2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Learning Flexible Job Shop Scheduling under Limited Buffers and Material Kitting Constraints

Shishun Zhang[1], Juzhan Xu[3], Yidan Fan[1], Chenyang Zhu[1], Ruizhen Hu[3], Yongjun Wang[1], Kai Xu[2,*]
[1]National University of Defense Technology [2]Institute of AI for Industries, Chinese Academy of Sciences
[3]Shenzhen University [*]Corresponding Author

***Abstract*— The Flexible Job Shop Scheduling Problem (FJSP) originates from real production lines, while some practical constraints are often ignored or idealized in current FJSP studies, among which the limited buffer problem has a particular impact on production efficiency. To this end, we study an extended problem that is closer to practical scenarios—the Flexible Job Shop Scheduling Problem with Limited Buffers and Material Kitting. In recent years, deep reinforcement learning (DRL) has demonstrated considerable potential in scheduling tasks. However, its capacity for state modeling remains limited when handling complex dependencies and long-term constraints. To address this, we leverage a heterogeneous graph network within the DRL framework to model the global state. By constructing efficient message passing among machines, operations, and buffers, the network focuses on avoiding decisions that may cause frequent pallet changes during long-sequence scheduling, thereby helping improve buffer utilization and overall decision quality. Experimental results on both synthetic and real production line datasets show that the proposed method outperforms traditional heuristics and advanced DRL methods in terms of makespan and pallet changes, and also achieves a good balance between solution quality and computational cost. Furthermore, a supplementary video is provided to showcase a simulation system that effectively visualizes the progression of the production line.**

I. Introduction

The Flexible Job-Shop Scheduling Problem (FJSP) is a core optimization challenge in modern manufacturing and has received increasing research attention in recent years [1], [2]. The main challenge lies in simultaneously optimizing the operation sequence and machine allocation decisions. As the number of machines and operations grows, the solution space becomes vast, making it a strong NP-hard problem. Recently, with the continuous development of Operations Research (OR) and Machine Learning (ML) fields, FJSP has gradually been addressed with improved solution methods.

However, the standard FJSP often overlooks complex resource constraints prevalent in real-world production lines. In high-mix scenarios such as steel plate processing and part sorting, parts must be temporarily stored in a limited number of buffer zones (e.g., pallets) under strict material kitting rules, where each pallet can only accommodate parts of the same category [3]. This results in a part-sorting bottleneck: insufficient pallets for diverse part types lead to frequent pallet changes, causing congestion and efficiency losses. This gives rise to the problem, we define the Flexible Job-Shop Scheduling Problem with Limited Buffers and Material Kitting (**FJSP-LB-MK**). The core challenge lies in balancing two objectives: optimizing the operation-to-machine assignment for compact scheduling, while consecutively scheduling jobs containing similar part types (i.e., steel plates) to maximize pallet utilization and reduce pallet switching caused by category mismatches.

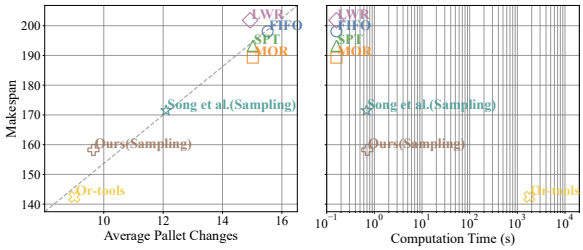


Fig. 1. Comparison of the makespan, pallet changes, and computation time (the closer to the bottom and left, the better). Our method establishes an effective balance between solution quality and computational cost.

There are numerous methods for solving FJSP and its variants. Traditional methods, such as constraint programming, rely on centralized search methods, and the computational time cost becomes prohibitive as the problem scales. Heuristic and metaheuristic algorithms often rely on manually designed rules, which leads to a lack of generalization. In recent years, Deep Reinforcement Learning (DRL) has demonstrated considerable promise in solving the standard FJSP and a range of combinatorial optimization problems [4]–[6]. It supports end-to-end decision-making with second-level inference time and consistently competitive performance, owing to its long-term reward mechanism in sequential decision-making and its ability to autonomously learnb effective heuristics for optimization tasks [7]. However, when faced with complex practical constraints in production line scenarios, existing DRL methods show significant limitations. Most of these methods rely on simplified state representations and struggle to capture long-term, non-local state dependencies arising from shared resources (e.g., pallets). As a result, the agent cannot anticipate how current decisions actually affect the availability of shared resources, leading to poor performance in scenarios under complex constraints.

To overcome these limitations, we build upon the strengths of DRL in long-sequence decision-making and leverage a heterogeneous graph network for global state modeling in the presence of buffer constraints. By constructing the dependency among different instances within the graph, we enable efficient message passing of critical information. During this process,

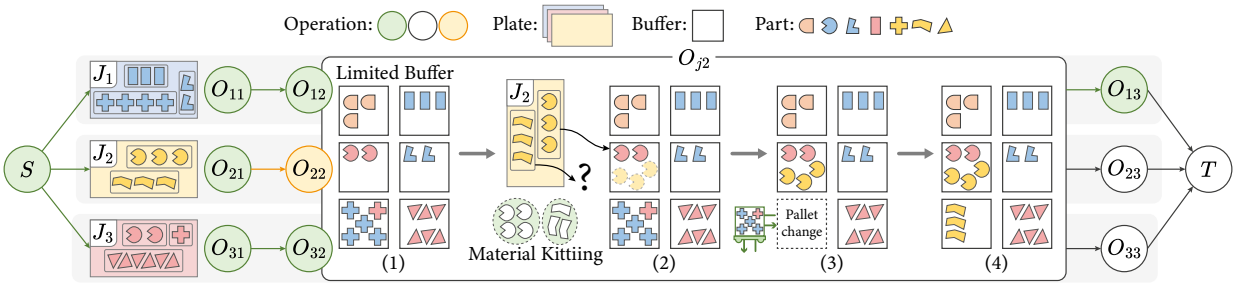


Fig. 2. FJSP with limited buffer and material kitting constraints. Green, white, and yellow circles indicate scheduled, unscheduled, and currently being scheduled operations. Parts of the same job share color; identical shapes denote the same part type. Operation O_{22} is subject to limited buffer and kitting constraints: (1) Six pallets are available, each preloaded with parts; (2) Job J_2's parts must be split across two pallets, but one type is not in existing categories; (3) A pallet must be replaced with an empty one; (4) Once an empty pallet is available, all parts can be properly assigned.

we give higher attention to decisions that might lead to costly pallet changes, providing the decision network with reliable state feature embeddings. Ultimately, the model learns a scheduling preference that inherently incorporates material kitting logic, decomposing complex global constraints into learnable local signals on the graph, thereby enhancing state representation in complex dynamics and contributing to overall decision making.

Our main contributions are as follows:

1) We are the first to address the FJSP problem under buffer constraints based on a DRL framework, demonstrating superior performance over baseline methods.
2) We effectively leverage and enhance a heterogeneous graph neural network (HGNN), enabling a more precise construction of state dependencies, allowing the model to perceive the costs introduced by pallet change (switch) and thereby focus on high-cost operations. As a result, the network produces reliable state feature embeddings that lead to improved decision performance.
3) We validate the proposed methods and multiple baseline approaches through experiments on synthetic and real production-line datasets. Experimental results across these datasets show that our method achieves a favorable balance between performance and computational efficiency, as depicted in Figure 1.

II. Related Works

A. Traditional Approaches for FJSP

Traditional approaches to FJSP include exact methods, heuristics, and metaheuristics. Exact methods such as Mixed Integer Linear Programming (MILP) [8] and Constraint Programming [9] can obtain optimal solutions for small-scale instances but become computationally prohibitive as problem size increases. Heuristic methods, such as Priority Dispatching Rules (PDRs) like FIFO [10], generate fast feasible solutions but suffer from myopic decisions and limited generalization due to manually designed rules. Metaheuristics, including Genetic Algorithms [11] and Differential Evolution [12], search for high-quality solutions but often struggle to balance solution quality and computational efficiency.

B. DRL Approaches for FJSP

DRL improves solution efficiency for FJSP through end-to-end policy learning. For example, [13] reduced action complexity using operation–machine pairs, while [14] integrated Graph Neural Networks (GNNs) with DRL to enhance PDRs. [15] employed heterogeneous graphs with attention mechanisms for state encoding, and [16] combined multi-policy generation with Proximal Policy Optimization (PPO) [17] to handle large-scale instances. However, existing DRL approaches still struggle to construct expressive state representations and adequately capture complex state dependencies.

C. FJSP Problems Under Constraints

Real-world FJSP often involves additional constraints. For example, [18] considered fixture constraints using a hybrid genetic algorithm. Limited buffer zones for work-in-progress storage have also been studied, as [19] optimized scheduling with a public buffer, and [20] proposed a hybrid differential evolution algorithm for multi-objective scheduling under buffer limitations. However, material kitting—a critical requirement in many production environments—remains largely unexplored. To fill this gap, we investigate the FJSP-LB-MK problem by jointly incorporating limited buffer and material kitting constraints into scheduling optimization.

III. Problem Statement

The FJSP-LB-MK is a variant of the classical FJSP that incorporates a more practical buffer constraint. An instance of the FJSP is defined by a set of jobs $J = \{J_1, J_2, ..., J_n\}$, and a set of machines $M = \{M_1, M_2, ..., M_m\}$. Each job $J_i \in J$ concludes a set of operations $O_i = \{O_{i1}, O_{i2}, ..., O_{il}\}$, and each operation O_{ij} can be processed on machines from a subset $M_{ij} \subset M$. The time taken by machine M_k to process operation O_{ij} is denoted as $p_{ijk} \in \mathbb{R}^+$. The most common objective function used in the FJSP is the Makespan, denoted as C_{max}, which is defined as:

$$C_{\max} = \max C_{ij}, \tag{1}$$

where C_{ij} represents the completion time of operation O_{ij} for job J_i. In addition to the classical FJSP, the FJSP-LB-MK introduces two key constraints: limited buffer and material kitting, as shown in Figure 2. Limited buffer means there

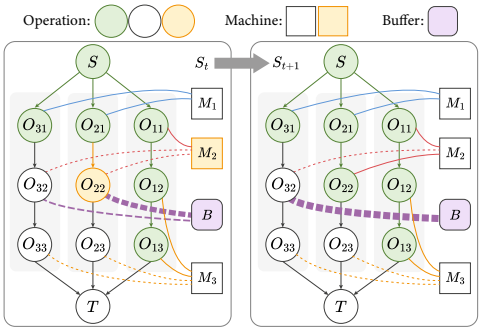


Fig. 3. Example of state transition. At timestep t, both unscheduled operations O_{32} and O_{22} are subject to buffer constraints; therefore, the buffer node is connected to both of them. The algorithm then prepares to execute the $O_{22} - M_2$ action. Upon execution, the graph transitions to state s_{t+1}. The newly scheduled operation node is marked green, and its associated connections are updated accordingly.

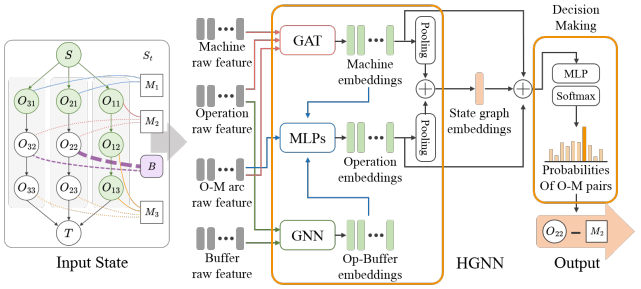


Fig. 4. Heterogeneous GNN uses multiple state features as input, and outputs the O-M pair decision.

is a limited number of buffer zones (each zone contains a pallet) available to store parts from each steel plate (the "job" in FJSP-LB-MK). Material kitting means each pallet can hold only parts of a single category (parts from different steel plates but having the same category can be grouped together).

Let $B = \{B_1, B_2, ..., B_k\}$ be the set of k buffer zones (pallets). The current total part categories in the buffer zone can be represented by $P_b = [p_1, p_2, ..., p_c]$. For a newly arrived steel plate J_i at the part-sorting operation, the part categories of which is given by $P_{J_i} = [p_1, p_2, ..., p_{J_i}]$, for parts in J_i belonging to existing categories of buffer zone, they are directly assigned to their corresponding pallets, for new part categories, they are assigned to the empty pallets. If new categories exceed the number of empty pallet, one or more pallets currently storing parts need to be moved to the warehouse and replaced with a new empty pallet to accommodate the new category of parts. The single pallet change time is defined as t_{switch}, the number of exceeded new part categories is N_{excess}, and since only one pallet can be changed at a time, the total pallet change time is,

$$T_{\text{replace}} = N_{\text{excess}} \times t_{\text{switch}}. \tag{2}$$

IV. Deep Reinforcement Learning for FJSP-LB-MK

A. MDP Formulation

Under the DRL framework, the scheduling process is typically modeled as a Markov Decision Process (MDP). At each decision step, the RL agent receives the current global state and selects an operation-machine pair as the scheduling action. This action triggers a state transition, and the updated state becomes the input for the next decision. The process continues until all operations are scheduled.

1) State: As shown in Figure 3, we use a heterogeneous graph $\mathcal{H}$ to represent the global state. This directed graph consists of multiple node and edge types. For state feature design, we primarily follow [15]. For operation features, we adopt the 6-D scheduling features in [15], and additionally introduce part category one-hot features **Type** $\in \mathbb{R}^T$, where T denotes the number of part categories, to represent the part types associated with the job of the operation, a binary indicator **PS** $\in \mathbb{R}$ to distinguish part-sorting operations, and **SwEst** $\in \mathbb{R}$ to estimate the pallet change count based on the current state. For buffer features, we use a concatenated vector to represent the status of all buffers, which includes: 1) part category one-hot features **Type** $\in \mathbb{R}^T$ to capture the part types stored in all buffers and 2) occupancy rate $\in \mathbb{R}$ to indicate the overall utilization of the buffers. As the scheduling progresses, the connections of edges and associated state features of these nodes will dynamically change.

2) Action: Our approach integrates operation selection and machine assignment into a compound decision. At each decision point t, the action space A_t consists of all eligible operation-machine pairs—where an operation O_{ij} is eligible once its predecessor completes and a machine M_k is idle. This action space shrinks over time. The set of operations in $A(t)$ is called the candidate operation set, $J_c(t)$.

3) Transition: As illustrated in Figure 3, after executing action a_t, the environment transitions to a new state s_{t+1}, updating the relevant operation and machine sets. The completed operation node and its associated edges are removed, and the features and connections of machine, operation, and buffer nodes are refreshed to reflect the updated schedule. For part-sorting operations with buffer constraints, a dedicated execution process is triggered. Specifically, these operations enter a *Pallet Change* module, which determines whether pallet changes are necessary and how many are required.

4) Reward: Our reward function has two components: the estimated change in makespan and the change in pallet changes, both based on the differences between states s_t and s_{t+1}. This dual-component reward guides the network toward learning a policy that minimizes makespan while taking into account pallet changes. The reward is defined as follows:

$$\begin{aligned} r(s_t, a_t, s_{t+1}) = C_{\max}(s_t) - C_{\max}(s_{t+1}) \\ +\lambda(P_{\max}(s_t) - P_{\max}(s_{t+1})), \end{aligned} \tag{3}$$

where $P_{\max}(s_t)$ represents the total pallet changes at time step t, λ is a weight factor to link the two objectives.

B. Heterogeneous GNN

To effectively extract and represent the global state information of FJSP-LB-MK, we draw on and enhance the HGNN proposed by [15], and the enhanced network architecture is illustrated in Figure 4. Specifically, the enhanced HGNN comprises three main stages: Machine Embedding,

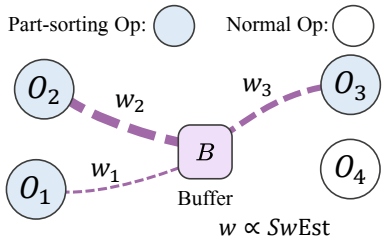


Fig. 5. Message passing between the buffer and the part-sorting operations, w is the weight of the edge

Operation-Buffer Embedding, and Operation Embedding. For the Machine Embedding and Operation Embedding, we mainly follow the original setting of HGNN. The Operation-Buffer Embedding is a novel module we introduced to effectively propagate the current state information of buffers to operations, and incorporate the new Operation-Buffer feature to the Operation Embedding.

For Operation-Buffer Embedding, as shown in Fig. 3, the heterogeneous graph of FJSP-LB-MK incorporates a unique buffer node B, connected exclusively to all part-sorting operation nodes to convey the buffer state information at each time step. To further inform the network of potential pallet change costs, we employ a weighted GNN message-passing mechanism, which propagates buffer features to its neighboring part-sorting operation nodes via weighted edges (Fig. 5). The edge weight is set directly proportional to the estimated number of pallet changes (switches) **SwEst** required for a specific operation O_{ij} at each time step:

$$w_{ij} = \text{sigmoid}(\alpha \cdot SwEst), \tag{4}$$

where α is a scaling factor (0.3 in our setting). The Operation-Buffer feature of $O_{ij} \in \mathcal{N}_t(B)$ is then computed as:

$$\delta_{ij} = w_{ij} B. \tag{5}$$

In this way, operations with higher anticipated pallet change costs receive a more heavily weighted message from the buffer during feature aggregation.

We highlight the key design choices in this component:

1) Selective connectivity: Instead of connecting the buffer node B to all operation nodes, we only connect it to the part-sorting operation nodes. This is because a GNN aggregates information from neighboring nodes, and indiscriminately broadcasting buffer features to all operations—regardless of their relevance—would not only introduce noise but also weaken the ability of the network model to identify which operations should attend to the pallet state. To demonstrate the advantages of Selective connectivity, we evaluate and compare various connectivity strategies between buffer and operation nodes in the ablation studies.

2) Cost-sensitive propagation: Instead of employing uniform message broadcasting, we dynamically modulate propagation through edge weights that are positively correlated with the estimated pallet change (switch) cost (**SwEst**). This mechanism, referred to as the *cost-avoiding* strategy, encourages the network model to prioritize decisions associated with higher switch costs. In contrast, we also investigate a *benefit-seeking* strategy, where edge weights are negatively correlated with **SwEst**, thereby guiding the network model toward decisions with lower immediate switch costs. Our ablation study confirm the superiority of the *cost-avoiding* strategy. We attribute this to the fact that benefit-seeking may lead to locally optimal but short-sighted choices, neglecting downstream constraints and creating bottlenecks. By contrast, cost-avoiding encourages the model to proactively mitigate high-cost decisions, achieving better global optimization and more robust scheduling performance.

TABLE I

PARAMETERS OF THE SYNTHETIC FJSP-LB-MK DATASETS

Size ($n \times m$)	n_i [1]	$\lvert\mathcal{M}_{ij}\rvert$ [2]	$\bar{p}_{ij}$ [3]	n_{ps} [4]	c_j [5]	C [6]	t_p [7]	t_r [8]	P [9]
10×5	U(4, 6)	U(1, 5)	U(1, 20)	1	U(3, 5)	10	2	5	6
20×5	U(4, 6)	U(1, 5)	U(1, 20)	1	U(3, 5)	10	2	5	6
15×10	U(8, 12)	U(1, 10)	U(1, 20)	1	U(3, 5)	10	2	5	6
20×10	U(8, 12)	U(1, 10)	U(1, 20)	1	U(3, 5)	10	2	5	6
30×10	U(8, 12)	U(1, 10)	U(1, 20)	1	U(3, 5)	10	2	5	6
40×10	U(8, 12)	U(1, 10)	U(1, 20)	1	U(3, 5)	10	2	5	6

[1] Number of operations in Job J_i; [2] Number of compatible machines for operation O_{ij}; [3] Average processing time of operation O_{ij}; [4] Number of part-sorting operations per job; [5] Number of part categories on each job; [6] Total number of part categories; [7] Part placement time (sec); [8] Pallet replacement time (sec); [9] Number of available pallets.

V. EXPERIMENTS

A. Dataset

To evaluate the performance of our proposed algorithm, we conducted a series of experiments on synthetic datasets of varying scales and real production line datasets.

1) Synthetic Dataset: Similar to most studies on FJSP, synthetic FJSP-LB-MK instances were generated for model training and testing, with the generation procedure following the methodology outlined in [21]. We considered six distinct problem scales, ranging from 10 jobs $\times$ 5 machines to 40 jobs $\times$ 10 machines. Taking into account the specific characteristics of the FJSP-LB-MK problem, we further customized these synthetic instances with modifications:

- Within the set of operations for each job, certain operations were designated as **part-sorting** operations. These operations are only executable by specific machines.
- A subset of machines was designated exclusively for executing part-sorting operations.
- For each problem scale, several variant instances were created based on the number of machines and operations.

As shown in Table I, the first four columns represent the original FJSP parameters, while the last five columns correspond to the extended FJSP-LB-MK parameters, including part sorting, pallet change, and other features. These parameters were adaptively adjusted with the problem scale.

2) Real Production Line Dataset: We constructed a real production-line dataset by collecting comprehensive data related to steel plate processing from four industrial production lines. In these production lines, each steel plate (job) J undergoes a sequence of operations. For each operation O_i, the factory information system automatically

TABLE II

PARAMETERS OF THE REAL PRODUCTION LINE DATASET

Size ($n \times m$)	n_i	$\lvert\mathcal{M}_{ij}\rvert$	n_{ps}	c_j	C	t_p	t_r	P
A: 20×16	8	$\{5:1, 1:3, 2:4\}$	1	1-11	18–47	14	90	18
B: 20×12	9	$\{8:1, 1:4\}$	3	1–12	16–38	14	180	48
C: 20×10	9	$\{8:1, 1:2\}$	3	1–9	17–28	14	180	24
D: 20×12	11	$\{10:1, 1:2\}$	4	1–8	26–55	14	90	20

records processing-related information. Combined with a dedicated processing-time calculation program, this enables the computation of the required processing time p_{ik} when the operation is executed on an eligible machine M_k. For each production line, we sampled 10,000 instances for training, 100 for validation, and 100 for testing. Each instance represents a production segment consisting of 20 jobs (steel plates). The key parameters for the four production lines are shown in Tables II. Unlike the uniform sampling of the synthetic dataset, the $|\mathcal{M}_{ij}|$ metric of the dataset reflects actual machine-operation mappings.

Additionally, part categories and their counts are derived from production lines, providing actual values for c_j and C, resulting in greater variability than in the synthetic dataset, and making the production line dataset a more rigorous benchmark for testing an algorithm's adaptability to buffer congestion. In this work, we idealize the production environment by not accounting for real-world disruptions such as machine failures or dynamic order insertions.

B. Implementation Details

Algorithm 1 Details of Training Procedure with PPO

Require: Initial heterogeneous GNN, policy network (actor π_θ), and value network (critic v_ϕ); total iterations I

Sample B instances
for iter $= 1$ to I **do**
  for all $b = 1$ to B **in parallel do**
    Set initial state s_t for instance b
    while s_t not terminal **do**
      Get embeddings from network
      Select $a_t \sim \pi_\theta(s_t)$; observe r_t, s_{t+1}
      $s_t \leftarrow s_{t+1}$
    end while
    Estimate advantage A_t and compute loss $\mathcal{L}$
    Update θ, ϕ, ω for R epochs
  end for
  if iter $\bmod 10 = 0$ **then**
    Validate policy
  end if
  if iter $\bmod 20 = 0$ **then**
    Resample instances
  end if
end for
return Updated model parameters.

1) Details of training: All training and experiments were performed on a device equipped with an NVIDIA GeForce RTX 3080 Ti GPU and an Intel Core i9-10980XE CPU. We use Proximal Policy Optimization (PPO) for training, which employs an actor-critic framework. The actor is the policy network π_ω, while the critic v_ϕ predicts the value $v(s_t)$ of a state s_t. The training process involves I iterations, where a batch of instances is processed in parallel by the DRL agent with instance replacement every 20 iterations. Additionally, the policy is validated on a set of independent validation instances every 10 iterations during training. The training procedure and some key training parameters are shown in Algorithm 1 and Table III.

TABLE III

TRAINING PARAMETERS.

Parameter	**Value**
lr (learning rate)	2e-4
gamma (discounting factors)	1.0
K_epochs (update epoch in PPO)	3
A_coeff (weight of policy loss)	1
vf_coeff (weight of value loss)	0.5
entro_coeff (weight of entropy loss)	0.05
KL_coeff (weight of KL regularization term)	0.05
parallel_iter (batch size of training)	20
save_timestep (validate interval)	10
max_iterations	10000
minibatch (update batchsize in PPO)	512
update_timestep (update interval)	5

2) Training on datasets: First, we train our model on small-scale synthetic instances with 10 jobs $\times$ 5 machines, with the training instances generated on-the-fly, and we apply the policy model learned from the small-scale to all scales of the synthetic test set to verify the performance.

For the real production line datasets, we use the model trained on the synthetic dataset as the pretrained model, and then fine-tune it on different production line datasets. Initially, when directly transferring the model from the synthetic dataset to the production line datasets, we observed a temporary deterioration in the makespan metric. To address this issue, we incorporated an additional KL-divergence regularization term in the PPO loss with respect to the pre-trained policy, which prevents overly rapid policy updates and leads to more stable performance improvements during training.

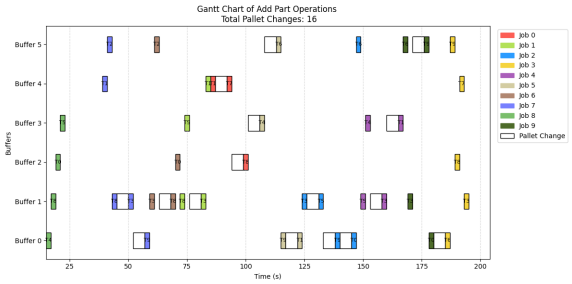
C. Baseline

We adapt several strong-performing algorithms from the standard FJSP to the FJSP-LB-MK setting. These include: 1) the CP-SAT solver from Google OR-Tools, 2) Priority Dispatching Rules (PDRs), and 3) a DRL method known for its effectiveness on FJSP.

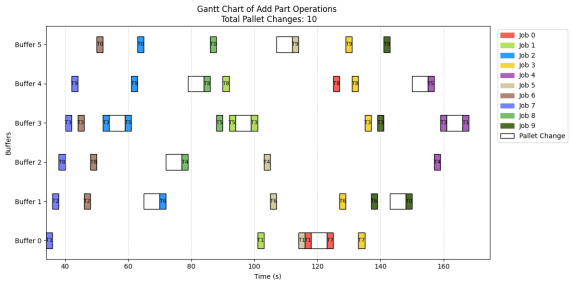
OR-Tools. OR-Tools is a widely used constraint programming solver. We use it with part sorting sequences obtained via MCTS (10,000 simulations) as static input. Each instance is solved within a 1800s time limit, and the best solution found serves as an upper bound for evaluation.

PDRs. The scheduling process is split into two stages: operation sequencing and machine assignment. For sequencing, we use rules such as Most Work Remaining (MWR), Least Work Remaining (LWR), and First in First Out (FIFO). For machine assignment, we use the Earliest End Time (EET).

DRL. We adapt the DRL framework from Song et al. [15], which performs well on standard FJSP benchmarks, to the FJSP-LB-MK. Two strategies are tested: 1) **DRL-Greedy**: selects the highest probability action, and 2) **DRL-Sampling**: samples actions based on the probability distribution. Each instance is run 100 times, reporting the best result.

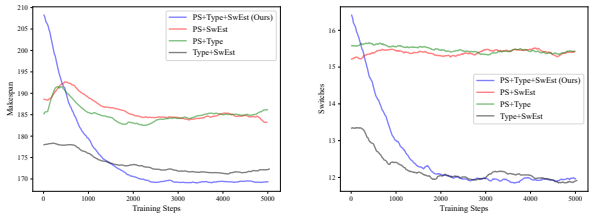


(a) [15], Pallet_change=16, Makespan=192



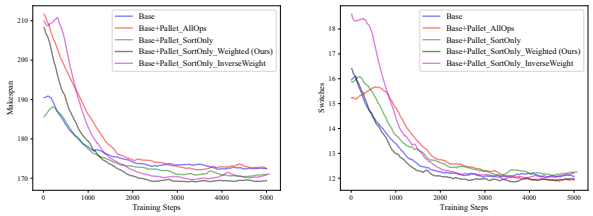
(b) Ours, Pallet_change=10, Makespan=167

Fig. 6. Gantt Chart Comparison (Test Case: 10j5m test_set instance). The white rectangles represent time delays caused by pallet changes. Rectangles of other colors represent the time intervals when parts from different jobs are loaded onto the pallet (where T_n denotes the n-th type of part).



(a) Makespan (b) Switches

Fig. 7. Training curve comparison on State Features.



(a) Makespan (b) Switches

Fig. 8. Training curve comparison on selective connectivity and cost-sensitive propagation.

D. Evaluation Metrics

To assess both the efficiency and quality of the proposed method, we adopt the following evaluation metrics commonly used in FJSP literature:

- **Makespan**: The average total completion time across all test instances, as the primary optimization objective.
- **Gap**: The average relative difference in makespan compared to the solutions obtained by OR-Tools, used to evaluate the quality of the solution.
- **Time**: The average computational time required to obtain solutions, reflecting the algorithm's efficiency.

In addition to the standard scheduling metrics, we introduce **Switches** as an indicator of the algorithm's performance with respect to pallet exchange frequency.

- **Switches**: The average total number of pallet changes across all test instances.

E. Results and Analysis

Table IV and Table V report the scheduling performance of our method compared with various baselines on both the synthetic datasets and the real production line datasets. As shown, our approach consistently achieves superior makespan performance across all problem sizes and production lines. While our method requires more computation time than PDRs, it yields significantly better performance, and on the production line C dataset, it even surpasses the exact solver (OR-Tools) with markedly lower computation time. Under both the *Greedy* and *Sampling* strategies, our method outperforms the DRL baseline [15], and as the problem scale grows, the performance gap continues to widen, demonstrating the superior scalability and generalization ability of our approach for large-scale problems.

More notably, in terms of the pallet changes (switches), our approach consistently outperforms the DRL baseline and all PDRs across all synthetic problem scales. This advantage can be attributed to the explicit incorporation of switch-related objectives into our model, where graph-based structural modeling and reward design guide the policy to reduce long-term pallet switches overhead. However, on the production line datasets, our method does not always achieve the lowest number of switches. This phenomenon can be attributed to the significant heterogeneity across different production lines. Specifically, the total number of part categories C, the number of part categories per job c_j, and the available pallets P vary substantially between lines, leading to disparate levels of buffer congestion. Consequently, while our strategy—pre-trained on synthetic instances and fine-tuned for production lines—primarily prioritizes makespan optimization, it may incur slightly more switches on certain real-world datasets. This reflects an inherent trade-off dictated by the specific buffer constraints of individual production environments.

Additionally, Figure 6 presents a comparative visualization of Gantt charts between our method and [15], using a representative case from the 10j5m synthetic test set. The results show that our method can shorten the overall completion time by effectively reducing the number of pallet changes.

In summary, our method achieves dual optimization of scheduling quality and buffer usage cost, and achieves a good balance between solution quality and computational cost. This makes it particularly well-suited for large-scale flexible job-shop scenarios with practical buffer constraints.

TABLE IV

PERFORMANCE ON SYNTHETIC FJSP-LB-MK DATASETS.

Size		PDRs FIFO	MOR	SPT	MWR	LWR	Greedy strategy [15]	Ours	Sampling strategy [15]	Ours	OR-Tools[1]
10×5	Makespan	198.11	189.27	193.23	191.74	201.81	177.02	169.85	171.48	**158.25**[*]	142.45 (9.01)
	Gap[2]	39.1%	32.9%	35.6%	34.6%	41.7%	24.3%	19.2%	20.4%	**11.1%**	
	Time (s)	0.16	0.16	0.16	0.16	0.16	0.39	0.43	0.68	0.71	
	Switches	15.52	15.02	15.02	15.08	14.93	14.50	12.10	12.01	**9.65**	
20×5	Makespan	359.06	373.83	361.26	373.95	373.54	342.69	300.70	327.17	**296.88**	279.45 (20.68)
	Gap	28.5%	33.5%	29.3%	33.8%	33.7%	22.6%	7.6%	17.1%	**6.2%**	
	Time (s)	0.32	0.32	0.32	0.32	0.32	0.78	0.84	1.30	1.41	
	Switches	32.21	32.86	33.01	33.18	33.50	32.40	21.61	27.66	**19.19**	
15×10	Makespan	335.05	306.45	327.73	298.91	352.88	326.25	279.47	294.91	**259.34**	239.79 (13.90)
	Gap	39.7%	27.8%	36.7%	23.8%	47.2%	36.1%	16.5%	23.0%	**8.2%**	
	Time (s)	0.51	0.51	0.50	0.50	0.50	1.17	1.33	1.91	2.02	
	Switches	24.46	24.07	24.24	24.21	23.83	23.62	16.03	20.76	**13.79**	
20×10	Makespan	413.55	393.96	420.79	388.03	452.13	411.05	342.13	385.22	**324.63**	297.75 (18.99)
	Gap	38.9%	32.3%	41.3%	30.3%	51.8%	38.1%	14.9%	29.4%	**9.0%**	
	Time (s)	0.71	0.71	0.71	0.71	0.71	1.68	1.76	3.02	3.20	
	Switches	33.66	33.14	33.53	33.34	33.90	33.17	19.89	29.07	**17.26**	
30×10	Makespan	595.15	566.64	593.82	554.79	624.53	579.29	452.11	547.34	**444.70**	410.74 (25.93)
	Gap	44.9%	38.0%	44.6%	35.1%	52.0%	41.0%	10.1%	33.3%	**8.3%**	
	Time (s)	1.25	1.25	1.25	1.25	1.25	2.26	2.64	6.82	7.19	
	Switches	51.98	51.05	52.26	51.73	51.13	50.57	26.31	45.23	**25.13**	
40×10	Makespan	757.67	741.12	762.47	723.96	794.56	742.75	567.47	715.23	**564.89**	518.18 (41.82)
	Gap	46.2%	43.0%	47.1%	39.7%	53.3%	43.3%	9.5%	38.0%	**9.0%**	
	Time (s)	2.10	2.10	2.10	2.10	2.10	3.21	3.47	13.18	14.05	
	Switches	70.12	70.02	68.89	69.56	68.33	67.66	33.76	62.73	**31.47**	

[*] Best among all methods excluding OR-Tools;
[1] For OR-Tools, the makespan and the switches (in brackets) of optimally solved instances within a 1800s time limit are reported;
[2] Gap is calculated based on the OR-Tools result as a reference.

TABLE V

PERFORMANCE ON REAL PRODUCTION LINE DATASETS.

Dataset		PDRs FIFO	MOR	SPT	MWR	LWR	Greedy strategy [15]	Ours	Sampling strategy [15]	Ours	OR-Tools
A	Makespan	8946.92	8946.92	9188.00	8618.30	10413.75	8057.74	7753.52	7656.92	**7392.72**	6706.06 (18.85)
	Gap	33.4%	33.4%	37.0%	28.5%	55.3%	20.2%	15.7%	14.2%	**10.2%**	
	Time (s)	0.92	0.92	0.92	0.92	0.92	2.17	2.31	4.57	4.77	
	Switches	20.36	20.36	21.93	**15.28**	27.12	16.36	18.34	15.86	16.42	
B	Makespan	11867.21	11867.21	12731.46	12425.45	12902.02	11372.99	10855.39	10737.66	**10207.62**	9949.58 (3.33)
	Gap	19.3%	19.3%	28.0%	24.9%	29.7%	14.3%	9.1%	7.9%	**2.6%**	
	Time (s)	1.09	1.09	1.09	1.09	1.09	2.65	2.71	5.31	6.16	
	Switches	3.26	3.26	3.24	3.30	3.23	3.00	3.08	3.04	**2.88**	
C	Makespan	42896.40	42896.40	43803.81	40565.81	45554.45	41364.82	40460.43	40994.28	**40359.80**	40377.03 (25.42)
	Gap	6.2%	6.2%	8.5%	0.5%	12.8%	2.5%	0.2%	1.5%	**-0.04%**	
	Time (s)	0.90	0.90	0.90	0.90	0.90	2.31	2.38	3.99	4.94	
	Switches	32.13	32.13	31.07	31.60	30.65	31.62	**25.11**	25.63	25.89	
D	Makespan	19801.34	19801.34	20502.43	20162.80	20535.88	18753.75	18475.30	18310.58	**17616.80**	17074.17 (19.58)
	Gap	16.0%	16.0%	20.1%	18.1%	20.3%	9.8%	8.2%	7.2%	**3.2%**	
	Time (s)	1.30	1.30	1.30	1.30	1.30	2.88	2.97	5.26	6.74	
	Switches	27.81	27.81	28.17	27.08	28.37	**19.17**	19.58	19.27	19.42	

TABLE VI

EFFECTIVENESS OF KEY STATE FEATURES.

	Makespan Value	Makespan Gap_to_ours	Switches Value	Switches Gap_to_ours
PS+SwEst	183.94	8.3%	15.41	27.4%
PS+Type	187.73	10.5%	15.42	27.4%
Type+SwEst	172.41	1.5%	**12.08**	-0.2%
PS+Type+SwEst (Ours)	**169.85**	0.0%	12.10	0.0%

TABLE VII

EFFECTIVENESS OF SELECTIVE CONNECTIVITY AND COST-SENSITIVE PROPAGATION.

	Makespan Value	Makespan Gap_to_ours	Switches Value	Switches Gap_to_ours
Base	172.62	1.6%	12.18	0.7%
Pallet_AllOps	172.80	1.7%	12.22	1.0%
Pallet_SortOnly	171.03	0.7%	12.31	1.7%
Pallet_SortOnly_InverseWeight	170.82	0.6%	12.28	1.5%
Pallet_SortOnly_Weighted (Ours)	**169.85**	0.0%	**12.10**	0.0%

F. Ablation Study

We perform ablation experiments on the smallest problem scale of the synthetic dataset (10 jobs $\times$ 5 machines), keeping all configurations consistent.

1) Effectiveness of Key State Features: To evaluate the impact of different input features, we remove components from the operation features. As mentioned in IV-A.1, the key features include: (1) 6-dimensional scheduling-related

features used in [15], (2) a binary indicator for part-sorting operations (**PS**), (3) a one-hot encoding of part types (**Type**), and (4) estimated pallet switches (**SwEst**). Figure 7 and Table VI show the results. Removing **Type** and **SwEst** significantly increases both *Makespan* and *Switches* (8.3%/10.5% and 27.4%/27.4% increases, respectively), highlighting their importance. Removing **PS** has a minor effect, suggesting its limited contribution in isolation. This emphasizes that **Type** and **SwEst** are crucial for estimating switch-related costs, while **PS** mainly helps identify operation types.

2) Effectiveness of Selective Connectivity and Cost-sensitive Propagation: In IV-B.1 and IV-B.2, we introduce the selective connectivity and cost-sensitive propagation strategy. Figure 8 and Table VII show results from different configurations: **Base** means no connections between operation nodes and the buffer node, **Pallet_AllOps** indicates all operations connected to the buffer node, and **Pallet_SortOnly** is that only part-sorting operations are connected to the buffer node, and the buffer message is broadcast uniformly to every part-sorting node without differentiation. This refinement enables the model to focus on critical decisions and eliminate noise from non-relevant operations, and achieve better performance than the above two configurations. The best performance is achieved with **Pallet_SortOnly_Weighted**, in which the buffer message is discriminally broadcast to part-sorting nodes, and the edge weights are proportional to the different estimated pallet changes of different part-sorting nodes. We have also compared two edge weight strategies: *cost-avoiding* and *benefit-seeking*, the latter is represented by **Pallet_SortOnly_InverseWeight**. The *benefit-seeking* strategy performs similarly to the baseline and worse than *cost-avoiding*. We analyze that *benefit-seeking* may lead to locally optimal but short-sighted choices while *cost-avoiding* better mitigates high-cost decisions with a long-term consideration, resulting in more efficient scheduling.

VI. Acknowledgments

This work was supported in part by the NSFC (62325211, 62132021), the Fundamental Research Funds for the Central Universities (2042025kf0014), the Major Program of Xiangjiang Laboratory (23XJ01009), Key R&D Program of Wuhan (2024060702030143).

VII. Conclusion

This work addresses the Flexible Job-Shop Scheduling Problem with Limited Buffers and Material Kitting (FJSP-LBMK) in manufacturing. We propose a DRL-based scheduling method using a heterogeneous graph neural network, which effectively models dependencies between machines, operations, and buffers. The method is validated on the synthetic dataset and real production line datasets, and compared with heuristics, constraint programming, and DRL methods. Experimental results demonstrate that our approach achieves a superior balance between performance and computational time, and ablation studies demonstrate the effectiveness of the key components in our method.

References

[1] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 904–916, 2019.

[2] X. Kai, Z. Hang, H. Ruizhen, Y. Min, L. Hao, Z. Hui, and Y. Haibin, "Embodied intelligence for flexible manufacturing: A survey," *ROBOT*, vol. 47, no. 4, pp. 581–624, 2025.

[3] J. Zhang, S. Wang, W. He, J. Li, Z. Cao, B. Wei, and M. Wang, "Material kitting in selective assembly: a manual order picking system based on augmented reality," *The International Journal of Advanced Manufacturing Technology*, vol. 123, no. 1, pp. 675–686, 2022.

[4] H. Zhao, C. Zhu, X. Xu, H. Huang, and K. Xu, "Learning practically feasible policies for online 3d bin packing," *Science China Information Sciences*, vol. 65, no. 1, p. 112105, 2022.

[5] H. Zhao, J. Xu, K. Yu, R. Hu, C. Zhu, and K. Xu, "Deliberate planning of 3d bin packing on packing configuration trees," *arXiv preprint arXiv:2504.04421*, 2025.

[6] S. Lin, H. Cui, Y. Wang, and Y.-H. Jia, "Decoupled training neural solver for dynamic traveling salesman problem," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 12 986–12 992.

[7] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[8] A. Zhao and J. F. Bard, "Batch scheduling in a multi-purpose system with machine downtime and a multi-skilled workforce," *International Journal of Production Research*, vol. 62, no. 12, pp. 4470–4493, 2024.

[9] D. Müller, M. G. Müller, D. Kress, and E. Pesch, "An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning," *European Journal of Operational Research*, vol. 302, no. 3, pp. 874–891, 2022.

[10] B. Chen and T. I. Matis, "A flexible dispatching rule for minimizing tardiness in job shop scheduling," *International Journal of Production Economics*, vol. 141, no. 1, pp. 360–365, 2013.

[11] R. Chen, B. Yang, S. Li, and S. Wang, "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem," *Computers & industrial engineering*, vol. 149, p. 106778, 2020.

[12] H. Li, X. Wang, and J. Peng, "A hybrid differential evolution algorithm for flexible job shop scheduling with outsourcing operations and job priority constraints," *Expert Systems with Applications*, vol. 201, p. 117182, 2022.

[13] E. Yuan, L. Wang, S. Cheng, S. Song, W. Fan, and Y. Li, "Solving flexible job shop scheduling problems via deep reinforcement learning," *Expert Systems with Applications*, vol. 245, p. 123019, 2024.

[14] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *Advances in neural information processing systems*, vol. 33, pp. 1621–1632, 2020.

[15] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1600–1610, 2022.

[16] I. Echeverria, M. Murua, and R. Santana, "Solving the flexible job-shop scheduling problem through an enhanced deep reinforcement learning approach," *arXiv preprint arXiv:2310.15706*, 2023.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[18] J. C. Chen, T.-L. Chen, Y.-Y. Chen, and M.-Y. Chung, "Multi-resource constrained scheduling considering process plan flexibility and lot streaming for the cnc machining industry," *Flexible Services and Manufacturing Journal*, pp. 1–48, 2023.

[19] Z. Han, C. Han, S. Lin, X. Dong, and H. Shi, "Flexible flow shop scheduling method with public buffer," *Processes*, vol. 7, no. 10, p. 681, 2019.

[20] J. Liang, P. Wang, L. Guo, B. Qu, C. Yue, K. Yu, and Y. Wang, "Multi-objective flow shop scheduling with limited buffers using hybrid self-adaptive differential evolution," *Memetic Computing*, vol. 11, no. 4, pp. 407–422, 2019.

[21] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations research*, vol. 41, no. 3, pp. 157–183, 1993.