Transactions on Computer Science and Intelligent Systems Research
ISSN: 2960-1800, eISSN: 2960-2238 | Volume 11
TPCEE 2025

# Research on Multi-Agent Task Scheduling Optimization Based on Deep Reinforcement Learning

Huiyu Hu, Xiuli Wang *

School of Economics and Management, Nanjing University of Science and Technology, Nanjing, Jiangsu 210000, China

* Corresponding Author Email huh_y2001@163.com

**Abstract.** For the task scheduling problem in multi-agent systems, this paper proposes a collaborative optimization method based on Graph Neural Network and Reinforcement Learning. Firstly, a heterogeneous graph structure is constructed to uniformly model the temporal dependencies, resource competition, and agent capability differences among tasks, and multi-dimensional node features are designed to fully describe the scheduling state. Secondly, the Proximal Policy Optimization algorithm is adopted to achieve efficient training and stable convergence of the policy network based on graph embedding, supporting rapid decision-making for large-scale instances. To verify its effectiveness, 30 test cases are generated for each of the three scales, totaling 90 cases. It is compared with Genetic Algorithm and Gurobi's exact solver. Through a large number of simulation experiments, the effectiveness and advantages of this method in solving the studied problem have been verified.

**Keywords:** Multi-agent scheduling; Graph neural network; Reinforcement learning.

## 1. Introduction

A multi-agent system is a kind of system that can intelligently and flexibly respond to changes in working conditions and the demands of surrounding processes. Through the collaborative work of multiple agents, the optimization of production resources and dynamic scheduling can be achieved. However, the traditional methods for solving scheduling problems in resource optimization mainly rely on dynamic programming and integer programming, which can be solved by precise algorithms or hand-designed heuristic methods. Although traditional precise algorithms can provide optimal solutions when solving scheduling problems in resource optimization, their computational costs are relatively high, which makes these methods face scalability challenges when dealing with large-scale problems. Moreover, these methods do not perform well in terms of real-time performance, making it difficult to respond quickly to real-time changes in production conditions, such as sudden increases in new tasks or machine failures. These limitations restrict the application of precise methods in dynamic and large-scale scheduling scenarios. In contrast, heuristic methods can quickly provide high-quality solutions in specific applications. These methods simplify the complexity of the problem, sacrificing some optimality in exchange for computational efficiency. However, developing effective heuristic methods is not an easy task; it requires interdisciplinary collaboration, including contributions from computer scientists, operations researchers, and others. This process requires significant time and resource investment, and the final effect may not always reach the expected optimal level. That is, traditional scheduling methods often rely on fixed rules and algorithms, which are difficult to adapt to complex and variable production environments. To overcome these challenges, introducing machine learning techniques, especially deep reinforcement learning, has become a key approach to improving the performance of multi-agent systems.

This paper proposes an innovative optimization method that ingeniously combines graph neural networks (GNN) with proximal policy optimization (PPO) to address the temporal dependency issues in complex task scheduling. This method models the scheduling problem as a graph structure, where tasks and agents are represented as nodes, and the dependency relationships between tasks and the resource capabilities of agents are modeled through the edges of the graph. GNN is employed to

extract an embedding representation for each node, fusing both local and global information; upon these embeddings, the PPO algorithm learns a scheduling policy that optimizes the task-assignment sequence and agent selection, aiming to minimize the makespan.

## 2. Problem Description and Modeling

### 2.1. Problem description

For the multi-agent scheduling optimization problem aiming at minimizing the makespan, this paper focuses on the single-task agent (ST), single-agent task (SR), and time-expanded allocation (TA) category problems under the iTax classification method that have cross-schedule dependencies (XD). That is, the constraints of the task not only depend on the scheduling of a single agent itself, but also are related to the task arrangement, and it is necessary to coordinate the schedules of multiple agents to achieve global optimization. Moreover, task allocation needs to consider time expansion, each agent can only execute one task at a time, and each task only requires one agent to complete.

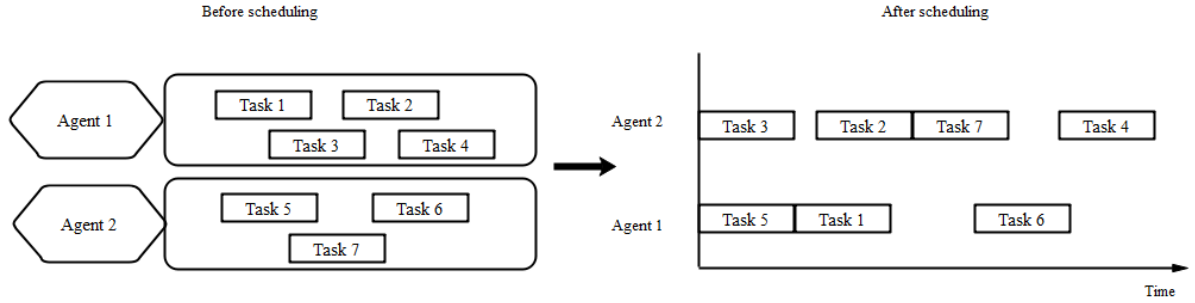


**Figure 1.** Task Scheduling Diagram

### 2.2. Assumptions

(1) All agents have the same task processing capabilities, and the time consumption of tasks does not vary depending on the agent.

(2) The time consumption of each task is determined and fixed, without considering randomness (such as machine failures or execution time fluctuations).

(3) Once a task starts to be executed, it must be completed continuously and cannot be interrupted to execute other tasks.

(4) The time for an agent to switch positions between different tasks is zero, or it has been included in the task time consumption, and does not affect the scheduling time.

(5) Each agent can only handle one task at the same time and cannot execute multiple tasks concurrently.

### 2.3. Symbol definitions

The notation used are as follows:

Sets

T: The set of all tasks

A: The set of all agents

E: The set of task dependencies, if $(i, j) \in E$, then task $i$ is completed before task $j$

Parameters

$p_i$: Task i processing time

$w_{ij}$: Interval time between task $i$ and task $j$

$d_i$: Deadline of task \ mathbit{i}, which can be set to \infty if there is no limit

M: A sufficiently large constant

Decision variables

$X_{r,i}$: Binary variable: If task $i$ is assigned to agent $r$, it is 1; otherwise, it is 0.

$Y_{i,j}$: Binary variable: If task $i$ must be completed after task $j$ is executed, it is 1; otherwise, it is 0.

$s_i$: Start time of task $i$

$C_{max}$: Maximum completion time

### 2.4. Mathematical Model

$$\text{Min obj}=C_{\max} \tag{1}$$

$$\sum_{r\in R} X_{r,i} = 1, \forall i \in T \tag{2}$$

$$s_i + p_i \le C_{max}, \forall i \in T \tag{3}$$

$$s_i + p_i \le d_i, \forall i \in T \tag{4}$$

$$s_i + p_i + w_{ij} \le s_j, \forall (i,j) \in E \tag{5}$$

$$s_i + p_i \le s_j + M\left(1 - Y_{i,j}\right) + M\left(1 - X_{r,i}\right) + +M\left(1 - X_{r,j}\right), \forall r \in R, \forall i,j \in T, i \ne j \tag{6}$$

$$s_i + p_i \le s_j + MY_{i,j} + M\left(1 - X_{r,i}\right) + +M\left(1 - X_{r,j}\right), \forall r \in R, \forall i,j \in T, i \ne j \tag{7}$$

$$X_{i,r}, Y_{i,j} \in \{0,1\} \tag{8}$$

$$s_{i,} C_{max} \ge 0, \forall i \in T \tag{9}$$

Among them, formula (1) is the objective function, aiming to minimize the makespan. Formula (2) ensures that each task is uniquely assigned to a certain agent $r$. Formula (3) sets the lower bound of makespan as the maximum completion time of all tasks, ensuring that the objective function can truly measure the moment when the "last task ends". Formula (4) ensures that the completion time of task $i$ is no later than its absolute deadline. Formula (5) imposes a "minimum interval" constraint on task pairs ($i, j$) with a precedence relationship, meaning that after task $i$ is completed, task $j$ must wait for at least $w_{ij}$ time units before starting. Formulas (6) and (7) jointly guarantee that if tasks $i$ and $j$ are assigned to the same agent $r$, their time periods must strictly not overlap; if they are not assigned to the same agent, the constraint is automatically relaxed by the big M. Formulas (8) and (9) give the domain of all decision variables, with the assignment variable and sequence variable being binary variables, and the start time and makespan being non-negative continuous variables.

## 3. Problem-solving based on Graph Neural Networks and Proximal Policy Optimization Algorithm

### 3.1. Extraction of Graph Structure Features by GNN

GNN is used to extract the local state embeddings of each agent. Through the message passing mechanism, the representation of each node is aggregated with the information of its neighboring nodes, thereby being able to capture the structural information in the graph and the relationships

between nodes, integrating the neighbor information to form an understanding of the global environment. This paper constructs a heterogeneous graph G=(V, E). Here, the nodes V include the task set T and the agent set A. The number of nodes $N = n_T + n_A$. Each node $v_i \in V$ has a feature vector $x_i$, forming a feature matrix:

$$X = [x_1, x_2, \ldots, x_n]^T \tag{10}$$

The edge set in the figure is divided into two categories: the static preceding edges $E_{prec}$, with the weights directly taking the interval time $w_{ij}$; the dynamic agent-task edges $E_{agent}$, which are temporarily connected only when the current task is ready and there is an idle agent, and dynamically increased or decreased through candidate masks to ensure that the policy network only outputs probabilities on feasible actions.

Adjacent matrix:

$$A_{ij} = \begin{cases} w_{ij}, If\ task\ i\ is\ a\ prerequisite\ of\ task\ j, the\ time\ interval\ is\ wij. \\ 0, else \end{cases} \tag{11}$$

In the graph modeling of this study, the ten-dimensional components of the node feature matrix are specifically as shown in the following table. All dimensions strictly fall within the range of [0, 1], ensuring the numerical stability of the forward and backward propagation of graph convolution, and allowing a single backward propagation to simultaneously affect graph embeddings, policy outputs, and value estimation.

**Table 1.** Node-based 10-dimensional feature embedding

| Serial number | Feature dimension | Feature description |
|---|---|---|
| 1 | Task processing duration | Time required to complete the task |
| 2 | Task status | Indicates whether the task is scheduled, 0 not started/1 in progress/2 completed |
| 3 | Earliest start time | The earliest start time of the task |
| 4 | Deadline | The deadline of the task |
| 5 | Executable flag | 0 the task is not executable/1 the task is executable |
| 6 | Urgency | The more urgent the task is, the closer it is to 1 |
| 7 | Number of downstream dependencies | The number of subsequent tasks that depend on this task |
| 8 | Waiting time | The time that the task has been waiting |
| 9 | Resource competition degree | How many tasks that have not started other tasks are competing with the current task for the same prerequisite task |
| 10 | Node type | 0 task node/1 proxy node |

Graph neural networks employ graph convolutional neural networks, and each layer performs neighbor aggregation:

$$h_i^{(l+1)} = \sigma\left( \hat{D}^{-\frac{1}{2}} \hat{A}\, \hat{D}^{-\frac{1}{2}} h^{(l)}\, W^{(l)} \right) \tag{12}$$

$h_i^{(l)}$: Node feature matrix of the l-th layer, h^{(0)} = X

$W^{(l)}$: Trainable weight matrix

$\hat{A}$: $\hat{A} = A + I$ (where I is the identity matrix and self-loops are introduced), $\widehat{D}$ is the degree matrix of $\hat{A}$ (diagonal matrix, $\widehat{D}_{ii} = \sum_j \hat{A}_{ij}$)

$\sigma$ : Activation function

The pooling operation aggregates the node-level embeddings through first-order statistical aggregation (mean pooling), outputting a graph-level global feature vector; this vector constitutes a sufficient statistic for the entire scheduling system state in the reproducing kernel Hilbert space, providing compressed representations of the critical path length, resource utilization, and remaining workload for the downstream policy network, thereby enabling low-dimensional and comprehensive perception of system bottlenecks.

$$h_{graph} = \text{Pool}\left(H^{(L)}\right) \in R^{d_{hidden}} \tag{13}$$

$h_{graph}$: The representation vector of the entire graph, used for the Critic network

Pool: Such as average pooling, summation pooling or attention pooling

In the code, the default is average pooling, that is:

$$h_{graph} = \frac{1}{n+m}\sum_i h_i^{(l)} \tag{14}$$

Graph neural network output:

Node-level embedding: $h_i$

Graph-level embedding: $h_{graph}$

### 3.2. Markov Decision Process (MDP) Mapping

(1) State Space (S)

The state needs to fully represent the task progress, the agent's status, and the constraint relationships, and is presented in the form of a graph structure and feature vectors.

Task progress: Each task has three states: "not started/ongoing/completed", along with dynamic information such as waiting time and remaining processing time.

Agent status: Each agent has two states: "idle/busy", along with dynamic information such as remaining busy time and used time.

Constraint relationships: The dependencies between tasks (predecessors, successors, lag time) are represented by an adjacency matrix to ensure no loops.

The state is represented as an heterogeneous graph.

(2) Action Space (A)

Actions are defined as "selecting 1 task from the currently executable tasks to assign to an idle agent".

(3) Reward Function (R)

The reward is designed with "minimizing Makespan" as the core, using the form of delayed penalty and terminal reward, guiding the strategy to prefer tasks that can shorten the total completion time.

Delayed penalty: A negative reward is given for each step of time advancement to encourage early completion.

Terminal reward: A completion reward is given when the task is completed, proportional to the remaining number of tasks and the completion time, encouraging early completion.

Deadlock penalty: If a deadlock is detected, an additional negative reward is given to encourage avoiding deadlock.

Target reward: The ultimate goal is to minimize the Makespan, and the reward is negatively correlated with Makespan.

### 3.3. PPO Strategy Optimization

#### 3.3.1. Strategy Network (Actor).

The core of the policy network is to learn a policy function, which outputs the probability of "selecting each action a" given the current state s, thereby guiding the agent to make decisions. The policy network takes the feature of candidate task nodes, graph pooling features, and global features as inputs, and outputs only the probability for the candidate tasks:

$$\pi_\theta(a \mid s) = \text{Softmax}\left(W_a \cdot \phi(s)\right) \tag{15}$$

$\phi(s)$: The state feature vector is composed of the candidate task node features $h_i$, the graph pooling features $h_{graph}$, and the global state feature $g$. The global feature $g$ is a normalized vector description of the macroscopic operation situation of the scheduling system, including the remaining workload of the current task, resource utilization rate, etc.

#### 3.3.2. Value Network (Critic).

The core of the value network is to learn a value function, that is, given the current state s, estimate "how much cumulative reward can be obtained from this state in the future", providing feedback signals for the policy network. The value network takes graph-level embeddings as input and outputs state value estimates:

$$V_\theta(s) = W_v \cdot MLP\left(\left[h_{graph}, g\right]\right) \tag{16}$$

#### 3.3.3. PPO Update.

The parameters of the policy network and the value network are updated through the PPO algorithm, while sharing the GNN encoder. The target function of PPO is updated in the reverse direction using PPO:

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{17}$$

The advantage function is directly incorporated into the core policy objective of PPO. The advantage function $A_t$ is estimated by GAE to measure the quality of the action.

$$A_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l},\ \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{18}$$

γ: Discount factor

λ: GAE parameter (controls bias and variance balance)

$r_t(\theta)$: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, the probability ratio represents the ratio of the probabilities of the new and old strategies

The PPO algorithm limits the range of variation of the probability ratio $r_t(\theta)$ to $1-\epsilon$ *and* $1+\epsilon$ to prevent excessive changes in the policy during a single update, thereby ensuring the stability of the training. $\epsilon$ is the PPO clipping parameter.

Total loss function:

$$L = c_1 L^{\text{CLIP}} + c_2 L^{\text{V}} - c_3 L^{ent} \tag{19}$$

The value loss $L^V = E[(V_\theta(s) - R_t)^2]$ is used to optimize the accuracy of the value estimation of the Critic. The entropy regularization $L^{ent} = -E[H(\pi_\theta)]$ is employed to encourage exploration of the policy and prevent premature convergence to local optima.

### 3.4. GNN-PPO framework flow

This algorithm framework first models the multi-agent scheduling problem as an heterogeneous graph MDP. Through the graph convolutional network, the task nodes, agent nodes and global state are encoded in one go, and node-level embeddings and graph-level embeddings are output. Then, the node embeddings are generated as action probability by the policy head, and the graph-level embeddings provide state value by the value head. Together, they constitute the Actor-Critic architecture. The old policy sampling trajectory is used, GAE is constructed to give the advantage, and PPO-Clip loss is back-propagated end-to-end. Adam is used to synchronize the update of graph convolution, policy and value parameters. Finally, the current network weights are periodically copied as the old policy, new trajectories are collected in a cycle and the above process is repeated until the Makespan converges on the validation set or the maximum number of iterations is reached.

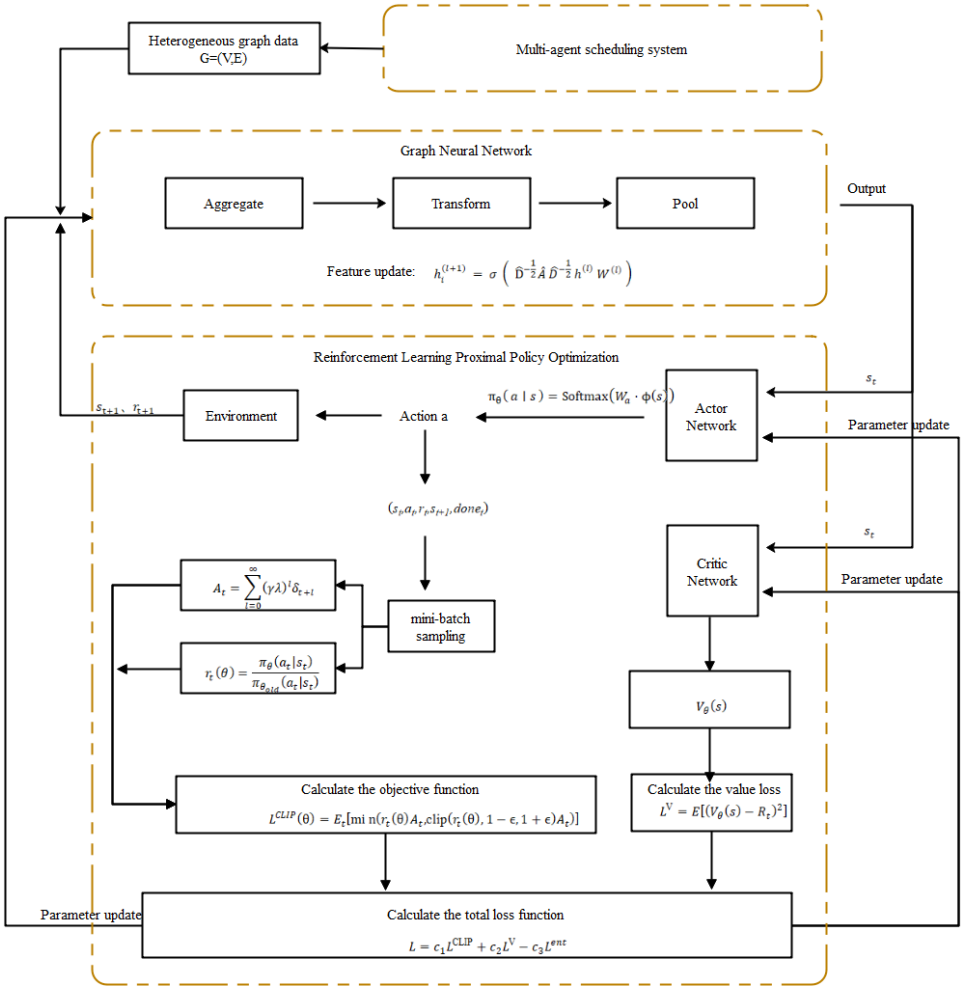


**Figure 2.** Multi-agent scheduling framework based on GNN-PPO algorithm

## 4. Experimental Simulation and Analysis

### 4.1. Experimental Design

To comprehensively evaluate the solution performance of the proposed GNN+PPO method in the multi-agent task scheduling problem, this paper designs a systematic experimental scheme. It conducts quantitative analysis in terms of solution quality and stability, and compares it with two representative algorithms:

Traditional heuristic algorithm: Genetic Algorithm (GA);

Exact solving method: Commercial solver Gurobi.

The experiments adopt a unified testing platform and evaluation metrics to ensure the fairness and reproducibility of the comparison results. All program codes in this chapter are tested using Python 3.6 software, and the program testing environment is as follows: Operating system: 64-bit Windows 10, Processor: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, Memory: 12GB.

Based on the random task generation model, three types of test cases are constructed: small scale (20 tasks × 5 agents), medium scale (50 tasks × 10 agents), and large scale (100 tasks × 20 agents). Each type generates 30 different instances, totaling 90 problem instances. The three algorithms (GA, Gurobi, GNN+PPO) run independently on each instance 30 times, and record the Makespan of each run. Statistical methods are used to compare the significance of differences between the algorithms, and the stability and robustness of the algorithms are analyzed through box plots, means, standard deviations, etc.

### 4.2. Experimental Results and Analysis

#### 4.2.1. Performance Comparison of Makespan.

The experimental results show that GNN_PPO outperforms the solver Gurobi and the GA algorithm on all test scales. Due to the NP-hard nature of MILP, in this paper, the running time limit of Gurobi is set to 3600 seconds, and the optimal feasible solution obtained within the time limit is taken as the benchmark. In Table 2, the makespan of Gurobi is the average feasible solution within 3600 seconds, and it does not represent the theoretical optimum; it is only used for comparing the quality at the same time with other algorithms. The specific results are as follows:

**Table 2.** Target value experimental results

| Scale | Method | C_avg | C_best | C_worst | σ | CV |
|---|---|---|---|---|---|---|
| Large | GA | 277.0667 | 246.0000 | 306.0000 | 16.1005 | 0.0581 |
| | GNN_PPO | *268.8000* | 241.0000 | 297.0000 | 14.4494 | 0.0538 |
| | GUROBI | 387.6667 | 266.0000 | 533.0000 | 63.9091 | 0.1649 |
| Medium | GA | 263.6333 | 225.0000 | 315.0000 | 20.8880 | 0.0792 |
| | GNN_PPO | *258.8667* | 218.0000 | 309.0000 | 20.8041 | 0.0804 |
| | GUROBI | 356.2000 | 268.0000 | 485.0000 | 62.9857 | 0.1768 |
| Small | GA | 207.9667 | 156.0000 | 250.0000 | 21.1407 | 0.1017 |
| | GNN_PPO | *205.6333* | 157.0000 | 248.0000 | 20.8963 | 0.1016 |
| | GUROBI | 315.2667 | 201.0000 | 459.0000 | 65.4254 | 0.2075 |

GNN_PPO achieved significant reductions in Makespan across all problem scales, and its advantages became more pronounced as the problem size increased, demonstrating excellent scalability. Compared to Gurobi, GNN_PPO performed better on medium and large-scale problems, indicating its superior search and generalization capabilities under complex constraints. Moreover, GNN_PPO had a smaller standard deviation, suggesting more stable and robust solution results. Additionally, the GA algorithm relies on general operators such as crossover and mutation, and is insensitive to graph structure information; GNN can directly encode topology into embeddings through message passing,

and the same set of parameters can handle any number of nodes and edge distributions without the need to redesign operators.

### 4.2.2. Stability and Robustness Analysis.

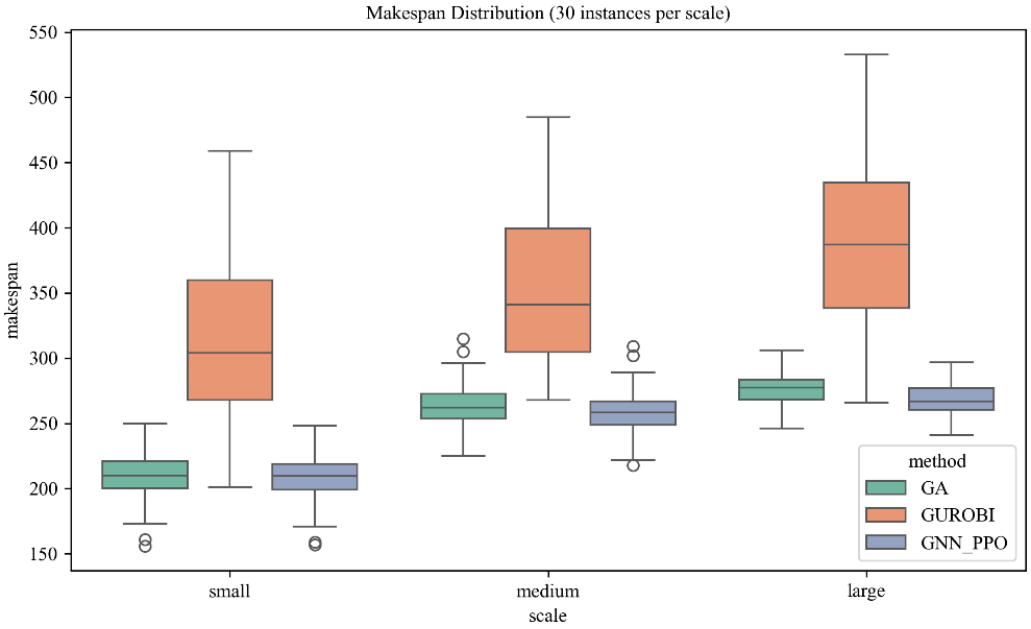


**Figure 3.** Three methods box plot distribution

By analyzing the distribution of Makespan through box plots, it can be seen that GNN-PPO has the lowest and narrowest box plot across all scales, indicating that its average Makespan is the smallest and the results are the most concentrated. In the code, the Wilcoxon signed-rank test is used to quantify the significant differences between GA and Gurobi relative to GNN-PPO: using GNN-PPO as the benchmark, a one-sided test is conducted for each scale (small/medium/large) to determine whether the comparison algorithms significantly achieve a smaller Makespan, thereby scientifically evaluating the statistical validity of each algorithm in terms of solution quality, rather than relying solely on intuitive comparisons of average values. The results show that the GNN_PPO method is significantly superior to GA and Gurobi. Especially in the large-scale scenario, the advantage of GNN-PPO is further amplified. Moreover, directly applying the small-scale trained model to large-scale inference, GNN+PPO still maintains the lowest and narrowest box plot, without the need for re-tuning parameters or re-training, demonstrating its outstanding generalization ability and significantly surpassing GA and Gurobi which require calibration on a scale-by-scale ba

## 5. Conclusion

This paper proposes an intelligent solution framework that integrates graph neural networks and proximal policy optimization to address the multi-agent task scheduling problem. By constructing a heterogeneous graph model to depict task dependencies and resource constraints, designing an efficient feature extraction mechanism, and combining the PPO algorithm to achieve policy learning, the experimental results show that the proposed method significantly outperforms traditional genetic algorithms and exact solvers in terms of solution quality and stability, and has good scalability and engineering application potential.

This study only considers a type of similar machine scheduling problem that minimizes the maximum service time. In subsequent research work, more application scenarios should be considered, and the deep reinforcement algorithm should be further optimized to improve the optimization performance of the algorithm.

## References


[1] Weiss G. Multiagent systems: a modern approach to distributed artificial intelligence [M]. Cambridge: MIT press; 1999.

[2] Gronauer S, Diepold K. Multi-agent deep reinforcement learning: a survey [J]. Artificial Intelligence Review. 2022, 55 (2): 895-943.

[3] Graham RL, Lawler EL, Lenstra JK, et al. Optimization and approximation in deterministic sequencing and scheduling: a survey [J]. Annals of Discrete Mathematics. 1979 (5): 287–326.

[4] Pinedo ML. Scheduling: Theory, Algorithms, and Systems [M]. 4th ed. Cambridge: Springer, 2008.

[5] Hoogeveen H. Multicriteria scheduling [J]. European Journal of operational research, 2005, 167 (3): 592- 623.

[6] Minella G, Ruiz R, Ciavotta M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem [J]. INFORMS Journal on Computing. 2008, 20 (3): 451-471.

[7] Peha JM. Heterogeneous-criteria scheduling: minimizing weighted number of tardy jobs and weighted completion time [J]. Computers & operations research. 1995; 22 (10): 1089-100.

[8] Balasubramanian H, Fowler J, Keha A, et al. Scheduling interfering job sets on parallel machines [J]. European Journal of Operational Research. 2009, 199 (1): 55-67.

[9] Elvikis D, Hamacher H, t'Kindt V. Scheduling two interfering job sets on uniform parallel machines with makespan and cost functions [C]. //4th Multidisciplinary International Conference on Scheduling: Theory and Applications. 2009: 645–654

[10] Perez-Gonzalez P, Framinan JM. A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems[J]. European Journal of Operational Research. 2014, 235 (1): 1-6.

[11] Prorok A, Hsieh MA, Kumar V. Fast redistribution of a swarm of heterogeneous robots [J]. EAI Endorsed Transactions on Scalable Information Systems. 2016, 3 (10): 249-55.

[12] Caridi M, Cavalieri S. Multi-agent systems in production planning and control: an overview [J]. Production Planning & Control. 2004, 15 (2): 106-18.

[13] Amador S, Okamoto S, Zivan R. Dynamic multi-agent task allocation with spatial and temporal constraints [C]. //Proceedings of the AAAI Conference on Artificial Intelligence. 2014, 28 (1).

[14] Le Pape C. A combination of centralized and distributed methods for multi-agent planning and scheduling [C]. // Proceedings., IEEE International Conference on Robotics and Automation. 1990: 488- 493.

[15] Nunes E, Gini M. Multi-robot auctions for allocation of tasks with temporal constraints [C]. //Proceedings of the AAAI conference on artificial intelligence. 2015, 29 (1).

[16] Das GP, McGinnity TM, Coleman SA, Behera L. A distributed task allocation algorithm for a multi-robot system in healthcare facilities [J]. Journal of Intelligent & Robotic Systems. 2015, 80 (1): 33-58.

[17] Cheng CY, Chen TL, Wang LC, et al. A genetic algorithm for the multi-stage and parallel-machine scheduling problem with job splitting–A case study for the solar cell industry [J]. International Journal of Production Research. 2013, 51 (16): 4755-77.

[18] Chen L, Dai SL, Dong C. Adaptive optimal tracking control of an underactuated surface vessel using actor–critic reinforcement learning [J]. IEEE Transactions on Neural Networks and Learning Systems. 2022.

[19] Vu VT, Tran QH, Pham TL, et al. Online actor-critic reinforcement learning control for uncertain surface vessel systems with external disturbances [J]. International Journal of Control, Automation and Systems. 2022, 20 (3): 1029-40.

[20] Dao PN, Liu YC. Adaptive reinforcement learning in control design for cooperating manipulator systems [J]. Asian Journal of Control. 2022, 24 (3): 1088-103. [53] Lillicrap T P, Hunt J J, Pritzel A. et al. Continuous control with deep reinforcement learning [J]. arXiv preprint arXiv: 1509. 02971. 2015.

[21] Wang Z, Liu C, Gombolay M. Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints [J]. Autonomous Robots. 2022, 46 (1): 249-68.

[22] Tampuu A, Matiisen T, Kodelja D, et al. Multiagent cooperation and competition with deep reinforcement learning [J]. PloS one. 2017, 12 (4): e0172395. [56] Hu K, Li M, Song Z, et al. A review of research on reinforcement learning algorithms for multi-agents [J]. Neurocomputing. 2024: 128068.

[23] Ning Z. A survey on multi-agent reinforcement learning and its application [J]. Journal of Automation and Intelligence. 2024.