}(\mathcal{S}_i), \forall i \in \{1, \dots, B\}$
  Compute $\mathcal{L}_{\text{PGMPO}}(\mathcal{D}_i), \forall i \in \{1, \dots, B\}$
  $\mathcal{L}_{\text{PGMPO}} \leftarrow \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}_{\text{PGMPO}}(\mathcal{D}_i)$
  $\theta \leftarrow \text{Adam}(\theta, \nabla_\theta \mathcal{L}_{\text{PGMPO}})$
end for
Output: Trained model parameters θ.

value differences into preference probabilities, as follows:

$$p_\theta(\boldsymbol{\tau}^* \succ \boldsymbol{\tau}^l \mid g) = \sigma(\beta(\log \pi_\theta(\boldsymbol{\tau}^* \mid g, z^*) - \log \pi_\theta(\boldsymbol{\tau}^l \mid g, z^l))). \quad (11)$$

In this work, we employ the Bradley–Terry (BT) model [Bradley and Terry, 1952] as the pairwise preference model. The BT model uses the sigmoid function as its mapping function σ. We also define β as $f(\boldsymbol{\tau}^l, g)/(f(\boldsymbol{\tau}^*, g) \cdot |\mathcal{O}|)$ instead of a fixed value. This adaptively controls β based on solution quality differences, eliminating extensive hyperparameter tuning and avoiding length bias [Liao *et al.*, 2025].

Finally, the preference probabilities p_θ are represented with respect to the parameterized policy π_θ. We therefore formulate the following maximum likelihood objective, referred to as PGMPO loss:

$$\mathcal{L}_{\text{PGMPO}}(\mathcal{D}) = -\frac{1}{|\mathcal{D}|} \sum\nolimits_{(\boldsymbol{\tau}^*, \boldsymbol{\tau}^l, z^*, z^l, g) \in \mathcal{D}} \log \sigma(\beta(\log \pi_\theta(\boldsymbol{\tau}^* \mid g, z^*) - \log \pi_\theta(\boldsymbol{\tau}^l \mid g, z^l))). \quad (12)$$

Mini-batch PGMPO training is summarized in Algorithm 1.

Gradient Analysis

Here, we analyze the gradient of PGMPO loss to better understand how PGMPO guides multiple policies to learn diverse problem-solving strategies. Let u denote $\beta(\log \pi_\theta(\boldsymbol{\tau}^* \mid g, z^*) - \log \pi_\theta(\boldsymbol{\tau}^l \mid g, z^l))$. Then, the gradient of $\mathcal{L}_{\text{PGMPO}}$ with respect to θ can be derived as follows (the detailed derivation is provided in Appendix B):

$$\nabla_\theta \mathcal{L}_{\text{PGMPO}} = \underbrace{-\beta}_{\text{Scaling factor}} \cdot \underbrace{(1 - \sigma(u))}_{\text{Regularization factor}} \cdot \underbrace{\left(\nabla_\theta \log \pi_\theta(\boldsymbol{\tau}^* \mid g, z^*) - \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}^l \mid g, z^l)\right)}_{\text{Policy update factor}}. \quad (13)$$

As shown in Eq. (13), PGMPO encourages the policy conditioned on z^* to assign higher likelihood to the preferred schedule $\boldsymbol{\tau}^*$, while reducing the likelihood assigned to dispreferred schedules $\boldsymbol{\tau}^l$ under z^l. As a result, each policy learns distinct preferred solution patterns for different instance sub-distributions, enabling multiple policies to explore diverse favorable regions of the search space. In addition, PGMPO leverages naturally induced preference relations from objective values as training signals, thereby eliminating the need for explicitly designed reward functions. Unlike the REINFORCE algorithm, which directly uses absolute objective values, PO leverages sigmoid-bounded relative log-likelihoods, substantially reducing gradient variance and enabling more stable training.

Problem Size ($n \times m$)	n_j	$\lvert\mathcal{M}_{ji}\rvert$	$\overline{p_{ji}}$
10×5	$U(4,6)$	$U(1,5)$	$U(1,20)$
20×5	$U(4,6)$	$U(1,5)$	$U(1,20)$
15×10	$U(8,12)$	$U(1,10)$	$U(1,20)$
20×10	$U(8,12)$	$U(1,10)$	$U(1,20)$
30×10	$U(8,12)$	$U(1,10)$	$U(1,20)$
40×10	$U(8,12)$	$U(1,10)$	$U(1,20)$

Table 1: Distributions for generating test instances.

5 Experiments

This section presents experimental settings and evaluation results for PGMPO. All experiments are conducted on a machine with an Intel i9-14900KS CPU and a single NVIDIA GeForce RTX 4090 GPU with 24GB of memory. Our PyTorch implementation, built on DAN, is publicly available.[1]

5.1 Experimental Settings

Training Settings

We train our model on instances with $n \times m = 10 \times 5$. Following the well-established SD_1 procedure [Song *et al.*, 2022], we sample the training instances from the corresponding discrete uniform distributions as follows: $n_j \sim U(4,6)$, $|\mathcal{M}_{ji}| \sim U(1,5)$, and $\bar{p}_{ji} \sim U(1,20)$, where $\bar{p}_{ji}$ is an average processing time of operation O_{ji}. p_{jik} is then sampled from $U(0.8\bar{p}_{ji}, 1.2\bar{p}_{ji})$. We set $C = 128$, which corresponds to 7-dimensional bit vectors, and train the model for 4000 epochs (E) with a batch size (B) of 16. All other hyperparameters follow the original DAN setting [Wang *et al.*, 2023]; we refer readers to the original paper for details.

Test Instances

For evaluation, we use six synthetic SD_1 datasets from [Wang *et al.*, 2023], covering sizes 10×5, 20×5, 15×10, 20×10, 30×10, and 40×10, each containing 100 instances. The test datasets follow the distribution shown in Table 1. This setup assesses both in-distribution performance and cross-size generalization.

To further validate cross-distribution generalization, we test the model without retraining on four public benchmark sets: MK [Brandimarte, 1993] and three groups from the Hurink dataset (rdata, edata, and vdata) [Hurink *et al.*, 1994]. The MK dataset consists of 10 instances covering seven different sizes, from 10×6 to 20×15. Each Hurink group includes 40 instances, covering eight problem sizes from 10×5 to 30×10, with five instances per size. These groups mainly

[1] https://github.com/Ingukchoi/PGMPO

Data	Size	Metric	FIFO*	MOR*	SPT*	MWKR*	HGNN*	MCGA*	ME-GNN*	RS*	DAN*	PGMPO	OR-Tools*
SD_1	10×5	MS	119.40	115.38	129.82	113.23	105.59	105.00	104.89	103.31	101.67	**99.03**	96.32 (5%)
		Gap	24.06%	19.87%	34.76%	17.58%	9.66%	9.01%	8.90%	7.26%	5.57%	**2.81%**	
		Time	0.16s	0.16s	0.16s	0.16s	1.11s	–	–	–	0.38s	0.39s	
	20×5	MS	216.08	214.16	230.48	209.78	207.53	203.88	203.74	201.73	192.78	**189.46**	188.15 (0%)
		Gap	14.87%	13.85%	22.56%	11.51%	10.31%	8.36%	8.29%	7.22%	2.46%	**0.70%**	
		Time	0.32s	0.32s	0.32s	0.32s	2.36s	–	–	–	0.92s	0.96s	
	15×10	MS	184.55	173.15	198.33	171.25	160.86	160.42	159.92	157.30	153.22	**150.12**	143.53 (7%)
		Gap	28.65%	20.68%	38.22%	19.41%	12.13%	11.77%	11.41%	9.59%	6.79%	**4.59%**	
		Time	0.51s	0.51s	0.50s	0.50s	3.98s	–	–	–	2.68s	2.79s	
	20×10	MS	233.48	219.80	255.17	216.11	214.81	211.08	210.72	207.86	193.91	**189.72**	195.98 (0%)
		Gap	19.22%	12.20%	30.25%	10.30%	9.64%	7.70%	7.52%	6.06%	-1.03%	**-3.20%**	
		Time	0.71s	0.71s	0.71s	0.71s	6.23s	–	–	–	4.72s	4.83s	
	30×10	MS	328.24	317.43	350.07	312.93	308.55	308.83	308.33	305.27	286.77	**274.02**	274.67 (6%)
		Gap	19.50%	15.57%	27.47%	13.96%	12.36%	12.43%	12.25%	11.14%	4.43%	**-0.24%**	
		Time	1.10s	1.10s	1.11s	1.10s	12.79s	–	–	–	9.56s	9.86s	
	40×10	MS	426.97	421.34	445.17	414.82	410.76	411.71	410.19	407.28	379.71	**363.91**	365.96 (3%)
		Gap	16.67%	15.13%	21.66%	13.37%	12.26%	12.50%	12.09%	11.29%	3.77%	**-0.56%**	
		Time	1.51s	1.51s	1.50s	1.51s	24.54s	–	–	–	15.94s	16.43s	

Table 2: Experimental results on six synthetic datasets. Reference method: OR-Tools. Results for the methods marked with * are taken from original papers. **Boldface** denotes the best MS and Gap among all methods except the reference method. For OR-Tools, we report both MS and the optimality ratio. MCGA, ME-GNN, and RS do not report computation times.

Data	Metric	MWKR*	HGNN*	MCGA*	ME-GNN*	RS*	DAN*	PGMPO
MK	MS	201.74	190.30	191.00	190.70	181.11	180.80	**177.80**
	Gap	28.91%	18.56%	18.67%	18.59%	15.40%	9.53%	**6.40%**
Hurink (rdata)	MS	1053.10	985.30	987.60	988.28	976.40	978.28	**975.32**
	Gap	13.86%	5.57%	5.71%	5.78%	4.72%	4.95%	**4.36%**
Hurink (edata)	MS	1219.01	1116.68	1115.05	1115.23	1112.57	1122.60	**1106.07**
	Gap	18.60%	8.71%	8.38%	8.39%	7.90%	9.08%	**6.85%**
Hurink (vdata)	MS	952.01	930.80	931.35	930.30	925.78	925.40	**924.15**
	Gap	4.22%	1.32%	1.40%	1.30%	0.70%	0.69%	**0.42%**

Table 3: Experimental results on benchmark datasets. Reference method: Best-known solutions from [Wang *et al.*, 2023].

differ in the degree of machine flexibility, with an average of 2, 1.15, and $0.5m$ eligible machines per operation for rdata, edata, and vdata, respectively. The processing time of an operation is the same on all its eligible machines.

Baseline Algorithms and Evaluation Metrics

We compare PGMPO against several traditional methods and state-of-the-art DRL-based approaches as follows:

- **Classic solver:** CP-SAT solver from OR-Tools, using a 1800-second (s) time limit per instance.
- **Priority dispatching rules (PDRs):** First-In-First-Out (FIFO), Most Operations Remaining (MOR), Shortest Processing Time (SPT), and Most Work Remaining (MWKR).
- **DRL-based methods:** HGNN [Song *et al.*, 2022], MCGA [Huang *et al.*, 2025a], ME-GNN [Huang *et al.*, 2025b], RS [Ho *et al.*, 2024], and DAN [Wang *et al.*, 2023].

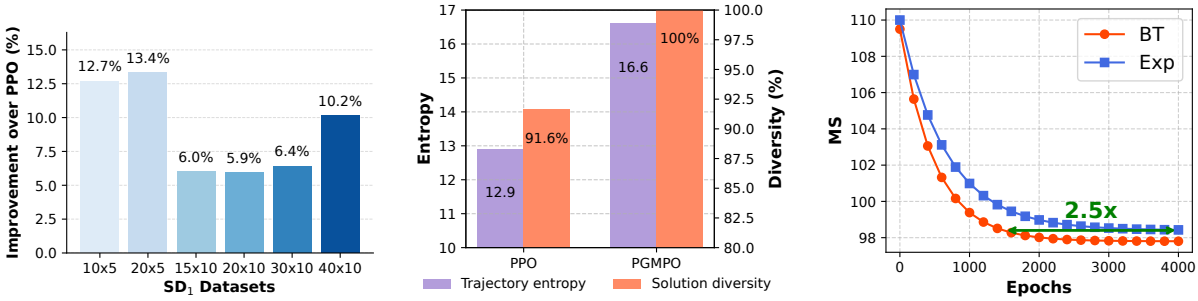
At test time, DRL-based baselines employ a sampling strategy that generates 100 solutions using probabilistic sampling from a single policy and reports the best. Similarly, PGMPO generates one solution per policy in parallel and reports the result of the best-performing policy z^*. For a fair comparison, during inference, PGMPO samples 100 bit vectors out of 128 to generate solutions from 100 policies.

For performance evaluation, we use three metrics: average makespan (MS), average gap (Gap), and average runtime (Time). The gap is calculated as $100 \times (C_{max} - C^{RF}_{max})/C^{RF}_{max}$, where $C^{RF}_{\max}$ denotes the makespan of the reference method.

5.2 Experimental Results

Experimental Results on Synthetic Datasets

We first evaluate PGMPO on six synthetic datasets, and the results are summarized in Table 2. From the table, we can observe that PGMPO greatly outperforms all PDRs and DRL-based methods across all instance sizes. In particular, compared with DAN, PGMPO yields statistically significant improvements on 10×5 and 20×5 datasets at the 5% level, and on the remaining four datasets at the 1% level, as confirmed by the Wilcoxon signed-rank test [Wilcoxon, 1992].



(a) Performance improvement (b) Solution diversity (c) Training curves

Figure 4: **(a)** Performance improvement over DAN with PPO under the total tardiness objective. **(b)** Trajectory entropy and solution diversity evaluated on the 10×5 SD_1 test set. **(c)** Training curves on training instances.

These results show that PGMPO can unlock the potential of the backbone model with minimal modifications. Notably, on unseen large-scale instances (20×10, 30×10, and 40×10), PGMPO even achieves better solution quality than OR-Tools while requiring $109\times$ to $375\times$ less computation time.

Experimental Results on Benchmark Datasets

We next compare PGMPO against the best-performing PDR (MWKR) and DRL-based methods on four benchmark datasets, with results provided in Table 3. As shown in the table, PGMPO considerably surpasses all competitors, demonstrating robust performance in out-of-distribution scenarios. Concretely, with only minimal computational overhead, PGMPO reduces the performance gap of DAN by 32.8%, 11.9%, 24.6%, and 43.5% on the MK, rdata, edata, and vdata datasets, respectively.

Applicability to Other Objectives

PGMPO is not restricted to makespan minimization. To evaluate its applicability to other objectives, we also consider minimizing total tardiness. Specifically, total tardiness is computed as $\Sigma_{J_j \in \mathcal{J}} \max(C_j - D_j, 0)$, where C_j denotes the completion time of job J_j, and D_j is its deadline. Following [Smit *et al.*, 2026], we set $D_j = \Sigma_{O_{ji} \in \mathcal{O}_j} (\min_{M_k \in \mathcal{M}_{ji}} p_{jik})$. Using these deadlines, we train both PGMPO and DAN with PPO on 10×5 instances from SD_1, and evaluate them across six problem sizes. For the PPO baseline, we adopt the lower-bound-based reward function from [Smit *et al.*, 2026]; details of the reward design are provided in the original paper. All other experimental settings follow Section 5.1.

Figure 4a shows that PGMPO consistently outperforms DAN with PPO under the total tardiness objective. This suggests that PGMPO can be easily applied to different objectives without requiring the redesign of reward functions for each one.

Exploration Capability

Our main claim is that PGMPO effectively guides multiple policies to learn diverse solution patterns. To empirically verify this claim, we compare PGMPO and PPO with an entropy bonus (the original DAN training approach) in terms of the sum of step-wise entropy (trajectory entropy) and the unique solution ratio. From Figure 4b, we can see that PGMPO generates 100% unique solutions. Even without any diversity-enforcing term, PGMPO exhibits significantly higher trajectory entropy than PPO with an entropy bonus. These results provide clear evidence that the policies learned with PGMPO exhibit distinct behavioral patterns.

Data	Metric	$C = 16$	$C = 32$	$C = 64$	$C = 128$
MK	Gap	7.11%	6.75%	6.54%	**6.40%**
rdata	Gap	5.12%	4.97%	4.82%	**4.36%**
edata	Gap	7.80%	7.59%	7.20%	**6.85%**
vdata	Gap	0.60%	0.52%	0.47%	**0.42%**

Table 4: The effect of C on model performance.

Preference Model

We examine the impact of the preference model, a crucial component of PO. For comparison, we train the model using an unbounded exponential function (Exp) as the preference model $\sigma(\cdot) = \exp(\cdot)$ and report the training curves in Figure 4c, while keeping all other settings unchanged. As shown in the figure, BT matches the performance of Exp in approximately $2.5\times$ fewer epochs. Based on this result, we use the BT function as the PGMPO preference model.

Sensitivity Analysis

We analyze the effect of C on model performance by training our model with different values of C, where $C \in \{16, 32, 64, 128\}$. For each model, we draw 100 bit vectors during inference. Table 4 shows the results on four benchmarks. As illustrated, training with a larger C consistently produces stronger models. These results may stem from the fact that a larger C increases the likelihood of finding a better anchor, $\boldsymbol{\tau}^*$, for generating preference pairs.

Ablation Study

This section presents ablation study results that analyze the impact of each component of PGMPO. For the bit-conditioning block ablation (w/o_BcB), we inject the bit vectors into the MLP in Eq. (6), following prior work [Chalumeau *et al.*, 2023; Choi *et al.*, 2025]. For the best-reference strategy ablation (w/o_BR), we generate all possible $\binom{C}{2}$ pairs for model training. We further compare PGMPO

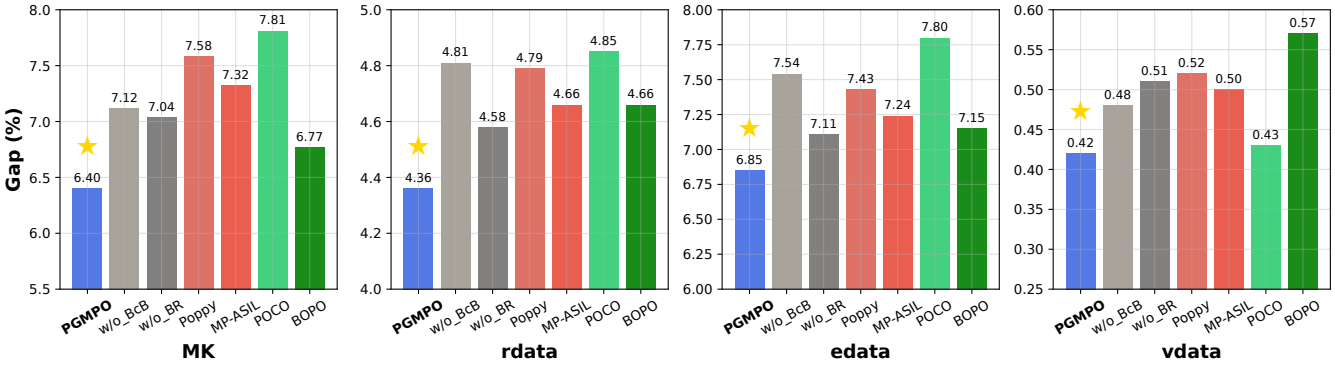


Figure 5: Ablation study results on four benchmark datasets.

with two representative multi-policy optimization methods, Poppy [Grinsztajn *et al.*, 2023], and MP-ASIL [Choi *et al.*, 2025], by training DAN with each method. Finally, to assess the benefit of multi-policy modeling, we train DAN with POCO [Pan *et al.*, 2025] and BOPO [Liao *et al.*, 2025]. These two recent PO-based methods for CO problems optimize a single policy. We evaluate all variants on four benchmark datasets, with results shown in Figure 5.

As shown in the figure, PGMPO consistently achieves the best performance across all benchmarks. Specifically, the results clearly indicate that both the BcB and BR strategies contribute to performance improvements. Moreover, PGMPO demonstrates stronger effectiveness in multi-policy optimization than Poppy and MP-ASIL. In comparison with POCO and BOPO, PGMPO successfully learns diverse schedule generators in the multi-policy setting, leading to significant performance gains.

6 Conclusion and Future Work

We propose PGMPO, a novel learning framework for the FJSSP that integrates multi-policy modeling with preference-based population optimization to learn diverse and specialized scheduling strategies. Extensive experiments on synthetic and benchmark datasets demonstrate that PGMPO significantly improves the performance of existing DRL-based solvers. Moreover, PGMPO maintains its effectiveness across different objective functions, highlighting its broad applicability. Ablation studies further provide strong empirical evidence supporting the contributions of the proposed method. For future work, we aim to extend PGMPO beyond FJSSP to a broader range of realistic scheduling scenarios.

A Applicability to Other Models

To evaluate the model-agnostic nature of PGMPO, we apply it to HGNN [Song *et al.*, 2022] as an additional backbone. Following Section 4.1, we add the projected bit embeddings to the initial operation and machine embeddings. We retain the original HGNN settings (e.g., embedding dimension), except for introducing multi-policy modeling and PO-based optimization. We train the model on the synthetic dataset SD_1 [Song *et al.*, 2022] using only 10×5 instances. The training parameters follow those described in Section 5.1.

Method	10×5	20×5	15×10	20×10	30×10	40×10
HGNN*	105.59	207.53	160.86	214.81	308.55	410.76
PGMPO	**103.14**	**201.98**	**158.11**	**204.87**	**300.49**	**396.04**

Table 5: Average makespan comparison between HGNN and PGMPO.

Table 5 provides the evaluation results on the SD_1 dataset. Despite being trained only on small-scale instances, PGMPO consistently enhances HGNN across all problem sizes, highlighting its versatility.

B Gradient of PGMPO Loss

In this section, we derive the gradient of the PGMPO loss following the procedure in [Liao *et al.*, 2025]. Let u denote $\beta(\log \pi_\theta(\boldsymbol{\tau}^* \mid g, z^*) - \log \pi_\theta(\boldsymbol{\tau}^l \mid g, z^l))$. The gradient of $\mathcal{L}_{\text{PGMPO}}$ with respect to θ can be expressed as:

$$\nabla_\theta \mathcal{L}_{\text{PGMPO}} = \frac{\partial \mathcal{L}_{\text{PGMPO}}}{\partial u} \cdot \nabla_\theta u. \tag{14}$$

The derivative of $-\log \sigma(u)$ can be written as:

$$\frac{\partial \mathcal{L}_{\text{PGMPO}}}{\partial u} = -(1 - \sigma(u)). \tag{15}$$

The gradient of u with respect to θ is:

$$\begin{aligned} \nabla_\theta u = \beta(&\nabla_\theta \log \pi_\theta(\boldsymbol{\tau}^* \mid g, z^*) \\ &-\nabla_\theta \log \pi_\theta(\boldsymbol{\tau}^l \mid g, z^l)). \end{aligned} \tag{16}$$

Putting these together, the total gradient of $\mathcal{L}_{\text{PGMPO}}$ is obtained as follows:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{\text{PGMPO}} = &-\beta(1 - \sigma(u)) \cdot \\ &\left(\nabla_\theta \log \pi_\theta(\boldsymbol{\tau}^* \mid g, z^*) - \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}^l \mid g, z^l)\right). \end{aligned} \tag{17}$$

Acknowledgments

This work was supported in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2024-00334171) and in part by the IITP(Institute of Information & Communications Technology Planning & Evaluation)-ITRC(Information Technology Research Center) grant funded by the Korea government(Ministry of Science and ICT) (IITP-2026-RS-2024-00437268).

References

[Berto *et al.*, 2025] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, et al. RL4CO: an extensive reinforcement learning for combinatorial optimization benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5278–5289, 2025.

[Bradley and Terry, 1952] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[Brandimarte, 1993] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.

[Chalumeau *et al.*, 2023] Felix Chalumeau, Shikha Surana, Clément Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and Tom Barrett. Combinatorial optimization with policy adaptation using latent space search. In *Advances in Neural Information Processing Systems*, volume 36, pages 7947–7959, 2023.

[Chen *et al.*, 2022] Ruiqi Chen, Wenxin Li, and Hongbing Yang. A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem. *IEEE Transactions on Industrial Informatics*, 19(2):1322–1331, 2022.

[Choi *et al.*, 2025] Inguk Choi, Woo-Jin Shin, Sang-Hyun Cho, and Hyun-Jung Kim. Towards generalizable multi-policy optimization with self-evolution for job scheduling. In *Advances in Neural Information Processing Systems*, volume 38, pages 24569–24616, 2025.

[Dauzère-Pérès *et al.*, 2024] Stéphane Dauzère-Pérès, Junwen Ding, Liji Shen, and Karim Tamssaouet. The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2):409–432, 2024.

[Du *et al.*, 2022] Yu Du, Junqing Li, Chengdong Li, and Peiyong Duan. A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):5695–5709, 2022.

[Feng *et al.*, 2021] Yi Feng, Lu Zhang, Zhile Yang, Yuanjun Guo, and Dongsheng Yang. Flexible job shop scheduling based on deep reinforcement learning. In *5th Asian Conference on Artificial Intelligence Technology (ACAIT)*, pages 660–666, 2021.

[Gheshlaghi Azar *et al.*, 2024] Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238, pages 4447–4455, 2024.

[Grinsztajn *et al.*, 2023] Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pages 48485–48509, 2023.

[Han and Yang, 2021] Baoan Han and J Yang. A deep reinforcement learning based solution for flexible job shop scheduling problem. *International Journal of Simulation Modelling*, 20(2):375–386, 2021.

[Ho *et al.*, 2024] Kuo-Hao Ho, Jui-Yu Cheng, Ji-Han Wu, Fan Chiang, Yen-Chi Chen, Yuan-Yu Wu, and I-Chen Wu. Residual scheduling: A new reinforcement learning approach to solving job shop scheduling problem. *IEEE Access*, 12:14703–14718, 2024.

[Hong *et al.*, 2024] Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic preference optimization without reference model. *arXiv preprint arXiv:2403.07691*, 2024.

[Hottung *et al.*, 2025] André Hottung, Mridul Mahajan, and Kevin Tierney. PolyNet: Learning diverse solution strategies for neural combinatorial optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

[Huang *et al.*, 2025a] Dailin Huang, Hong Zhao, Jie Cao, Kangping Chen, and Lijun Zhang. Optimizing the flexible job shop scheduling problem via deep reinforcement learning with mean multichannel graph attention. *Applied Soft Computing*, 177:113128, 2025.

[Huang *et al.*, 2025b] Dailin Huang, Hong Zhao, Weiquan Tian, and Kangping Chen. A deep reinforcement learning method based on a multiexpert graph neural network for flexible job shop scheduling. *Computers & Industrial Engineering*, 200:110768, 2025.

[Hurink *et al.*, 1994] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.

[Iklassov *et al.*, 2023] Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa De Retana, and Martin Takac. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 5350–5358, 2023.

[Kim *et al.*, 2021] Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems*, volume 34, pages 10418–10430, 2021.

[Lei *et al.*, 2022] Kun Lei, Peng Guo, Wenchao Zhao, Yi Wang, Linmao Qian, Xiangyin Meng, and Liansheng Tang. A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. *Expert Systems with Applications*, 205:117796, 2022.

[Liao *et al.*, 2025] Zijun Liao, Jinbiao Chen, Debing Wang, Zizhen Zhang, and Jiahai Wang. BOPO: Neural combinatorial optimization via best-anchored and objective-guided preference optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pages 37456–37475, 2025.

[Meng *et al.*, 2024] Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, volume 37, pages 124198–124235, 2024.

[Pan *et al.*, 2025] Mingjun Pan, Guanquan Lin, You-Wei Luo, Bin Zhu, Zhien Dai, Lijun Sun, and Chun Yuan. Preference optimization for combinatorial optimization problems. In *Forty-second International Conference on Machine Learning*, 2025.

[Park *et al.*, 2019] In-Beom Park, Jaeseok Huh, Joongkyun Kim, and Jonghun Park. A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities. *IEEE Transactions on Automation Science and Engineering*, 17(3):1420–1431, 2019.

[Peters and Schaal, 2007] Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th International Conference on Machine Learning*, pages 745–750, 2007.

[Rafailov *et al.*, 2023] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Smit *et al.*, 2026] Igor G. Smit, Yaoxin Wu, Pavel Troubil, Yingqian Zhang, and Wim P.M. Nuijten. Neural multi-objective combinatorial optimization for flexible job shop scheduling problems. In *The Fourteenth International Conference on Learning Representations*, 2026.

[Song *et al.*, 2022] Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19(2):1600–1610, 2022.

[Song *et al.*, 2024] Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. Preference ranking optimization for human alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18990–18998, 2024.

[Wang *et al.*, 2023] Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. Flexible job shop scheduling via dual attention network-based reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3091–3102, 2023.

[Waschneck *et al.*, 2018] Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72:1264–1269, 2018.

[Wilcoxon, 1992] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics: Methodology and Distribution*, pages 196–202. Springer, 1992.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[Wirth *et al.*, 2017] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017.

[Xin *et al.*, 2021] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12042–12049, 2021.

[Yuan *et al.*, 2024] Erdong Yuan, Liejun Wang, Shuli Cheng, Shiji Song, Wei Fan, and Yongming Li. Solving flexible job shop scheduling problems via deep reinforcement learning. *Expert Systems with Applications*, 245:123019, 2024.

[Zeng *et al.*, 2022] Zhengqi Zeng, Xiaoxia Li, and Changbo Bai. A deep reinforcement learning approach to flexible job shop scheduling. In *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 884–890, 2022.

[Zhang *et al.*, 2020] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 1621–1632, 2020.

[Zhao and Deng, 2024] Cong Zhao and Na Deng. An actor-critic framework based on deep reinforcement learning for addressing flexible job shop scheduling problems. *Math. Biosci. Eng*, 21(1):1445–1471, 2024.

[Zhao *et al.*, 2016] Zhou Zhao, Hanqing Lu, Deng Cai, Xiaofei He, and Yueting Zhuang. User preference learning for online social recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2522–2534, 2016.

[Zhao *et al.*, 2022] Fuqing Zhao, Tao Jiang, and Ling Wang. A reinforcement learning driven cooperative meta-heuristic algorithm for energy-efficient distributed no-wait flow-shop scheduling with sequence-dependent setup time. *IEEE Transactions on Industrial Informatics*, 19(7):8427–8440, 2022.

[Zhao *et al.*, 2025] Peng Zhao, You Zhou, Di Wang, Zhiguang Cao, Yubin Xiao, Xuan Wu, Yuanshu Li, Hongjia Liu, Wei Du, Yuan Jiang, et al. Dual operation aggregation graph neural networks for solving flexible job-shop scheduling problem with reinforcement learning. In *Proceedings of the ACM on Web Conference*, pages 4089–4100, 2025.