



ARTICLE

A Cross-Domain Meta-Reinforcement Learning Method for the Permutation Flow Shop Scheduling Problem with Sequence-Dependent Setup Times

Bo Wei and Hongfeng Wang*

College of Information Science and Engineering, Northeastern University, Shenyang, China

*Corresponding Author: Hongfeng Wang, Email: hfwang@mail.neu.edu.cn

Received: 06 July 2026; Accepted: 02 September 2026; Published: 22 October 2026

ABSTRACT: The Permutation Flow Shop Scheduling Problem (PFSP) is one of the most classic combinatorial optimization problems in manufacturing systems, with wide applications in semiconductor fabrication, textile processing, and steelmaking. The Permutation Flow Shop Scheduling Problem with Sequence-Dependent Setup Times (PFSP-SDST) is more representative of real-world production, where machine changeovers depend on the sequence of jobs. However, SDST destroys the optimal substructure of standard PFSP, substantially increasing solution difficulty. To address this challenge, we propose a meta-reinforcement learning method based on Model-Agnostic Meta-Learning (MAML) and Evolution Strategy (ES). Meta-training is conducted only on standard PFSP instances to learn general policy initialization parameters, enabling the model to generalize to new SDST instances with only simple fine-tuning at test time without requiring any SDST information during the training phase. Architecturally, a multi-head self-attention encoder captures complex inter-job relationships and a pointer network decoder autoregressively constructs scheduling sequences, with neighborhood search further improving solution quality. Experimental results on the PFSP-SDST benchmark dataset demonstrate that the proposed method achieves an average ARPD of 3.27 across four SDST types and twelve instance scales. It outperforms seven state-of-the-art comparison algorithms across all SDST types and time budgets, achieving a 49.5% reduction against the average ARPD of 6.48 across all baselines, validating its cross-domain generalization capability from standard PFSP to PFSP-SDST.

KEYWORDS: Permutation Flow Shop Scheduling; Sequence-Dependent Setup Times; meta-reinforcement learning; Model-Agnostic Meta-Learning; Evolution Strategy; attention mechanism; cross-domain generalization

1 Introduction

The Permutation Flow Shop Scheduling Problem (PFSP) [1] is one of the most classic production scheduling problems in manufacturing. In this problem, n jobs are processed on m machines in the same order, with the optimization objective of minimizing the maximum completion time (makespan). When $m \geq 3$, PFSP has been proven to be NP-hard, with a solution space size of $n!$ [2]. Although PFSP research has yielded rich results, standard models ignore practical production factors, and their industrial applicability has been questioned [3]. Among these, Sequence-Dependent Setup Times (SDST) is one of the most important practical constraints: before processing different jobs on a machine, setup operations such as mold changes are required, and the setup time depends on the types of consecutive jobs [4]. After introducing SDST, completion time calculation must include setup times, and SDST destroys the optimal substructure of standard PFSP [5]; the classic NEH heuristic [6] requires specialized adaptation to be applied, significantly increasing solution difficulty.

Regarding solution methods, PFSP-SDST has long been dominated by heuristic and metaheuristic approaches [1,3,4], with deep reinforcement learning [7,8] and meta-learning [9,10] methods gradually attracting attention in recent years. However, all of the above methods—whether traditional metaheuristics, DRL, or meta-learning methods—assume that the SDST matrix is fully known during the training or solution phase. More critically, in actual manufacturing systems, when facing new product introductions, equipment status changes, or production process adjustments, the SDST matrix may change with production conditions, making it impossible to enumerate all possible SDST scenarios during training; therefore, models must be trained in standard PFSP environments to handle diverse new SDST instances.

In this context, meta-learning is a particularly suitable paradigm: unlike conventional deep reinforcement learning, which optimizes performance on a single fixed task configuration, it extracts transferable meta-knowledge from source tasks and optimizes the ability to adapt to new tasks with only a few gradient steps. These twin capabilities enable the model to generalize across the vast space of SDST configurations without enumerating them during training, and to handle a new SDST instance with lightweight fine-tuning rather than full retraining.

To address the above challenge, we propose a cross-domain meta-reinforcement learning framework. The core idea is to perform meta-training on standard PFSP (without SDST), learning general scheduling policy initialization parameters, so that the model can rapidly transfer and generalize to new SDST instances with only fine-tuning at test time. This method operates without access to true SDST values during the meta-training phase, achieving cross-domain transfer from standard PFSP to PFSP-SDST.

The main contributions of this work are as follows: (1) A cross-domain meta-learning paradigm is proposed that trains on standard PFSP benchmark instances and transfers to new PFSP-SDST instances. The meta-training phase requires no SDST information, using only standard PFSP instances to learn general scheduling policy initialization; during testing, when facing new instances with unknown SDST, only fine-tuning steps are needed for rapid transfer and generalization. (2) A multi-head self-attention encoder-pointer network architecture is designed for PFSP-SDST, where SDST features are filled with tiny uniform noise during meta-training to maintain parameter responsiveness, and activated with real SDST values during the test fine-tuning phase, enabling the same architecture to operate in both scenarios without structural modification. (3) An ES-based test-time fast fine-tuning mechanism is proposed, requiring only a few steps to adapt meta-parameters to new instances. (4) The cross-domain generalization capability is systematically validated on the PFSP-SDST benchmark dataset.

The remainder of this paper is organized as follows: [Section 2](#) reviews relevant literature, tracing three research threads of traditional PFSP-SDST methods, DRL scheduling methods, and meta-learning methods, identifying the research gap; [Section 3](#) establishes the mathematical programming model of PFSP-SDST; [Section 4](#) elaborates on the proposed meta-reinforcement learning algorithm; [Section 5](#) conducts experimental validation; [Section 6](#) concludes the paper and discusses future directions.

2 Literature Review

This section systematically reviews relevant research from three dimensions: traditional solution methods for PFSP-SDST, applications of DRL in scheduling, and meta-learning with cross-domain generalization, identifying the research gap.

2.1 Traditional Solution Methods for PFSP-SDST

As an important extension of classic PFSP, PFSP-SDST has been predominantly solved by heuristic and metaheuristic approaches of various types since Ruiz et al. [4] first systematically studied it and proposed

various metaheuristic algorithms. Ruiz and Maroto [1] provided a comprehensive review of PFSP heuristics, and Allahverdi [5] conducted a systematic survey of scheduling problems with setup times, providing the theoretical foundation for this field.

In single-objective optimization, Ruiz and Stützle [3] proposed an iterated greedy algorithm for PFSP-SDST, laying the experimental foundation for this field; Li and Zhang [11] proposed an adaptive hybrid genetic algorithm improving solution quality through multiple integrated local search strategies; Sioud and Gagné [12] proposed an enhanced migrating birds optimization algorithm improving solution performance through improved neighborhood search and restart mechanisms. In recent years, learning methods have begun to combine with metaheuristics: Du et al. [13] combined an iterated greedy algorithm with imitation learning, treating jobs as graph nodes, using AFF-GNN encoding and generating expert solutions via critical path IG to guide training, but still requiring the complete SDST matrix to be known in advance; Guo et al. [14] integrated the NEH heuristic into Q-learning, achieving significant progress on the Taillard benchmark set, yet applicable only to standard PFSP.

In multi-objective optimization, Li and Ma [15] proposed a decomposition-based multi-objective discrete artificial bee colony algorithm (MODABC/D), achieving trade-offs among multiple conflicting objectives through decomposition; Jiang and Wang [16] proposed an improved multi-objective evolutionary algorithm based on decomposition, incorporating both energy efficiency and makespan into the optimization. Lv and Wang [17] studied PFSP-SDST with peak power constraints, simultaneously considering learning and deterioration effects. In distributed scenarios, Wang et al. [18] proposed the CMOEA/I multi-objective evolutionary algorithm for group scheduling, Li et al. [19] designed a two-stage IG for flexible assembly scenarios, Song and Lin [20] proposed a genetic programming hyper-heuristic for distributed assembly PFSP-SDST, and Pan et al. [21] designed improved metaheuristics for distributed lot-streaming PFSP.

Although the above methods for PFSP-SDST achieve good optimization performance under different problem settings, their common prerequisite is that SDST is fully known during the solution phase. Once the SDST matrix changes, these methods must be re-executed from scratch. Furthermore, traditional metaheuristic methods require re-optimization from scratch for each instance, lacking knowledge transfer across instances capability.

2.2 Deep Reinforcement Learning in Scheduling

Moreover, DRL has combined the pattern recognition strength of deep learning with the sequential decision-making capability of reinforcement learning, and has achieved significant progress in scheduling by directly mapping scheduling states to dispatching actions in recent years.

In value-function-based DRL methods, Yan et al. [8] addressed dynamic FJSP-SDST using D3QN to train a scheduling agent, designing eight composite rules to guide action selection; Lin et al. [22] proposed two DQN methods for reentrant hybrid flow shop scheduling; Yang et al. [23] designed a DDQN architecture incorporating problem features for dynamic multi-objective PFSP, enabling transfer to large-scale instances after training on small-to-medium scales.

In GNN-based methods, Zhao et al. [7] transformed HFSP into a heterogeneous graph, using HGNN encoding followed by PPO for policy learning; Huang et al. [24] designed a splicing disjunctive graph GNN architecture for distributed job shop scheduling, improving policy quality through probability enhancement mechanisms.

However, the generalization scope of the above DRL methods is strictly limited to the training distribution. These methods assume that training and testing share the same state distribution. Once the test scenario introduces SDST features not present during training, the model tends to fail due to state

distribution shift. Therefore, existing DRL methods cannot handle the requirement of training on standard PFSP and testing on PFSP-SDST.

2.3 Meta-Learning and Cross-Domain Generalization

Meta-learning enables models to rapidly generalize to new tasks with few samples by leveraging experience from a diverse set of training tasks, providing a promising approach to addressing distribution shift problems.

Zhang et al. [10] proposed the MRLM algorithm for the HFSP-LF problem with learning-forgetting effects, achieving automatic selection of search operators through meta-training and Q-learning dual-stage learning. However, the meta-knowledge learned by this method is specific to a particular problem variant and cannot generalize to different problem types. Wu et al. [9] combined MAML with PPO for dynamic FJSP, achieving rapid adaptation through a modular framework and adaptive MDP, but the adaptation scope is limited to instances of the same problem type. Similarly, Chen et al. [25] developed an enhanced MAML-based DRL framework for multi-workflow dynamic scheduling in product design, enabling generalization across different workflow structures. Qin et al. [26] combined Q-learning with quality-diversity algorithms for FJSP with transportation constraints. More recently, Hou et al. [27] proposed a graph meta-reinforcement learning approach for proactive energy-carbon aware scheduling in SMT production lines, combining heterogeneous GNN encoding with a semi-Markov meta-learner.

In distributed scheduling, Wang and Wang [28] proposed the MLIGA algorithm with memory and learning mechanisms for distributed PFSP; Liu et al. [29] designed a Q-learning-based two-stage multi-thread LTMIG algorithm for collaborative scheduling of distributed factories and AGVs. The above meta-learning methods all assume that meta-training and meta-testing share the same problem setting, differing only in instance parameters. Cross-variant transfer from standard PFSP to PFSP-SDST, i.e., the training phase faces standard PFSP instances without SDST while the testing phase faces new instances with unknown SDST, remains an unexplored direction.

2.4 Research Gap and Positioning

Therefore, existing research exhibits three levels of systematic deficiencies: (1) traditional metaheuristic methods rely on SDST being fully known at solution time and require re-optimization for each instance, lacking knowledge transfer across instances capability; (2) DRL methods, although capable of rapidly mapping instances to policies, are strictly limited to the training distribution and cannot handle scenario shifts between training and testing distributions; (3) existing meta-learning methods only support instance-level adaptation within the same problem type and do not touch cross-domain generalization. These deficiencies point to a widely overlooked research gap: when SDST information is unavailable during the training phase, no method can transfer from standard PFSP to PFSP-SDST. This paper proposes a cross-domain meta-reinforcement learning framework to fill this gap. The framework performs meta-training on standard PFSP instances without any SDST information to learn a general policy initialization, and at test time, given a new PFSP-SDST instance with arbitrary SDST, only a few fine-tuning steps are needed to adapt the meta-parameters, enabling the model to rapidly transfer from the standard variant to the SDST-extended variant. Table 1 provides a comparative summary of this work against representative methods from the three categories.

Table 1: Comparison of this study with recent related studies.

Category	Method	Key Feature	Cross-Domain?
Metaheuristic	AHA [11], EMBO [12], IL + AFF-GNN [13], NEH + QL [14], etc.	Require SDST known at solution time; no knowledge transfer across instances	No
DRL	HGNN + DRL [7], DRL + MH [8], DQN [22], DDQN [23], etc.	Limited to training distribution; cannot handle distribution shift	No
Meta-Learning	MRLM [10], MAML + DRL [9,25], GMeta-MATD3 [27], etc.	Instance-level adaptation only within same problem type; no cross-variant transfer	No
Ours	MAML + ES	Cross-variant transfer from standard PFSP to unseen SDST instances with only lightweight fine-tuning	Yes

3 Problem Modeling

The Permutation Flow Shop Scheduling Problem with Sequence-Dependent Setup Times (PFSP-SDST) can be formally described as follows: Given n jobs and m machines, each job must be processed on all machines in the same order (Machine $1 \rightarrow 2 \rightarrow \dots \rightarrow m$). The processing time of job i on machine j is $p_{i,j}$, which is known, fixed, and non-negative. Additionally, when switching from job i to job k on machine j , a setup time $s_{j,i,k}$ is required, which depends on the preceding job i and the current job k . Setup times are separable from processing times, i.e., setup operations can be performed in advance before the job arrives at the machine. Compared to standard PFSP, the key distinction is the introduction of the setup time matrix $s_{j,i,k}$, which must be incorporated into the completion time recursion and destroys the optimal substructure that standard PFSP relies on.

The scheduling objective of PFSP-SDST is to find a job permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ that minimizes the maximum completion time (makespan). Let $C_{\pi_i,j}$ denote the completion time of job π_i on machine j . The optimization objective is:

$$\min C_{max} = C_{\pi_n,m} \quad (1)$$

The calculation of completion times must consider both setup times and processing times. For job π_i at position i on machine j , its start time depends on two factors: the time when machine j completes the previous job and becomes available after setup time, and the completion time of job π_i on the previous machine $j-1$.

Therefore, the completion time formulas for PFSP-SDST are:

$$C_{\pi_1,1} = p_{\pi_1,1} \quad (2)$$

$$C_{\pi_i,1} = C_{\pi_{i-1},1} + s_{1,\pi_{i-1},\pi_i} + p_{\pi_i,1}, \quad i = 2, \dots, n \quad (3)$$

$$C_{\pi_1,j} = C_{\pi_1,j-1} + p_{\pi_1,j}, \quad j = 2, \dots, m \quad (4)$$

$$C_{\pi_i,j} = \max(C_{\pi_{i-1},j} + s_{j,\pi_{i-1},\pi_i}, C_{\pi_i,j-1}) + p_{\pi_i,j}, \quad i = 2, \dots, n; j = 2, \dots, m \quad (5)$$

where n is the number of jobs, m is the number of machines, $p_{i,j}$ is the processing time, $s_{j,i,k}$ is the setup time on machine j when switching from job i to job k , π is the job permutation, and $C_{\pi_i,j}$ is the completion time of job π_i on machine j .

The computational hardness of PFSP-SDST originates from two sources. On one hand, the underlying PFSP is NP-hard for $m \geq 3$ with a solution space of $n!$ permutations, so exhaustive search is intractable even for moderate instances. On the other hand, the sequence-dependent setup matrix $s_{j,i,k}$ makes the completion time of each job depend on the whole preceding sequence, destroying the optimal substructure that standard PFSP relies on. These two sources of hardness jointly explain why existing paradigms are stretched: Metaheuristic algorithms must restart their entire optimization procedure whenever the SDST configuration changes, while deep reinforcement learning algorithms trained on a fixed distribution cannot be expected to perform well when the test-time SDST configuration differs from the training one. The proposed meta-reinforcement learning framework addresses both aspects simultaneously.

4 Algorithm Design

4.1 Overall Framework

We propose a meta-reinforcement learning framework based on Model-Agnostic Meta-Learning (MAML) and Evolution Strategy (ES) for solving the PFSP-SDST problem. The core innovation of this framework is that the meta-training phase requires only standard PFSP benchmark instances (without SDST information), learning parameter initialization for rapid transfer from standard to SDST scenarios; during testing, when facing new instances with arbitrary SDST, only a few ES fine-tuning steps are needed for transfer, enabling meta-training to proceed independently of true SDST values, relying only on placeholder noise to maintain parameter responsiveness.

Problem Setup: During meta-training, the agent faces a series of standard PFSP tasks $\{\mathcal{T}_1, \mathcal{T}_2, \dots\}$ (without SDST); during meta-testing, the agent faces new PFSP-SDST tasks $\{\mathcal{T}'_1, \mathcal{T}'_2, \dots\}$ (with SDST matrices). The task distributions differ between the two phases: training task completion time calculation does not use $s_{j,i,k}$, while testing tasks require it. The objective of meta-learning is to learn a parameter initialization θ_0 such that for a new PFSP-SDST task $\mathcal{T}'$, only K gradient update steps are needed to obtain well-performing task-specific parameters $\theta_{\mathcal{T}'}$.

Three-Component Architecture: The algorithm consists of three core components: a state encoder, a policy network, and a meta-learner. The state encoder employs a multi-head self-attention network, treating each job as a graph node. Job features include processing times, scheduling status, SDST features (or placeholder noise), etc.; complex inter-job relationships are encoded through multiple layers of multi-head attention, and all node embeddings are average-pooled to obtain a graph-level global embedding (Section 4.2.2). The policy network is based on a pointer network architecture, sequentially selecting jobs in an autoregressive manner: at each step, attention scores are computed for unscheduled jobs, selection probabilities are obtained via softmax, and legal permutations are ensured through a masking mechanism (Section 4.2.3). The meta-learner adopts the MAML bi-level optimization framework: the inner loop performs K steps of ES fine-tuning for each training task to obtain task-specific parameters, and the outer loop aggregates post-fine-tuning performance across all tasks to compute meta-gradients and update θ_0 (Section 4.3). Standard policy gradient methods require backpropagation through the entire autoregressive decoding process, which becomes prohibitively expensive under the MAML bi-level optimization framework due to second-order derivative computation. ES is chosen over traditional policy gradient methods such as REINFORCE because ES, as a derivative-free optimization method, directly estimates gradients through parameter-space Gaussian perturbations without maintaining a value network, offering lower memory

overhead and higher parallel efficiency in high-dimensional parameter spaces and large-scale population parallel evaluation.

A natural consideration in designing the meta-learner is the choice of the meta-learning algorithm. MAML is adopted because of several desirable properties. It is model-agnostic and places no requirement on the inner-loop optimizer, which allows the inner loop to be implemented with the ES gradient estimation adopted in this work. Moreover, since the inner loop is updated by ES, the outer-loop meta-gradient is obtained directly from the fine-tuned parameters without second-order derivative computation. Most importantly, MAML explicitly optimizes the post-adaptation performance in the outer loop, which directly matches the goal of learning an initialization from which only a few fine-tuning steps suffice for cross-domain adaptation. These properties are not jointly possessed by other meta-learning variants.

The proposed framework is designed for practical deployment in manufacturing. Meta-training is carried out offline on standard PFSP benchmark instances, and the resulting initialization is then applied to production lots: each time a new batch of products arrives and its SDST configuration is determined, a lightweight ES fine-tuning is triggered for that batch. The optimization thus proceeds on a per-production-lot basis, and the few fine-tuning steps are completed quickly enough to support online real-time optimization on the shop floor. The output of the framework is a complete job permutation that specifies the processing order of all jobs on every machine; this permutation serves directly as the scheduling instruction released to the production execution system for the line.

The training and testing process is outlined in Algorithm 1 and the overall framework is illustrated in Fig. 1: during meta-training on standard PFSP instances, each epoch samples a batch of tasks, performs inner-loop ES fine-tuning, aggregates meta-gradients, and updates θ_0 ; during testing, θ_0 is loaded, a few ES fine-tuning steps (typically 5–10 steps) are performed on PFSP-SDST instances to obtain customized scheduling policies, followed by neighborhood search (Section 4.4) for further optimization. The algorithm involves two key design choices: first, during meta-training, SDST feature slots are filled with tiny uniform noise $\mathcal{U}(10^{-5}, 10^{-4})$ rather than zeros, using noise to maintain parameter responsiveness (ES Parameter Responsiveness) of SDST channels, enabling the network to process real SDST features during testing without structural modification (Section 4.2.1); second, the policy network is vectorized using vmap, enabling all P perturbed policies to be evaluated in parallel, which significantly reduces the time overhead of population evaluation.

Algorithm 1: MAML-ES based meta-learning for PFSP-SDST

```

1: Randomly initialize meta-parameters  $\theta_0$ 
2: for epoch = 1, 2, ...,  $E$  do
3:   Sample a batch of standard PFSP instances  $B$  from  $D_{train}$ 
4:    $g_{meta} \leftarrow 0$ 
5:   for each  $\tau_i$  in  $B$  do
6:      $\theta_i \leftarrow \text{InnerLoopFine-Tuning}(\theta_0, \tau_i, K, P, \sigma, \alpha)$  // Section 4.3
7:      $\pi_{baseline} \leftarrow \text{Decode}(\theta_i, \tau_i)$  // Section 4.2
8:      $F_{baseline} \leftarrow \text{Makespan}(\pi_{baseline}, \tau_i)$ 
9:      $\varepsilon_1, \dots, \varepsilon_P \sim \mathcal{N}(0, I)$ 
10:    for  $p = 1, \dots, P$  do
11:       $\pi_p \leftarrow \text{Decode}(\theta_i + \sigma \cdot \varepsilon_p, \tau_i)$ 
12:       $F_p \leftarrow \text{Makespan}(\pi_p, \tau_i)$ 
13:    end for

```

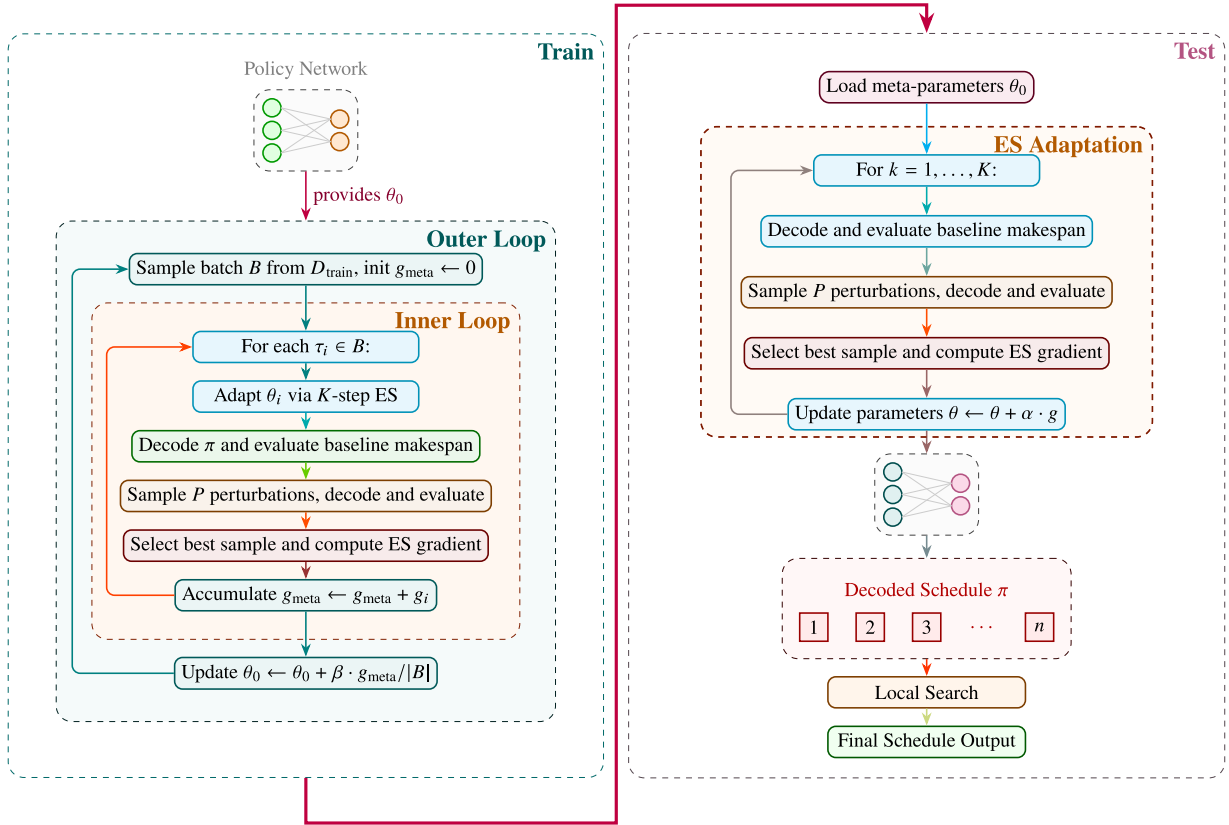
(Continued)

Algorithm 1 (continued)

```

14:    $p^* \leftarrow \arg \min_p F_p$ 
15:    $g_i \leftarrow ((F_{baseline} - F_{p^*})/\sigma) \cdot \varepsilon_{p^*}$ 
16:    $g_{meta} \leftarrow g_{meta} + g_i$ 
17: end for
18:    $\theta_0 \leftarrow \theta_0 + \beta \cdot g_{meta}/|B|$  // outer-loop meta-update
19: end for
20:  $\pi \leftarrow \text{TestTimeFine-Tuning}(\theta_0, \tau_{test})$  // Section 4.4
21: return  $\pi$ 

```

**Figure 1:** Full algorithm architecture.

The following sections detail the design of the three components: [Section 4.2](#) introduces the network architecture, [Section 4.3](#) elaborates on the meta-learning training process, [Section 4.4](#) describes the meta-learning testing process, and [Section 4.5](#) presents the complexity analysis.

4.2 State and Network Design**4.2.1 State Representation and Feature Engineering**

For the PFSP-SDST problem, the state feature representation is designed as follows. For each job i , its feature vector f_i contains the following components:

(1) Processing Time Features: $p_{i,1}, p_{i,2}, \dots, p_{i,m}$, the processing times of job i on m machines, normalized.

(2) Scheduling Status Feature: A binary indicator variable indicating whether job i has already been scheduled.

(3) Total Processing Time Feature: $\sum_{j=1}^m p_{i,j}$, normalized.

(4) Earliest Possible Start Time Feature: For each machine j , compute the earliest possible start time of job i , considering precedence constraints and machine availability, normalized.

(5) SDST Dynamic Feature: This feature enables the model to handle cross-domain adaptation. During the meta-testing phase (PFSP-SDST environment), for each machine j , extract the normalized setup time $s_{j,l_j,i}$ from the last processed job to job i (let l_j be the last job processed on machine j , initially $l_j = -1$, in which case $s_{j,l_j,i} = 0$); during the meta-training phase (standard PFSP environment), real setup times are absent, and the SDST feature slot cannot be filled with zeros. Under a constant zero input, the ES gradient estimate for the connected weights merely reflects the randomly sampled perturbation component without meaningful information. The feature slot is therefore filled with uniformly distributed tiny random noise $\mathcal{U}(10^{-5}, 10^{-4})$ instead of zeros, so that the network can immediately adapt when real SDST data replaces the noise at test time.

Combining the above features, the feature dimension for each job i is $3m + 2$, where processing times contribute m dimensions, scheduling status contributes 1 dimension, total processing time contributes 1 dimension, earliest start time contributes m dimensions, and SDST features (or meta-training placeholder noise) contribute m dimensions.

The state features are organized as a 3D tensor of shape $[batch_size, n_jobs, feature_dim]$, where $batch_size$ corresponds to the meta-batch size during meta-training and to the population size during population evaluation.

4.2.2 Multi-Head Self-Attention Encoder

We use a multi-head self-attention network as the encoder. Each job is treated as a node in a graph, with a fully connected graph among jobs, allowing all job pairs to interact directly. Attention weights are adaptively learned by the model, without requiring predefined edge connections. Compared to the fixed-weight aggregation of Graph Convolutional Networks (GCN), self-attention adaptively assigns different importance to different neighbor nodes through learnable attention weights, aligning with the asymmetry of inter-job relationships in PFSP-SDST. The encoder-decoder architecture is illustrated in Fig. 2.

The encoder consists of L multi-head attention layers stacked together, each layer using self-attention mechanisms to capture inter-job relationships.

Multi-Head Attention Mechanism: Given input features $X \in \mathbb{R}^{n \times d_{in}}$, multi-head attention is computed as:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O \quad (6)$$

where h is the number of attention heads, and $W^O \in \mathbb{R}^{hd_v \times d}$ is the output projection matrix that maps the concatenation of h attention heads back to embedding dimension d .

Each attention head and the attention function are respectively defined as:

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V) \quad (7)$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \quad (8)$$

where $d_k = d/h$ is the key/query dimension per attention head, and $\sqrt{d_k}$ is the scaling factor to prevent excessively large dot products that would cause softmax gradient vanishing. In this context, $Q = XW_i^Q$ (query), $K = XW_i^K$ (key), and $V = XW_i^V$ (value) represent different linear projections of the same set of job features. QK^T measures the similarity between each pair of job features—jobs with similar processing time patterns receive higher attention scores; when SDST features are activated during testing, job pairs with smaller setup times are also assigned higher weights. V carries the actual content information of job features. After softmax normalization, the attention weights are multiplied with V to achieve adaptive aggregation of inter-job relationship information.

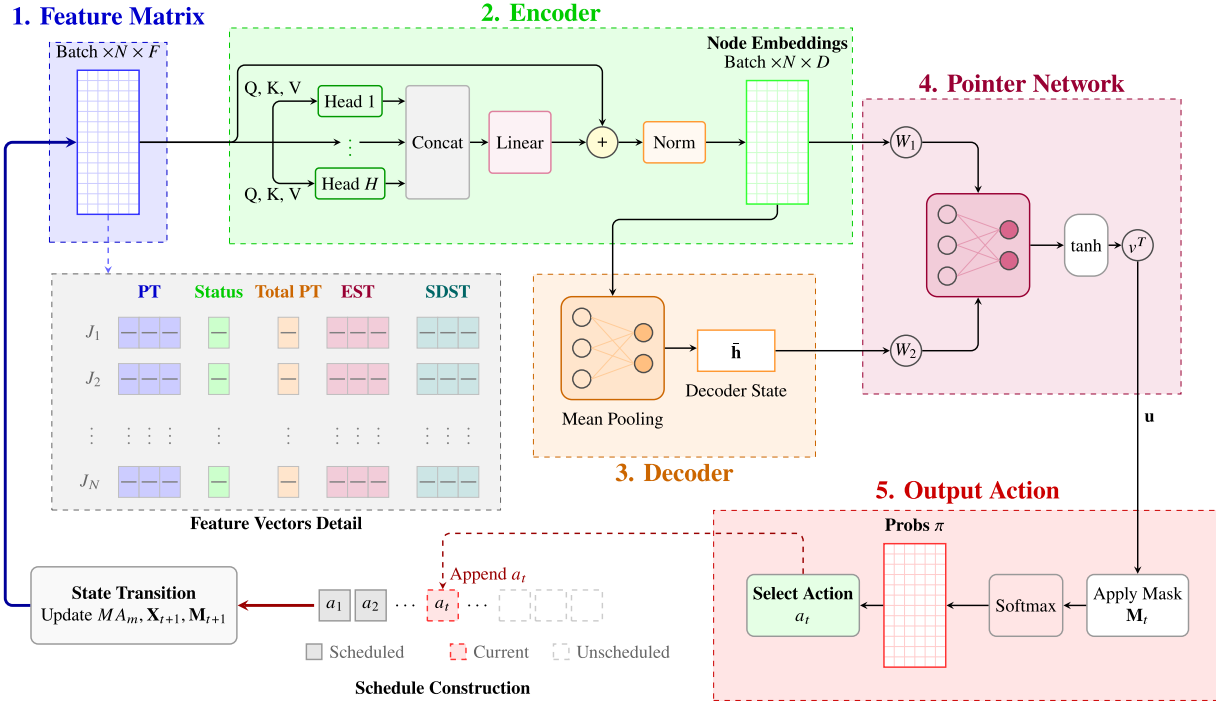


Figure 2: Self-attention encoder-pointer network decoder architecture.

Encoder Architecture: The encoder contains L attention layers, each consisting of a multi-head attention sublayer and a feed-forward network sublayer, using residual connections and batch normalization (BatchNorm):

$$H^{(l)} = \text{BatchNorm}(H^{(l-1)} + \text{MultiHead}(H^{(l-1)})) \quad (9)$$

$$H^{(l)} = \text{BatchNorm}(H^{(l)} + \text{FFN}(H^{(l)})) \quad (10)$$

where $H^{(l)} \in \mathbb{R}^{n \times d}$ represents the job node embedding matrix of the l -th encoder layer (d is the embedding dimension), FFN is a two-layer fully connected feed-forward network (with ReLU activation in the hidden layer), and BatchNorm is the batch normalization layer. $H^{(0)}$ is the input feature, and $H^{(L)}$ is the final node embedding. Additionally, the encoder outputs a graph-level embedding (average pooling over all node embeddings) to represent the global scheduling state.

4.2.3 Pointer Network Decoder

The decoder adopts a Pointer Network architecture, sequentially generating the scheduling sequence. At each decoding step, the pointer network selects the next job to schedule based on the current state.

Decoder State: The graph embedding h_{graph} output by the encoder serves as the decoder state.

Attention Mechanism: The decoder computes pointer scores for each candidate job using an attention mechanism:

$$u_i = v^T \tanh(W_1 \cdot e_i + W_2 \cdot h_{graph}), \quad i \in \text{unscheduled} \quad (11)$$

where e_i is the encoder output embedding of job i (i.e., the i -th row of $H^{(L)}$), h_{graph} is the graph-level global embedding output by the encoder, serving as the decoder state of the pointer network. Since the state feature matrix X changes after each job selection, the self-attention encoder is re-run at each step, so both node embeddings e_i and graph embedding h_{graph} are dynamically updated at each step; W_1, W_2, v are learnable parameters.

Masking Mechanism: To avoid repeatedly selecting already scheduled jobs, a masking mechanism sets the scores of scheduled jobs to negative infinity:

$$u_i = \begin{cases} u_i, & \text{if job } i \text{ is unscheduled} \\ -\infty, & \text{otherwise} \end{cases} \quad (12)$$

Selection Probability: Softmax is applied to the pointer scores to obtain selection probabilities:

$$p_i = \frac{\exp(u_i)}{\sum_{j \in \text{unscheduled}} \exp(u_j)} \quad (13)$$

where p_i is the probability of job i being selected as the next scheduled job, and u_i is the pointer score of job i . During decoding, the job with the largest p_i is selected.

Decoding Process: The decoder constructs the scheduling sequence in an autoregressive manner. Starting from an empty sequence, at each step the pointer network computes attention scores over all unscheduled jobs using the current encoder embeddings and the global graph embedding as the decoder state. The scores are normalized via softmax to form a probability distribution, from which the next job is selected. After each selection, the state features are updated to reflect the new scheduling status—removing the chosen job from the candidate set and recomputing the earliest start times of remaining jobs based on the partial schedule. This process repeats until a complete permutation of length n is obtained. The complete encode-decode procedure is detailed in Algorithm 2.

Algorithm 2: Policy network encode-decode

```

1:  $\pi \leftarrow []$ 
2: while  $|\pi| < n$  do
3:    $H^{(0)} \leftarrow X$  // current state feature matrix
4:   for  $l = 1, \dots, L$  do
5:      $H \leftarrow \text{BatchNorm}(H + \text{MultiHead}(H))$ 
6:      $H \leftarrow \text{BatchNorm}(H + \text{FFN}(H))$ 
7:   end for
```

(Continued)

Algorithm 2 (continued)

```

8:    $e_1, \dots, e_n \leftarrow H^{(L)}$ 
9:    $h_{graph} \leftarrow \text{MeanPool}(e_1, \dots, e_n)$ 
10:  for  $i \in \text{unscheduled}$  do
11:     $u_i \leftarrow v^T \cdot \tanh(W_1 \cdot e_i + W_2 \cdot h_{graph})$     // pointer score
12:  end for
13:   $u_i \leftarrow -\infty$  for  $i \in \text{scheduled}$     // hard masking
14:   $p \leftarrow \text{softmax}(u)$ ;  $i^* \leftarrow \arg \max(p)$ 
15:   $\pi.\text{append}(i^*)$ 
16:   $X \leftarrow \text{UpdateFeatures}(X, i^*)$     // update state features
17: end while
18: return  $\pi$ 

```

4.3 Meta-Learning Training Process**4.3.1 MAML Framework and Cross-Domain Learning**

This section elaborates on the MAML-based cross-domain meta-learning training process, including ES gradient estimation, inner/outer loop optimization, and the complete training workflow.

Inspired by [30], we adopt Model-Agnostic Meta-Learning (MAML) as the meta-learning framework and embed it into the ES optimization framework for cross-domain scheduling. Compared to standard MAML, our meta-training is conducted on standard PFSP (without SDST), while the testing phase faces PFSP-SDST instances. The core idea of this cross-domain meta-learning is to learn a parameter initialization that can rapidly fine-tune from the standard scenario to the extended scenario.

In practice, each new batch of products brings a different SDST configuration, and retraining the model from scratch for every such batch requires repeating the full training procedure, which is prohibitively expensive in real manufacturing environments. Meta-learning addresses this issue by accumulating general scheduling knowledge during meta-training without any SDST information and embedding it into the initialization parameters θ_0 . Specifically, the outer-loop objective explicitly optimizes the post-adaptation performance across a distribution of training instances, so the learned θ_0 resides in a parameter region from which only a few fine-tuning steps are sufficient to reach high-quality solutions. Consequently, when a new batch of products with a different SDST configuration arrives and its SDST information becomes available, lightweight ES fine-tuning on the meta-learned initialization rapidly adapts the policy to the newly observed SDST data and generalizes to the new configuration without retraining from scratch.

MAML consists of two levels of optimization:

Inner-Loop Fine-Tuning: For each training instance τ_i (standard PFSP), starting from θ_0 , perform K steps of ES fine-tuning to obtain fine-tuned parameters θ'_i :

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \alpha \cdot \nabla_{\theta} F_i(\theta_i^{(k)}) \quad (14)$$

where $\theta_i^{(0)} = \theta_0$, α is the inner-loop learning rate, and F_i is the fitness function on instance τ_i .

Outer-Loop Meta-Update: Across all training instances, compute meta-gradients based on the fine-tuned parameters θ'_i , and update the meta-parameters:

$$\theta_0 \leftarrow \theta_0 - \beta \cdot \nabla_{\theta_0} \left(\frac{1}{|B|} \sum_{\tau_i \in B} F_i(\theta'_i) \right) \quad (15)$$

where β is the meta-learning rate. Since the inner loop adopts ES gradient estimation (derivative-free optimization), the outer-loop meta-gradients are directly estimated through ES based on the inner-loop fine-tuned parameters, without requiring second-order derivatives.

The construction principle of the integrated framework is that MAML and ES play complementary roles. MAML provides the bi-level optimization structure that explicitly targets fast adaptation, optimizing the post-adaptation performance rather than the performance at the initial parameters themselves. ES provides a derivative-free gradient estimator that operates directly on the makespan objective, so that the inner-loop fine-tuning does not require backpropagation through the scheduling process. Consequently, the outer-loop meta-gradient is obtained directly from the inner-loop fine-tuned parameters without second-order derivatives, which is a natural advantage of this integration over gradient-based MAML variants.

Building on this integration, three structural modifications distinguish the proposed framework from the standard MAML formulation. First, the task distribution is deliberately constructed across domains: meta-training is performed on standard PFSP instances without any SDST information, while meta-testing faces PFSP-SDST instances, deviating from the standard MAML assumption that training and testing tasks share the same distribution. Second, the SDST feature slots are filled with tiny uniform noise during meta-training to maintain parameter responsiveness, so that the network can process real SDST inputs at test time without structural modification. Third, a first-improvement local search is integrated after the test-time fine-tuning to further refine the solution. Together, these modifications constitute the algorithmic innovations of this work.

4.3.2 ES Gradient Estimation

Evolution Strategy (ES) is the gradient estimator for both inner/outer loop optimization in the meta-learning framework of this paper. Unlike value-function-based policy gradient methods such as REINFORCE, ES directly estimates gradients through parameter-space perturbations, without maintaining a value network, offering lower memory overhead and higher parallel efficiency in high-dimensional parameter spaces and large-scale population parallel evaluation. This section sequentially introduces the ES gradient estimation principle, population-level parallel implementation, and the best-sample estimation strategy.

Given policy network parameters θ and a PFSP-SDST instance, the fitness function $F(\theta)$ on that instance is defined as directly equal to the makespan value generated by the scheduling policy (the optimization objective is to minimize F):

$$F(\theta) = \text{Makespan}(\pi_\theta) \quad (16)$$

where π_θ denotes the scheduling sequence (permutation) generated by decoding after loading the policy network with parameters θ , and $\text{Makespan}(\pi_\theta)$ is the maximum completion time of that sequence on the given instance.

ES estimates gradients by sampling parameter perturbations. Specifically, sample P perturbation vectors $\varepsilon_1, \dots, \varepsilon_P \sim \mathcal{N}(0, I)$ from a standard normal distribution, generating perturbed parameters $\theta_p = \theta + \sigma \cdot \varepsilon_p$. Then evaluate the fitness $F(\theta_p)$ of each perturbed parameter on that instance. To avoid the absolute fitness value affecting the gradient direction, we adopt centered fitness, using the baseline makespan $F(\theta)$ under the current parameters to compute relative advantage. To improve population evaluation efficiency, this paper adopts vmap vectorized mapping to achieve population-level parallel evaluation. The policy network is vectorized using `torch.func.vmap`, enabling forward propagation for all parameters in the population simultaneously, with the environment rollout phase using a batch size equal to the population size. This

parallelization design significantly reduces the time overhead of population evaluation, with particularly notable effects under GPU acceleration.

This paper adopts the best-sample-based gradient estimation strategy in both training and testing, i.e., using only the single best-performing member of the population for gradient estimation:

$$\nabla_{\theta} F \approx \frac{F_{p^*} - F(\theta)}{\sigma} \cdot \varepsilon_{p^*}, \quad (17)$$

where p^* is the index of the individual with the smallest makespan, $F(\theta)$ is the baseline makespan under the current parameters. Since $F_{p^*} - F(\theta)$ is the makespan difference between the best perturbed solution and the current solution, the parameter update direction is consistent with the best perturbation ε_{p^*} , ensuring movement toward reducing makespan. This strategy only needs to compute and save the gradient estimate of the best perturbation direction, avoiding the additional computational overhead of full-population averaging.

4.3.3 Cross-Domain Training Procedure

The specific procedure of meta-training is shown in Algorithm 3.

Algorithm 3: Inner-loop ES fine-tuning

```

1:  $\theta \leftarrow \theta_0$ 
2: for  $k = 1, 2, \dots, K$  do
3:    $F_{baseline} \leftarrow \text{Makespan}(\text{Decode}(\theta, \tau), \tau)$  // baseline fitness
4:    $\varepsilon_1, \dots, \varepsilon_P \sim \mathcal{N}(0, I)$  // sample perturbations
5:   for  $p = 1, \dots, P$  do
6:      $\theta_p \leftarrow \theta + \sigma \cdot \varepsilon_p$ 
7:      $\pi_p \leftarrow \text{Decode}(\theta_p, \tau)$ 
8:      $F_p \leftarrow \text{Makespan}(\pi_p, \tau)$  // evaluate fitness
9:   end for
10:   $p^* \leftarrow \arg \min_p F_p$ 
11:   $g \leftarrow ((F_{baseline} - F_{p^*})/\sigma) \cdot \varepsilon_{p^*}$ 
12:   $\theta \leftarrow \theta + \alpha \cdot g$  // update parameters
13: end for
14: return  $\theta$ 

```

During meta-training, the following strategies are adopted: (1) Random Instance Sampling: In each epoch, standard PFSP instances are sampled with replacement from the training set, preventing bias introduced by fixed ordering. (2) Scale-Grouped Training: Since different $n \times m$ scales correspond to different feature dimensions and network parameter counts, the training data are grouped by scale, and a separate model is trained for each group independently. (3) Best Model Preservation: The fitness is tracked throughout training, and the historically best model parameters are saved.

4.4 Meta-Learning Testing Process

After meta-training is completed, for a new PFSP-SDST instance, rapid online fine-tuning of the meta-parameters θ_0 is performed through ES gradient steps, followed by neighborhood search to further improve solution quality. The test-time fine-tuning procedure is illustrated in Algorithm 4: after loading θ_0 , each step samples P perturbation vectors, generates scheduling sequences via decoding, and evaluates the resulting

makespan values; the best-performing perturbation is selected to compute the centered gradient $((F_{baseline} - F_{p^*})/\sigma) \cdot \varepsilon_{p^*}$, which is then used to update the parameters. After K fine-tuning steps, the final parameters are decoded to obtain the scheduling solution.

After the ES fine-tuning loop, a first-improvement local search is applied to the decoded schedule. Starting from the current permutation, the algorithm explores the insert neighborhood: each job is removed from its position and reinserted at every other position, and the first move yielding a makespan improvement is accepted. Upon reaching a local optimum, a perturbation that randomly swaps 5% of the job pairs is applied to escape the current basin of attraction, and the search restarts. This process continues until the time budget is exhausted, and the best schedule found is returned as the final solution.

Algorithm 4: Test-time ES fine-tuning

```

1:  $\theta \leftarrow \theta_0$ 
2: for  $k = 1, 2, \dots, K_{test}$  do
3:    $F_{baseline} \leftarrow \text{Makespan}(\text{Decode}(\theta, \tau_{test}), \tau_{test})$ 
4:    $\varepsilon_1, \dots, \varepsilon_P \sim \mathcal{N}(0, I)$ 
5:   for  $p = 1, \dots, P$  do
6:      $\theta_p \leftarrow \theta + \sigma \cdot \varepsilon_p$ 
7:      $\pi_p \leftarrow \text{Decode}(\theta_p, \tau_{test})$ 
8:      $F_p \leftarrow \text{Makespan}(\pi_p, \tau_{test})$ 
9:   end for
10:   $p^* \leftarrow \arg \min_p F_p$ 
11:   $g \leftarrow ((F_{baseline} - F_{p^*})/\sigma) \cdot \varepsilon_{p^*}$ 
12:   $\theta \leftarrow \theta + \alpha \cdot g$ 
13: end for
14:  $\pi \leftarrow \text{Decode}(\theta, \tau_{test})$ 
15:  $\pi \leftarrow \text{LocalSearch}(\pi)$  // optional LS
16: return  $\pi$ 

```

4.5 Complexity Analysis

The proposed method consists of encoding, decoding, and makespan evaluation. Due to the dynamic encoding mechanism for SDST state transitions, the self-attention encoder is invoked at each of the n autoregressive steps. A single forward pass of the encoder requires $\mathcal{O}(L \cdot (n^2 \cdot d + n \cdot d^2))$; repeated n times, the cumulative encoder complexity becomes $\mathcal{O}(n \cdot L \cdot (n^2 \cdot d + n \cdot d^2))$. The pointer network decoder requires $\mathcal{O}(n^2 \cdot d + n \cdot d^2)$, and the makespan evaluation costs $\mathcal{O}(n \cdot m)$. Since L and d are fixed model hyperparameters, and the number of machines m does not exceed the number of jobs n in the adopted benchmark instances, the encoder term $\mathcal{O}(L \cdot n^3 \cdot d)$ dominates the overall complexity. Therefore, the overall time complexity of the proposed method is $\mathcal{O}(L \cdot n^3 \cdot d)$.

5 Experiments

5.1 Experimental Setup

Experiments were conducted on a cloud GPU server (GPU: NVIDIA GeForce RTX 3090 $\times$ 1, 24 GB VRAM, CUDA 12.1; CPU: AMD EPYC 7742; 62 GB RAM; Ubuntu 20.04; PyTorch 2.1.0). Standard Taillard PFSP instances [31] were used for meta-training, and the PFSP-SDST benchmark dataset proposed by Ruiz et al. [4] was used for testing. The evaluation metric is the Relative Percentage Deviation (RPD), defined as $\text{RPD} = (C_{\text{alg}} - C_{\text{best}})/C_{\text{best}} \times 100\%$, where C_{best} is the best known makespan for each instance. The Average

Relative Percentage Deviation (ARPD) is the arithmetic mean of RPD across all instances. Runtime is controlled by parameter $\rho \in \{30, 60, 90\}$, with the time budget set to $\rho \cdot n \cdot m$ milliseconds.

5.2 Parameter Calibration

Bayesian optimization was performed using the TPE sampler in the Optuna framework, with makespan minimization as the objective over 50 trials. The search ranges are listed in Table 2.

Table 2: Hyperparameter search ranges.

Parameter	Search Range
σ (noise_std)	0.15, 0.20, 0.25, 0.30, 0.35
K (inner_steps)	3, 4, 5, 6, 7
B (batch_size)	1, 2, 4, 8, 16
β (meta_step_size)	0.005, 0.01, 0.015, 0.02, 0.025
α (inner_step_size)	0.005, 0.01, 0.015, 0.02, 0.025

The main effects plot for each parameter is shown in Fig. 3. The makespan varies smoothly around the optimum for all parameters, indicating robust hyperparameter sensitivity. The final configuration is $\sigma = 0.20$, $K = 5$, $B = 8$, $\beta = 0.01$, and $\alpha = 0.01$.

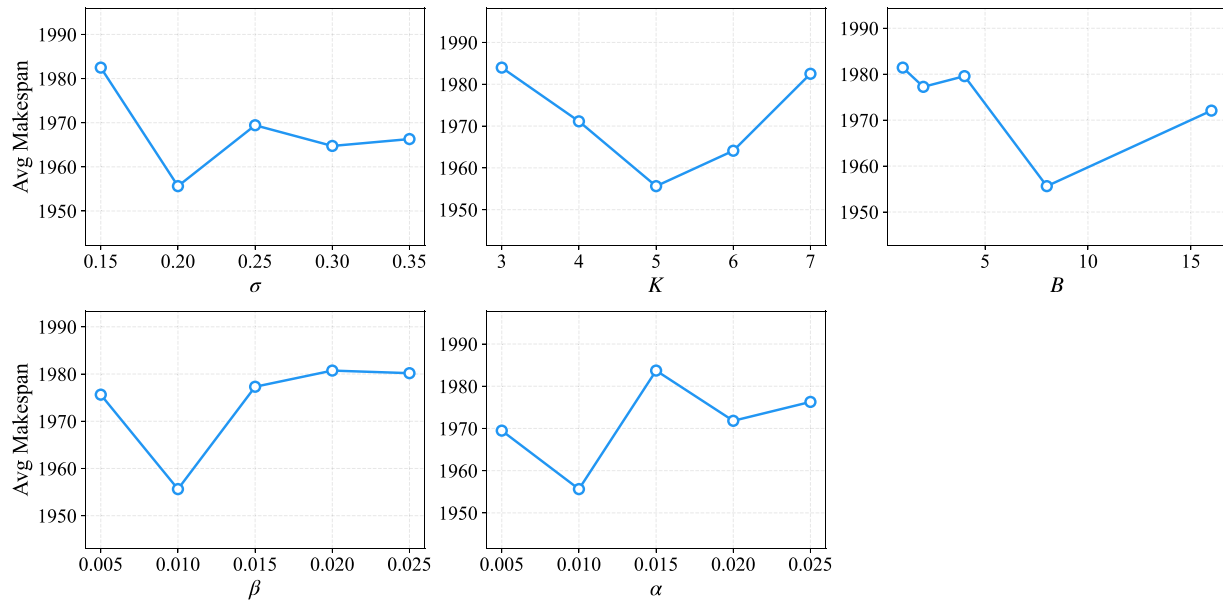


Figure 3: Parameter main effects plot.

5.3 Ablation Experiments

To verify the effectiveness of each component, four ablation variants were designed: (1) No Meta-Learning: Remove the MAML bi-level structure and directly train with a DRL method end-to-end on PFSP-SDST; (2) No Fine-Tuning: Skip the test-time fine-tuning phase and directly use the meta-trained initialization parameters for scheduling; (3) No Multi-Head Attention: Replace the multi-head self-attention encoder with a per-node MLP, eliminating inter-job relationship modeling; (4) No Local Search: Remove the

local search post-processing step from the test-time pipeline, using only the ES-fine-tuned policy network to directly generate schedules.

Fig. 4 presents the RPD distributions of the four ablation variants and the full model across 4 SDST types and 12 instance scales. The median RPD of all four ablation variants is higher than that of the full model.

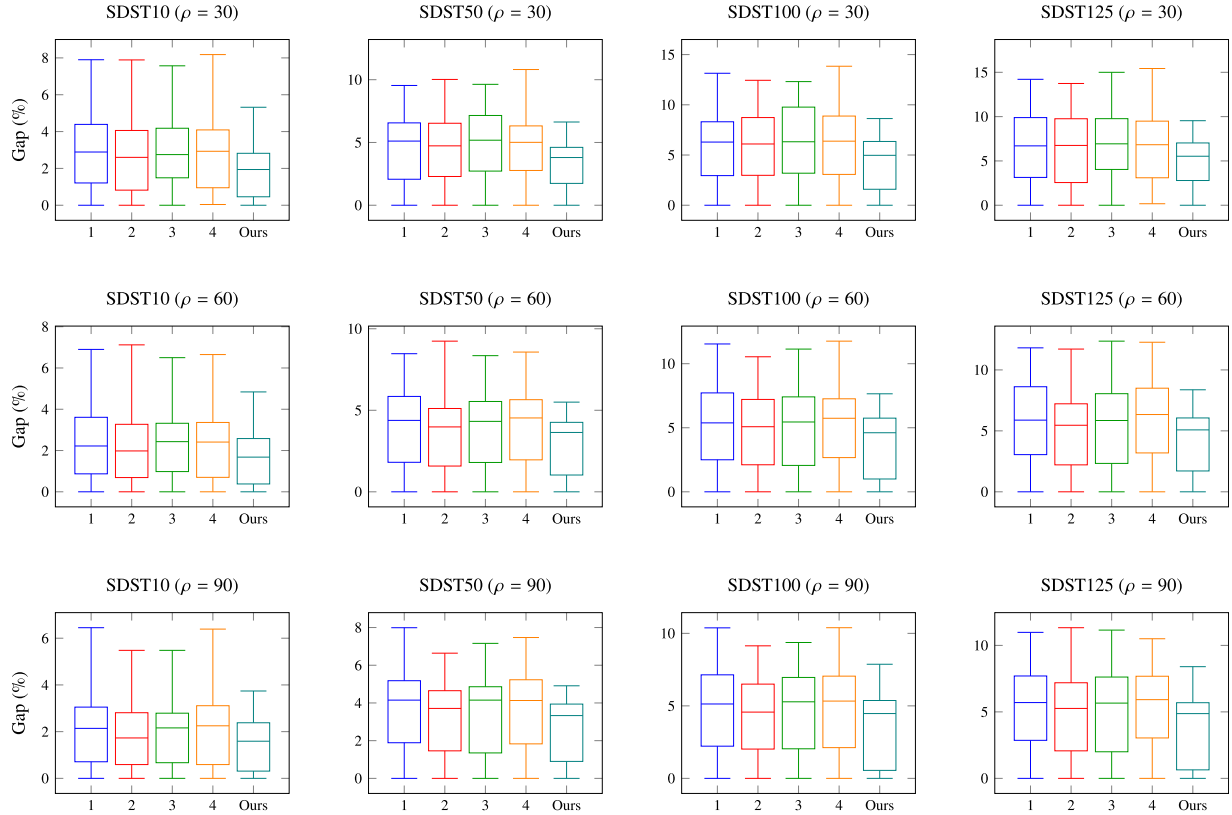


Figure 4: Ablation experiment box plots (12 subplots, 4 SDST $\times$ 3 ρ).

The statistical results of the ablation experiments are shown in Table 3, where Mean ARPD is averaged across three ρ values and p -values are computed via the Wilcoxon Signed-Rank test over 480 instances. All p -values are below $p < 0.001$ at $\alpha = 0.05$, confirming that the performance differences between the full model and each ablation variant are statistically significant.

Table 3: Ablation experiment statistical results (480 instances).

Variant	Mean ARPD	p ($\rho = 30$)	p ($\rho = 60$)	p ($\rho = 90$)
Full Model (Ours)	3.27	—	—	—
No Meta-Learning	4.22	$p < 0.001$	$p < 0.001$	$p < 0.001$
No Fine-Tuning	4.52	$p < 0.001$	$p < 0.001$	$p < 0.001$
No Multi-Head Attention	4.49	$p < 0.001$	$p < 0.001$	$p < 0.001$
No Local Search	4.55	$p < 0.001$	$p < 0.001$	$p < 0.001$

As shown in Table 3, all four ablation variants exhibit significantly higher ARPD than the full model ($p < 0.001$), confirming the effectiveness of each component. The No Local Search variant shows the largest

degradation, demonstrating the decisive contribution of neighborhood search post-processing to solution quality. The No Fine-Tuning variant shows the second largest degradation, indicating that online parameter fine-tuning provides substantial gains beyond the meta-trained initialization alone. The No Multi-Head Attention variant further validates that the multi-head self-attention mechanism is critical for inter-job relationship modeling. The No Meta-Learning variant validates the core role of MAML bi-level optimization in cross-domain generalization.

5.4 Comparative Experiments

Furthermore, five recently published representative algorithms and two additional meta-learning-based algorithms were selected for comparison to evaluate the proposed method against the state of the art. All comparison algorithms were configured with their originally recommended parameters and run under identical termination conditions ($\rho = 30/60/90$). The basic information of each algorithm is listed in Table 4.

Table 4: Comparison algorithm information.

Algorithm	Type	Core Strategy
DABC [32]	Metaheuristic	ABC + Metropolis + dynamic neighborhood (insert/swap/two-opt)
EGA [33]	Metaheuristic	NEH init + Iterated Greedy + multi-start
RLMA [34]	RL + Metaheuristic	Sarsa (λ) local search selection + cooperative crossover + memetic
HGNN-DRL [35]	GNN + DRL	Heterogeneous GNN (HGNN) + PPO
PPO-GAT [36]	GNN + DRL	Graph Attention Network (GAT) + PPO + multi-signal differential reward
MDQTS [37]	Meta-RL	MAML dual-loop meta-learning + Deep Q-Network (DenseNet) + online fine-tuning
MetaPNet [38]	Meta-RL	Meta-learning REINFORCE + Pointer Network + beam search + active search

The first five algorithms were selected according to two criteria. First, they are recently published representative algorithms covering the two mainstream solution approaches in scheduling: metaheuristic methods (DABC, EGA, RLMA) and deep reinforcement learning methods (HGNN-DRL, PPO-GAT). Second, all five algorithms have demonstrated competitive performance on scheduling benchmarks in their original publications. Note that none of these algorithms was originally designed for PFSP-SDST. For the metaheuristic baselines, the sequence-dependent setup times are incorporated into the makespan evaluation of each candidate solution. For the DRL baselines, which were originally designed for related flow shop and job shop scheduling problems, the SDST information is encoded into the state features of their network inputs, and the models are retrained on the PFSP-SDST instances.

In addition, two meta-learning-based algorithms, MDQTS and MetaPNet, were included to further verify the generalization-oriented comparison: both are meta-trained on standard PFSP instances and then adapted to the test PFSP-SDST instances via online fine-tuning, following their original meta-learning paradigms. For the five baselines, the comparison is conducted under an asymmetric setting: the meta-heuristic baselines (DABC, EGA, RLMA) directly solve the PFSP-SDST instances with complete SDST

information, and the DRL baselines (HGNN-DRL, PPO-GAT) are trained on PFSP-SDST instances with SDST features fully available. In contrast, the proposed method performs meta-training exclusively on standard PFSP instances without any SDST information, and only lightweight ES fine-tuning is performed on the test instances.

Fig. 5 presents the ARPD comparison of the eight algorithms across three ρ values and four SDST types using grouped bar charts. Ours achieves the lowest ARPD under all settings, and the gap with traditional metaheuristics (DABC, EGA, RLMA) narrows as ρ increases.



Figure 5: ARPD grouped bar chart for 8 algorithms across 3 ρ values and 4 SDST types.

To verify the statistical significance of the performance differences, the Friedman test [39] was conducted at each ρ value separately. The Friedman statistics are $\chi^2 = 2191.23$ ($\rho = 30$), $\chi^2 = 2201.59$ ($\rho = 60$), and $\chi^2 = 2226.21$ ($\rho = 90$), all reaching extremely significant levels ($p < 0.001$), confirming systematic differences among the algorithms. To further quantify the performance gap, Table 5 reports the per-rho ARPD mean and 95% confidence interval for each algorithm, estimated from 480 instances using the normal approximation. Confidence intervals that do not overlap with Ours indicate a statistically significant difference at the $\alpha = 0.05$ level.

As shown in Table 5, each cell reports the mean ARPD followed by the 95% confidence interval in brackets, estimated from 480 instances using the normal approximation. The confidence interval of Ours does not overlap with those of any comparison algorithm at any ρ value, confirming that Ours significantly outperforms all competitors across all tested time budgets.

Table 5: Per-Rho ARPD mean and 95% confidence interval (480 instances).

Algorithm	$\rho = 30$	$\rho = 60$	$\rho = 90$
Ours	3.60 [3.37, 3.83]	3.21 [3.00, 3.41]	3.00 [2.80, 3.19]
MDQTS	7.89 [7.46, 8.32]	5.96 [5.62, 6.30]	5.16 [4.86, 5.45]
MetaPNet	5.70 [5.44, 5.96]	5.70 [5.44, 5.95]	5.74 [5.48, 5.99]
PPO-GAT	4.75 [4.44, 5.06]	3.98 [3.73, 4.23]	3.64 [3.41, 3.87]
HGNN-DRL	5.35 [5.03, 5.68]	4.64 [4.35, 4.92]	4.27 [4.01, 4.54]
RLMA	9.87 [9.31, 10.43]	7.83 [7.37, 8.29]	7.36 [6.93, 7.79]
EGA	9.52 [8.98, 10.05]	7.38 [6.94, 7.83]	6.56 [6.17, 6.95]
DABC	9.04 [8.68, 9.39]	8.17 [7.84, 8.51]	7.62 [7.30, 7.94]

6 Conclusion and Future Work

This paper proposes a cross-domain meta-reinforcement learning method based on MAML and ES for the PFSP-SDST. Meta-training is conducted exclusively on standard PFSP instances, enabling rapid generalization to unseen SDST instances with only lightweight fine-tuning. Unlike existing meta-learning approaches that assume identical problem settings for meta-training and meta-testing, this work establishes a cross-variant transfer paradigm for scheduling.

The current method has several limitations, including the substantial computational cost of meta-training on multiple instance scales, the fixed number of fine-tuning steps, and the exclusive focus on single-objective makespan optimization. Future work will therefore address these issues by developing adaptive fine-tuning strategies, extending the method to multi-objective scenarios considering energy consumption and tardiness, and generalizing to more complex scheduling variants such as distributed and flexible flow shops.

Acknowledgement: None.

Funding Statement: This work was supported in part by the Fundamental Research Funds for the Central Universities in China under Grant N2404004, in part by the National Natural Science Foundation of China under Grant No. 72271048.

Author Contributions: Study conception and design: Bo Wei; data collection: Bo Wei; analysis and interpretation of results: Bo Wei and Hongfeng Wang; draft manuscript preparation: Bo Wei; review and editing: Hongfeng Wang. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: Given his role as Editorial Board Member of this journal, Hongfeng Wang had no involvement in the peer review of this article and had no access to information regarding its peer review. Full responsibility for the editorial process for this article was delegated to another journal editor. The authors declare no conflicts of interest.

References

1. Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. *Eur J Oper Res.* 2005;165(2):479–94. doi:10.1016/j.ejor.2004.04.017.
2. Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. *Math Oper Res.* 1976;1(2):117–29. doi:10.1287/moor.1.2.117.
3. Ruiz R, Stützle T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *Eur J Oper Res.* 2008;187(3):1143–59. doi:10.1016/j.ejor.2006.07.029.
4. Ruiz R, Maroto C, Alcaraz J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *Eur J Oper Res.* 2005;165(1):34–54. doi:10.1016/j.ejor.2004.01.022.
5. Allahverdi A. The third comprehensive survey on scheduling problems with setup times/costs. *Eur J Oper Res.* 2015;246(2):345–78. doi:10.1016/j.ejor.2015.04.004.
6. Nawaz M, Ensore EE Jr, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega.* 1983;11(1):91–5. doi:10.1016/0305-0483(83)90088-9.
7. Zhao Y, Luo X, Zhang Y. The application of heterogeneous graph neural network and deep reinforcement learning in hybrid flow shop scheduling problem. *Comput Ind Eng.* 2024;187:109802. doi:10.1016/j.cie.2023.109802.
8. Yan B, Liu X, Lu S, Hu C, Wang X, Zhou Z. Deep reinforcement learning approach for a dynamic flexible job shop problem with sequence dependent setup times. *Comput Ind Eng.* 2025;207(2):111310. doi:10.1016/j.cie.2025.111310.
9. Wu L, Li X, Lu X, Feng Z, Jing Y. An adaptive meta-reinforcement learning framework for dynamic flexible job shop scheduling. *IEEE Trans Autom Sci Eng.* 2025;22(10):24036–52. doi:10.1109/TASE.2025.3627282.
10. Zhang Z, Shao Z, Shao W, Chen J, Pi D. MRLM: a meta-reinforcement learning-based metaheuristic for hybrid flow-shop scheduling problem with learning and forgetting effects. *Swarm Evol Comput.* 2024;85(1):101479. doi:10.1016/j.swevo.2024.101479.
11. Li X, Zhang Y. Adaptive hybrid algorithms for the sequence-dependent setup time permutation flow shop scheduling problem. *IEEE Trans Autom Sci Eng.* 2012;9(3):578–95. doi:10.1109/TASE.2012.2192729.
12. Sioud A, Gagné C. Enhanced migrating birds optimization algorithm for the permutation flow shop problem with sequence dependent setup times. *Eur J Oper Res.* 2018;264(1):66–73. doi:10.1016/j.ejor.2017.06.027.
13. Du ZS, Li JQ, Song HN, Gao KZ, Xu Y, Li JK, et al. Solving the permutation flow shop scheduling problem with sequence-dependent setup time via iterative greedy algorithm and imitation learning. *Math Comput Simul.* 2025;234(20):169–93. doi:10.1016/j.matcom.2025.02.026.
14. Guo D, Liu S, Ling S, Li M, Jiang Y, Li M, et al. The marriage of operations research and reinforcement learning: integration of NEH into Q-learning algorithm for the permutation flowshop scheduling problem. *Expert Syst Appl.* 2024;255(8):124779. doi:10.1016/j.eswa.2024.124779.
15. Li X, Ma S. Multiobjective discrete artificial bee colony algorithm for multiobjective permutation flow shop scheduling problem with sequence dependent setup times. *IEEE Trans Eng Manag.* 2017;64(2):149–65. doi:10.1109/TEM.2016.2645790.
16. Jiang ED, Wang L. An improved multi-objective evolutionary algorithm based on decomposition for energy-efficient permutation flow shop scheduling problem with sequence-dependent setup time. *Int J Prod Res.* 2019;57(6):1756–71. doi:10.1080/00207543.2018.1504251.
17. Lv DY, Wang JB. Considering the peak power consumption problem with learning and deterioration effect in flow shop scheduling. *Comput Ind Eng.* 2024;197(4):110599. doi:10.1016/j.cie.2024.110599.
18. Wang Y, Han Y, Wang Y, Pan QK, Wang L. Sustainable scheduling of distributed flow shop group: a collaborative multi-objective evolutionary algorithm driven by indicators. *IEEE Trans Evol Comput.* 2024;28(6):1794–808. doi:10.1109/TEVC.2023.3339558.
19. Li YZ, Meng LL, Zhang B. Two-stage iterated greedy algorithm for distributed flexible assembly permutation flowshop scheduling problems with sequence-dependent setup times. *AIMS Math.* 2025;10(5):11488–513. doi:10.3934/math.2025523.
20. Song HB, Lin J. A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times. *Swarm Evol Comput.* 2021;60:100807. doi:10.1016/j.swevo.2020.100807.

21. Pan Y, Gao K, Li Z, Wu N. Improved meta-heuristics for solving distributed lot-streaming permutation flow shop scheduling problems. *IEEE Trans Autom Sci Eng.* 2023;20(1):361–71. doi:10.1109/TASE.2022.3151648.
22. Lin CC, Peng YC, Chang YS, Chang CH. Reentrant hybrid flow shop scheduling with stockers in automated material handling systems using deep reinforcement learning. *Comput Ind Eng.* 2024;189(3):109995. doi:10.1016/j.cie.2024.109995.
23. Yang YY, Qian B, Hu R, Li Z, Zhang ZQ, Jin HP, et al. Deep reinforcement learning algorithm incorporating problem characteristics for dynamic multi-objective permutation flow-shop scheduling problem. *Swarm Evol Comput.* 2025;96(3):101973. doi:10.1016/j.swevo.2025.101973.
24. Huang JP, Gao L, Li XY. An end-to-end deep reinforcement learning method based on graph neural network for distributed job shop scheduling problem. *Expert Syst Appl.* 2024;238(2):121756. doi:10.1016/j.eswa.2023.121756.
25. Chen Z, Zhang L, Cai W, Laili Y, Wang X, Wang F, et al. Multi-workflow dynamic scheduling in product design: a generalizable approach based on meta-reinforcement learning. *J Manuf Syst.* 2025;79(7):334–46. doi:10.1016/j.jmsy.2025.01.010.
26. Qin H, Xiang Y, Liu F, Han Y, Wang Y. Enhancing quality-diversity algorithm by reinforcement learning for flexible job shop scheduling with transportation constraints. *Swarm Evol Comput.* 2025;93(2):101849. doi:10.1016/j.swevo.2025.101849.
27. Hou L, Shen W, Ge L, Shi Y, Su Y. Proactive energy-carbon aware production scheduling in SMT lines: a novel graph meta-reinforcement learning approach. *J Manuf Syst.* 2026;88(3):333–49. doi:10.1016/j.jmsy.2026.06.009.
28. Wang B, Wang H. An iterated greedy algorithm with memory and learning mechanisms for the distributed permutation flow shop scheduling problem. *Comput Mater Contin.* 2025;82(1):371–88. doi:10.32604/cmc.2024.058885.
29. Liu Z, Sang H, Zhang B, Meng L, Meng T. A learning-based two-stage multi-thread iterated greedy algorithm for co-scheduling of distributed factories and automated guided vehicles with sequence-dependent setup times. *IEEE Trans Emerg Top Comput Intell.* 2025;9(3):2208–18. doi:10.1109/TETCI.2025.3540405.
30. Qing H, Zhang M, Wu Y, Huang W, Yang J, Song W, et al. Instance-wise adaptive scheduling via derivative-free meta-learning. In: *Proceedings of the Fourteenth International Conference on Learning Representations (ICLR)*; 2026 Apr 23–27; Rio de Janeiro, Brazil.
31. Taillard E. Benchmarks for basic scheduling problems. *Eur J Oper Res.* 1993;64(2):278–85. doi:10.1016/0377-2217(93)90182-M.
32. Wang J, Lei D, Li D, Li X, Tang H. A dynamic artificial bee colony for fuzzy distributed energy-efficient hybrid flow shop scheduling with batch processing machines. *J Manuf Syst.* 2025;78:94–108. doi:10.1016/j.jmsy.2024.10.019.
33. Li QY, Pan QK, Wang L, Gao L, Li WM. Dynamic cascaded flow-shop scheduling using an evolutionary greedy algorithm. *IEEE Trans Evol Comput.* 2025;29(5):1490–501. doi:10.1109/TEVC.2025.3541959.
34. Wang Y, Han Y, Wang Y, Wang X, Liu Y, Gao K. Reinforcement learning-assisted memetic algorithm for sustainability-oriented multiobjective distributed flow shop group scheduling. *IEEE Trans Syst Man Cybern Syst.* 2025;55(4):2399–413. doi:10.1109/TSMC.2024.3518625.
35. Song W, Chen X, Li Q, Cao Z. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Trans Ind Inform.* 2023;19(2):1600–10. doi:10.1109/TII.2022.3189725.
36. Liu Y, Fan J, Shen W. A deep reinforcement learning approach with graph attention network and multi-signal differential reward for dynamic hybrid flow shop scheduling problem. *J Manuf Syst.* 2025;80(1):643–61. doi:10.1016/j.jmsy.2025.03.028.
37. Song P, Chen H, Cui K, Wang J, Shi D. Meta-learning for dynamic multi-robot task scheduling. *Comput Oper Res.* 2025;182(1):107109. doi:10.1016/j.cor.2025.107109.
38. Kim H, Barde S. A meta-learning enhanced deep reinforcement learning approach for generalizing across orienteering problem with time windows. *Transp Res Part C Emerg Technol.* 2026;183(6):105450. doi:10.1016/j.trc.2025.105450.
39. Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J Am Stat Assoc.* 1937;32(200):675–701. doi:10.1080/01621459.1937.10503522.