The CSI Journal on
Computer Science and Engineering
Vol. 20, No. 1, 2026
Pages 67-77
doi.org/10.22034/jcse.2025.558772.1066
Regular Paper

Using deep reinforcement learning networks to provide dynamic scheduling in workplace environments

Zahra Salimifard[1], Alireza Chamkoori[2*], Nooshin Rabiee[3]

[1]Department of Computer Engineering, Faculty of Computer, Lian Institute of Higher Education, Bushehr, Iran
[2]Department of Computer Engineering, Faculty of Computer, Islamic Azad University, Bushehr, Iran
[3]Department of Electrical Engineering, Faculty of Electrical Engineering, Lian Institute of Higher Education, Bushehr, Iran
*Corresponding Author Email: Chamkoori@iau.ac.ir

Abstract

The dynamic scheduling problem in modern work environments, such as data centers and cloud computing systems, is considered one of the complex challenges in the field of system optimization due to the heterogeneous and variable nature of input tasks. Traditional and static scheduling methods such as FIFO and SJF, due to their inability to adapt to the real-time and dynamic conditions of the environment, do not show the necessary efficiency for optimizing key performance metrics. In this paper, an asynchronous conditional policy factoring algorithm is presented for dynamic scheduling. This algorithm, by utilizing the policy factoring mechanism, enables learning complex and coordinated policies and improves the convergence speed and efficiency of the learning process by using asynchronous updates. This approach allows the system to effectively deal with the uncertainty and dynamics of the environment and allocate resources optimally. The experimental results clearly demonstrated the absolute superiority of the proposed algorithm in all evaluation criteria, including total task completion time and average task waiting time. The proposed algorithm was able to reduce Makespan by 5% and average waiting time by 13% compared to the best reference algorithm, namely QMIX.

Keywords: Deep reinforcement learning, dynamic scheduling, multi-agent systems, policy factorization.

1. Introduction

In recent decades, with the increasing complexity and scale of industrial and service systems, scheduling has become a key challenge in various fields such as manufacturing, cloud computing, supply chain management, and logistics. The dynamic nature of these environments, characterized by unexpected arrivals, resource failures, and changing priorities, has rendered traditional and static scheduling approaches based on deterministic models and fixed predictions ineffective [1]. Despite their theoretical optimality in ideal conditions, these approaches quickly lose their effectiveness when faced with uncertainty and real-world dynamics, leading to reduced productivity, increased costs, and delays in service delivery. In response to these challenges, the need for dynamic and adaptive scheduling solutions that are capable of making intelligent and real-time decisions in changing conditions is felt more than ever.

Deep reinforcement learning (DRL), as a powerful paradigm in artificial intelligence, has opened up new horizons for solving complex sequential decision-making problems, including dynamic scheduling, with the ability to learn optimal policies through direct interaction with the environment [2]. By incorporating deep neural networks to approximate value or policy functions, deep reinforcement learning algorithms are able to handle large and continuous state and action spaces, which are a hallmark of real-scale scheduling problems. Approaches such as the DQN network and its derivatives have demonstrated their ability to solve finite-dimensional scheduling problems by learning an optimal value-action function [3]. However, these algorithms face challenges in discrete and high-dimensional action spaces and often do not perform well for more complex

problems. In contrast, policy gradient-based algorithms, such as REINFORCE and A2C/A3C, demonstrate greater flexibility in handling large continuous or discrete action spaces by directly learning the policy and, therefore, have attracted more attention in the scheduling field [4].

However, many real-world scheduling problems are inherently multi-agent in nature. In a manufacturing environment or a data center, each machine or server can be considered as an independent agent that must make decisions to allocate resources and execute tasks. These decisions affect not only the performance of the agent itself, but also the performance of the entire system. In such a situation, the use of single-agent reinforcement learning algorithms, which consider the entire system as a single agent with a centralized policy, faces two major challenges [5]: First, as the number of resources (agents) increases, the dimensions of the state and action space grow exponentially, which is known as the "curse of dimensionality," making it practically impossible to learn a centralized policy. Second, this approach is inherently centralized and lacks the scalability and flexibility required for distributed systems. These limitations have paved the way for the emergence and development of multi-agent deep reinforcement learning (MADRL).

The centralized training and decentralized execution (CTDE) method has been proposed as one of the most effective approaches in MADRL [6]. In this framework, during the training phase, agents have access to each other's additional information (such as observations, actions, and rewards) to learn a value function or optimal policy in a centralized manner. This additional information helps to solve the problem of environmental instability caused by simultaneous changes in the policies of other agents. After the training phase is completed, in the execution phase, each agent makes decisions based only on its local observations and in a completely decentralized manner, which ensures the scalability and practical application of the algorithm in real systems.

Actor-Critic algorithms have become popular choices, especially in the CTDE framework. Algorithms such as MADDPG, which use a centralized critic that evaluates the policies of all agents, have achieved considerable success in environments with complex interactions [7]. However, MADDPG and similar algorithms often assume that the agents' decision-making is synchronous; that is, all agents choose and execute their actions at the same time step. This assumption is not true in many dynamic scheduling applications, such as work environments where machines asynchronously complete their tasks and prepare for the next task. Ignoring the asynchronous nature of these systems can lead to unnecessary delays, reduced productivity, and suboptimal resource allocation, as the system has to wait for the slowest agent to be ready to make the next decision. More recent research has attempted to fill this gap by developing asynchronous algorithms. These algorithms allow each agent to make independent decisions as soon as they become available, which is more consistent with operational realities and can significantly improve the system's efficiency [8].

Another fundamental challenge in multi-agent reinforcement learning is the issue of credit allocation. In a multi-agent system, the reward received is usually a collective, team-based signal that reflects the performance of the entire system. It is very difficult to identify the exact contribution and influence of each agent to this collective reward. Without an effective credit allocation mechanism, agents cannot understand which of their actions were beneficial and which were harmful, which makes the learning process slow and unstable. Traditional Actor-Critic algorithms usually use a benefit function to reduce the variance in the policy gradient estimate. However, in multi-agent environments, using a common benefit function for all agents does not solve the credit allocation problem. More modern approaches such as COMA attempt to calculate the contribution of each agent by comparing the current reward with a baseline reward where the agent's action is ignored using a counterfactual advantage function [9]. Although this approach is an important step forward, the exact calculation of the reward function can be computationally difficult in complex environments with a large number of agents. Furthermore, these approaches may not fully account for the conditional and state-dependent effects of agents' actions on each other, which is crucial in scheduling problems where the decision of one machine depends strongly on the state of other machines. In this paper, we introduce the proposed algorithm as a novel multi-agent Actor-Critic approach to solve the dynamic scheduling problem. The proposed algorithm aims to overcome the aforementioned limitations, and its key innovations are focused on three main areas. First, it fully exploits the centralized learning and decentralized execution method to achieve a coordinated and optimal policy at the system level, while providing scalability and distributed execution capability. Second, the algorithm is specifically designed for asynchronous decision-making environments. Each agent executes its policy independently and based on local events (such as the completion of a task), which leads to increased responsiveness and efficiency of the system in real-world environments [10]. Third and most importantly, the main innovation of the proposed method is in providing a novel mechanism for credit allocation based on conditional advantage. Instead of using a general advantage function or a simple counterfactual baseline, the centralized critic learns a advantage function that evaluates the impact of an agent's action by considering (conditionalizing on) the simultaneous or recent actions of other relevant agents. This approach allows for much more accurate credit allocation, because it explicitly answers the question: "How useful was this agent's action given what other agents were doing?" This subtle distinction is of great importance, especially in dynamic scheduling problems where coordination between resources is crucial to avoid bottlenecks and minimize idle time [11]. For example, assigning a long-running task to a machine when the next machine in the production line is idle is more valuable than when the next machine is itself busy with another long-running task. Conditional advantage models these interdependencies and provides a much richer and more accurate learning signal to each agent.

While MADRL algorithms have made significant progress, most of them either assume concurrency or use credit allocation mechanisms that are not optimized for modeling the complex and conditional dependencies in dynamic scheduling problems. By combining three key features (CTDE, asynchrony, and conditional advantage-based credit allocation), our proposed algorithm provides a comprehensive and powerful solution specifically designed to meet the unique needs of dynamic work environments. It not only seeks to improve classical performance metrics such

as average completion time and resource efficiency, but also has the potential to be applied to large-scale intelligent and automated industrial systems by providing a flexible and distributed decision-making framework.

In the second part of the paper, the research foundations and a review of related works are presented. In the third part, we describe in detail the proposed architecture, the mathematical formulation of conditional advantage. In the fourth part, we present the results of experiments and performance evaluation of the proposed algorithm in dynamic scheduling environments, and show how the proposed algorithm can significantly outperform existing advanced algorithms. The conclusion is given in the fifth part.

2. Theoretical foundations of the research and a review of related works

2.1. Reinforcement learning in dynamic scheduling problem

Reinforcement learning is defined as a quintet (S,A,P,R,γ) for modeling the scheduling problem under uncertainty [12].

State space (S): The state $s_t \in S$ at any time t represents a complete picture of the state of the scheduling system. The current state should contain all the information necessary for future decision-making. In a dynamic work environment, the state includes a set of properties such as the state of each machine (idle, running, under repair), the characteristics of the jobs waiting in the queue of each machine (processing time, due date), information about the jobs being executed (remaining processing time), and general system properties such as the current time and the total number of jobs in the system.

Action space (A): At each decision step, the agent chooses an action $a_t \in A$ based on the current state s_t. In scheduling problems, actions are usually reduced to choosing a prioritization rule to assign the next job to an idle machine. These rules are simple heuristics that prioritize tasks based on their characteristics. Instead of using a fixed rule, the reinforcement learning algorithm learns which rule will perform best in each case. The set of actions includes the known rules in Table 1:

TABLE I. ACTIONS TAKEN BY AGENTS

OPERATION	DEFINITION
SPT	Selects the job with the shortest processing time for the next operation.
LPT	Selects the job with the longest processing time for the next operation.
SRPT	Selects the job with the shortest total remaining processing time.
LRPT	Selects the job with the longest total remaining processing time.
STPT	Selects the job with the shortest total processing time.
LTPT	Selects the job with the longest total processing time.
FIFO	Selects the job that entered the queue first.
LIFO	Selects the job that entered the queue last.
LOR	Selects the job with the fewest remaining operations.
MOR	Selects the job with the most remaining operations.

State transition function (P): The function $P(s_{t+1}|s_t,a_t)$ specifies the probability of transitioning from state s_t to state s_{t+1} after performing operation a_t. In scheduling environments, this transition is usually deterministic (assigning a job to a machine changes the state of the system in a certain way), but environmental dynamics such as the arrival of new jobs or machine failures can give this function a stochastic nature.

Reward function (R): The reward function $r_t=R(s_t,a_t,s_{t+1})$ is a numerical signal that evaluates the desirability of transitioning from state s_t to s_{t+1}. The design of the reward function is crucial to align the agent's goals with the overall goals of the scheduling problem (such as minimizing the average completion time or delay). For example, a reward function can be defined as the negative of the tardiness of all jobs. If C_j is the completion time and Dj is the due date of job j, the job's tardiness will be $T_j=\max(0,C_j-D_j)$. The reward at each step is defined as follows:

$$r_t = -\sum_{j \in J_{completed}} T_j \quad (1)$$

$J_{completed}$ is the set of tasks completed in the most recent time step.

Discount factor (γ): $\gamma \in [0,1]$ specifies the importance of future rewards compared to immediate rewards. A value close to 1 gives more importance to long-term rewards. The agent's goal in this framework is to learn a policy $\pi(a_t|s_t)$, which is a mapping from states to probability distributions over actions, such that the sum of future rewards with a discount factor, called the expected return (Gt), is maximized:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (2)$$

The optimal policy π^* is the policy that maximizes the following optimal value-action function:

$$Q^*(s,a) = max_\pi E[G_t \mid s_t = s, a_t = a, \pi] \quad (3)$$

2.2. Multi-Agent Actor-Critic Architecture and CTDE Method

In industrial scheduling problems involving multiple machines, modeling the problem as a single agent faces a scalability challenge. A more natural approach is to consider each machine as an independent agent, resulting in a multi-agent system. In multi-agent reinforcement learning, each agent learns its own policy, but this process is challenged by the instability of the environment; from the perspective of each agent, the environment changes as the policies of other agents change, making it difficult for the algorithm to converge [12]. The Actor-Critic architecture provides an effective solution for learning in large or continuous action spaces. This architecture consists of two distinct networks:

Actor: The actor is responsible for learning and implementing the policy $\pi_\theta(a|s)$. The network receives a state as input and produces an action (or probability distribution over actions) as output. The parameters of this network (θ) are updated using policy gradient methods.

Critic: The critic is responsible for evaluating the current policy of the Actor. This network estimates a value function, such as $V(s)$ (the value of a state) or $Q(s,a)$ (the value of a state-action pair). The output of the critic is used to compute a learning signal with lower variance (such as the benefit function $A(s,a)=Q(s,a)-V(s)$) and guide the Actor update.

To overcome the instability challenge in MARL, the centralized training and decentralized execution paradigm of CTDE has been widely used [13]. The main idea of CTDE is that in the training phase, which is performed offline, system-wide information can be used to learn optimal policies. In this framework:

Centralized training: A centralized critic is trained for each agent (or for the entire system). This critic has access not only to the observations and actions of the local agent, but also to the observations and actions of other agents. This additional information allows the critic to provide a stable and accurate estimate of the value function and overcome the problem of instability of the environment. The critic update is usually done using the time difference error:

$$L(\phi)=E_{(s,a,r,s')\sim D}[(y-Q_{\phi(s,a)})^2] \quad (4)$$

where $y=r+\gamma Q_{\phi^-}(s',\pi_{\theta^-}(s'))$ and s and a are global information (observations and actions of all agents).

Decentralized execution: Each agent makes decisions based only on its local observations. The policy of each agent $\pi_{\theta i}(a_i|o_i)$ depends only on the local observation o_i. The policy gradient for each agent is calculated using the evaluation signal provided by the centralized critic:

$$\nabla_{\theta_i} J(\theta_i) \approx E_{s,a\sim D}[\nabla_{\theta_i} log\pi_{\theta_i}(a_i|\ o_i)A_i(s,a)] \quad (5)$$

where $A_i(s,a)$ is the advantage function calculated by the centralized critic for agent i. This structure allows each agent to operate completely independently and scalable in the real environment after training.

2.3. Asynchronous Decision Making in Scheduling Systems

Many MARL algorithms, including the classical CTDE-based algorithms, operate based on synchronous decision making. In this case, there is a global clock and all agents receive their observations, choose their actions, and execute them simultaneously at discrete and coordinated time steps. This model may be suitable for some applications, such as computer games, but it conflicts with the reality of many real-world systems, especially dynamic work environments [14]. In a manufacturing plant, machines finish their operations at different times. Forcing a machine that has finished its work to wait for other machines to make a collective decision leads to a waste of time and a significant reduction in efficiency.

In contrast, the asynchronous or event-driven model allows each agent to make its next decision immediately upon the occurrence of a local event (e.g., the completion of a task), without waiting for other agents. This approach is inherently more compatible with the distributed and dynamic nature of scheduling problems and can lead to higher responsiveness and better resource efficiency [15]. However, the design of asynchronous MARL algorithms comes with new challenges, especially in the areas of coordination and credit allocation, since the actions of the agents no longer occur in a common time frame. The algorithm proposed in this paper is specifically designed to operate in such an asynchronous environment.

2.3. Review of Modern Dynamic Scheduling Algorithms

In recent years, extensive research has been conducted on the use of DRL and MARL in solving dynamic scheduling problems. Early research was mainly focused on single-agent algorithms such as DQN and PPO, but as the complexity of the problems increased, the focus has shifted towards multi-agent approaches and more advanced architectures.

An important branch of research is the use of graph neural networks (GNNs) to represent the structure of the scheduling problem. For example, in reference [16] a GNN-based and reinforcement learning model for flexible job shop scheduling (FJSP) was presented. In this approach, jobs and machines are considered as nodes of a graph, and GNN is used to extract structural features and relationships between them. This approach allows the agent to make more informed decisions. The main advantage of this approach is its generalizability to problems of different sizes, but the computational complexity of GNN can significantly increase the training time.

Another approach focuses on improving multi-agent Actor-Critic architectures. Research [17] proposes an algorithm called MAPPO for dynamic scheduling in manufacturing environments. MAPPO attempts to improve convergence in multi-agent environments by using the PPO proximal policy optimization algorithm, which is known for its robustness in training. This method utilizes the CTDE framework and shows strong performance in standard benchmarks. However, this algorithm still relies on the assumption of simultaneous decision-making and its credit allocation mechanism is relatively simple and may not perform optimally in environments with complex interactions between agents. To deal with uncertainty and unexpected events such as machine failures, research [17] has developed a robust reinforcement learning framework. By training the policy in the face of various failure scenarios, this algorithm attempts to learn a scheduling strategy that not only performs well under normal conditions but is also robust to disturbances. The advantage of this approach is to increase the stability and reliability of the system, but its disadvantage is the need for accurate and diverse simulation of possible disturbance types, which is not always possible in practice.

In the field of asynchronous scheduling, [10] introduces an asynchronous Actor-Critic algorithm for scheduling in warehouses. This algorithm allows each robot to make decisions independently and event-driven. Although this is an important step in the right direction, this research does not focus mainly on the complex credit allocation problem arising from interactions between agents in intensive manufacturing environments and uses a general benefit function.

Inspired by hierarchical learning, [19] proposes a two-level approach to scheduling. At the top level, a metacontroller agent determines the overall resource allocation policy, and at the bottom level, local agents choose prioritization rules based on this policy. This structure helps to decompose the problem into simpler subproblems and can improve scalability. However, designing the hierarchical structure and reward functions for each level is itself a complex engineering challenge and may lead to local optimality.

Paper in [20] addresses the credit allocation problem using

an attention mechanism in the focused critic. The critic tries to estimate the importance of each factor in the overall team reward by weighting the influence of different factors based on the current state. This method is more flexible than simpler approaches, but the attention mechanism may not fully model the conditional dependencies and counterfactuals of the agents' actions, which is exactly the problem that the proposed algorithm with its conditional advantage aims to solve.

2.5. Comparison table of related works

Table 2 provides a summary of the reviewed modern algorithms and their comparison in terms of the main method, advantages and disadvantages.

TABLE II. COMPARISON OF MODERN SCHEDULING ALGORITHMS

Reference	Main Method	Advantages	Disadvantages
[16]	Deep reinforcement learning using Graph Neural Networks (GNNs) for state feature extraction.	Powerful and rich representation of problem structure; generalizability to different problem sizes.	High computational complexity in GNN training; unsuitable for real-time decision-making.
[17]	Use of the MAPPO algorithm within the CTDE framework to improve training stability.	More stable convergence compared to simple policy gradient algorithms; strong performance on standard benchmarks.	Assumes simultaneous decision-making; uses a simple credit assignment mechanism; does not focus on complex inter-agent dependencies.
[18]	Training a robust policy against disturbances using diverse failure scenarios in a simulated environment.	Produces stable and reliable scheduling policies in uncertain environments; mitigates the impact of unexpected events.	Requires accurate and comprehensive modeling of various disturbances; highly dependent on simulation environment quality.
[10]	Multi-agent Actor-Critic algorithm for asynchronous, event-driven decision-making in warehouse environments.	More realistic modeling of operational environments; improves system responsiveness and efficiency by eliminating unnecessary waiting times.	Lacks an advanced credit assignment mechanism; may not perform optimally in environments with dense agent interactions.
[19]	Hierarchical reinforcement learning with a high-level meta-controller and low-level local agents.	Improves scalability by decomposing the problem into simpler sub-problems; capable of managing both long-term and short-term objectives.	Complex design of hierarchical structure and reward functions; risk of converging to locally optimal policies.
[20]	Use of an attention mechanism in a centralized critic to dynamically weight the importance of different agents in team reward.	More flexible and dynamic credit assignment compared to traditional methods; focuses on key agents at each decision step.	Attention mechanism may not fully capture complex causal or conditional relationships; increased computational complexity for the critic.

Analysis of previous works shows that despite significant progress, there is still a research gap in the development of algorithms that simultaneously cover three key features: scalability through the CTDE framework, realism through asynchronous decision modeling, and accuracy through an advanced credit allocation mechanism capable of modeling conditional dependencies between agents' actions.

3. Proposed algorithm

We call the proposed algorithm, abbreviated as ACPF-DS, which is a multi-agent Actor-Critic approach based on the CTDE framework and asynchronous decision making and uses the asynchronous policy conditional factorization mechanism for dynamic scheduling, credit allocation, and policy coordination.

3.1. General architecture of the proposed algorithm

The architecture of the proposed method consists of two main components: decentralized agents and a centralized learner.

Decentralized agents: Each agent i (representing a work unit) is equipped with an actor network with parameters θ_i. This network models the policy $p_i(a_i|o_i;\theta_i)$ that maps the local observation o_i to a probability distribution over possible actions. The network operates completely autonomously and asynchronously.

Centralized learner: This unit is active only in the training phase, it consists of a critic network with parameters Φ. Unlike the actors, the critic network has access to global information and approximates a joint action-state value function $Q_{tot}(s,\boldsymbol{a}|a;\Phi)$. s is the reconstructed global state and $\boldsymbol{a}$ is the vector of joint actions. The main task of this learner is to assess the quality of cooperation of the agents and provide accurate training signals for updating their policies.

3.2. Joint optimization through conditional policy factorization

The main innovation of the proposed algorithm lies in the updating of the actor networks. The goal is to optimize individual policies in such a way that the overall performance of the system improves. This is achieved through a policy gradient driven by the centralized critic. We call this approach "conditional policy factorization" because the optimal joint policy $p_i(\{\boldsymbol{a}\}|s)$ is implicitly factored into the product of the individual policies $prod_i{*}p_i(a_i|o_i)$ and this factorization is coordinated through the centralized critic conditioned on s. The update of the critic and actor networks is as follows:

Critic network update: The critic network is updated by minimizing the mean square error between its prediction and a TD target. By sampling a minibatch of experiences from the buffer, the cost function for the critic is defined as:

$$L(\phi)=\frac{1}{|B|}\sum_{(s,a,r,s')\in B}(y-Q_{tot}(s,a;\phi))^2 \tag{6}$$

where $|B|$ is the sampled batch and y is the TD target. This objective is computed using objective networks for greater stability:

$$y=r_{tot}+\gamma Q_{tot}'(s',\mathrm{a}';\phi') \tag{7}$$

Here $r_{tot}=sum(r_i)$ is the team reward, γ is the discount factor, ϕ' is the parameters of the target critical network, and $\mathbf{a}'=p_{i'1}(o'_1)\times p_{i'N}(o'_N)$ is the vector of next actions chosen by

the target policies of the actors.

Updating the actor networks: The update of each actor i is done based on the policy gradient theorem. The gradient of the objective function J_{θ_i} with respect to the parameters of actor θ_i is as follows:

$$\nabla_{\theta_i} J(\theta_i) = E_{s,a}[\nabla_{\theta_i} log \pi_i(a_i | o_i)]; \quad (8)$$

In the credit allocation process, the important point is the precise definition of the advantage function A_i. In this advantage function, it measures the contribution of agent i to the overall value by comparing its actual action with a baseline that holds the actions of other agents constant:

$$A_i(s, \mathbf{a}) = Q_{tot}(s, \mathbf{a}; \phi) - b_i(s, \mathbf{a}_{-i}) \quad (9)$$

Where a_{-i} represents the actions of all agents except i. The baseline b_i calculates the expected value by averaging over all possible actions of agent i based on its current policy:

$$b_i(s, \mathbf{a}_{-i}) = \sum \pi_i(a_i' | o_i; \theta_i) Q_{tot}(s, (a_i', \mathbf{a}_{-i}); \phi) \quad (10)$$

This formulation effectively informs each agent how much better its chosen action (a_i) was than an "average" action from its perspective in that particular situation. This approach provides a rich and focused learning signal to each agent and facilitates convergence towards a coordinated and optimal policy. The cost function for each actor i is in the form of maximizing the expected benefit:

$$L(\theta_i) = -E_{s,\mathbf{a}}[log\ \pi_i(a_i | o_i; \theta_i) \cdot A_i(s, \mathbf{a})] \quad (11)$$

To reduce the variance, an entropy term can also be added to the cost function to encourage exploration:

$$L(\theta_i) = -E_{s,\mathbf{a}}[log\ \pi_i(a_i | o_i) \cdot A_i(s, \mathbf{a}) + \beta H(\pi_i(\cdot | o_i))] \quad (12)$$

where H is the policy entropy and β is its weight coefficient.

3.3. Adaptive update mechanism

To improve the stability and learning speed, the proposed algorithm uses an adaptive update mechanism that consists of two components:

Experience buffer with prioritization: Instead of uniformly sampling the experience buffer, more priority is given to the experiences that have caused the most surprises or errors to the model. The priority of an experience j is defined as the absolute value of its TD error:

$$p_j \propto |\delta_j|^{\alpha} \quad (12)$$

Where $\delta_j = y_j - Q_{tot}(s_j, \mathbf{a}_j)$ and α is a hyperparameter that controls the amount of prioritization. This allows the algorithm to focus on the "hardest" examples and accelerate the learning process.

Learning rate tuning: The learning rate is a critical meta-parameter. In the proposed method, a gradual learning rate reduction strategy is used. The learning rate is set high at the beginning of training to enable fast exploration, and as training progresses and approaches the optimum point, it is gradually reduced to ensure stable convergence. Decreasing functions such as exponential or step can be used for this purpose.

3.4. Actor and critic network architecture

For the dynamic scheduling problem, the observations of each agent (o_i) can include the attributes "current agent status (free/busy)", "number of tasks in the queue", "next task attributes (processing time, priority)" and "state of neighboring agents".

Actor Network Architecture: This network is a multilayer perceptron (MLP) as follows, which maps a local observation to a probability distribution over discrete actions (e.g., choosing one of N available or waiting tasks):

Input layer: Dimension equal to the size of the observation vector o_i.

Hidden layers: 2 hidden layers with 256 neurons in each layer.

Activation function: ReLU for the hidden layers due to computational efficiency and avoiding the problem of gradient fading.

Output layer: A layer with the number of neurons equal to the number of possible actions ($|A_i|$).

Output activation function: Softmax to produce a valid probability distribution over the actions.

Critic Network Architecture: This network is also an MLP, but its input is larger and includes the global state s (which can be the concatenation of observations from all agents) and the common action vector $\boldsymbol{a}$.

Input layer: Dimension equal to $|s| + sum_i|\boldsymbol{a}_i|$.

Hidden layers: 2 hidden layers with 512 neurons per layer (larger than the actor due to the higher input complexity).

Activation function: ReLU for the hidden layers.

Output layer: A layer with 1 neuron that outputs the value of Q_{tot} as a scalar.

Output activation function: Linear (Linear/Identity) because the output is a continuous and unbounded value.

4. Implementation and Results

In the simulation, a toolkit based on the Python 3.12 programming language was used. The laptop used had an Intel Core i7-13620H processor and 32GB RAM and NVIDIA RTX 4070 8GB graphics.

TABLE III. AN EXAMPLE OF FT06 SCHEDULING PROBLEM IN STATIC AND DYNAMIC MODE

2	1	0	3	1	6	3	7	5	3	4	6
1	8	2	5	4	10	5	10	0	10	3	4
2	5	3	4	5	8	0	9	1	1	4	7
1	5	0	5	2	5	3	3	4	8	5	9
2	9	1	3	4	5	5	4	0	3	3	1
1	3	3	3	5	9	0	10	4	4	2	1

a) ft06 scheduling problem in static mode

0	34	1	2	1	0	3	1	6	3	7	5	3	4	6
2	57	1	1	8	2	5	4	10	5	10	0	10	3	4
4	62	3	2	5	3	4	5	8	0	9	1	1	4	7
8	51	1	1	5	0	5	2	5	3	3	4	8	5	9
10	43	1	2	9	1	3	4	5	5	4	0	3	3	1
11	65	3	1	3	3	3	5	9	0	10	4	4	2	1

b) ft06 scheduling problem in dynamic mode

4.1. Dataset

The Starjob dataset [21] was used for the simulation, which was specifically designed for modeling task scheduling environments with dynamic task input. This dataset has features such as dynamics and uncertainty, multi-

agent and distributed nature, asynchronous decision-making, and a complex state and action space. Therefore, the Starjob dataset provides a challenging yet realistic testbed to measure the ability of the proposed ACPF-DS algorithm to learn dynamic, synchronous, and asynchronous scheduling policies. The Starjob dataset is divided into 70% for training, 15% for validation, and 15% for testing.

The matrix of an example of the ft06 (6 machines, 6 jobs) scheduling problem dataset in static and dynamic mode is shown in Table 3.

4.2. Reference algorithms

The performance of the proposed algorithm is compared with several reference algorithms. These algorithms cover a wide range of approaches, from simple heuristic rules to advanced reinforcement learning algorithms.

FIFO: A simple and common algorithm in which tasks are processed in the same order in which they are entered [22]. This method is fair but does not consider the characteristics of the tasks (such as processing time or priority).

SJF: A heuristic that selects the task with the shortest processing time to execute [23]. This method reduces the average waiting time but may lead to “long waiting” of long tasks.

MADDPG: A well-known algorithm in the MADRL domain that follows the CTDE framework and is designed for environments with continuous action space, but can also be adapted to discrete space [24].

QMIX: Another popular algorithm based on CTDE that factors a joint value function as the sum of the individual value functions of each agent. This algorithm imposes a uniform assumption on the value function, which may be restrictive in some complex problems [25].

4.3. Evaluation Criteria

The performance of the algorithms is evaluated based on the following three key criteria that are widely used in the scheduling literature:

Makespan (C_{max}): The completion time of the last task in the entire set. This criterion indicates the overall efficiency of the system in processing a batch of tasks. The lower the value, the better.

$$C_{max}=max(C_1,...,C_J) \tag{12}$$

Average Waiting Time (AWT): The average time that tasks wait in the queue to begin processing. This metric is directly related to customer satisfaction and inventory holding costs. The lower the value, the better.

$$AWT=\frac{1}{|J|}\sum_{j\in J}(start_{timej}-arrival_{timej}) \tag{13}$$

Machine Efficiency: The percentage of time that machines are working. This metric indicates the optimal use of available expensive resources. The higher the value, the better.

$$Utilization=\frac{\sum busy_{time\ m}}{|M|\times C_{max}} \tag{14}$$

4.4. Analysis of the obtained results

4.4.1. Analysis of the training data

The goal of tuning the hyperparameters is to improve the performance of the base agent and achieve the desired learning behavior. The desired learning behavior is characterized by the convergence of the agent's reward to the estimate Q_0. The estimate Q_0 is calculated by performing inference on the critic at the beginning of each episode. This estimate represents the expected long-term reward based on the current observation. Ideally, this estimate corresponds to the total actual reward collected during the episode. During training, several things are considered:

Reward per episode to reflect the agent's momentary performance.

Average reward to identify the overall learning progress.

The **value of Q_0** represents the critic's estimate of the expected long-term reward.

The desired learning behavior can be inferred from these graphs:

Convergence of the value of Q_0: The value of Q_0 should converge towards the average reward, which indicates that the critic's estimates are becoming more accurate and consistent with the agent's performance.

Stable average reward: A stable average reward with low fluctuations indicates that the agent has effectively learned the task and consistently performs well.

Reduced fluctuations: As training progresses, the reward per part should show less fluctuations, indicating that exploration is decreasing and the agent is focusing more on exploiting its learned knowledge.

Figure 1 shows the learning behavior of an agent (here agent number 15). This figure shows that the reward trends and *Q0* values in the agent have achieved the desired learning behavior according to the criteria stated for examining the behavior of agents.

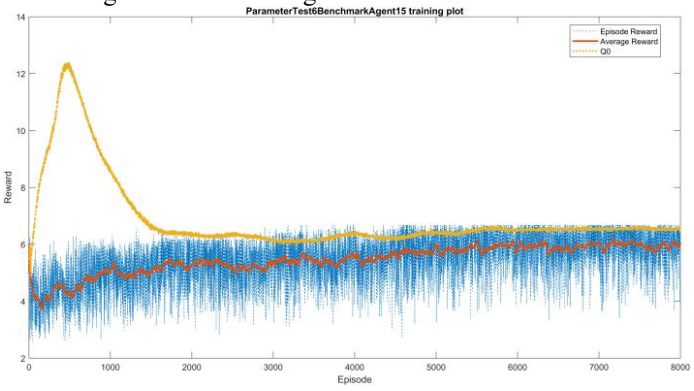

Figure 1. Example of an agent training graph

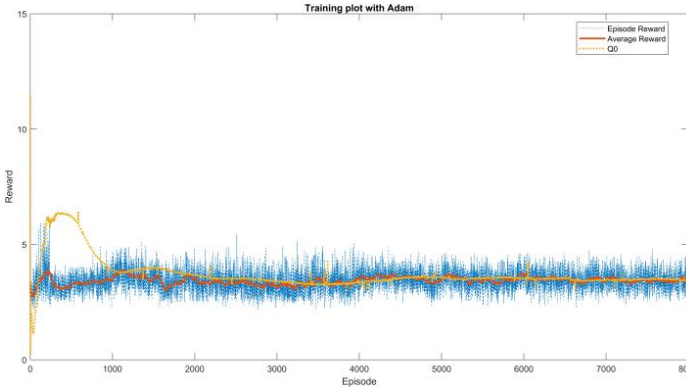

Figure 2. Appropriate behavior of the Adam optimization algorithm and its convergence

4.4.2. Testing optimization algorithms and creating the base agent

Three base agents were created with different optimization algorithms to select the best algorithm. These agents were trained for the ft06 problem (6 jobs, 6 machines). The obtained results show that:

RMSprop has unstable and undesirable learning behavior. SGDM and Adam have stable behavior. Adam was chosen because of its greater stability in estimating Q_0 and its more common application (Figure 2). The agent-based test on the ft06 problem showed that the Makespan value is 79.

Figure 3 shows the Gantt chart (task scheduling chart) that shows the solution found by the agent. From this Gantt chart, it can be concluded that the agent, looking at the ft06 problem (6 machines and 6 jobs), schedules the jobs and their operations in the correct order.

The pie chart in Figure 4 shows that the agent mostly uses the three dispatch rules MOR, SPT, and LPT, and the other rules are ignored.

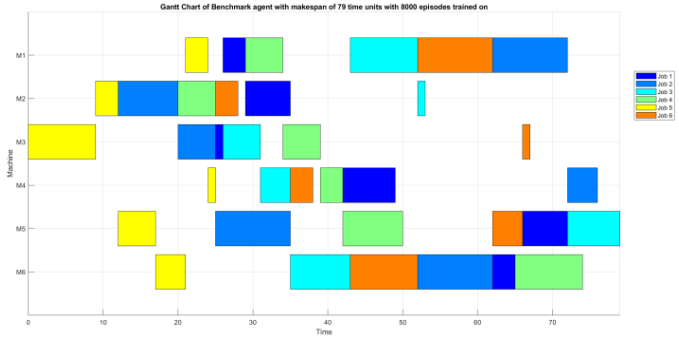

Figure 3. Gantt chart of the best schedule and task completion time obtained by the benchmark agent

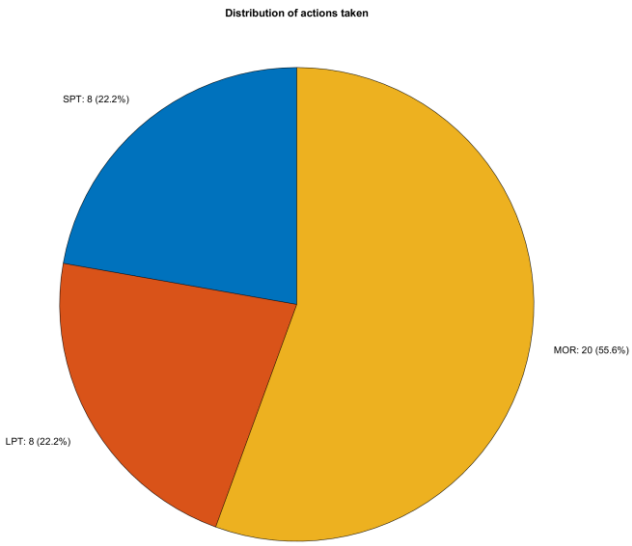

Figure 4. Pie chart of different actions performed by the agent

In five iterations, a set of agents were trained with different parameters. The four agents with the best performance were as follows:

Agent 13: Reward 6.64, Makespan = 61

Agent 18: Reward 6.22, Makespan = 70

Agent 26: Reward 5.73, Makespan = 72

Agent 29: Reward 5.73, Makespan = 67

The learning behavior of agent 13 was more favorable (Figure 5). The selected hyperparameters of this agent, which we use in the following, are listed in Table 4.

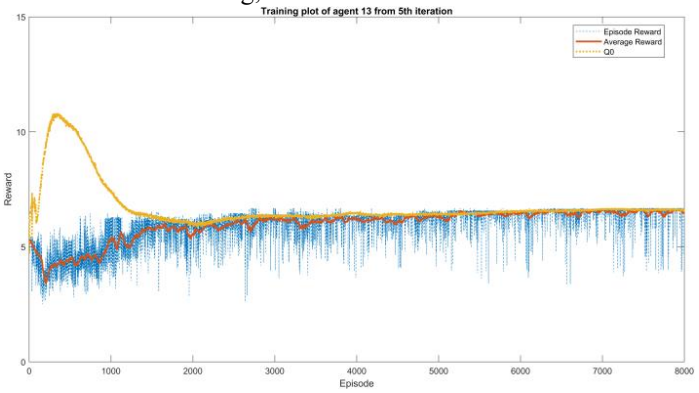

Figure 5. Learning graph of agent 13 in the fifth iteration

TABLE IV. SUGGESTED HYPERPARAMETERS FOR TRAINING THE PROPOSED ALGORITHM

HYPERPARAMETER	SYMBOL	SUGGESTED VALUE
Discount Factor	gamma	0.99
Soft Update Rate	tau	0.005
Experience Buffer Capacity	C	10^6
Batch Size	Batch Size	256
Actor Learning Rate	alpha_actor	10^-4
Critic Learning Rate	alpha_critic	10^-3
Optimizer	-	Adam
Entropy Coefficient	beta	0.01
Prioritization Exponent	alpha	0.6
Number of Training Episodes	-	8000

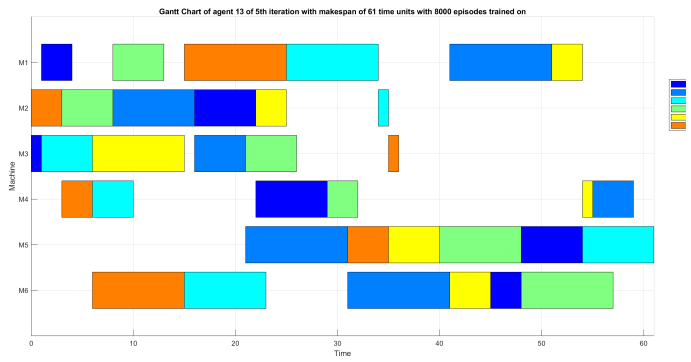

Figure 6. Gantt chart of the solution found by agent 13 from the fifth iteration

The Gantt chart in Figure 6 shows an 18-point decrease in Makespan compared to Figure 3. The pie chart in Figure 7 also shows that the agent used 9 dispatch rules and ignored only LOR.

4.4.3. Results of the proposed algorithm for different numbers of jobs and machines

The results of the proposed algorithm in different problems (for different numbers of jobs and machines) in terms of reward, completion time and computational time are presented in Table 5. As the input size increases, a linear growth in computational time is observed. As a result, the steps required to solve a job scheduling problem increase with the number of machines or jobs. Also, problems with more machines require more computational time with the same number of inputs.

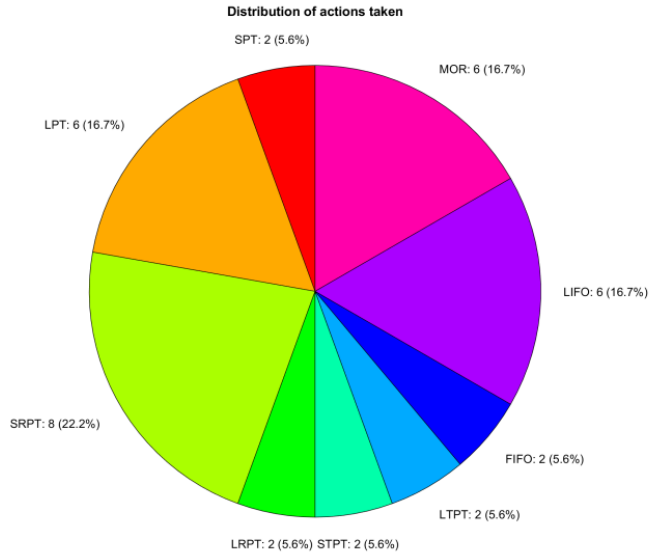

Figure 7. Distribution of actions performed by the final selected agent

TABLE V. COMPARISON OF THE PERFORMANCE OF THE ALGORITHMS BASED ON THE EVALUATION CRITERIA (MEAN ± STANDARD DEVIATION)

Example	Number of Machines	Number of Jobs	R (Reward)	C (Completion Time)	Computational Time (CT)
abz8	15	20	5.014	1010	5.8
ft06	6	6	6.661	61	0.76
la04	5	10	6.473	253	0.92
la09	5	15	8.761	321	1.38
la15	5	20	7.582	436	1.91
la25	10	15	5.511	627	2.81
la35	10	30	6.099	1169	5.60
svw01	10	20	5.224	983	3.79
svw15	10	50	4.978	1568	13.16
ta01	15	15	4.827	874	4.26
ta31	15	30	5.418	1205	9.71
ta41	20	30	4.595	1472	16.66

4.4.4. Comparison of the proposed algorithm with other algorithms

The experiments were run on 10 different examples of the Starjob dataset (with 50 jobs and 10 machines) and the average results after 8000 training steps for the reinforcement learning algorithms are reported in Table 6.

TABLE VI. COMPARISON OF ALGORITHM PERFORMANCE BASED ON EVALUATION CRITERIA (MEAN ± STANDARD DEVIATION)

Algorithm	Makespan (time units)	Average Waiting Time (AWT)	Machine Utilization (%)
FIFO	1854 ± 121	552 ± 84	81.2 ± 3.5
SJF	1725 ± 105	415 ± 69	86.8 ± 2.9
MADDPG	1657 ± 98	381 ± 62	89.5 ± 2.5
QMIX	1619 ± 92	358 ± 58	91.3 ± 2.1
ACPF-DS (Proposed)	1544 ± 85	312 ± 51	94.6 ± 1.8

The proposed algorithm ACPF-DS has shown better performance than the reference algorithms in all the evaluation criteria.

Superiority over classical methods (FIFO, SJF): Reinforcement learning algorithms (MADDPG, QMIX, ACPF-DS) have significantly outperformed simple heuristic rules. This shows that static policies are not able to optimally handle the dynamics and complexity of the environment, while RL algorithms can learn adaptive and state-dependent policies.

Comparison with MADRL algorithms: The proposed algorithm has been able to reduce Makespan and average waiting time by about 5% and 13%, respectively, compared to the best reference algorithm (QMIX). This improvement can be attributed to the counterfactual credit allocation mechanism. While QMIX provides a global view of each agent's contribution, ACPF-DS allows each agent to accurately assess the impact of its action by keeping the actions of others constant. This richer training signal leads to learning more coordinated and efficient policies. Also, the improvement in Makespan has directly led to increased resource efficiency. By reducing idle times and making smarter choices, the proposed method has been able to increase the efficiency level to nearly 95%, which is a significant achievement in production environments.

In summary, the results show that the proposed architecture, especially through the combination of asynchronous decision-making and accurate credit allocation mechanism, is able to provide more efficient and robust solutions to the complex dynamic scheduling problem compared to existing methods.

5. Conclusion

In this paper, a multi-agent deep reinforcement learning

framework was presented to solve the dynamic scheduling problem. The proposed ACPF-DS algorithm, by utilizing the policy factorization and asynchronous update mechanism, allows each agent to make optimal decisions based on local observations and a global view of the system state. This approach facilitates effective coordination among agents and avoids the phenomenon of incoherence that is common in other multi-agent algorithms. The obtained results clearly demonstrated the superiority of the proposed algorithm. The proposed algorithm succeeded in reducing Makespan by about 5% and average waiting time by 13% compared to the best compared reference algorithm, namely QMIX. This significant improvement indicates the high ability of the proposed method to learn complex and efficient scheduling policies in dynamic and uncertain environments and proves the high capacity of deep reinforcement learning to solve large-scale combinatorial optimization problems.

References

[1] [1] M. Zhang, Y. Wang, and L. Zhang, "Dynamic Scheduling in Smart Manufacturing: Challenges and Opportunities," IEEE Trans. Ind. Inform., vol. 19, no. 2, pp. 1885–1895, Feb. 2024.

[2] H. Wang, C. Chen, and Q. Zhang, "Reinforcement Learning for Dynamic Job Shop Scheduling: A Survey," IEEE Trans. Autom. Sci. Eng., vol. 20, no. 1, pp. 436–453, Jan. 2023.

[3] Z. Ma, Z. Li, H. Zhang, and L. Gao, "A Survey of Deep Reinforcement Learning for Job Shop Scheduling Problems," IEEE Transactions on Automation Science and Engineering, vol. 30, no. 1, pp. 1-18, 2024.

[4] J. Lee, S. Kim, and J. Park, "Deep Reinforcement Learning for Dynamic and Stochastic Resource Allocation in Cloud Computing," IEEE Transactions on Cloud Computing, vol. 12, no. 2, pp. 345-359, 2024.

[5] Y. Wang, H. He, and X. Tan, "Scalable Multi-Agent Reinforcement Learning for Large-Scale Traffic Signal Control," IEEE Transactions on Intelligent Transportation Systems, vol. 25, no. 3, pp. 1120-1135, 2024.

[6] S. K. Samudrala, A. Kumar, and P. Stone, "A Survey on Centralized Training for Decentralized Execution in Multi-Agent Reinforcement Learning," Journal of Artificial Intelligence Research, vol. 79, pp. 201-245, 2024.

[7] L. Chen, Y. Wu, and Z. Yang, "MADDPG with Attention Mechanism for Cooperative Multi-Agent Pathfinding," IEEE Transactions on Games, vol. 16, no. 1, pp. 55-68, 2024.

[8] H. Nguyen, T. Pham, and D. Kim, "Asynchronous Advantage Actor-Critic for Multi-Robot Warehouse Commissioning," IEEE Robotics and Automation Letters, vol. 9, no. 4, pp. 3012-3019, 2024.

[9] F. Wei, Z. Li, and D. Zhao, "Counterfactual Credit Assignment in Multi-Agent Reinforcement Learning: A Survey," ACM Computing Surveys, vol. 57, no. 1, Article 14, pp. 1-36, 2025.

[10] M. Schmidt, T. Schneider, and S. Rinderle-Ma, "Asynchronous Deep Reinforcement Learning for Flexible Job Shop Scheduling," IEEE Transactions on Industrial Informatics, vol. 20, no. 5, pp. 5678-5687, 2024.

[11] G. Palmer, K. Tuyls, and D. Bloembergen, "Shapley-Based Credit Assignment in Multi-Agent Reinforcement Learning," Journal of Machine Learning Research, vol. 25, no. 45, pp. 1-48, 2024.

[12] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. MIT Press, 2023.

[13] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey of multi-agent deep reinforcement learning," Artificial Intelligence Review, vol. 56, no. 2, pp. 1-45, 2023.

[14] S. K. Samudrala, A. Kumar, and P. Stone, "A Survey on Centralized Training for Decentralized Execution in Multi-Agent Reinforcement Learning," Journal of Artificial Intelligence Research, vol. 79, pp. 201-245, 2024.

[15] M. Schmidt, T. Schneider, and S. Rinderle-Ma, "Asynchronous Deep Reinforcement Learning for Flexible Job Shop Scheduling," IEEE Transactions on Industrial Informatics, vol. 20, no. 5, pp. 5678-5687, 2024.

[16] J. Park, S. Lee, and H. Kim, "Graph-based Deep Reinforcement Learning for Scalable Flexible Job Shop Scheduling," IEEE Transactions on Automation Science and Engineering, vol. 30, no. 2, pp. 112-126, 2024.

[17] Y. Zhang, W. Li, and G. Wang, "Solving Dynamic Job Shop Scheduling Problem with Multi-Agent Proximal Policy Optimization," Journal of Manufacturing Systems, vol. 72, pp. 34-47, 2024.

[18] Z. Chen, L. Gao, and P. Li, "Robust Deep Reinforcement Learning for Dynamic Scheduling under Uncertainties and Machine Breakdowns," IEEE Transactions on Reliability, vol. 73, no. 1, pp. 258-271, 2024.

[19] A. Farhadi and M. H. Fazel Zarandi, "A Hierarchical Reinforcement Learning Approach for Large-Scale Production Scheduling," Computers & Industrial Engineering, vol. 188, p. 109845, 2024.

[20] X. Liu, Y. Zhou, and J. Sun, "Attention-based Critic for Multi-Agent Reinforcement Learning in Dynamic Scheduling Environments," IEEE Transactions on Games, vol. 16, no. 2, pp. 210-222, 2024.

[21] https://github.com/starjob42/Starjob

[22] M. Pinedo, Scheduling: Theory, Algorithms, and Systems, 6th ed. Cham, Switzerland: Springer, 2024.

[23] J. Y.-T. Leung, ed., Handbook of Scheduling: Algorithms, Models, and Performance Analysis, 2nd ed. Boca Raton, FL, USA: CRC Press, 2024, pp. 45–89.

[24] Y. Zhang et al., "A Survey on Multi-Agent Deep Reinforcement Learning for Manufacturing Scheduling: From MADDPG to GNN-Based Approaches," IEEE Transactions on Automation Science and Engineering, vol. 21, no. 2, pp. 1245–1267, Mar. 2024.

[25] X. Li et al., "Beyond Monotonicity: Non-Monotonic Value Factorization for Multi-Agent Reinforcement Learning," Advances in Neural Information Processing Systems 36 (NeurIPS 2024), Dec. 2024, pp. 7890–7905.

Zahra Salimifard was born in Bushehr, Iran, in 1988. She holds an Associate Degree in Computer Software from Al-Zahra Technical and Vocational Institute in Bushehr and a Bachelor of Science in Computer Engineering (Software) from the Lian Institute (a private non-profit) in Bushehr. She is currently pursuing a Master's degree in Computer Engineering and works as a computer teacher in schools.

Email: Zsalimi085@gmail.com

Alireza Chamkoori was born in Abadan, Iran, in 1973. He received the B.Sc. degree in Electronics Engineering from Islamic Azad University, Bushehr, Iran, and the M.Sc. and Ph.D. degrees in Computer Engineering (Software) from the Islamic Azad University, Science and Research Branch, Tehran, Iran. He is currently with the Department of Computer Engineering, Islamic Azad University, Bushehr, Iran. His research interests include cloud computing security, network communication, encryption and access control algorithms, and machine learning-based threat detection.

Email: Chamkoori@iau.ac.ir (Corresponding Author)

Nooshin Rabiee received her M.Sc. and Ph.D. degrees in Electrical Engineering from the Islamic Azad University of Bushehr and Shiraz in 2012 and 2019. She has been as a lecturer in the Islamic Azad University and Lian Institute of Higher Education ,Bushehr, Iran. Her research interests are Signal Processing ,Image Processing, Radar Recognition. She published more than 20 referable papers in international journal and conferences.

Email: Nooshin.rabiee@yahoo.com