Vietnam Journal of Science and Technology
DOI: https://doi.org/10.15625/2525-2518/23452

Resource-constrained production scheduling problem with multi-robot task in smart factory using deep reinforcement learning

Toan Thanh Phan*, Tuan Anh Nguyen, Thoai Minh Vu

Thang Long University, Nghiem Xuan Yem Street, Dinh Cong Ward, Ha Noi, Viet Nam

*Email: *pttoan@hnue.edu.vn*

Abstract. Smart factories integrate diverse components such as humans, robots, artificial intelligence, cloud computing, and IoT systems, promising to enhance production efficiency, customization, and waste reduction to meet the demanding requirements of modern industry. A critical component is the Manufacturing Execution System (MES), which bridges the planning and production execution phases and performs functions including data collection, production coordination, equipment maintenance, detailed scheduling, resource allocation, and quality control. However, most current MES systems have not yet integrated resource scheduling capabilities, creating an urgent need for efficient scheduling algorithms that increase productivity, reduce workflow execution time, and optimize resource utilization. The core challenges involve multiple multi-stage tasks with complex constraints on completion time, system resources, and spatial positioning, including managing job allocation for multiple heterogeneous robots, coordinating resources to avoid delays from buffer limitations or bottlenecks, and optimizing execution time to prevent production-line stoppages. This research develops a machine learning model based on reinforcement learning to optimize resource-constrained production scheduling in smart factories, designated DRL-RCPSSP, focusing on improving performance, resource allocation, and adaptability under dynamic conditions.

Keywords: reinforcement learning, smart factory, resource-constrained projects scheduling problem, automated guided vehicles, machine learning, neural networks, artificial intelligence.

Classification numbers: 4.7.1, 4.7.4.

1. INTRODUCTION

Industry 4.0 represents a revolutionary process that integrates intelligent technologies to optimize production processes and methodologies, primarily based on the deep integration of modern technologies such as artificial intelligence (AI), Internet of Things (IoT), automated robots, big data, and cloud computing. One of the representative applications of this revolution is the emergence of smart factories. These are advanced production models where equipment, machinery, robots, and management systems are connected and automatically exchange information in real time. This enables smart factories to automate production processes, make intelligent decisions, minimize resource waste, and increase productivity and quality of manufactured products [1].

The Resource-Constrained Production Scheduling Problem (RCPSSP) specifically, and the Resource-Constrained Project Scheduling Problem (RCPSP) generally, represent some of the most classic and complex problems in the optimization field, with the objective of minimizing workflow completion time (makespan) while adhering to job precedence constraints and resource availability limitations. RCPSP appears widely in fields such as manufacturing, construction, software engineering, logistics, and supply chain management. As an NP-hard problem, RCPSP has been studied since the 1960s with various solution approaches, including exact algorithms, heuristics, and metaheuristics [2, 3].

However, in the context of smart factories, the integration of technologies such as PLC control systems, Internet of Things (IoT), and various robot types into real production environments makes production systems increasingly complex, flexible, and dynamic. This leads to frequent emergence of uncertain factors such as resource changes, job interruptions, technical failures, or order modifications. These factors render traditional methods, which rely on deterministic models, inflexible and difficult to adapt to reality.

Reinforcement Learning (RL), a branch of machine learning, has demonstrated strong potential in solving optimization problems with state-action characteristics in real time, where systems need to learn optimal action sequences through environmental interaction. RL is particularly suitable for scheduling problems in smart factories, where conditions such as robot status, job priorities, or unexpected incidents constantly change [4].

In recent years, many reinforcement learning methods have been applied to solve scheduling problems such as JSSP and FJSSP [5], achieving competitive results compared to traditional algorithms. Unlike traditional algorithms that require manual heuristic design, RL can automatically learn scheduling strategies. Some studies also combine Graph Neural Networks (GNN) with RL to solve problems with graph structures, where GNN is used to extract features from graph-structured data. Compared to other feature extraction methods like CNN or MLP, GNN has the advantage of being applicable to graphs of different sizes without changing network architecture, and has demonstrated good generalization capability in JSSP problems. The application of reinforcement learning has opened new directions for solving RCPSSP scheduling problems in smart manufacturing environments, where requirements include not only optimality but also adaptability, flexibility, and real-time autonomous decision-making capabilities [6–8].

This paper focuses on developing and deploying an advanced machine learning model based on reinforcement learning methods, capable of learning effective scheduling policies to optimize makespan in the RCPSSP scheduling problem in smart factories. The research aims to enhance performance for factory production lines, optimize resource distribution, and ensure adaptation to real production variations such as resource changes and job interruptions.

This paper makes two main contributions to the field of production scheduling in smart manufacturing environments. First, we develop and deploy an advanced machine learning framework that combines Reinforcement Learning (RL) with Graph Neural Networks (GNN) to address resource scheduling challenges in smart factory production lines. Our approach transforms the scheduling process into a sequential decision-making problem formulated as a Markov Decision Process (MDP). The GNN architecture effectively captures the structural dependencies within factory workflows and extracts salient problem features, which are subsequently mapped to action probability distributions through a policy network that guides the reinforcement learning agent's scheduling decisions.

Second, we establish a comprehensive evaluation framework by constructing datasets based on standard benchmarks from PSPLIB and iMOPSE [9, 10], supplemented with real-world production data collected from Company 10, one of Vietnam's leading textile manufacturers. Through extensive experimentation across these diverse datasets, we demonstrate that our proposed DRL-RCPSSP model consistently outperforms traditional heuristic algorithms while exhibiting strong generalization capabilities across different problem scales and complexities.

2. LITERATURE REVIEW

2.1. Machine learning applications in smart factories

In the context of increasingly deep global integration, enterprises compete not only on quality and cost but also face supply chain volatility, high product customization requirements, and fast delivery speeds [11]. This creates complex competitive pressure in industrial production, demanding systems that are more flexible, accurate, and efficient than ever before. Therefore, optimizing operational efficiency and minimizing resource waste in production lines becomes a survival factor. This not only helps enterprises enhance competitiveness but also contributes to sustainable development, cost savings, emission reduction, and meeting stringent environmental standards in international markets.

The Resource-Constrained Production Scheduling Problem (RCPSSP) is precisely the representative mathematical model to address this issue. In RCPSSP, production activities need to be arranged to comply with resource constraints and job sequences, with the objective typically being to shorten total workflow completion time (makespan) [12].

Traditional scheduling methods, despite their foundational role, often cannot meet requirements in smart manufacturing environments characterized by high dynamism and complexity. Machine Learning (ML) has emerged as a breakthrough tool, providing optimal scheduling solutions, cost reduction, performance enhancement, and real-time adaptation to changes.

In this paper, we focus on analyzing the development of production scheduling methods, the role of machine learning in industrial production, and advances in ML-based optimization techniques, while identifying challenges and existing gaps in this research field.

2.2. Traditional production scheduling methods

Scheduling for production lines is an important step to optimize human resources and materials in production line processes while ensuring constraints on job sequence and resource limitations in factories. Before advanced technologies like artificial intelligence and machine learning emerged, numerous research works addressed this issue. Several traditional research methods have been proposed following approaches ranging from simple rule-based heuristics to mathematical optimization models and modern metaheuristics [13, 14].

Algorithms such as First-Come-First-Served (FCFS), Longest Processing Time (LPT), and Shortest Processing Time (SPT) are widely used due to their simplicity, ease of application, and low computational resource requirements. However, they are unsuitable for complex systems as they cannot handle dynamic conditions such as equipment failures or changing demands. Mathematical optimization methods such as Linear Programming (LP), Mixed-Integer Programming (MIP), and Dynamic Programming (DP) have been proposed for scheduling models

with complex production systems having multiple constraints. However, this approach is still limited by computational complexity and difficulty in scaling for large problems [15–19]. Metaheuristic algorithms such as Genetic Algorithm (GA) [20], Simulated Annealing (SA), and Particle Swarm Optimization (PSO) have partially addressed scalability issues, allowing efficient approximate solutions for large-scale problems [21]. Nevertheless, they typically require complex parameter tuning and have not adapted well to real-time changes.

2.3. Machine learning in production scheduling

In the context of smart factories with integrated environments incorporating diverse resources such as humans, various robot types, IoT systems, etc., this model presents many complex issues such as production environment volatility, where market demand, raw material supply, and equipment status can all change in real time. Production planning and coordination become more difficult when factories must handle multiple customized orders, limited resources, and tight time constraints. Additionally, objectives of minimizing waste, optimizing efficiency, and maintaining operational stability constantly pressure traditional operational systems.

In this context, Machine Learning emerges as a powerful tool to address challenges in production scheduling. Advanced machine learning models, especially reinforcement learning, through their ability to process big data, recognize complex patterns, and predict outcomes, enable smart factories to build flexible, self-learning, and adaptive scheduling systems. Machine learning models not only help predict job completion times and efficiently allocate resources but also proactively detect risks such as equipment failures or production line bottlenecks [22].

Unsupervised learning is a machine learning model that autonomously seeks patterns and structures in data without requiring preexisting labels. Instead of relying on pre-labeled data, this algorithm analyzes and groups data points based on similarities and differences from input datasets. Typical algorithms of this method include k-means clustering and hierarchical clustering, which have been applied to RCPSP scheduling problems by grouping jobs or resources with similar characteristics. This clustering helps achieve more efficient resource allocation and reduces scheduling complexity, especially in multi-machine environments with diverse job requirements [23].

Reinforcement Learning (RL) is becoming a promising approach to address adaptive scheduling needs in smart factories which is one of the most complex scheduling problems in production and project management. In this problem, the objective is to determine job execution sequences such that total workflow completion time (makespan) is minimized while still adhering to resource and job precedence constraints. Unlike traditional methods such as heuristics or mathematical optimization that lack adaptability in dynamic environments, reinforcement learning allows models to learn scheduling policies through repeated interaction with simulation environments. At each step, the agent chooses actions (such as deciding which job to execute next), then receives feedback (rewards or penalties) from the environment and updates its policy to gradually approach optimal solutions [24–26].

Convolutional Neural Networks (CNN) are among the most widely used networks in many fields due to their effective feature extraction capability from structured data such as images, videos, time series, etc. CNN operates by applying filters to local spatial data regions, thereby detecting characteristic data patterns. Through combining multi-layer convolution architecture, pooling operations, nonlinear activation functions, and fully connected layers, CNN has demonstrated superior effectiveness in recognition, classification, and prediction problems.

In production scheduling optimization, particularly the Resource-Constrained Production Scheduling Problem (RCPSSP), CNN networks have been designed to automatically learn and select effective scheduling strategies. Instead of manually designing priority rules as in traditional algorithms, CNN is trained on sample datasets to predict optimal scheduling rules based on training data characteristics—such as complexity level, resource load, relationships between jobs [27, 28].

3. MATHEMATICAL MODEL FOR PRODUCTION SCHEDULING PROBLEM AND SOLUTION METHOD

3.1. Problem description and assumptions

In a smart factory, there are multiple parallel production lines, each executing a series of jobs according to a predetermined sequence following workflow structure. Each operation requires the use of limited resources, specifically different robot types such as robotic arms, AGV (Automated Guided Vehicle) robots, inspection robots, etc. Each robot type has a finite quantity, with each robot capable of serving only one job at a time. The objective is to find a production scheduling method to complete all jobs with minimum workflow completion time (makespan) while ensuring constraints on job sequence and robot resource limitations [29].

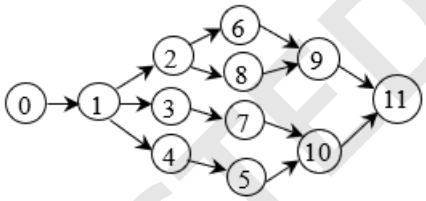

Figure 1. The relationship between jobs in a workflow.

The detailed mathematical model of the problem is as follows:

We denote the workflow as a Directed Acyclic Graph (DAG) represented by G = (J, E), where:

- J is the set of vertices, each vertex represents a job;

- J={1, 2,…,N } is the set of jobs, N is the number of jobs;

- E represents the data dependencies between these jobs. The edge (i, j) ∈ E means job i must complete before job j can be executed;

- R = {1, 2, ..., M} is the set of robot resources.

Parameters:

- $p_i \geq 0$: is the processing time of job i;

- $H = \sum_{i \in J} p_i$; T ={0, 1, 2,..., H};

- $R_{i,m} \geq 0$: amount of resource m needed to execute job i;

- R_m: is the total units of resource m available in the system.

Decision variables:

- $x_{i,t} \in \{0, 1\}$;

- $x_{i,t} = \begin{cases} 1; \text{if job i starts at time t} \\ 0; \text{otherwise} \end{cases}; \ \forall i \in V, t \in T = \{0,1,2,\dots,H\};$

- S_i: start time of job i; $S_i \geq 0$; $\forall i \in V$;

- C_i: completion time of job i; $C_i = S_i + p_i \ \ \forall i \in V$;

- C_{max}: $C_{max} \geq C_j$; $\forall j \in J$ (completion time of the entire workflow);

- Objective function: $C_{max} \rightarrow$ min.

Problem constraints:

- Each job i is executed only once:

$$\sum_{t \in T} x_{i,t} = 1; \ \forall i \in V$$

- Completion time of job i:

$$C_i = S_i + p_i; \ \forall i \in V$$

- Precedence constraint: for all (i, j) ∈ E; job j cannot start when job i has not completed

$$S_j \geq C_i + p_i; \ \forall (i,j) \in E$$

- Resource constraint: if job i starts at time t, it will occupy resources at all time points

The explanation of examples is below:

Table 1. Example of resource types and quantities.

Resource	Notation	Available Quantity
Robot resource type 1	R1	12
Robot resource type 2	R2	13
Robot resource type 3	R3	4
Robot resource type 4	R4	12

Table 2. Example of robot resource requirements for jobs.

Job	Time p_i	R1	R2	R3	R4
J_1	5	1	0	1	0
J_2	8	4	0	0	0
J_3	4	10	0	2	0
J_4	6	0	0	0	3
J_5	3	3	0	1	0
J_6	8	0	0	0	8
J_7	5	4	0	1	0
J_8	9	0	1	0	2
J_9	2	6	0	0	0

In this data table, job J_2 has an execution time of 8 time units and requires 4 type-1 robot resources, while job J_3 has an execution time of 4 and requires 10 type-1 robot resources and 2 type-3 robot resources.

3.2. Development and implementation of deep reinforcement learning network for RCPSSP problem

The Resource-Constrained Production Scheduling Problem (RCPSSP) is one of the most common NP-hard combinatorial optimization problems in practice. In this research, we propose a Deep Reinforcement Learning (DRL) model combined with Proximal Policy Optimization (PPO) algorithm to learn optimal scheduling policies through repeated environmental interaction. The Agent is constructed with two main components: the Policy Network (Actor), which is a neural network responsible for generating probability distributions over the action space, i.e., selecting which job should be scheduled next at each time step. The second component is the Value Network (Critic), a separate neural network responsible for estimating the expected return of the current state. This network supports the policy update process by providing more accurate evaluations of chosen actions. The Proximal Policy Optimization (PPO) algorithm is also used during agent training to limit the change magnitude per update, avoiding overly large updates that could destabilize the neural network [27, 30].

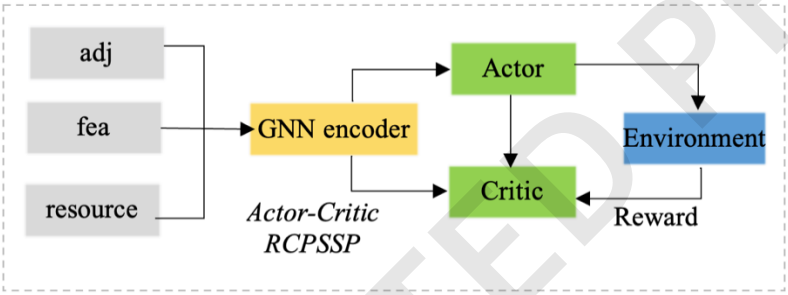

Figure 2. Architecture of the proposed DRL-RCPSSP framework for production scheduling in smart factories.

We have constructed a deep reinforcement learning model for the production scheduling problem according to the schematic in Figure 2, comprising the following main components:

- Environment env = (adj, fea, resource) comprising three basic components:

+ Adjacency matrix adj representing dependencies between jobs in the production schedule:

$$Adj[i,j] = \begin{cases} 1 \text{ nếu } j \in successors(i) \\ 0 \text{ nếu } j \notin successors(j) \end{cases}$$

where i is the current job index and j is the next job index. If job j is a successor of job i, then A[i][j] = 1, otherwise A[i][j] = 0.

Example:
[0, 1, 0, 0], # job 0 → job 1
[0, 0, 1, 1], # job 1 → job 2, job 3
[0, 0, 0, 0],
[0, 0, 0, 0]]

+ Feature matrix fea comprising three basic information types: job numbers, execution time required for jobs, and resource requirements of jobs. The fea matrix has the following structure:

$$fea[i] = (j_i, Duration_i, R_i^1, R_i^2, ..., R_i^k)$$

where n is the number of jobs and k is the number of robot types in the factory, for example:

$$fea = \begin{pmatrix} job\,1 & 10 & 3 & 2 \\ job\,2 & 5 & 2 & 1 \\ job\,3 & 8 & 1 & 3 \end{pmatrix}$$

+ Resource vector to store the number of various robot types in the factory

- State s ∈ S: describes the current state of jobs in the schedule:

+ List of completed and uncompleted jobs

+ Current resource status (remaining quantities)

+ Earliest available time to execute each job

+ Dependency structure between jobs

- Action a ∈ A: used to select a feasible job (ready to execute with sufficient resources) for scheduling. Each action corresponds to the index of the selected job.

- In the DRL environment, job selection is formulated as a priority-based scheduling process with hard feasibility constraints. At each iteration step, the environment generates a runnable job set containing only unscheduled jobs that satisfy precedence constraints. The agent selects a job exclusively from this set using a GNN-based policy network, which evaluates job priorities based on job features, global workflow context, and current resource status. During training, actions are sampled from the resulting probability distribution to promote exploration, while greedy selection is applied during evaluation. After a job is selected, the environment advances time until all resource and completion constraints are met and then executes the job in a non-preemptive manner, updating the runnable set for subsequent decisions.

Algorithm: Job Selection by the DRL Agent

Input:state=(adj,node_features, resource_state, runnable_jobs)

h_nodes, h_global ← GNN(adj, node_features)

for each job j in runnable_jobs do

  φ(j) ← concat(h_nodes[j], h_global, resource_state)

score(j) ← Actor(φ(j))

π(j|state) ← softmax(score(j)) ∀ j ∈ runnable_jobs

if training then

  a ← sample from Categorical(π)

else

  a ← a ← $\arg \max_{j \in \text{runnable\_jobs}} \pi(j|s)$

return selected job a

- Reward r:

$$r^{(k)} = \frac{C_{max}^{(k-1)} - C_{max}^{(k)}}{C_{max}^{(k-1)} + \varepsilon} \quad (1)$$

where: $C_{max}^{(k-1)}$ is the makespan at k-1 iteration; $C_{max}^{(k)}$ is the makespan at k iteration; and ε is the constant, $\varepsilon = 10^{-3} \times C_{max}^{(0)}$.

The proposed normalized reward measures the relative improvement of makespan between two consecutive steps. It is positive when the makespan decreases, negative when it increases, and scale-invariant due to normalization by the previous makespan. We evaluate its effectiveness by analyzing reward density and variance during training, and by measuring the correlation between cumulative reward and final makespan. Ablation studies against unnormalized and alternative normalization schemes are conducted to assess convergence speed, stability across random seeds, and final solution quality.

- Policy $\pi_\theta(a|s)$ is a neural network (policy network) that outputs probability distributions for action selection from the current state.

- Value V(s): estimation of expected reward from the current state

$$V(s) = E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s] \quad (2)$$

The deep reinforcement learning model is trained and optimized using a loss function comprising three main components:

- Policy Loss: The main objective of the reinforcement learning model is to maximize the expected value of the reward function after each step. However, to avoid overly aggressive updates, we used a clipping mechanism to limit changes in action probability ratios between old and new policies, aiming to find actions that maximize profit while not changing the current policy too drastically [31].

$$\text{Policy loss} = E[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)] \quad (3)$$

where: $r_t(\theta)$: ratio of action probability selected at time step t in new policy compared to old policy

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{t-1}(a_{t-1}|s_{t-1})} \quad (4)$$

where $\boldsymbol{\pi_\theta(a_t|s_t)}$ is the probability of selecting action at in state s_t under policy $\boldsymbol{\pi_\theta}$

A_t is the advantage at time step t, calculated using Generalized Advantage Estimation (GAE):

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots \quad (5)$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \quad (6)$$

where: r_t is the reward received at step t; $V(s_{t+1})$ is the predicted value for the next state s_{t+1}; $V(s_t)$ is the predicted value for the current state s_t; γ is the discount factor that controls the importance of future rewards; λ is the GAE parameter that controls the variance-bias tradeoff during training.

To improve the accuracy of advantage estimation while simultaneously reducing the variance of this estimate, the model employs the Generalized Advantage Estimation (GAE) method. The advantage at each time step represents the difference between the actual cumulative reward received by the agent and the predicted value from the value network (Critic). By combining predicted values from multiple steps (using the γ and λ coefficients), GAE helps reduce reward variance, thereby decreasing volatility during the training process.

In our implementation, the discount factor γ is not hard-coded but defined as a configurable parameter (args.gamma). In the experiments reported in the paper, γ is set to 1.0, which is consistent with the finite-horizon, episodic nature of the production scheduling problem. The use of normalized rewards, PPO clipping, and Generalized Advantage Estimation ensures stable convergence despite γ = 1.

- Value Loss: This is part of the critic network, whose purpose is to predict the value of each state. If this prediction is too inaccurate, there will be a penalty for adjustment. The algorithm uses clipping in value loss to avoid excessively high or low values causing training instability.

$$Value\ Loss = E\left[max\left(\left(V_\theta(s_t) - V_{target}(s_t)\right)^2, \left(V_{clip} - V_{target}(s_t)\right)^2\right)\right] \quad (7)$$

where: $V_\theta(s_t)$ is the predicted value of state s_t from the critic network; $V_{target}(s_t)$ is the target value at time step t, computed from the reward value and the next value; V_{clip} is the clipped value calculated from $V_\theta(s_t)$ with deviations constrained within the range $[-\epsilon, +\epsilon]$, and is computed according to the formula:

$$V_{clip} = V_{old} + clip(V_\theta(s_t) - V_{old}, -\epsilon, \epsilon) \quad (8)$$

- Entropy Loss: To encourage exploration during training, the model also uses an entropy loss component. The purpose is to make action probability distributions have high dispersion (more exploration), avoiding situations where agents only follow known actions without trying new ones.

$$Entropy\ Loss = E[H(\pi_\theta(a_t|s_t))] \quad (9)$$

where: $\boldsymbol{H(\pi_\theta(a_t|s_t))}$ is the entropy of the action probability distribution a_t corresponding to state s_t under policy π_θ, and is calculated as follows:

$$H(\pi_\theta(a_t|s_t)) = \sum_a \pi_\theta(a|s_t) \log(\pi_\theta(a|s_t)) \quad (10)$$

In the production scheduling problem model for smart factories, jobs typically do not exist independently but are organized into workflows, each being a sequence of jobs to be executed in order. However, these workflows do not operate independently but have close relationships in terms of resources and time, forming a complex and structurally rich dependency network. In this context, extracting representative features for each job in each workflow becomes an important step for input into machine learning or decision-making models. Instead of relying on manual features or simple statistics, we propose using Graph Neural Networks (GNN) as a deep learning tool to learn representations for each job, considering the context of the entire workflow and overall job network [32].

The proposed GNN architecture comprises 3 GNN layers, each containing two MLP layers. The GNN is responsible for aggregating information from each node in the graph based on relationships with neighboring nodes and information about completion time, required robot resources for jobs, and generating embedding feature vectors. These feature vectors will be used as input for the critic network.

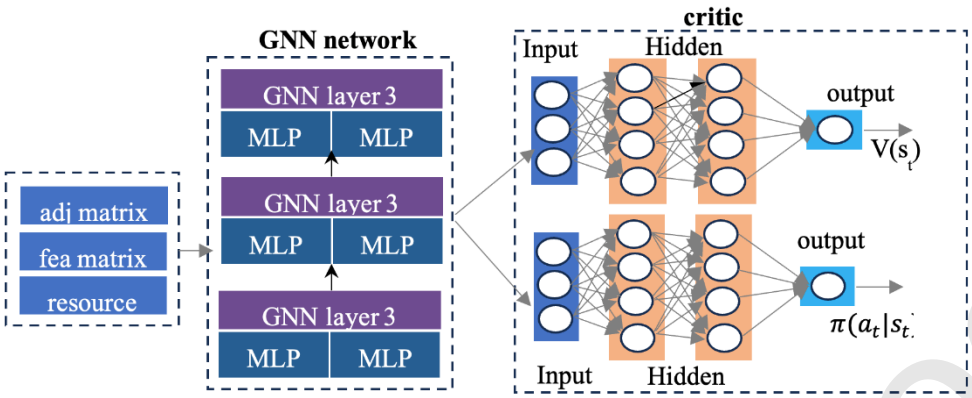

Figure 3. The proposed GNN architecture.

The model input includes adjacency matrix adj representing job order, feature matrix fea with job execution time attributes, and resource requirements for each job.

Notation:

- $h_i^{(l-1)}$: feature vector of node i at layer 1; corresponding to row v of matrix fea;

- N(v): set of nodes adjacent to node v;

- adj: adjacency matrix representing job execution order;

- H(l): features of all nodes at layer 1 in GNN;

- H(0) = fea: input feature matrix of GNN;

- $MLP^{(l)}$: MLP network in GNN layer 1, comprising two linear layers and ReLU activation.

Information aggregation process for each node:

$$H^{(l)} = adj \times H^{(l-1)} \tag{11}$$

For each node i, node features are aggregated according to:

$$h_i^l = \sum_{j \in N(i)}^{n} h_j^{(l-1)} \tag{12}$$

where N(i) is the set of predecessor jobs of job i and $h_j^{(l-1)}$ is the feature of node j at layer 1. Output of GNN layer l is passed through two MLP layers:

$$H^{(l)} = MLP^{(l)}\left(H^{(l-1)} + H^{(l)}\right) \tag{13}$$

with: $MLP^{(l)}(z) = Linear_2^{(l)}\left(ReLU(Linear_1^{(l)}(z))\right)$.

The GNN output consists of embedding feature vectors of jobs in the workflow and will be used as input for the MLP critic network to evaluate the value of the current state V(s).

3.3. Computational complexity and inference efficiency

Notation:

- n be the number of jobs (nodes) in the precedence DAG;

- |E| be the number of precedence edges;

- L be the number of GNN message-passing layers;

- d be the embedding dimension of node representations;

- $F(s_t)$ be the feasible (runnable job) job set at decision step t;

- T be the number of decision steps required to generate one complete schedule.

During inference, the trained DRL policy constructs a schedule through a sequence of job-selection decisions. At each step, a GNN computes embeddings over the precedence graph, and the actor evaluates only the feasible (runnable) job set, resulting in negligible cost for action selection compared to graph embedding. With a dense adjacency representation, the per-step inference complexity scales as $O(L \times n^2 \times d)$, leading to an overall complexity of approximately $O(L \times n^3 \times d)$ for one complete schedule. The environment enforces precedence and resource constraints via event-driven simulation, whose cost grows with the number of scheduling events rather than quadratically with the number of jobs.

4. EXPERIMENTS AND EVALUATION OF DEEP REINFORCEMENT LEARNING MODEL (DRL-RCPSSP)

4.1. Experimental data

To evaluate model quality, we constructed a dataset based on the standard PLIB_dataset, iMOPSE, and practical data from Company 10, one of Vietnam's leading companies in textiles. The dataset was built with workflows simulating 30 to 120 jobs. PSPLIB is a popular international standard dataset widely used in RCPSP scheduling problem research. Instances in PSPLIB include multiple simulated workflows with job numbers from 30 to 120 jobs (denoted as J30, J60, J90, and J120) and various constraints on execution order, time, and resources. Each problem has approximately 400 to 500 instances, 4 to 5 robot resource types, job completion times set from 1 to 10 time units, precedence constraints between jobs built with complexity levels from simple to complex with constraint pairs between jobs ranging from 5 % to 25 %.

iMOPSE is a standard dataset built for multi-skill MRCPSP scheduling problems. Based on this dataset, we constructed a dataset for production scheduling problems with job numbers from 50 to 90 jobs (denoted as I50, I60, I80, I90), 4 robot resource types, job completion times synchronized with PSPLIB instances from 1 to 10 time units. The complexity regarding constraint ratios between jobs in workflows was also built based on iMOPSE descriptions with constraint ratios around 15 % and divided into 3 levels: H(High), M(Medium), and L(Low). Additionally, we collected practical data from Company 10 and built datasets for problems with 30 to 90 jobs, 4 robot resource types, workflows with constraint ratios from 5 % to 10 %.

Table 3. Dataset specifications.

Dataset	Job Count	Instances	Resource Types	AVG density
J30	30	480	4	0.12
J60	60	480	4	0.063
J90	90	480	4	0.04
J120	120	600	4	0.03
I50	50	480	5	0.035

I60	60	480	5	0.07
I90	90	600	5	0.042
G_30	30	300	4	0.21
G_60	60	300	4	0.054
G_90	90	300	4	0.037

Workflow complexity is evaluated through density coefficient:

$$density = \frac{2e}{n(n-1)}$$

where: n is the number of jobs and e is the number of precedence constraints.

4.2. Deep reinforcement learning training parameters

From the prepared dataset, we divided the data into two parts: one for training and one for testing the model, with a ratio of train dataset: test dataset = 70 % : 30 %. Other training parameters are detailed in Table 4.

Table 4. Training parameters.

Hyperparameter	Value	Hyperparameter	Value
Optimizer	Adam	Number of episodes per update	10
Total timesteps	400,000	Update epoch	4
Parallel environments	4	Layers of GNN	4
Learning rate (η)	1×10^{-5}	Hidden dimensions of GNN	32
Discount factor (γ)	1	Layer of policy network	2
GAE parameter (λ)	0.97	Hidden dimensions of policy network	32
Clipping parameter (ε)	0.2	Layer of value network	3
Value function coefficient	0.5	Hidden dimensions of value network	32
Entropy bonus coefficient	0.02	Collect steps of each environment	32

To train the model for the RCPSP production scheduling problem, we used hardware with the following specific configuration:

Table 5. Hardware specification.

Component	Specification	Component	Specification
GPU	NVIDIA A100	VRAM	24GB
CPU	Intel Core i9	CUDA	CUDA 11.2 / cuDNN 8.1
RAM	32 GB DDR4	Framework	TensorFlow 2.x / PyTorch 1.8

4.3. Experimental results

During training, we adjusted the learning rate using Learning Rate Annealing strategy [33], where the learning rate decreases at specific intervals after several epochs. This adjustment helps the model quickly learn features from training data in early epochs, then gradually reduces the learning rate to prevent overshooting optimal values during training, leading to better convergence.

Learning rate update formula:

$$\alpha = \frac{total\_timesteps}{batch\_size}$$

$$\beta = 1.0 - \frac{t-1}{\alpha}$$

$$learning\_rate = \beta \times 1 \times 10^{-5}$$

where total_timesteps = 400,000, batch_size = 256, and t is the t-th iteration step.

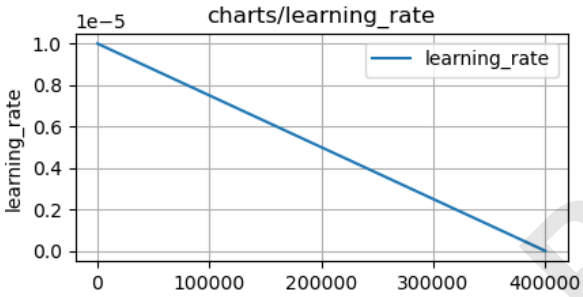

Figure 4. Learning rate adjustment chart.

The value_loss/losses chart plays a crucial role in monitoring model learning quality across phases. The value_loss represents error in estimating expected value of the current state, a fundamental component of many reinforcement learning algorithms like PPO or A2C. Accurate prediction of state expected value is crucial for optimal next job selection in scheduling [34].

Through loss/value_loss charts, we observe that in early phases, value loss fluctuates strongly with high tendencies, indicating the model is still unstable and exploring optimal policies. However, as training steps increase, we observe decreasing and stabilizing trends in value loss, especially after approximately 200,000 steps. This demonstrates the model gradually learns the RCPSP scheduling problem environment structure and improves accuracy in action value estimation.

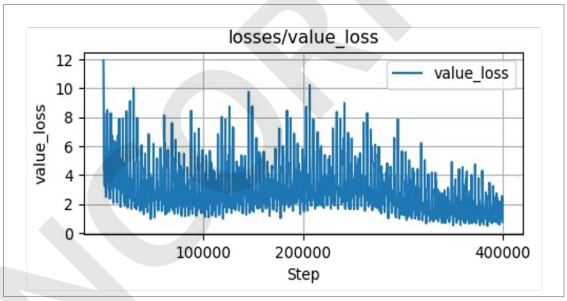

Figure 5. Losses/value_loss Chart for dataset J30

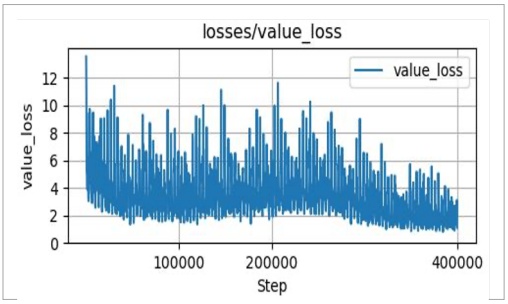

Figure 1. Losses/value_loss Chart for dataset I50

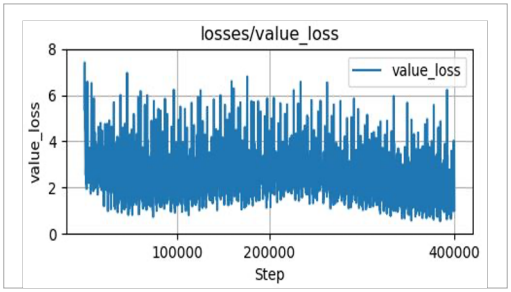

Figure 2. Losses/value_loss Chart for dataset J60

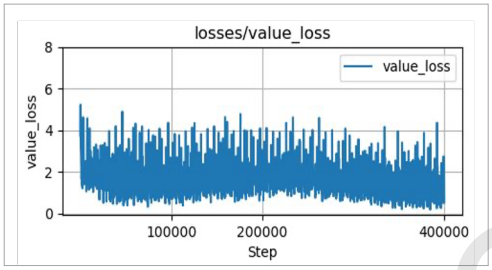

Figure 3. Losses/value_loss Chart for dataset G60

In the context of deep reinforcement learning models, entropy is calculated based on the probability distribution of actions generated by the policy. The entropy value reflects the level of randomness in the agent's action selection behavior. High entropy values indicate that the agent tends to explore various different strategies, while low entropy values suggest that the agent tends to exploit a well-established policy, which may lead to overfitting and missed opportunities to discover better solutions.

Through the entropy loss charts in Figures 9-12 below, the entropy values are observed to fluctuate within the range of 1.0 to 1.4. This entropy fluctuation range demonstrates that the model maintains a good balance between exploration and exploitation, which is a crucial factor for ensuring that the scheduling policy finds optimal solutions [35].

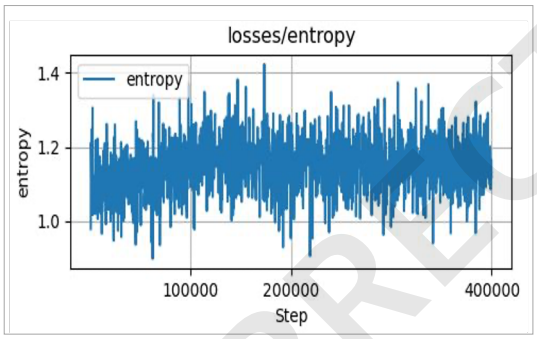

Figure 5. Entropy/losses chart for dataset J30

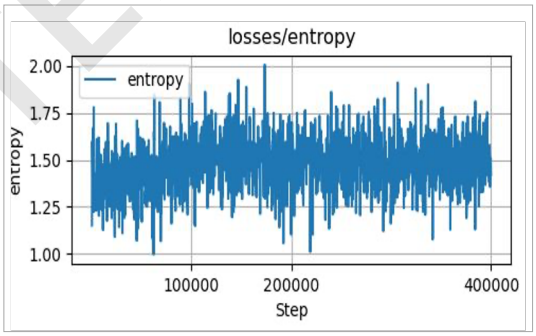

Figure 5. Entropy/losses chart for dataset G30

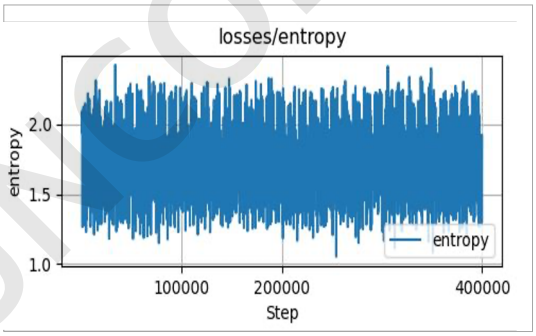

Figure 7. Entropy/losses chart for dataset J90

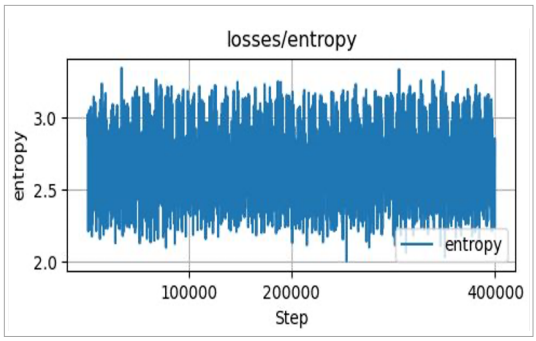

Figure 7. Entropy/losses chart for dataset I90

In the Resource-Constrained Production Scheduling Problem (RCPSSP), one of the most critical optimization objectives is to minimize the total workflow completion time, measured by the makespan metric. Therefore, we evaluate whether the model has learned optimal scheduling strategies and generalization capabilities to new instances based on the changes in average makespan values on both training and testing datasets.

Based on the training average makespan chart presented in Figure 13, we observe that in the initial phase, the makespan rapidly decreases from approximately 76 to 71-72 and maintains stability at this level throughout the training process. This demonstrates that the model has learned effective scheduling strategies to optimize workflow completion time on the training dataset. Similarly, the test average makespan chart shows that the makespan trend also decreases significantly from approximately 61 to 57.5 and maintains stability around this value. These results have been validated across workflows with different complexity levels and scales, as shown in Figures 13-14. This proves that the model not only learns well on the training set but also has the capability to generalize and learn good scheduling strategies on the test dataset. Particularly, through the two charts in Figures 15 and 16, we observe rapid and stable convergence of makespan, with no signs of performance degradation from training to testing models, indicating that the model does not suffer from overfitting. Therefore, the model demonstrates the ability to find scheduling strategies in an optimal and efficient manner.

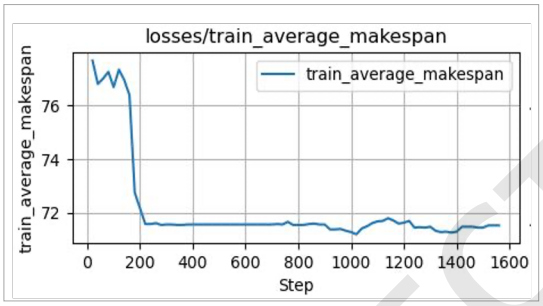

Figure 9. Chart of losses/train-average-makespan-J30

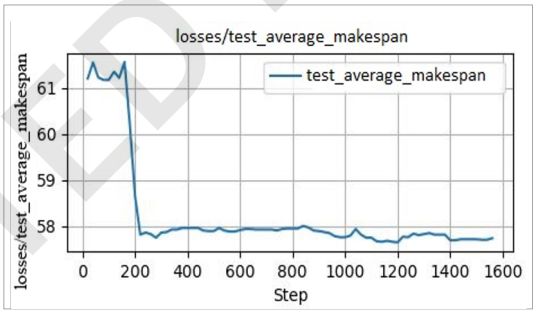

Figure 9. Chart of losses/test-average-makespan-J30

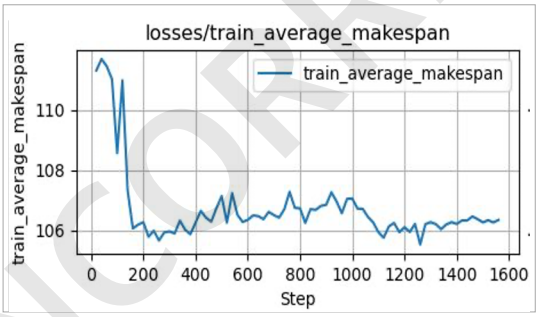

Figure 11. Chart of losses/train-average-makespan-J90

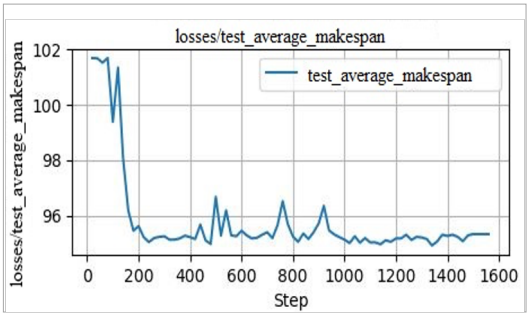

Figure 11. Chart of losses/test-average-makespan-J90

To evaluate the effectiveness of the deep reinforcement learning model implemented for the RCPSP production scheduling problem, it is necessary to compare the optimal results obtained by our developed model with traditional heuristic methods to determine solution quality, applicability, and model stability. In this study, after training the model on the training dataset

and using the RL model to find optimal solutions on the test dataset, we performed solution quality comparisons based on average makespan metrics between the RL model and several popular algorithms including: LFT (Latest Finish Time), LST (Latest Start Time), GRPW (Greatest Resource Position Weight), and MTS (Minimum Total Slack). The comparison results are presented in detail in Table 6. The comparative chart of optimal values obtained by the DRL-RCPSSP model and heuristic algorithms is presented in Figure 17.

Table 6. Comparison of the average makespan of different algorithms in RCPSSP.

Size/ AVG_Makespan	LFT	LST	GRWP	MTS	DRL- RCPSSP
J30	65.30	69.51	74.24	71.45	57.42
J60	92.78	92.67	104.58	99.12	78.43
J90	112.07	116.89	125.01	121.87	93.05
J120	151.08	152.01	164.12	159.48	113.22
I50	85.07	87.17	89.01	88.18	71.03
I60	95.16	95.89	101.61	99.87	78.62
I90	116.29	115.08	118.27	117.21	92.09
G30	57.15	55.21	59.05	58.85	55.27
G60	88.28	90.14	92.46	91.09	76.85
G90	101.09	106.29	108.32	110.67	90.04

Through Table 6 and the comparative chart in Figure 17, we observe that the DRL-RCPSSP deep reinforcement learning model achieves superior results compared to traditional heuristic algorithms in the resource-constrained production scheduling problem. The DRL-RCPSSP model consistently finds the best schedules across all 10 experimental datasets, with makespan values obtained by the deep reinforcement learning model always being 10 % to 30 % smaller than those of heuristic algorithms. This performance gap becomes more pronounced in larger-scale instances such as J90, I90, and J120. The DRL-RCPSSP model not only demonstrates high quality in finding optimal schedules but also exhibits extremely fast inference time ranging from approximately 0.5 to 2.8 ms for workflows with scales from 30 to 120 jobs. With such prediction speed, the DRL-RCPSSP deep reinforcement learning model can completely meet the stringent tact time requirements in industrial production lines.

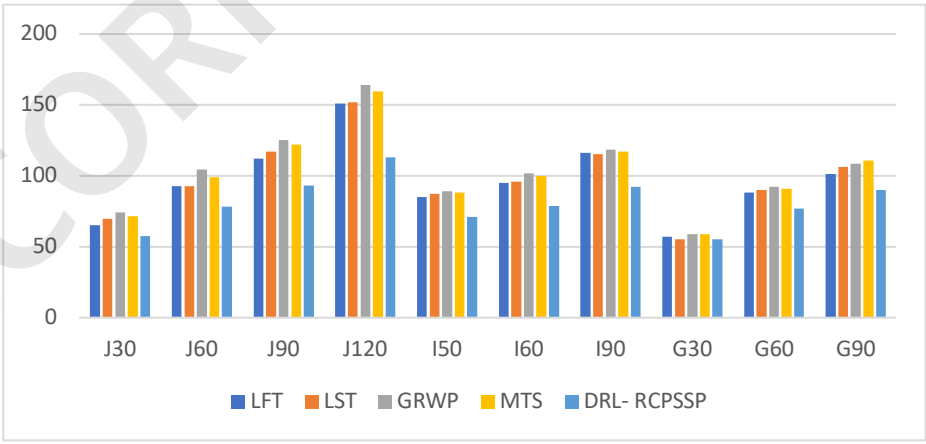

Figure 17. Comparison of different algorithms in RCPSSP.

5. CONCLUSION

This paper constructed a mathematical model for the Resource-Constrained Production Scheduling Problem (RCPSSP) based on the RCPSP scheduling problem, while proposing a DRL-RCPSSP deep reinforcement learning model that combines Graph Neural Networks (GNN) and deep reinforcement learning networks to learn production scheduling strategies for RCPSSP problems. The GNN structure is used to extract RCPSSP problem features (order, job completion times, resource requirements), while the Proximal Policy Optimization (PPO) algorithm is used to train the deep reinforcement learning model to find optimal scheduling policies. This paper also presented a learning rate adjustment method using Learning Rate Annealing strategy with linear learning rate reduction according to iteration numbers (epoch or step) during training, helping the model learn quickly in early phases and achieve high accuracy in final training phases.

Future research directions include improving reward function calculation accuracy, which determines scheduling policy quality in deep reinforcement learning models. Therefore, we will focus on proposing better reward functions while improving Graph Neural Networks (GNN) with Attention mechanisms and Evolutionary Algorithms (EA) to increase feature representation capability, enhance action policy quality, and search for optimal network architectures or parameters.

CRediT authorship contribution statement. Toan Phan Thanh: Conceptualization, Project administration, Supervision. Tuan Anh Nguyen, Thoai Minh Vu: Methodology, Formal analysis, Writing – original draft.

Declaration of competing interest. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

1. Vespoli S., Mattera G., Marchesano M. G., Nele L., Guizzi G. – Adaptive manufacturing control with deep reinforcement learning for dynamic WIP management in industry 4.0. Comput. Ind. Eng., **202** (2025) 110966. https://doi.org/10.1016/j.cie.2025.110966.
2. Blazewicz J., Lenstra J. K., Rinnooy Kan A. H. G. – Scheduling subject to resource constraints: Classification and complexity. Discrete Appl. Math., **5** (1983) 11–24. https://doi.org/10.1016/0166-218x(83)90012-4.
3. Graves S. C. – A review of production scheduling. Oper. Res., **29** (1981) 646–675. https://doi.org/10.1287/opre.29.4.646.
4. Khadivi M., Charter T., Yaghoubi M., Jalayer M., Ahang M., Shojaeinasab A., Najjaran H. – Deep reinforcement learning for machine scheduling: Methodology, the state-of-the-art, and future directions. Comput. Ind. Eng., **200** (2025) 110856. https://doi.org/10.1016/j.cie.2025.110856.
5. Yuan E., Cheng S., Wang L., Song S., Wu F. – Solving job shop scheduling problems via deep reinforcement learning. Appl. Soft Comput., **143** (2023) 110436. https://doi.org/10.1016/j.asoc.2023.110436.
6. Zhang Y., Zhu H., Tang D., Zhou T., Gui Y. – Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. Robot. Comput.-Integr. Manuf., **78** (2022) 102412. https://doi.org/10.1016/j.rcim.2022.102412.
7. Burggräf P., Wagner J., Saßmannshausen T., Ohrndorf D., Subramani K. – Multi-agent-based deep reinforcement learning for dynamic flexible job shop scheduling. Procedia CIRP, **112** (2022) 57–62. https://doi.org/10.1016/j.procir.2022.09.024.
8. Takenaka T., Ashima A., Nishino N. – Strategies for evolving IoT-based product–service systems from emergent synthesis perspective. Procedia CIRP, **112** (2022) 1–5. https://doi.org/10.1016/j.procir.2022.09.014.

9. Zhang C., Song W., Cao Z., Zhang J., Tan P. S., Xu C. – Learning to dispatch for job shop scheduling via deep reinforcement learning. In: *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. Vancouver, Canada, (2020).
10. Myszkowski P. B., Laszczyk M., Nikulin I., Skowroński M. – iMOPSE: a library for bicriteria optimization in multi-skill resource-constrained project scheduling problem. Soft Comput., **23** (2018) 3397–3410. https://doi.org/10.1007/s00500-017-2997-5.
11. Hozdić E. – Smart factory for Industry 4.0: A review. Int. J. Mod. Manuf. Technol., **7** (2015) 28–35.
12. Modrak V., Sudhakarapandian R., Balamurugan A., Soltysova Z. – A review on reinforcement learning in production scheduling: An inferential perspective. Algorithms, **17** (2024) 343. https://doi.org/10.3390/a17080343.
13. Dong P., Xie J., Tang W., Xiong N., Zhong H., Vasilakos A. V. – Performance evaluation of multipath TCP scheduling algorithms. IEEE Access, **7** (2019) 29818–29825. https://doi.org/10.1109/access.2019.2898110.
14. Klein R. – Resource-constrained scheduling problems. In: *Scheduling of Resource-Constrained Projects*. Springer US, (2000) 73–109. https://doi.org/10.1007/978-1-4615-4629-0_3.
15. K.Suri P., Mittal S. – Design of stochastic simulator for analyzing the impact of scalability on CPU scheduling algorithms. Int. J. Comput. Appl., **49** (2012) 4–9. https://doi.org/10.5120/7717-1065.
16. Yohanes K. N. – Dual sourcing strategy in mass customized manufacturing. In: *2008 IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE, (2008) 1088–1092. https://doi.org/10.1109/ieem.2008.4738038.
17. Ghassemi Tari F., Olfat L. – Heuristic rules for tardiness problem in flow shop with intermediate due dates. Int. J. Adv. Manuf. Technol., **71** (2013) 381–393. https://doi.org/10.1007/s00170-013-5478-8.
18. Modrak V., Pandian R. S. – Flow shop scheduling algorithm to minimize completion time for n-jobs m-machines problem. Tehn. Vjesn., **17** (2010) 273–278.
19. Thenarasu M., Rameshkumar K., Rousseau J., Anbuudayasankar S. P. – Development and analysis of priority decision rules using MCDM approach for a flexible job shop scheduling: A simulation study. Simul. Model. Pract. Theory, **114** (2022) 102416. https://doi.org/10.1016/j.simpat.2021.102416.
20. Sadegheih A. – Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. Appl. Math. Model., **30** (2006) 147–154. https://doi.org/10.1016/j.apm.2005.03.017.
21. Pongchairerks P. – Particle swarm optimization algorithm applied to scheduling problems. ScienceAsia, **35** (2009) 89–94. https://doi.org/10.2306/scienceasia1513-1874.2009.35.089.
22. Sunday A. P., Emeka U. C., Chukwumuanya E. O., Chikwendu O. C. – Machine learning applications for production scheduling optimization. J. Explor. Dyn. Probl., **2** (2025) 63–79. https://doi.org/10.31004/edp.v2i4.137.
23. Cao F., Servranckx T., Vanhoucke M., He Z. – A comparison of different clustering algorithms for the project time buffering problem. Comput. Ind. Eng., **199** (2025) 110752. https://doi.org/10.1016/j.cie.2024.110752.
24. Jędrzejowicz P., Ratajczak-Ropel E. – Reinforcement learning strategy for A-Team solving the resource-constrained project scheduling problem. In: *Computational Collective Intelligence. Technologies and Applications*. Springer Berlin Heidelberg, (2013) 457–466. https://doi.org/10.1007/978-3-642-40495-5_46.
25. Mao H., Schwarzkopf M., Venkatakrishnan S. B., Meng Z., Alizadeh M. – Learning scheduling algorithms for data processing clusters. In: *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, (2019) 270–288. https://doi.org/10.1145/3341302.3342080.
26. Choi J., Realff M. J., Lee J. H. – A Q-learning-based method applied to stochastic resource constrained project scheduling with new project arrivals. Int. J. Robust Nonlinear Control, **17** (2007) 1214–1231. https://doi.org/10.1002/rnc.1164.
27. Golab A., Gooya E. S., Falou A. A., Cabon M. – A convolutional neural network for the resource-constrained project scheduling problem (RCPSP): A new approach. Decis. Sci. Lett., **12** (2023) 225–238. https://doi.org/10.5267/j.dsl.2023.2.002.

28. Teichteil-Königsbuch F., Povéda G., González de Garibay Barba G., Luchterhand T., Thiébaux S. – Fast and robust resource-constrained scheduling with graph neural networks. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, **33**. Association for the Advancement of Artificial Intelligence (AAAI), (2023) 623–633. https://doi.org/10.1609/icaps.v33i1.27244.
29. Devanga A., Badilla E. D., Dehghanimohammadabadi M. – Applied reinforcement learning for decision making in industrial simulation environments. In: *2022 Winter Simulation Conference (WSC)*. IEEE, (2022) 2819–2829. https://doi.org/10.1109/wsc57314.2022.10015282.
30. Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. – Proximal policy optimization algorithms 2017.
31. Zhao X., Song W., Li Q., Shi H., Kang Z., Zhang C. – A deep reinforcement learning approach for resource-constrained project scheduling. In: *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE / IEEE, (2022) 1226–1234. https://doi.org/10.1109/ssci51031.2022.10022122.
32. Cai H., Bian Y., Liu L. – Deep reinforcement learning for solving resource constrained project scheduling problems with resource disruptions. Robot. Comput.-Integr. Manuf., **85** (2024) 102628. https://doi.org/10.1016/j.rcim.2023.102628.
33. Nakamura K., Derbel B., Won K.-J., Hong B.-W. – Learning-rate annealing methods for deep neural networks. Electronics, **10** (2021) 2029. https://doi.org/10.3390/electronics10162029.
34. Patterson A., Liao V., White M. – Robust losses for learning value functions. IEEE Trans. Pattern Anal. Mach. Intell. (2022) 1–12. https://doi.org/10.1109/tpami.2022.3213503.
35. Chen Q., Zhang Q., Liu Y. – Balancing exploration and exploitation in episodic reinforcement learning. Expert Syst. Appl., **231** (2023) 120801. https://doi.org/10.1016/j.eswa.2023.120801.