ISSN 2707-1898 (print)

Український журнал інформаційних технологій
Ukrainian Journal of Information Technology
http://science.lpnu.ua/uk/ujit
https://doi.org/10.23939/ujit2025.01.035
Article received 10.04.2025 p.
Article accepted 01.05.2025 p.
UDC 004.8

Correspondence author
K. S. Hrishchenko
k.hrishchenko@kpi.ua

K. S. Hrishchenko, O. O. Pysarchuk

National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

ACTION-MASKED REINFORCEMENT LEARNING TECHNOLOGY FOR ORDER SCHEDULING

The problem of high-performance and efficient order scheduling is a common combinatorial optimization problem in various industrial contexts. Creation of a model capable of generating schedules balanced in terms of quality and computational time poses a significant challenge due to the large action space. This study proposes a high-performant environment and a reinforcement learning model for allocating orders to resources using a mechanism of invalid action masking. The developed reinforcement learning solution overcomes the limitations of traditional heuristic and exact methods regarding computation performance and efficiency. The research included the design of a Gymnasium-compatible simulation environment, performance benchmarking, development of optimized environment state updating procedures, feature generation strategies, and evaluation of PPO and MaskablePPO models. The environment implemented incremental updates of the features state and action masks with extensive NumPy vectorization, significantly reducing computational overhead and improving compatibility with deep learning policies. The invalid action masking replaced penalty-based constraints by mandatory restrictions on the agent's action space to feasible decisions, thus enhancing the policy's accuracy by focusing on valid and more optimal choices. Datasets containing up to 500 orders were generated, on which PPO and MaskablePPO models from the Stable-Baselines3 library were trained. Each model was trained for 100,000 iterations. Training progress was monitored using TensorBoard. The masked version required 1.49 minutes for training, while the unmasked model completed training in 1.2 minutes. For Masked PPO, the average per-step penalty was 2.41, while for PPO it was 325,000. These results demonstrated that the standard PPO frequently selected invalid actions, collecting heavy penalties, whereas the MaskedPPO accumulated penalties solely related to the schedule length. As a result, on the test dataset, MaskedPPO completed the schedule calculation in 0.18 seconds, producing a schedule with a total duration of 4.590 minutes, compared to 5.4 seconds and 5,127 minutes for standard PPO, which made invalid action attempts in 96 % of the cases. It was found that action masking significantly improved the model's accuracy and convergence despite a slightly longer training time. The results reveal the strong potential of reinforcement learning approaches for order scheduling and combinatorial optimization problems in general. The proposed Masked PPO model resulted in reduced order allocation time compared to the traditional exact CP-SAT method while maintaining higher schedule quality than SPT heuristic approache on problems with 50500 jobs. This work established the foundation for future research involving more complex model architectures based on Set Transformers, Graph Neural Networks, and Pointer Networks. These enable effective generalization and allow the trained policies to be applied to problem instances with higher input dimensions than those seen during training.

Keywords: production scheduling, reinforcement learning, deep learning, artificial intelligence, PPO, Maskable PPO, resource allocation, Gymnasium, intelligent agents.

Introduction

The problem of production-schedule planning and resource allocation is one of the most challenging core problems in mass and small-batch manufacturing, supply chain optimization, and project management. It concerns assigning a graph of interdependent operations to alternative resources while considering technological constraints, resource availability, and expected due dates. This problem belongs to the class of NP-complete combinatorial optimization problems and is related to the traveling salesman and vehicle routing problems. Classical solution approaches are divided into two types: exact methods and heuristics. Exact methods include integer programming, branch and bound, and dynamic programming. Although they guarantee optimality, exact methods reveal extensive computational complexity on large-scale instances and require extensive manual parameter tuning. As the problem size grows, their

practical application becomes difficult, whereas heuristics, despite quick solution generation, cannot ensure acceptable schedule quality. Consequently, approaches that balance schedule-generation speed and quality under limited computation time are needed, as this balance confers a decisive competitive advantage on production planning systems. Advances in machine-learning technologies, particularly deep reinforcement learning, have enabled new alternative approaches to such problems. Specifically, for the traveling salesman problem, reinforcement-learning models have achieved better solutions [1] than well-known heuristics and exact methods.

The Object of study – the process of generating operation schedules in a flexible manufacturing environment.

The Subject of study – reinforcement learning methods with action masking for scheduling problems.

The Aim of the study – development and experimental validation of a production-order planning technology based on the Maskable PPO reinforcement-learning model, with an evaluation of both schedule quality and computational performance.

To achieve the stated aim, the following principal *research tasks* have been set:

- develop a custom environment that models production;
- enable step by step planning simulation with invalid actions masking;
- implement incremental updates of the environment state and the action mask to improve efficiency;
- implement the computation of reward / penalty for the model's action;
- train the Maskable PPO model;
- evaluate the effectiveness of the trained model.

Materials and methods. Based on the Gymnasium package, a specialized simulation environment, JobShopEnv, was developed and is compatible with common reinforcement-learning frameworks. The environment supports a variable number of orders and operations, alternative resources for each operation, as well as dependencies between operations. The internal state of the environment includes the current time, resource availability, the set of completed operations, and an action mask that is updated at every step. State updates are implemented incrementally, i.e., by updating only the dependencies in response to an action, without recalculation of the entire state at each step. The action space is represented as a vector that encodes the choice of operation and its assignment to a given resource. ActionMasker, an environment wrapper, was used to exclude invalid actions, presenting the agent only with allowed actions, with mask entries set to 1.

As the reinforcement learning algorithm, the Maskable PPO model from the SB3-Contrib package was employed. The training was conducted with a fixed number of steps for problem instances of varying sizes, from 5 to 500 orders. Each order in the set comprises 10 operations with variable durations and resource requirements, so processing a set of 50 orders corresponds to scheduling 500 operations. TensorBoard monitoring functionality was used to record the average reward, episode completion time, and the number of steps. Experiments were also run for comparison purposes using the PPO model provided by stable_baselines3 without action masking. The choice of Proximal Policy Optimization (PPO) is justified by its compatibility with action-masking mechanisms and its easy integration with environments with a discrete action space. The alternative Soft Actor-Critic (SAC) algorithm is oriented toward problems with continuous action spaces and requires adaptation for application in future research. The Advantage Actor-Critic (A2C) algorithm is compatible with the developed discrete environment; however, additional modifications are needed to enforce action constraints.

A uniform setup of 100,000 training steps, initial state, and environment parameters was maintained. Accumulated reward, percentage of invalid actions, and episode duration were analyzed. Profiling of environment state updates and model action processing using the cProfile utility revealed significant time overhead on mask computation and feature-vector updates. Consequently, optimizations were applied: incremental updates of the action mask and the array of available operations. All experiments were conducted in a Python 3.10 environment using NumPy and PyTorch without GPU support.

Analysis of recent research and publications. Reinforcement learning is finding increasingly broad applications in the field of combinatorial optimization. This growing interest has been driven by successful demonstrations of deep reinforcement learning in gaming environments and classical NP-hard combinatorial problems such as the traveling salesman problem. These environments feature large state and action spaces and exhibit clear advantages of the considered approach over classical techniques. Study [2] demonstrated the effectiveness of deep learning in the problem of scheduling information transmission in a high-performance network. The resulting model reduced data-transfer time through optimal allocation of data channels while still satisfying strict computation-time constraints.

Among contemporary deep reinforcement learning algorithms, Proximal Policy Optimization (PPO) is a key in combining training stability with efficiency [3]. PPO has been successfully applied to various combinatorial optimization problems due to the agent's ability to adapt to complex and dynamic environment conditions [4]. In the referenced publication, the model was applied to branching direction selection in a branch and bound algorithm. For the set-covering problem, it was shown that PPO achieved performance and accuracy gains over the FSB heuristic as well as over a pre-trained GCNN model.

One of the challenges is training agents in environments with invalid actions. Classical reinforcement-learning approaches, such as DQN or PPO, do not support action prohibition, which leads to the need for large penalties since the model may select actions that violate scheduling rules. Modifications have appeared to address this shortcoming, notably an action mask for PPO [5], which allows for the

specification of a binary vector of permissible actions and thus guarantees that the model will select only those actions. In [6], the influence of different mask types on model behavior was studied in depth, and experiments were conducted in three distinct environments. The experimental results showed that "standard PPO converges significantly slowly or not at all". This demonstrates the value of applying a mask for successful model training in environments with a large number of invalid actions.

The next aspect affecting the successful application of reinforcement learning to combinatorial problems in practice is the variable dimensions of input vectors. The classic PPO model, like Masked-PPO, assumes a fixed dimensions for both the action and feature vectors. In [7], the application of transformers to adapt models to combinatorial problems with variable dimensions was proposed. The model's performance substantially exceeded that of classical CPLEX and LSTM algorithms in terms of efficiency and solution quality. Transformers were also successfully applied in [8] to production scheduling. Decision Transformers allowed the problems to be modeled as a sequential decision making process, where the agent learns to reproduce optimal solutions via the series of actions that led to the optimization of the objective function. Based on this source analysis, it can be concluded that reinforcement learning represents a new stage in developing approaches to combinatorial optimization problems. The development of a suitable and optimized training environment will enable effective model training, evaluation, and comparison.

Research results and their discussion

In the implemented technology, a reinforcement learning (RL) approach was used to solve the production-order scheduling problem. The technology is presented by:

- an environment that simulates the production-planning process (Job Shop Environment);
- an agent wrapper that restricts the action space based on a mask;
- an agent implemented based on the Proximal Policy Optimization (PPO) model.

At each step the agent receives from the environment a reward and a mask (action_mask) that restricts the choice of actions and on this basis updates its policy to determine the optimal operation allocation.

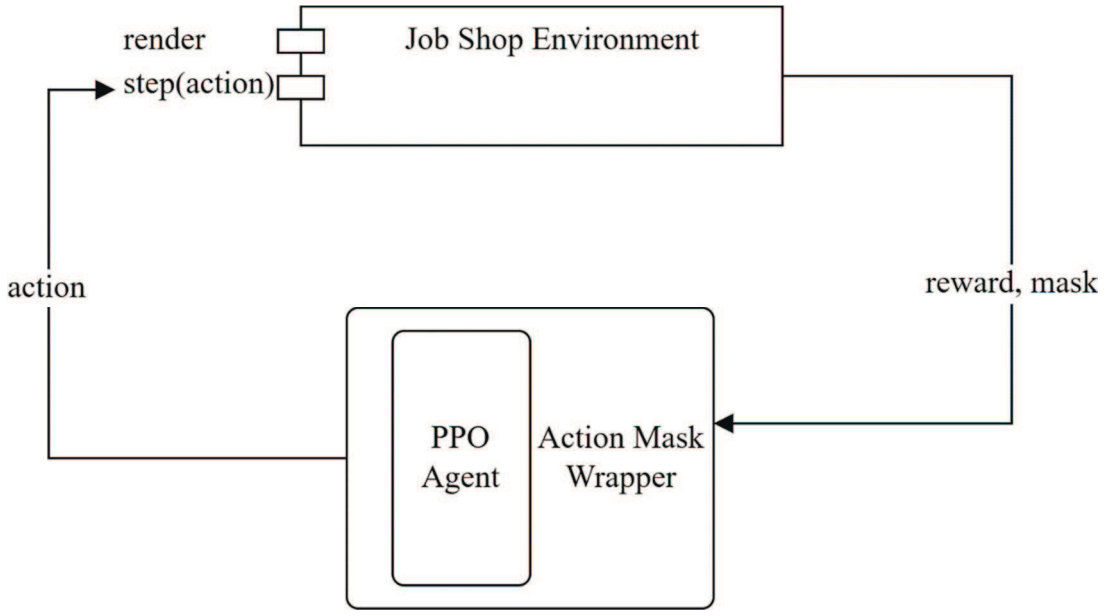

Fig. 1. Reinforcement Learning scheduling method scheme

For environment adaptation to a wide range of models, the Gymnasium standards [9] require the implementation of the following environment methods:

1. Method **step(action)** – defines the agent's interaction with the environment. The method receives an action from the agent, applies it in the environment, and returns the new state, reward, and a flag indicating whether the terminal state has been reached. The structure of the returned data includes:

a. Feature vector – the updated environment state after the performed action.

b. Agent reward – a numerical value indicating how beneficial the action was for achieving the goal.

c. Done indicator signals that the environment has reached a terminal state.

2. Method **render()** – used to visualize the environment state. It is implemented via both console and graphical interfaces. It provides real-time observation of the agent's training and monitors its behavior.

3. Method **close()** – closes the environment after the agent has completed training. It is necessary to properly clean and release of resources after the work is completed.

For the production-order planning environment, the following action encoding has been developed for the Python method **step(action)**: *action* – an integer encoding two

elements: the operation index and the resource index to which it is assigned, *action* = *op_id* * *max_alternatives* + *alt_id*. Where *op_id* – operation index, *alt_id* – resource index, *max_alternatives* – the total number of resources in the environment. Applying the action to the environment involves decoding it into a pair of values by the **step(action)** method. The pair of values (*op_id, alt_id)* is decoded from *action* as follows:

- op_id = action // max_alternatives
- alt_id = action % max_alternatives.

Example of encoding: *op_id* = 2, *max_alternatives* = 5, *alt_id* = 4, then *action* = 2 * 5 + 4 = 14. Reverse decoding: *op_id* = 14 // 5 = 2, *alt_id* = 14 % 5 = 4.

Each action received by the agent is submitted to the environment and processed. The environment rejects some actions that violate scheduling rules with penalties applied to the agent, whereas valid actions change the environment state. This way, a step by step simulation of the planning process occurs [10]. After decoding the action, the environment in the **step(action)** method validates correctness according to the following rules:

1. Whether the received operation *op_id* has already been scheduled in previous steps.
2. Whether all preceding operations have been scheduled.
3. Whether the selected resource *alt_id* is allowed for performing operation *op_id*.

In case any of these conditions are not met, the agent receives the maximum penalty, and the action is rejected by the environment. As a result, the environment state remains unchanged, and the agent must select *action* again. In case of successful completion of the above checks, the environment applies the received action and updates its internal state. The sequence of internal state updates is:

1. Operation start time calculation based on the completion times of preceding operations and resource availability.
2. Operation end time calculation based on the start time and the processing time on the selected resource.
3. Adding the operation entry to the selected resource's schedule.
4. Updating the resource's available time.
5. Adding subsequent operations to the list of operations available for allocation in the next step.

As a result of the state update, the reward for the agent is calculated. Since the production order scheduling problem is formulated as the minimization of total schedule completion time, and the PPO agent minimizes the cumulative penalty for assigning all operations within an episode implemented as the sum of negative rewards at each step – the reward function is defined as the negative increase in schedule duration at the current step. The reward function at step is expressed by the equation:

$$r_t = -(C_t - C_{t-1}), \quad (1)$$

where r_t – the reward at step t; C_t – the schedule duration at the current step; C_{t-1} – the schedule duration at the previous step.

It is important to note that the agent cannot decrease the schedule duration relative to the previous step, only increase it. The only case in which the agent receives no penalty is when an operation is added such that the schedule duration remains unchanged. In all other cases, the agent accumulates a penalty for the increased schedule duration. After completing all steps of an episode, the accumulated sum of r_t will equal the negative of the total schedule duration. This incentivizes the agent to produce a shorter overall schedule throughout the episode. Thus, the reward function reflects the problem objective – minimization of schedule duration in a form compatible with the PPO model. Throughout the training process, PPO accumulates rewards over a fixed window of steps equal to the batch_size and uses this aggregated return to compute updates to the policy parameters, adjusting the probability distribution for selecting the agent's subsequent actions.

At each step, the action mask is updated based on the availability of operations and alternatives. The mask restricts the agent's action space to ensure the selection of an action that allocates only available operations to allowed resources. The action mask is a vector whose elements indicate the availability of each choice for the agent. If an action is available, the mask entry at the corresponding index is 1; if it is not available, the entry is 0. For the scheduling environment, the mask has dimension *n_tasks* * *max_alternatives*, and each cell index corresponds to the encoded *action*. Consider an example of filling the action mask for *n_tasks* = 3, *max_alternatives* = 5. Suppose the current environment state is as follows:

- Operation 1 is available for allocation, allowed resources: 4, 5.
- Operation 2 is unavailable for allocation, allowed resources: 3, 2.
- Operation 3 is available for allocation, allowed resources: 1, 2.

The action mask for the described state has the following value: *action_mask* = [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0].

Visualization of the current environment state is implemented by the **render()** method. The method supports two modes: "console" – displays text-based information about the environment state; "visual" – provides a graphical interface. The visual mode is used to display the final state of the environment in order to present planning results to production stakeholders. Gantt charts give a clear view of time intervals for each operation and resource workload, enabling production specialists to identify delays or scheduling issues. An example of the visualized environment state with 7 resources and 5 scheduled orders is shown in Fig. 2.

Text mode is required for real-time monitoring of the environment state during training and debugging. The console displays the environment's key characteristics:

- Current simulation time.
- Resource availability.
- Allocated operations list.
- List of operations available for allocation.

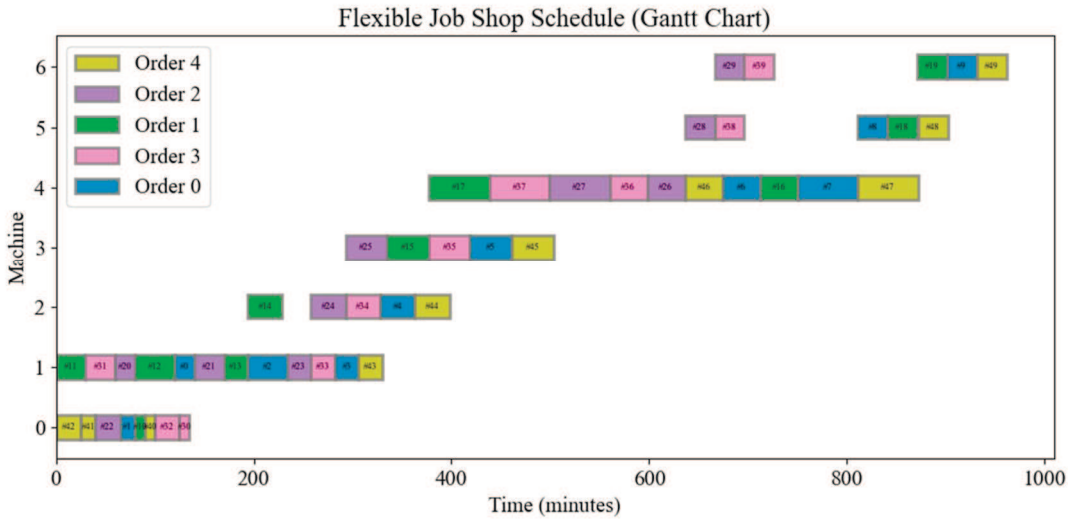

Fig. 2. Visual environment rendering mode – Gant chart

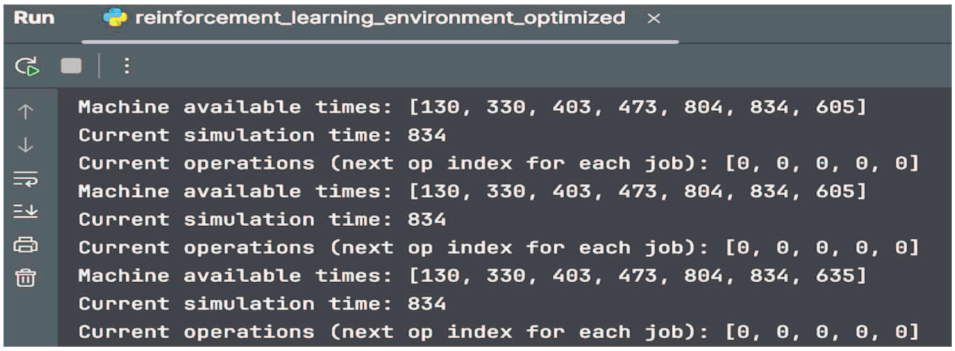

Fig. 3. Console environment rendering mode

An example of the console text-mode representation of the current state is shown in Fig. 3.

The developed environment was used to train agents built on the PPO and MaskablePPO models. The network architectures of both models are identical, with the key feature being two output heads: state value estimation (value_net) and action probability distribution (action_net). The output dimension of value_net is always 1, while the output dimension of action_net is determined by the problem size: *n_tasks* * *max_alternative*. The input-layer dimension also depends on the problem size and is calculated as: *n_tasks* + *max_alternatives* + 1. For 500 operations and 7 resources, the network configuration is:

1. Feature processing, a Flatten layer with output dimension – 508.

2. Dense layer with Tanh activation, output dimension – 64.

3. Dense layer with Tanh activation, output dimension – 64.

4. Linear output layer action_net, output dimension – 3 500.

5. Linear output layer value_net, output dimension – 1.

The difference between the models lies in how action-selection probabilities are computed. PPO uses a standard Softmax over the entire action output vector (including invalid actions), whereas MaskablePPO applies the *action_mask* via Masked-Softmax according to the equation:

$$p(a_i \mid m) = \begin{cases} 0, \ if \ \ m[i] = 0 \\ \dfrac{e^{h_i}}{\sum\limits_{j,m[j]=1} e^{h_j}}, \ if \ \ m[i] = 1 \end{cases}, \qquad (2)$$

where $p(a_i,m)$ – the probability of the agent selecting action; a_i – the action with index i; m – the vector *action_mask*, h_i – the model's output for action i.

Each model was trained for 100,000 steps using the hyperparameters specified in Table 1.

As a result of the model training experiments, employing the Maskable PPO algorithm with incremental action-mask updates significantly improved training stability and convergence rate.

Table 1. PPO model hyperparameters

Hyperparameter	Value
policy	"MultiInputPolicy"
learning_rate	1 e-4
batch_size	64
n_steps	2048
n_epochs	10

The TensorBoard plot presented in Fig. 4 demonstrates the dynamics of the mean reward per episode (ep_rew_mean) during training of the PPO models in two variants: PPO and MaskedPPO. Since the problem is defined as a minimization problem, the goal of training is to decrease the penalty's absolute value, bringing the negative reward closer to zero. For comparing agent training quality, the average penalty per step is used, computed by the equation:

$$r_{mean} = \frac{r_{ep}}{T}, \tag{3}$$

where r_{mean} – the average penalty per step; r_{ep} – the accumulated penalty for the episode; T – the number of steps in the episode.

For MaskablePPO, the mean penalty at the start of training is 2.41 and steadily improves during training, decreasing by 7 % over 100,000 steps. By contrast, for PPO the value starts at 325,883 and fluctuates up to 22 % worse. This indicates instability in the PPO training process. In the case of PPO without masking, the penalty additionally accounts for all unsuccessful action attempts (e.g., selecting an occupied resource), and a fixed penalty of 100,000 is applied for each, explaining the high per-step penalty values. The model faces an excessively large action space [11] and, as a result, performs a significant number of invalid actions, accumulating excessive penalties. These excessive penalties negatively affect the formation of associations between actions and their outcomes, as noted in [12].

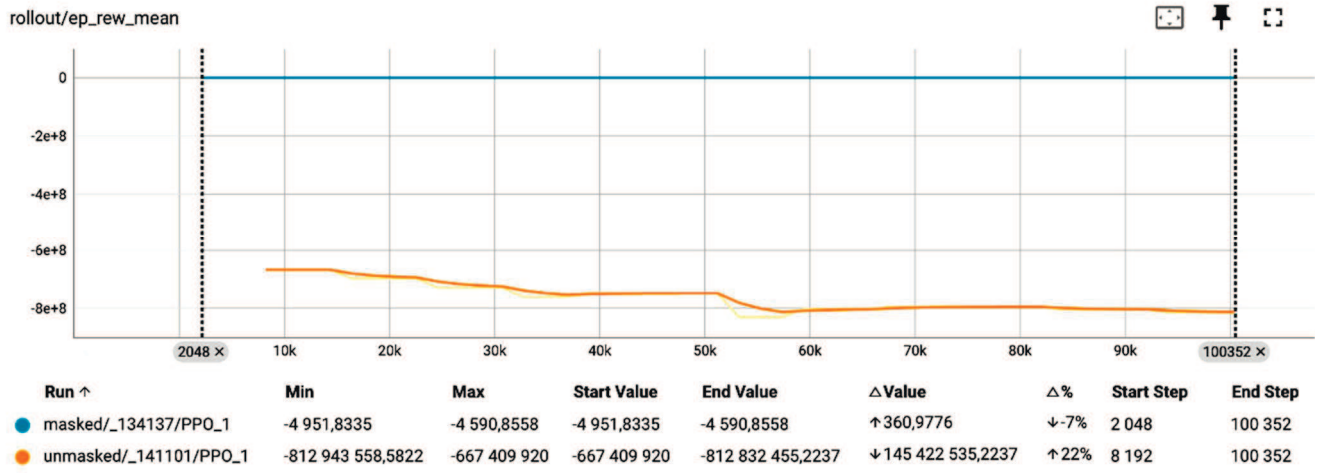

Fig. 4. Episode penalty over training step

In the unmasked environment model training is characterized by extremely high values of loss and value loss on the order of 10^{12}, while entropy remains stable at around –6,4, which may indicate problem complexity for the model or convergence issues due to the large search space and lack of efficient action constraints [13]. In contrast, in a masked environment, loss values are much lower 10^4; policy entropy stabilizes at approximately –3.5 to –3.6. Approximate KL divergence values remain low, indicating training stability and the adequacy of the applied action-masking mechanism. Meanwhile, the key performance metric (FPS) in the masked environment is somewhat lower, a consequence of the additional computational overhead of the masking process. Additionally, the percentage of invalid actions is computed:

$$IR = \frac{N_{inv}}{N} \cdot 100\,\%, \tag{4}$$

where N_{inv} – the number of invalid action attempts, N – the total number of action attempts.

Each attempt to select an invalid action is counted separately, even if it occurs within the same state, since such an error leads to wasted time without advancing toward the final solution. The experimental results are included in Table 2 for model comparison.

Table 2. PPO, MaskedPPO model training result comparison

Parameter	PPO	Masked PPO
Total Timesteps	100 352	100 352
Training time, s	120	141
Loss range	$1.21 \cdot 10^{12}$ – $1.41 \cdot 10^{12}$	$2.78 \cdot 10^{4}$ – $3.96 \cdot 10^{4}$
Value Loss range	$2.34 \cdot 10^{12}$ – $2.79 \cdot 10^{12}$	$6.33 \cdot 10^{4}$ – $7.25 \cdot 10^{4}$
Policy Gradient Loss	$\approx 10^{-6}$	$\approx 10^{-3}$
Approximate KL divergence	≈ 0	$\approx 10^{-4}$
Policy entropy	–6.4	–3.6
Percentage of invalid actions after training	96 %	0 %
Scheduling time for 50 orders after training, s	5.1	0.19

The obtained results reveal that the application of masking significantly improves the efficiency and stability of PPO model training, ensuring rapid convergence to an adequate behavior policy under specified conditions. Moreover, a substantial reduction in the percentage of invalid actions was observed from over 96 % in the baseline PPO to 0 % in the masked variant. As a result, the decision making time of the trained model for a set of 50 orders decreased from 5.1 s to 0.19 s, and the demonstrated acceleration confirms the

practical feasibility of using the masking mechanism when training PPO models for the order planning problem. To assess the stability of the training results, both models – PPO and Masked PPO – were evaluated over three runs with explicitly fixed *seed* values. Table 3 presents the outcomes of the trained models, demonstrating schedule quality and computational performance.

Table 3. Performance of PPO and Masked PPO models with different seed values

Model	seed	Schedule duration, min	invalid actions, %	Inference time, s
PPO	37	5158	86.91	4.14
	73	5440	97.85	6.02
	141	5287	96.26	5.39
Masked PPO	37	4891	0.00	0.17
	73	4932	0.00	0.18
	141	4974	0.00	0.17

Masked PPO delivers stable results across all metrics – zero percent invalid actions, and consistent schedule durations and inference times regardless of the *seed*. In contrast, the unmasked PPO model exhibits unstable behavior, with a high percentage of invalid actions (over 85 %), leading to longer computation times and significantly extended schedule durations.

Three configuration sets from Table 4 were tested to assess the impact of hyperparameters on the stability and training speed of the Masked PPO model.

Table 4. Hyperparameter sets for sensitivity analysis of the Masked PPO model

Set	learning_rate	batch_size	n_steps
basic	$1 \cdot 10^{-4}$	64	2048
aggressive	$4 \cdot 10^{-4}$	32	1024
precise	$5 \cdot 10^{-5}$	512	4096

Fig. 5 shows the mean reward per episode ep_rew_ mean over the number of training steps.

The plot shows the aggressive configuration is characterized by a rapid initial increase in mean reward during the early training steps. In contrast, the precise configuration demonstrates a more stable trajectory and ensures predictable result quality as the number of training steps increases. Despite its high peak reward, the basic configuration exhibits greater variability and less predictable convergence compared to the precise. Plots illustrating changes in the loss function (loss) were also generated for each model configuration to evaluate the convergence of the training process.

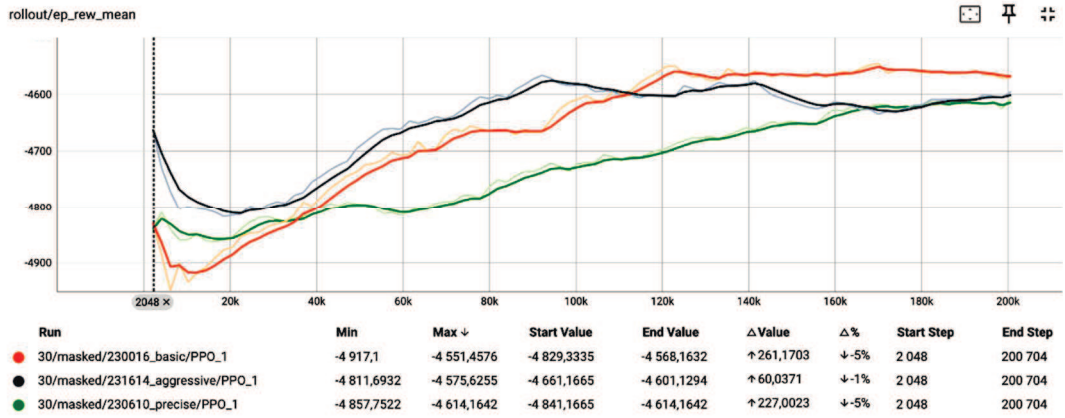

Fig. 5. Dependence of the mean reward (ep_rew_mean) on training steps for the aggressive, precise, basic sets

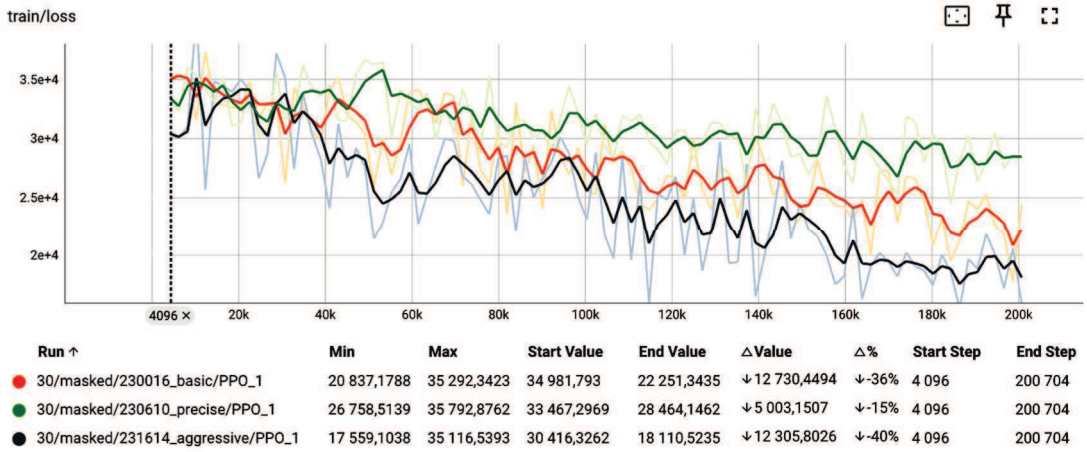

Fig. 6. Dependence of the loss function value on training steps for the aggressive, precise, and basic configurations

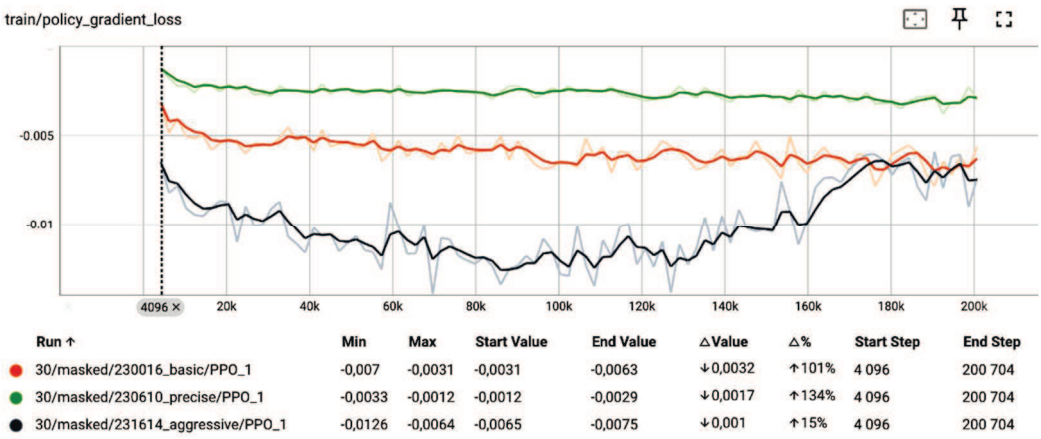

Fig. 7. Dependence of the policy gradient loss (policy_gradient_loss) on training steps for the aggressive, precise, and basic configurations

The aggressive configuration demonstrates the fastest decrease in loss value (–40 %), indicating a high learning performance. At the same time, spikes in the loss trajectory suggest that this configuration delivers learning speed but not overall model stability or effectiveness. In addition to the overall loss value (loss), the dynamics of policy_ gradient_loss were analyzed; this metric reflects the magnitude of gradients applied to update the policy parameters and characterizes the intensity of the agent's policy changes during training.

The precise configuration maintains the greatest stability, maintaining a consistent gradient magnitude with only minor fluctuations throughout training, indicating cautious policy updates. In contrast, the aggressive configuration exhibits high initial gradient dynamics and intensity, but after a sharp decline in policy_gradient_loss values, training slows markedly, potentially signaling a degradation of update efficiency.

Table 5. Comparison of schedule quality and computation time

Number of orders	Number of operations	Method	Schedule duration, min	Computation time, s
50	500	CP-SAT	4 620	11.2
		SPT	5 332	0.02
		PPO	5 127	5.1
		Masked PPO	4 951	0.18
100	1000	CP-SAT	9 451	34.7
		SPT	11 720	0.051
		PPO	12 234	14.1
		Masked PPO	10 202	0.51
500	5000	CP-SAT	Not found	Not found
		SPT	65 678	0.324
		PPO	66 129	81.2
		Masked PPO	57 053	2.4

The trained PPO and Masked PPO models were also compared with classical methods on extended input datasets. Specifically, CP-SAT (Google OR-Tools) was employed as the exact approach, and the shortest processing time dispatching rule (SPT) served as the heuristic method. The results are presented in Table 5.

As shown in Table 5, CP-SAT successfully found solutions for problems with 50 and 100 orders; for 500 orders, a solution was not obtained within the imposed time limit of 5 minutes. CP-SAT solutions serve as the standard for schedule quality. The Masked PPO model delivers the best balance between solution speed and schedule quality across all problem sizes, outperforming the SPT heuristic and standard PPO. Moreover, its computation time is substantially lower than that of CP-SAT. As the problem scale grows, the model maintains a low solution time while its schedule quality remains acceptable and only marginally below that of CP-SAT. This demonstrates the production applicability of the approach for medium- and large-scale planning problems, especially when CP-SAT cannot meet strict time constraints.

Discussion of research results. Employing the Proximal Policy Optimization (PPO) algorithm, a production-order scheduling model was developed in two operational modes: PPO without action masking and MaskedPPO with action masking. The PPO model is characterized by instability in the training process and high loss values, on the order of 10^{12}, while the proportion of invalid actions on application reaches 96 %, and the scheduling time for 50 orders is 5.1 seconds. With the masking approach, a stable model variant was obtained, featuring significantly lower losses (on the order of 10^4), the absence of invalid actions, and reduced computation time to 0.19 s for the same 50 orders.

The scientific novelty of the obtained research – for the first time, a reinforcement-learning training environment has been formalized and developed that effectively enforces constraints on operation sequencing and allocation to alternative resources with varying performance and applies

mechanisms of incremental state updating and invalid action masking to accelerate the training process. The performance of production plan computation is improved relative to classical exact approaches through the application of the Masked PPO reinforcement learning model.

The practical significance of the research – high efficiency of model training within the developed environment. Applying the proposed technology in production planning information systems significantly reduces scheduling time and produces schedules with durations close to optimal.

Conclusions

An environment that simulates the real production-planning process was implemented, and a model for efficient production order scheduling based on the Maskable PPO reinforcement learning method was developed. The research aim was achieved by constructing an optimized environment supporting dynamic action masking and incremental state updates. The resulting framework enabled rapid and efficient training of scheduling models.

Experimental results confirmed the advantage of the action masking approach over classical implementations, particularly in convergence rate and training stability. Employing incremental mask and internal state updates introduced only a minor performance overhead for action mask generation. These findings establish a basis for the development of specialized models and the integration of reinforcement-learning techniques into production-planning systems. The trained Maskable PPO model scheduled 50 orders in just 0.18 seconds, demonstrating its suitability for environments with highly dynamic production processes. Future research will investigate Set Transformers and attention-based graph neural networks. Application of the mentioned architectures will enable the model to generalize: training can be performed on small schedules, while it can be applied to handle significantly larger and more complex configurations.

References

[1] K. Li, T. Zhang, R. Wang, Y. Wang, Y. Han and L. Wang (Dec. 2022). Deep Reinforcement Learning for Combinatorial Optimization: Covering Salesman Problems, in *IEEE Transactions on Cybernetics*, 52(12), 13142–13155. https://doi.org/ 10.1109/TCYB.2021.3103811

[2] Kim, H., Kim, Y.-J., & Kim, W.-T. (2024). Deep reinforcement learning-based adaptive scheduling for wireless time-sensitive networking. *Sensors*, 24(16), 52–81. https://doi.org/ 10.3390/s24165281

[3] Cheng, Y., Huang, L., & Wang, X. (2022). Authentic Boundary Proximal Policy Optimization. *IEEE Transactions on Cybernetics*, 52(9), 9428–9438. https://doi.org/10.1109/ TCYB. 2021.3051456

[4] Zhang, T., Banitalebi-Dehkordi, A., & Zhang, Y. (2022, August). Deep reinforcement learning for exact combinatorial optimization: Learning to branch. *In 2022 26th International Conference on Pattern Recognition (ICPR)* (pp. 3105–3111). IEEE. https://doi.org/10.1109/ICPR56361. 2022.9956256

[5] Zhang, Y., Zhang, Z., & Zhang, L. (2020). Implementing action mask in proximal policy optimization (PPO) algorithm. *Procedia Computer Science*, 176, 2749–2758. https://doi.org/ 10.1016/j.procs.2020.09.122

[6] Wang, Z., Li, X., Sun, L., Zhang, H., Liu, H., & Wang, J. (2024). Learning State-Specific Action Masks for Reinforcement Learning. *Algorithms*, 17(2), 60. https:// doi.org/ 10.3390/ a17020060

[7] Jung, M., Lee, J., & Kim, J. (2024). A lightweight CNN-transformer model for learning traveling salesman problems. *Applied Intelligence*, 54, 7982–7993. https://doi.org/10.1007/ s10489-024-05603-x

[8] Waubert de Puiseau, C., Wolz, F., Montag, M., Peters, J., Tercan, H., & Meisen, T. (2025). Applying Decision Transformers to Enhance Neural Local Search on the Job Shop Scheduling Problem. *AI*, 6(3), 48. https://doi.org/ 10.3390/ai6030048

[9] Krishnan, S., Boroujerdian, B., Fu, W., Chen, Y., Sharma, P., & Bindel, D. (2021). Air Learning: A deep reinforcement learning gym for autonomous aerial robot visual navigation. *Machine Learning*, 110(9), 2501–2540. https://doi.org/ 10.1007/ s10994-021-06006-6

[10] Han, B., & Yang, J.-J. (2021). A deep reinforcement learning based solution for flexible job shop scheduling problem. *International Journal of Simulation Modelling, 20*(2), 375–386. https://doi.org/10.2507/IJSIMM20-2-CO7

[11] Zhang, X., Wang, Y., & Wang, J. (2022). Entropy regularized reinforcement learning with policy gradient. *Information Sciences, 607*, 1063–1079. https://doi.org/10.1016/j.ins. 2022.06.057

[12] Eschmann, J. (2021). Reward function design in reinforcement learning. In Reinforcement Learning Algorithms: Analysis and Applications (pp. 25–33). *Springer*. https://doi.org/10.1007/ 978-3-030-41188-6_3

[13] Hou, Y., Liang, X., Zhang, J., Yang, Q., Yang, A., & Wang, N. (2023). Exploring the use of invalid action masking in reinforcement learning: A comparative study of on-policy and off-policy algorithms in real-time strategy games. *Applied Sciences*, 13(14), 82–83. https://doi.org/10.3390/ app13148283

[14] Sahu, A., Venkatraman, V., & Macwan, R. (2023). Reinforcement learning environment for cyber-resilient power distribution system. *IEEE Access, 11*, 127216–127228. https://doi.org/10.1109/ACCESS.2023.3282182

К. С. Гріщенко, О. О. Писарчук

Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", м. Київ, Україна

ТЕХНОЛОГІЯ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ІЗ МАСКОЮ ДІЙ ДЛЯ ПЛАНУВАННЯ ЗАМОВЛЕНЬ

Високопродуктивне й ефективне планування замовлень – поширена комбінаторна оптимізаційна задача, що виникає в різноманітних контекстах. Побудова моделі, здатної формувати збалансовані за якістю та обчислювальними витратами розклади, – істотний виклик через масштабний простір допустимих дій. У роботі запропоновано високопродуктивне середовище та модель навчання з підкріпленням для розподілу замовлень на ресурси із маскуванням недопустимих дій. Розроблене рішення на основі навчання з підкріпленням долає обмеження традиційних евристичних та точних підходів стосовно швидкості розрахунків та ефективності. Дослідження передбачало проєктування сумісного із інтерфейсом Gymnasium середовища, аналіз продуктивності середовища, розроблення оптимізованих процедур оновлення стану та формування ознак, оцінювання якості навчання моделей PPO та MaskablePPO. Для середовища реалізовано інкрементне оновлення ознак стану та маски дій із широким застосування векторизації NumPy, що дало змогу істотно зменшити обчислювальні витрати на підтримання актуального стану та підвищило сумісність із політиками глибинного навчання. Маскування недопустимих дій замінило політику штрафів, обмежуючи множину вибору моделі лише коректними, чим підвищило точність моделі, зосередившись на виборі оптимальніших та коректніших дій. Сформовано набори даних розміром до 500 замовлень, на яких здійснено навчання PPO та MaskablePPO моделей, наданих пакетом Stable-Baselines3. Виконано 100 000 ітерацій для навчання кожної моделі. Моніторинг процесу навчання забезпечено за допомогою засобів TensorBoard. Час навчання версії з маскуванням становив 1,49 хв, модель без маски витратила на навчання 1,2 хв. Для Masked PPO середній штраф на кроці становив 2,41, тоді як для PPO – 325 000. Результати експериментів свідчать про те, що звичайна PPO часто вибирала недопустимі дії, накопичуючи штрафи за них, тоді як MaskedPPO накопичила штраф лише за довжину складеного розкладу. Завдяки цьому на тестовому наборі даних MaskedPPO здійснила розподіл замовлень за 0,18 с, отримавши розклад тривалістю 4 590 хв, а звичайна PPO за 5,4 с – розклад тривалістю 5 127 хв, здійснивши 96 % помилкових спроб дій. Встановлено, що маскування недопустимих дій покращило якість моделі, забезпечивши вищу збіжність попри довший час навчання. Продемонстровано значний потенціал підходу навчання із підкріпленням у плануванні та розподілі замовлень і оптимізаційних комбінаторних задач загалом. Запропонована модель Masked PPO забезпечила розподіл замовлень швидше за традиційний точний метод CP-SAT, зберігши якість отриманого розкладу, вищу за евристику SPT на задачах із 50–500 замовленнями. Закладено основу для подальших досліджень, розроблення складніших моделей на основі Set Transformers, Graph Neural Networks, Pointer Networks, які забезпечують ефективне узагальнення, можливість застосування моделі на задачах із більшою розмірністю вхідних векторів, ніж під час навчання.

Ключові слова: планування виробництва, навчання із підкріпленням, глибинне навчання, штучний інтелект, PPO, Maskable PPO, розподіл ресурсів, Gymnasium, інтелектуальні агенти.

Інформація про авторів:

Костянтин Сергійович Гріщенко, аспірант, кафедра обчислювальної техніки. **Email:** k.hrishchenko@kpi.ua; https://orcid.org/0009-0008-9251-0222

Олексій Олександрович Писарчук, доктор технічних наук, професор кафедри обчислювальної техніки. **Email:** platinumpa2212@gmail.com; https://orcid.org/0000-0001-5271-0248

Цитування за ДСТУ: Гріщенко К. С., Писарчук О. О. Технологія навчання з підкріпленням із маскою дій для планування замовлень. *Український журнал інформаційних технологій*. 2025, т. 7, № 1. С. 35–44.

Citation APA: Hrishchenko, K., & Pysarchuk, O. (2025). Action-masked reinforcement learning technology for order scheduling. *Ukrainian Journal of Information Technology, 7*(1), 35–44. https://doi.org/10.23939/ujit2025.01.035