SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

February 2025, Vol. 68, Iss. 2, 122302:1–122302:18
https://doi.org/10.1007/s11432-024-4261-1

Graph reinforcement learning with relational priors for predictive power allocation

Jianyu ZHAO & Chenyang YANG*

School of Electronics and Information Engineering, Beihang University, Beijing 100191, China

Received 26 June 2024/Revised 15 September 2024/Accepted 24 December 2024/Published online 13 January 2025

Abstract Deep reinforcement learning for resource allocation has been investigated extensively owing to its ability of handling model-free and end-to-end problems. However, its slow convergence and high time complexity during online training hinder its practical use in dynamic wireless systems. To reduce the training complexity, we resort to graph reinforcement learning for leveraging two kinds of relational priors inherent in many wireless communication problems: topology information and permutation properties. To harness the two priors, we first conceive a method to convert the state matrix into a state graph, and then propose a graph deep deterministic policy gradient (DDPG) algorithm with the desired permutation property. To demonstrate how to apply the proposed methods, we consider a representative problem of using reinforcement learning, predictive power allocation, which minimizes the energy consumption while ensuring the quality-of-service of each user requesting video streaming. We derive the time complexity required by training the proposed graph DDPG algorithm and fully-connected neural network-based DDPG algorithm in each time step. Simulations show that the graph DDPG algorithm converges much faster and needs much lower time and space complexity than existing DDPG algorithms to achieve the same learning performance.

Keywords reinforcement learning, graph neural network, relational priors, resource allocation

Citation Zhao J Y, Yang C Y. Graph reinforcement learning with relational priors for predictive power allocation. Sci China Inf Sci, 2025, 68(2): 122302, https://doi.org/10.1007/s11432-024-4261-1

1 Introduction

Deep reinforcement learning (DRL) has been introduced to optimize a variety of resource allocation problems, due to its ability of learning wireless policies from the optimization problems without closed-form objectives and constraints, making decisions in an end-to-end manner, and online training [1].

Most existing studies usually train the deep neural networks (DNNs) in DRL algorithms in simulated environments during the training stage, and then use the well-trained DRL algorithms to make decisions in the inference stage [2]. However, a well-trained DRL algorithm may suffer from performance degradation when it is used in non-stationary environments [2, 3]. This is because the statistics of the environmental parameters change over time, which causes the statistics difference between the real and simulated environments [2, 3]. For example, the channel statistics may vary owing to the high mobility of users.

To adapt to dynamic environments, a DRL algorithm needs to be trained online, which continually trains the DNNs based on the interactions with the environments as follows. (i) In each time step, the agent of DRL observes a state, and executes an action (i.e., inference). Then, the agent receives a reward, observes a new state after taking the action, and stores the state, action, reward as well as the new state as an experience (i.e., a sample in the DRL algorithm) in a replay buffer. (ii) The agent selects a batch of stored experiences from the replay buffer to update the model parameters of DNNs (i.e., training) every one time step (if the statistics change rapidly over time) or every several time steps (if the statistics change slowly over time).

Existing DRL algorithms suffer from high training complexity, i.e., sample complexity, time complexity, and space complexity, which hinders their practical use in dynamic environments. The sample complexity of a DRL algorithm refers to the minimal number of experiences needed for converging to an expected performance; hence high sample complexity incurs large signaling overhead for collecting experiences. Since only one experience is collected in each time step, the sample complexity amounts to the minimal

* Corresponding author (email: cyyang@buaa.edu.cn)

number of time steps. In other words, high sample complexity indicates slow convergence. Moreover, before a DRL algorithm converges, the decision made by the algorithm is with unacceptable performance. High time complexity and space complexity (i.e., the number of model parameters in the fine-tuned DNNs) result in large energy consumption [4]. Both high sample complexity and high time complexity impede the DRL algorithms to be online trained in real-time.

To reduce the training complexity, an effective approach is to incorporate relational priors into the DNNs in a DRL algorithm [5]. One kind of prior is the topology prior [5], which is captured by the adjacency matrix of a graph. Another kind of prior is the permutation properties of the input-output relations of the functions to be learned [5]. By incorporating graph neural networks (GNNs), DRL algorithms can harness both kinds of relational priors.

To showcase how to reduce the training costs of DRL algorithms by leveraging the priors, we consider a representative resource allocation problem that is a canonical application of reinforcement learning: predictive power allocation for video streaming [6], which is considered for the following reasons. (i) Video-on-demand services account for a large portion of traffic load [7] in prevalent and future cellular networks. (ii) Allocating resources to mobile users with predicted channels has shown promise for such non-real-time service in using radio resources efficiently [6,8,9]. (iii) This resource allocation problem can be naturally formulated as a Markov decision process (MDP) [10], the objective function is not with a closed-form expression, and the channel prediction and power allocation can be optimized in an end-to-end manner. (iv) The state design is non-trivial due to the different time scales of the objective function, constraints, and optimization variables, which need a trial-and-error process.

1.1 Related studies

1.1.1 *GNNs for resource allocation*

GNNs have been employed to optimize resource allocation in a variety of wireless systems [11–30]. It has been shown that GNNs are with improved performance compared to other DNNs [15–17,20–24,26–28,30], good scalability [11,13,26,28,30], size generalization ability [11–16,18–23,25,27,28,30], and low training complexity [14–16,21–23,29]. So far in literature, there are only a few studies considering graph reinforcement learning, where GNNs were incorporated with DRL for optimizing channel allocation [24], data uploading [25], task offloading [26,27], joint routing, and power control [28]. All these studies on graph reinforcement learning do not consider training complexity and permutation prior.

The performance of a GNN depends on its structure and the graph that the GNN learns over.

Modeling appropriate graphs is the premise of applying GNNs. By learning over graphs, the topology prior can be exploited by GNNs, where the hidden representation of each vertex is updated by aggregating the information from its neighboring vertices and edges with a processor and a pooling function, and combining with its hidden representation in the previous layer with a combiner [31]. However, modeling graphs is non-trivial.

The structures of GNNs depend on the choices of processors, pooling functions, and combiners, which determine the permutation properties of the functions represented by GNNs. If the permutation property of a designed GNN does not match with the property of the function to be learned, then the GNN may not perform well [15] or exhibit high training complexity [23].

A function can be a mapping from a graph or a set into a matrix or a vector. When a function is defined on a homogeneous graph with only one type of vertices and edges, the function is with the permutation properties induced by a single set [22]. A vanilla GNN learning over the homogeneous graph can match with the permutation property of the function simply using the same processors, pooling functions, and combiners for all vertices [11,12,21]. When a function is defined on a heterogeneous graph with multiple types of vertices or edges, the permutation property of the function is induced by multiple sets, which is complex. In [15,22,23], GNNs were designed to satisfy such complex properties. However, designing the structures of GNNs to satisfy the complex permutation properties is not an easy task.

1.1.2 *DRL for resource allocation*

DRL algorithms have been widely applied for resource allocation. For example, they have been used for optimizing power and bandwidth allocation [32], resource block allocation [33], subcarrier and power allocation [34], beamforming and artificial noise vectors as well as reconfigurable intelligent surface phase shift [35], channel allocation [36], user association, and power allocation [37].

The motivation of using DRL in these studies is learning wireless policies from the optimization problems without closed-form objective functions in [32,34], adapting to dynamic wireless systems in [33,37], and harnessing the powerful function approximation ability in [35].

1.1.3 *Time complexity*

The time complexity of an algorithm can be measured using two metrics: the running time when the algorithm halts [38] and the number of multiplications [38].

The time complexity of DRL algorithms during the inference stage has been measured in [39, 40]. However, the time complexity of DRL in the training stage, which is vital for online training in non-stationary environments, is rarely addressed.

1.1.4 *Resource allocation for video streaming*

To satisfy the quality of service (QoS) of video streaming, early resource allocation methods transmitted to users with constant bit-rate [41]. To minimize the energy consumption for ensuring QoS, adaptive resource allocation methods without leveraging future channel information were designed in [42, 43].

Predictive resource allocation has shown remarkable gain over the non-predictive counterparts [6, 8–10, 44, 45]. In [6, 44], the average data rates or transmit powers in a prediction window were optimized to ensure the QoS of video streaming under the assumption that perfect future channels were available. In [8, 45], the channels were first predicted with off-line trained learning models and then the future resources were optimized with the predicted information. In [10], the prediction and optimization for a single user were accomplished in a single step using DRL algorithms, which showed superior performance to the first-predict-then-optimize solution.

1.2 Motivation and contribution

In this paper, we strive to reduce the training complexity of DRL algorithms by exploiting relational priors. To this end, we design graph reinforcement learning by incorporating GNN with DRL algorithms.

Before designing a GNN, we need to construct a graph, which is non-trivial because the graph is not unique for a wireless problem [15]. To enable a DRL algorithm to perform well, we need to design a state, which is challenging for a MDP problem because the state is also not unique [46]. For example, for the considered predictive resource allocation problem with different time scales of the objective function, constraints, and optimization variables, it is not straightforward to identify which variables in each time step will affect the decision made by a DRL algorithm. Noticing that state, action, and reward have been designed for many resource allocation problems [32–37] where the states are judiciously designed and represented by matrices [1], we provide a method for converting state matrix into state graph to avoid another trial-and-error procedure.

We take the deep deterministic policy gradient (DDPG) algorithm as an example, and propose a graph DDPG algorithm, where the actor and critic networks are respectively a composite of a GNN and an output layer. Then, we analyze the permutation properties of the policy function and the action-value function for predictive power allocation, which are respectively learned by the actor network and critic network. To match with the permutation properties, we design the GNN and the output layer in the proposed graph DDPG algorithm. We proceed to derive the time complexity of training the graph DDPG and fully-connected neural network (FNN)-based DDPG algorithms in each time step.

For easy exposition and simplify notations, we do not consider multi-user interference and inter-cell interference for the considered problem as in majority studies in the literature of predictive resource allocation [6, 8–10, 44, 45], but our methods are applicable to the settings with interference. The major contributions are summarized as follows.

- We propose a method to convert a matrix-based MDP into a graph-based MDP. The method is also applicable to both DDPG and other DRL algorithms, with which the already-designed states in literature can be translated to their graph-based counterparts.
- We propose a graph DDPG algorithm by introducing GNN into the DDPG algorithm. We design the GNN and the output layer to satisfy the permutation properties of the policy function and the action-value function for the predictive power allocation.
- Simulation results show that the graph DDPG algorithm needs much lower training complexity than other DRL algorithms to achieve the same performance by harnessing the relational priors.

The rest of the paper is organized as follows. In Section 2, we introduce predictive power allocation for video streaming. In Section 3, we introduce the method of converting the matrix-based MDP into graph-based MDP. In Section 4, we design a graph DDPG algorithm with desired permutation properties. In Section 5, we derive the time complexity of the graph-based DDPG and FNN-DDPG algorithms. Simulation results are provided in Section 6, and the conclusion is provided in Section 7.

2 Predictive power allocation for video streaming

In this section, we first introduce a system model and a predictive power allocation problem for video streaming. Then, we present the matrix-based MDP for the problem.

2.1 System model and problem formulation

Consider a cloud radio access network (C-RAN), where M distributed units (DUs) connected with a centralized unit (CU) serve K mobile users requesting video streaming. The CU is used for radio resource allocation and centralized computing, and the DU is used for transmitting to each scheduled user with the allocated resources. When using reinforcement learning for the predictive resource allocation, the CU first decides the user association according to the large-scale channel gains of users, then collects the state of every user via DUs, and finally controls the DUs to allocate resources by sending instructions.

Each video is divided into N_v segments. The playback duration of each segment is divided into N_f frames, each with duration ΔT. Each frame contains N_s time slots, each with duration τ. The durations of each frame and each time slot are defined according to the coherence time of large-scale and small-scale channel gains, respectively.

Assume that the users are associated to the DUs with the strongest large-scale channel gains with them and the user association does not change within each frame. Each mobile user is served by only one DU in every frame and may be served by different DUs in different frames during video streaming. When multiple users are associated with a single DU, the DU transmits to the users using frequency division multiple access with optimized powers. Adjacent DUs transmit with frequency reuse. Hence, there are no multi-user interference and inter-cell interference. Then, the data rate of the kth user in the jth time slot of the tth frame can be expressed as $R_{tj}^k = W\log_2(1+\frac{\alpha_t^k g_{tj}^k p_{tj}^k}{\sigma_0^2})$, where α_t^k and g_{tj}^k are respectively the large-scale channel gain in the tth frame and the small-scale channel gain in the jth time slot of the tth frame between the kth user and its associated DU, p_{tj}^k is the transmit power allocated to the kth user in the jth time slot of the tth frame, W is the bandwidth for each user, and σ_0^2 is the noise power.

After a user initiates a request for a video, the first segment (denoted as the 0th segment) of the video has to be conveyed to the user in a best effort manner (i.e., transmitted by the associated DU with all available resources), during which the large-scale channel gains of the user are gathered to facilitate channel prediction for delivering subsequent segments. If each segment can be downloaded to the buffer of a user before playback, then no stalling will occur and the QoS of the user will be satisfied.

When the large-scale channel gains in a prediction window (consisting of future $(N_v-1)N_f$ frames) and the small-scale channel gain of each mobile user at the beginning of each time slot are known, the transmit powers allocated to the users in the future $(N_v-1)N_fN_s$ time slots can be optimized to minimize the total average transmit energy consumed by all the DUs for satisfying the QoS constraint of every user during video streaming [6], i.e.,

$$\mathrm{P1}: \min_{\substack{p_{tj}^k, t=1,\ldots,(N_v-1)N_f,\\ j=1,\ldots,N_s, k=1,\ldots,K,}} \mathbb{E}_g\left[\sum_{k=1}^{K}\left(\sum_{t=1}^{(N_v-1)N_f}\left(\sum_{j=1}^{N_s}\tau p_{tj}^k\right)\right)\right], \tag{1a}$$

$$\text{s.t.} \quad \sum_{n=1}^{n_v}\sum_{t=(n-1)N_f+1}^{nN_f}\sum_{j=1}^{N_s}\tau R_{tj}^k \geqslant \sum_{n=2}^{n_v+1} d_n^k, \quad n_v=1,\ldots,N_v-1, k=1,\ldots,K, \tag{1b}$$

$$\sum_{k=1}^{K} p_{tj}^k I_{t,m}^k \leqslant P^{\max}, \quad m=1,\ldots,M, \tag{1c}$$

where $\mathbb{E}_g[\cdot]$ indicates the expectation taken over small-scale channel gains, d_n^k is the number of bits of the nth segment in the video requested by the kth user, $P^{\max}$ is the maximal transmit power of each DU,

$I_{t,m}^k \in \{0,1\}$ is an association indicator, $I_{t,m}^k = 1$ if the kth user is associated with the mth DU in the tth frame, and $I_{t,m}^k = 0$ otherwise. Eq. (1b) is the QoS constraint of every user, which is satisfied by multiple DUs that serve the user in the prediction window, and Eq. (1c) is the power constraint of each DU.

To predict the future large-scale channel gains meanwhile optimize the future transmit powers, we resort to DRL thanks to its ability to solve an end-to-end problem, which makes the optimization with the implicit prediction.

2.2 Establishing matrix-based MDP

When using reinforcement learning (say the DDPG algorithm) to solve a problem, one first needs to design action, state, and reward. For optimizing the predictive power allocation, we can simply define the instantaneous power p_{tj}^k in the problem as an action. Then, the CU has to collect small-scale channel gains via DUs in every time slot and train the DNNs in DRL (say the actor and critic networks in DDPG) every one or several time slots. To reduce the resulting large signaling overhead and computational cost, we first decompose problem P1 into two nested problems.

Proposition 1. Solving P1 is equivalent to solving P2 and P3 in the following:

$$\mathrm{P2}: \min_{\substack{\bar{R}_t^k, k=1,\ldots,K,\\ t=1,\ldots,(N_v-1)N_f,}} \sum_{t=1}^{(N_v-1)N_f} \sum_{m=1}^{M} \bar{E}_t^m, \qquad \mathrm{P3}: \min_{p_{tj}^k, k\in\mathcal{K}_{t,m}} \bar{E}_t^m,$$

$$\text{s.t.} \sum_{n=1}^{n_v} \sum_{t=(n-1)N_f+1}^{nN_f} \Delta T \bar{R}_t^k - \sum_{n=2}^{n_v+1} d_n^k \geqslant 0, \qquad \text{s.t. } \mathbb{E}_g\left[W \log_2\left(1+\frac{\alpha_t^k g_{tj}^k p_{tj}^k}{\sigma_0^2}\right)\right] - \bar{R}_t^k = 0, \forall k \in \mathcal{K}_{t,m},$$

$$n_v = 1,\ldots,N_v-1,\ k=1,\ldots,K, \qquad \sum_{k\in\mathcal{K}_{t,m}} p_{tj}^k \leqslant P^{\max},$$

where $\bar{E}_t^m \triangleq \mathbb{E}_g[\sum_{k\in\mathcal{K}_{t,m}} \sum_{j=1}^{N_s} \tau p_{tj}^k]$ and $\bar{R}_t^k \triangleq \mathbb{E}_g[R_{tj}^k]$ are the total average energy consumed at the mth DU and the average data rate of the kth user in the tth frame, respectively, and $\mathcal{K}_{t,m}$ is the set of users associated to the mth DU in the tth frame. Since $I_{t,m}^k$ is an association indicator, $\sum_{k\in\mathcal{K}_{t,m}} p_{tj}^k$ in P3 is the equivalent expression of $\sum_{k=1}^{K} p_{tj}^k I_{t,m}^k$ in P1.

Proof. See Appendix A.

P2 is a problem to optimize the average rate allocation with known future large-scale channel gains, and P3 is a problem to optimize instantaneous power allocation with the small-scale channel gains in the jth time slot. Since the solution of P3 affects the values of the objective and constraints of P2, P1 can be solved using a nested optimization [47] as follows: $\bar{R}_t^k, k=1,\ldots,K$ are optimized from P2 (at the CU in each frame) in the outer loop, and $p_{tj}^k, k \in \mathcal{K}_{t,m}$ are optimized from P3 with the optimized average rates (at each DU in each time slot) in the inner loop.

Given that P3 is not a MDP problem, one can use a DNN trained in an off-line and unsupervised manner to optimize the instantaneous power allocation. Let $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$ denote the DNN, where $\boldsymbol{\phi}$ consists of trainable parameters. For example, $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$ can be designed as an FNN, whose inputs are $\boldsymbol{\alpha}_t^m \triangleq [\alpha_t^k, k \in \mathcal{K}_{t,m}]$, $\boldsymbol{g}_{tj}^m \triangleq [g_{tj}^k, k \in \mathcal{K}_{t,m}]$, and $\bar{\boldsymbol{R}}_t^m \triangleq [\bar{R}_t^k, k \in \mathcal{K}_{t,m}]$, and the output is $\boldsymbol{p}_{tj}^m \triangleq [p_{tj}^k, k \in \mathcal{K}_{t,m}]$. Due to the dynamic user association, the dimensions of the input and output vectors are time-varying across frames. To deal with this issue, we can pad the vectors $\boldsymbol{\alpha}_t^m$, $\boldsymbol{g}_{tj}^m$, $\bar{\boldsymbol{R}}_t^m$, and $\boldsymbol{p}_{tj}^k$ with zeros to make their dimensions constant in all frames.

To minimize the total average energy (i.e., the objective of P1 or P2) under the QoS constraint of every user and the power constraint of every DU, we define the action and state for optimizing the predictive rate allocation with implicit channel prediction by extending the matrix-based MDP for a single user in [10] into multiple users, and derive the reward that is relevant to $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$.

Action. The action vector in the tth time step is $\boldsymbol{a}_t = [a_t^1,\ldots,a_t^K]^\mathsf{T} \triangleq [\bar{R}_t^1,\ldots,\bar{R}_t^K]^\mathsf{T}$, where a_t^k is the action of the kth user in the time step. Then, the duration of a time step of DRL is equal to the duration of a frame.

State. Defining state is non-trivial for optimizing P2 with implicit prediction, since it is not straightforward to identify which variables in each time step affect the action. After designing with a trail-and-error procedure, the state of the user (say the kth user in the multi-user scenario) at time step t in [10] is composed of B_t^k, f_t^k, η_t^k, and $\boldsymbol{\alpha}_{t,1}^k,\ldots,\boldsymbol{\alpha}_{t,M}^k$. B_t^k is the amount of data in the buffer of the kth user, and

$f_t^k \in [0, \ldots, N_v - 1]$ is the frame index of the video segment that the kth user plays back, both at the tth time step. η_t^k is the ratio of the amount of data having been delivered to the kth user at time step t to the total amount of data in the video requested by the user. $\boldsymbol{\alpha}_{t,m}^k \triangleq [\alpha_{t-T,m}^k, \ldots, \alpha_{t,m}^k] \in \mathbb{R}^{1\times(T+1)}$, where $T > 1$ is a pre-determined value affecting the performance of the implicit channel prediction, and $\alpha_{t,m}^k$ is the large-scale channel gain between the kth user and the mth DU at time step t.

Different from [10], in the considered scenario a DU may serve multiple users in a time step. To help the agent learn the rate allocation to the users associated with each DU, we introduce the association indicator $I_{t,m}^k$ in P1 into the state. When the kth user is not associated with the mth DU at time step t, the large-scale channel gain between them is not useful for rate allocation. Hence we replace $\boldsymbol{\alpha}_{t,m}^k$ in the state of the single-user scenario in [10] by $\tilde{\boldsymbol{\alpha}}_{t,m}^k \triangleq [\alpha_{t-T,m}^k I_{t-T,m}^k, \ldots, \alpha_{t,m}^k I_{t,m}^k] \in \mathbb{R}^{1\times(T+1)}$. Then, if the kth user is associated with the mth DU at time step t, $\alpha_{t,m}^k I_{t,m}^k = \alpha_{t,m}^k$, otherwise $\alpha_{t,m}^k I_{t,m}^k = 0$. Finally, the state matrix and the state vector of the kth user are, respectively,

$$\boldsymbol{S}_t = [\boldsymbol{s}_t^1, \ldots, \boldsymbol{s}_t^K]^\mathsf{T} \in \mathbb{R}^{K\times((T+1)M+3)}, \quad \boldsymbol{s}_t^k \triangleq [B_t^k, f_t^k, \eta_t^k, \tilde{\boldsymbol{\alpha}}_{t,1}^k, \ldots, \tilde{\boldsymbol{\alpha}}_{t,M}^k]^\mathsf{T} \in \mathbb{R}^{((T+1)M+3)\times 1}. \tag{3}$$

Reward. In the following, we design the reward to minimize the objective function meanwhile satisfy the QoS constraints in P1. Since the solution of P3 affects the value of the constraints of P2, the following two conditions should be satisfied in order to ensure the QoS of every user. (i) The learned action satisfies the $K(N_v - 1)$ constraints in P2, and (ii) the learned action is achievable under the maximal power constraint of each DU (i.e., P3 has a feasible solution).

To satisfy the first condition, we introduce a safe layer into the actor network to map the action $\boldsymbol{a}_t$ into $\tilde{\boldsymbol{a}}_t = [\tilde{a}_t^1, \ldots, \tilde{a}_t^K]^\mathsf{T}$ at the time steps $t = lN_f, l = 1, \ldots, N_v - 1$ when the constraints cannot be satisfied by taking the action $\boldsymbol{a}_t$, where $\tilde{a}_t^k$ is the adjusted action for the kth user. As a safe layer, the mapping from its input to its output should be a function with closed-form expression, otherwise the gradient of the output with respective to the input cannot be obtained for training the actor network [48]. In what follows, we derive the mapping from $\boldsymbol{a}_t$ to $\tilde{\boldsymbol{a}}_t$.

According to the definition of B_t^k and d_l^k, we can obtain $B_{t+1}^k = B_t^k + \tilde{a}_t^k \Delta T - d_{l-1}^k$ at time step $t = lN_f$ (i.e., when the playback of the lth segment is finished). To ensure the QoS constraint of the kth user, i.e., $B_{t+1}^k \geqslant d_l^k$, we should ensure $\tilde{a}_t^k \geqslant (d_l^k + d_{l-1}^k - B_t^k)/\Delta T$. If a_t^k can satisfy the QoS constraint, then it does not need to be adjusted. Otherwise, the learned action vector should be adjusted by the safe layer as $\tilde{\boldsymbol{a}}_t = (\boldsymbol{d}_l + \boldsymbol{d}_{l-1} - \boldsymbol{B}_t)/\Delta T$, where $\boldsymbol{B}_t \triangleq [B_t^1, \ldots, B_t^K]^\mathsf{T}$ and $\boldsymbol{d}_l \triangleq [d_l^1, \ldots, d_l^K]^\mathsf{T}$. The input-output relation of the safe layer can be expressed as

$$\tilde{\boldsymbol{a}}_t = ((\boldsymbol{d}_l + \boldsymbol{d}_{l-1} - \boldsymbol{B}_t)/\Delta T) \odot (1 - \boldsymbol{i}_t) + \boldsymbol{a}_t \odot \boldsymbol{i}_t, \tag{4}$$

where $\odot$ denotes element-wise product, $\boldsymbol{i}_t = [i_t^1, \ldots, i_t^K]^\mathsf{T}$, $i_t^k = 1$ if the QoS constraint of the kth user can be satisfied by taking the action a_t^k and $i_t^k = 0$ otherwise.

The second condition cannot be satisfied, if the learned average rates of the users associated to a DU (say the mth DU) cannot be achieved even when the DU transmits with the maximal power (i.e., the instantaneous powers obtained by $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$ satisfies $\sum_{k\in\mathcal{K}_{t,m}} p_{tj}^k = P^{\max}$). To satisfy the condition, we impose a penalty on the reward at time step t, which is

$$r_t = -\sum_{m=1}^{M}\sum_{k\in\mathcal{K}_{t,m}}\sum_{j=1}^{N_s} \tau p_{tj}^k - \lambda \sum_{m=1}^{M}\sum_{k\in\mathcal{K}_{t,m}} \max\left\{\left(\tilde{a}_t^k I(t = lN_f) + a_t^k\,(1 - I(t = lN_f)) - \tilde{R}_t^k\right), 0\right\}, \tag{5}$$

where p_{tj}^k is the transmit power learned by $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$ using $\tilde{\boldsymbol{a}}_t$ at time steps $t = lN_f, l = 1, \ldots, N_v - 1$ and using $\boldsymbol{a}_t$ at other time steps, $\tilde{R}_t^k$ is the average rate of the kth user at time step t achieved by respectively transmitting $p_{t1}^k, \ldots, p_{tN_s}^k$ in the first, $\ldots, N_s$th time slots, λ is a tunable coefficient, and $I(t = lN_f)$ equals 1 if $t = lN_f$ and 0 otherwise. The first term helps minimize the objective function of P1, and the second term is the penalty.

Since the action lies in continuous space, we employ the DDPG algorithm [49] to solve the problem, where the actor and critic networks need to be trained. The actor network learns a policy function, which maps the state into action. The critic network learns an action-value function, which maps state-action pair into expected return $q_t \triangleq \mathbb{E}[\sum_{i=t}^{(N_v-1)N_f} \gamma r_i]$, where the matrix-based state-action pair can be expressed as $[\boldsymbol{S}_t, \boldsymbol{a}_t] \in \mathbb{R}^{K\times((T+1)M+4)}$ and $\gamma \in [0, 1]$ is the discount factor.

Using the DDPG algorithm, the predictive power allocation can be optimized, which is implemented as follows. After the first segment has been downloaded to each user and the large-scale channel gains have been collected, the CU sends $\tilde{\boldsymbol{a}}_t$ or $\boldsymbol{a}_t$ obtained with the DDPG algorithm to each DU in time step t. Within the time step each DU serves its associated users with the optimized transmit powers using $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$ in each time slot. At the end of the time step t, each DU computes and uploads its energy consumption and penalty in the time step to the CU for calculating r_t. At the end of the playback of each segment (i.e., at time step $t = lN_f$), the CU decides if the safe layer needs to be used to ensure the QoS. When all the videos requested by the users have been delivered, an episode terminates.

3 Converting matrix-based MDP into graph-based MDP

The state and state-action pair can either be represented by matrices or by graphs. The matrix-based and graph-based MDPs differ in the way of representing the state and state-action pair, while their rewards are the same.

In this section, we introduce a general method of converting the matrix-based state and state-action pair into the graph-based state and state-action pair. We then take the matrix-based MDP in Subsection 2.2 as an example to demonstrate how to apply the method.

3.1 General method of the conversion

A graph consists of a set of vertices and a set of edges, where each vertex and each edge may be associated with a feature and belong to a type. A graph can be represented by adjacency matrix and feature matrices [50]. The adjacency matrix reflects the topology of a graph, whose element at the ith row and the jth column is a non-zero integer if there is an edge between the ith vertex and the jth vertex, and zero otherwise. Feature matrix contains the features of vertices or edges.

In the following, we show how to obtain the adjacency matrix and feature matrix of the state and state-action graphs from the matrix-based state and state-action pair.

All entities in the system (e.g., users and DUs in the predictive power allocation problem) relevant to each element in the state matrix (or state-action matrix) are defined as vertices. If the element involves only one entity, the element is defined as a feature of the corresponding vertex. If the element involves two entities (e.g., the channel between a user and a DU), then there is an edge connecting the corresponding two vertices, and the element is defined as a feature of the edge. With the elements in the state matrix (or state-action matrix), we can obtain the vertex set, edge set, and feature matrix of the state graph (or state-action graph). Then, we can obtain an adjacency matrix with the defined vertices and edges.

3.2 Graph-based MDP for predictive power allocation

In the sequel, we use the general method to convert the state and state-action matrices for predictive power allocation into graphs.

We can obtain a state graph according to the state matrix in (3). Since the elements in $[B_t^k, f_t^k, \eta_t^k]^{\mathsf{T}} \triangleq \boldsymbol{u}_t^k$ only involve the kth user, each user is defined as a vertex and $\boldsymbol{u}_t^k$ is the feature vector of the kth user vertex. Then, the feature matrix of all user vertices in the state graph at time step t is $\boldsymbol{U}_t = [\boldsymbol{u}_t^1, \ldots, \boldsymbol{u}_t^K]^{\mathsf{T}}$. Since $[\tilde{\boldsymbol{\alpha}}_{t,1}^k, \ldots, \tilde{\boldsymbol{\alpha}}_{t,M}^k]^{\mathsf{T}}$ involves the kth user and all DUs, hence each DU is another type of vertex and there is an edge between every user and every DU, where $\tilde{\boldsymbol{\alpha}}_{t,m}^k \triangleq \boldsymbol{e}_{t,m}^k$ is the feature vector of the edge between the kth user vertex and the mth DU vertex. The features of all edges in the state graph at time step t can be expressed as a tensor $\boldsymbol{E}_t \in \mathbb{R}^{K\times M\times(T+1)}$, where a slice along the third dimension is $\boldsymbol{E}_t[k, m, :] = \boldsymbol{e}_{t,m}^k$.

After identifying vertices and edges, the vertex set is defined as $\mathcal{V} = \{1, \ldots, K+M\}$. We represent a vertex by its index. Then, $1, \ldots, K$ are the indices of user vertices and $K+1, \ldots, K+M$ are the indices of DU vertices. The edge set can be expressed as $\mathcal{E} = \{\{v_1, v_2\} | v_1 = 1, \ldots, K, v_2 = K+1, \ldots, K+M\}$, where $\{v_1, v_2\}$ represents the edge between the v_1th vertex and the v_2th vertex. Let $\boldsymbol{A} \in \mathbb{I}^{(K+M)\times(K+M)}$ denote the adjacency matrix, which is constant over time, whose element in the ith row and jth column is 1 if $\{i, j\} \in \mathcal{E}$, and is 0 otherwise.

In the state-action graph, the action (i.e., the average rate of a user) only involves the user, hence is a feature of user vertex. Then, the feature vector of the kth user is $\widetilde{\boldsymbol{u}}_t^k \triangleq [B_t^k, f_t^k, \eta_t^k, \bar{R}_t^k]^{\mathsf{T}}$, and the

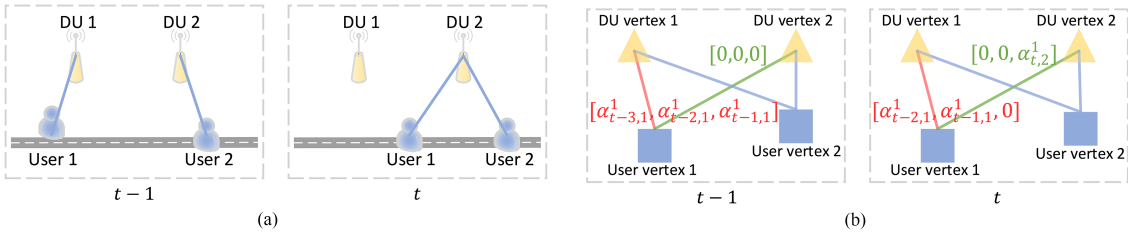

Figure 1 (Color online) User 1 is associated with DU 1 at time steps $t-3$, $t-2$, $t-1$ and with DU 2 at time step t. In the state graph, $T=2$. (a) User association at time steps $t-1$ and t, where the lines indicate the association relationships; (b) state graphs at time steps $t-1$ and t, where the lines indicate the edges and the vectors beside the lines are the features of the edges.

feature matrix of all user vertices at time step t is $\widetilde{\boldsymbol{U}}_t = [\widetilde{\boldsymbol{u}}_t^1, \ldots, \widetilde{\boldsymbol{u}}_t^K]^\mathsf{T}$. The feature matrix of all edges and adjacency matrix in the state-action graph at time step t are also $\boldsymbol{E}_t$ and $\boldsymbol{A}$, respectively.

For both the state and state-action graphs, the DU vertices do not have features.

Finally, the state graph and state-action graph can be, respectively, expressed as

$$\mathtt{S}_t^G = (\boldsymbol{U}_t, \boldsymbol{E}_t, \boldsymbol{A}), \qquad \mathtt{SA}_t^G = (\widetilde{\boldsymbol{U}}_{\boldsymbol{t}}, \boldsymbol{E}_t, \boldsymbol{A}). \tag{6}$$

Remark 1. There is an edge between every user and every DU in the state and state-action graphs. This is because the features of edges contain historical channels, which are essential for optimizing power allocation with implicit prediction. We take the scenario in Figure 1 as an example to illustrate this. At time step t, although user 1 is not associated with DU 1 (see Figure 1(a)), there is an edge between user vertex 1 and DU vertex 1 whose feature consists of historical channels (i.e., $\alpha^1_{t-2,1}$ and $\alpha^1_{t-1,1}$) that are necessary for implicit channel prediction (see Figure 1(b)).

4 Graph DDPG algorithm with desired permutation properties

In this section, we first introduce a vanilla GNN and several permutation properties to be used in the sequel. Then, we present the graph DDPG algorithm and the permutation properties of the policy function or action-value function defined on the graphs for the predictive power allocation.

4.1 Vanilla GNN and several permutation properties

Vanilla GNN. As a kind of DNNs, GNN is proposed for learning over graphs. In each layer of a GNN, the hidden representation of each vertex is obtained by aggregating the processed information of each neighboring vertex and edge and the combining with its own hidden representation in the previous layer, where the neighboring vertices of a vertex are the vertices connected to the vertex with an edge, and the edge is called the neighboring edge of the vertex. The aggregation and combination of a vanilla GNN are presented as follows, where each vertex uses the same processor, combiner, and pooling function.

Aggregation. The aggregated output at the vth vertex in the lth layer is

$$\boldsymbol{d}_v^{'(l)} = \mathrm{PL}_{n\in\mathcal{N}(v)}\left(q(\boldsymbol{d}_n^{(l-1)}, \boldsymbol{e}_v^n; \boldsymbol{\phi}^{(l)})\right), \tag{7}$$

where $\mathrm{PL}_{n\in\mathcal{N}(v)}(\cdot)$ is the pooling function to aggregate information from the vertices in $\mathcal{N}(v)$ and the corresponding neighboring edges, $q(\cdot,\cdot;\boldsymbol{\phi}^{(l)})$ is the processor with model parameters $\boldsymbol{\phi}^{(l)}$, $\mathcal{N}(v)$ is the set of neighboring vertices of the vth vertex that can be obtained from adjacency matrix, $\boldsymbol{d}_n^{(l-1)}$ is the representation of the nth vertex in the $(l-1)$th layer, and $\boldsymbol{e}_v^n$ is the feature of the edge between the vth vertex and the nth vertex.

Combination. The hidden representation of the vth vertex in the lth layer is

$$\boldsymbol{d}_v^{(l)} = \mathrm{CB}(\boldsymbol{d}_v^{(l-1)}, \boldsymbol{d}_v^{'(l)}; \boldsymbol{\theta}^{(l)}), \tag{8}$$

where $\mathrm{CB}(\cdot,\cdot,\boldsymbol{\theta}^{(l)})$ is the combiner with model parameters $\boldsymbol{\theta}^{(l)}$.

Permutation properties. Let $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ denote permutation matrices that can permute the rows or columns of a matrix $\boldsymbol{X}$ (or a vector $\boldsymbol{y}$) by multiplying with the matrix (or the vector), which are unitary matrices with "0" and "1". A multivariate function $\boldsymbol{y} = f(\boldsymbol{X})$ is one-dimension (1D)-permutation equivariance (PE) to $\boldsymbol{X}$ if $\boldsymbol{\Pi}_1\boldsymbol{y} = f(\boldsymbol{\Pi}_1\boldsymbol{X})$, 1D-permutation invariance (PI) to $\boldsymbol{X}$ if $\boldsymbol{y} = f(\boldsymbol{\Pi}_1\boldsymbol{X})$, two-dimension (2D)-PE to $\boldsymbol{X}$ if $\boldsymbol{\Pi}_1\boldsymbol{y} = f(\boldsymbol{\Pi}_1\boldsymbol{X}\boldsymbol{\Pi}_2^\mathsf{T})$, 2D-PI to $\boldsymbol{X}$ if $\boldsymbol{y} = f(\boldsymbol{\Pi}_1\boldsymbol{X}\boldsymbol{\Pi}_2^\mathsf{T})$, joint-PE to $\boldsymbol{X}$ if $\boldsymbol{\Pi}_1\boldsymbol{y} = f(\boldsymbol{\Pi}_1\boldsymbol{X}\boldsymbol{\Pi}_1^\mathsf{T})$, and joint-PI to $\boldsymbol{X}$ if $\boldsymbol{y} = f(\boldsymbol{\Pi}_1\boldsymbol{X}\boldsymbol{\Pi}_1^\mathsf{T})$ [15,51].

4.2 Graph DDPG algorithm and desired permutation properties

4.2.1 *Graph DDPG algorithm*

Let $\mu(\cdot)$ denote the policy function, and $Q(\cdot)$ denote the action-value function, which are the functions defined on the state graph and state-action graph, respectively.

The actor network that learns the policy function or the critic network that learns the action-value function is a composite of a GNN and an output layer. The GNN is used to learn the vertex representation. The output layer plays the role of mapping the vertex representation into the action for the actor network or the expected return for the critic network.

4.2.2 *Desired permutation properties for predictive rate allocation*

In the sequel, we treat the edge feature $\boldsymbol{E}_t \in \mathbb{R}^{K\times M\times(T+1)}$ as a $K \times M$ matrix, whose element in the kth row and the mth column is $\boldsymbol{e}_{t,m}^k$, since the elements in the third dimension are for prediction and is not permutable. This is a little misuse of notation, but can simplify the expressions of permutation properties significantly and enables easy understanding of the properties.

The policy function defined on the state graph in (6) is $\boldsymbol{a}_t = \mu(\boldsymbol{U}_t, \boldsymbol{E}_t, \boldsymbol{A})$. If we re-order the user vertices in the state graph, then the elements in $\boldsymbol{a}_t$, the rows of $\boldsymbol{U}_t$ and $\boldsymbol{E}_t$, the first K rows and columns of $\boldsymbol{A}$ will be permuted. If we re-order the DU vertices, then the columns of $\boldsymbol{E}_t$, the last M rows and columns of $\boldsymbol{A}$ will be permuted. The action-value function defined on the state-action graph in (6) is $q_t = Q(\widetilde{\boldsymbol{U}}_t, \boldsymbol{E}_t, \boldsymbol{A})$. The value of q_t will remain unchanged if we re-order the user or DU vertices.

Then, the permutation properties of $\mu(\cdot)$ and $Q(\cdot)$ can be, respectively, expressed as

$$\boldsymbol{\Pi}_1\boldsymbol{a}_t = \mu(\boldsymbol{\Pi}_1\boldsymbol{U}_t, \boldsymbol{\Pi}_1\boldsymbol{E}_t\boldsymbol{\Pi}_2^{\mathsf{T}}, \tilde{\boldsymbol{\Pi}}_1\boldsymbol{A}\tilde{\boldsymbol{\Pi}}_1^{\mathsf{T}}), \tag{9a}$$

$$q_t = Q(\boldsymbol{\Pi}_1\widetilde{\boldsymbol{U}}_t, \boldsymbol{\Pi}_1\boldsymbol{E}_t\boldsymbol{\Pi}_2^{\mathsf{T}}, \tilde{\boldsymbol{\Pi}}_1\boldsymbol{A}\tilde{\boldsymbol{\Pi}}_1^{\mathsf{T}}), \tag{9b}$$

where $\tilde{\boldsymbol{\Pi}}_1 = \mathrm{diag}(\boldsymbol{\Pi}_1, \boldsymbol{\Pi}_2)$ is a block diagonal matrix, $\boldsymbol{\Pi}_1 \in \mathbb{I}^{K\times K}$ is the permutation matrix for the indices of user vertices, and $\boldsymbol{\Pi}_2 \in \mathbb{I}^{M\times M}$ is the permutation matrix for the indices of DU vertices.

The permutation property of a function defined on a graph comes from the fact that the vertices with the same type do not have ordering. Since the elements in the feature matrices and adjacency matrix (i.e., the input of the function) are arranged according to the indices of vertices, when the vertices with the same type are re-ordered, the feature and adjacency matrices are permuted accordingly. Then, the output of the function may be permuted accordingly or remain unchanged.

To help understand the permutation property of a function defined on a graph, we take the policy function $\mu(\cdot)$ defined on the state graph constructed for the predictive power allocation problem with two user vertices (called Alice and Bob) as an example. For easy understanding, we replace the superscript k in the feature of user vertex $\boldsymbol{u}_t^k$, the feature of edge $\boldsymbol{e}_{t,m}^k$, and the action $\bar{R}_t^k$ to Alice and Bob. The two users do not have ordering, hence either Alice or Bob can be defined as user vertex 1. When Alice is defined as user vertex 1, the state graph is shown in Figure 2(a), the input (i.e., the feature matrices $\boldsymbol{U}_t$ and $\boldsymbol{E}_t$ as well as the adjacency matrix $\boldsymbol{A}$) and output (i.e., the action vector $\boldsymbol{a}_t$) of the policy function $\mu(\cdot)$ can be expressed as follows:

$$\boldsymbol{U}_t = \begin{bmatrix} \boldsymbol{u}_t^{\mathrm{Alice}} \\ \boldsymbol{u}_t^{\mathrm{Bob}} \end{bmatrix}, \boldsymbol{E}_t = \begin{bmatrix} \boldsymbol{e}_{t,1}^{\mathrm{Alice}} & \cdots & \boldsymbol{e}_{t,M}^{\mathrm{Alice}} \\ \boldsymbol{e}_{t,1}^{\mathrm{Bob}} & \cdots & \boldsymbol{e}_{t,M}^{\mathrm{Bob}} \end{bmatrix}, \boldsymbol{a}_t = \begin{bmatrix} \bar{R}_t^{\mathrm{Alice}} \\ \bar{R}_t^{\mathrm{Bob}} \end{bmatrix}, \boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_1 \\ \boldsymbol{I}_1^{\mathsf{T}} & \boldsymbol{0} \end{bmatrix},$$

where $\boldsymbol{I}_1 \in \mathbb{R}^{2\times M}$ is an identity matrix. When we re-order the user vertices as shown in Figure 2(b), since both input and output are arranged according to the indices of vertices and edges, the input and output are permuted accordingly by multiplying with permutation matrix. The permutation matrix (defined as $\boldsymbol{\Pi}$ and $\tilde{\boldsymbol{\Pi}}$), the input and output of the policy function after permutation can be expressed as follows:

$$\boldsymbol{\Pi} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \boldsymbol{\Pi}\boldsymbol{U}_t = \begin{bmatrix} \boldsymbol{u}_t^{\mathrm{Bob}} \\ \boldsymbol{u}_t^{\mathrm{Alice}} \end{bmatrix}, \boldsymbol{\Pi}\boldsymbol{E}_t = \begin{bmatrix} \boldsymbol{e}_{t,1}^{\mathrm{Bob}} & \cdots & \boldsymbol{e}_{t,M}^{\mathrm{Bob}} \\ \boldsymbol{e}_{t,1}^{\mathrm{Alice}} & \cdots & \boldsymbol{e}_{t,M}^{\mathrm{Alice}} \end{bmatrix}, \boldsymbol{\Pi}\boldsymbol{a}_t = \begin{bmatrix} \bar{R}_t^{\mathrm{Bob}} \\ \bar{R}_t^{\mathrm{Alice}} \end{bmatrix}, \tilde{\boldsymbol{\Pi}} = \begin{bmatrix} \boldsymbol{\Pi} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_2 \end{bmatrix}, \tilde{\boldsymbol{\Pi}}\boldsymbol{A}\tilde{\boldsymbol{\Pi}}^{\mathsf{T}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_1 \\ \boldsymbol{I}_1^{\mathsf{T}} & \boldsymbol{0} \end{bmatrix},$$

where $\boldsymbol{I}_2 \in \mathbb{R}^{M\times M}$ is another identity matrix.

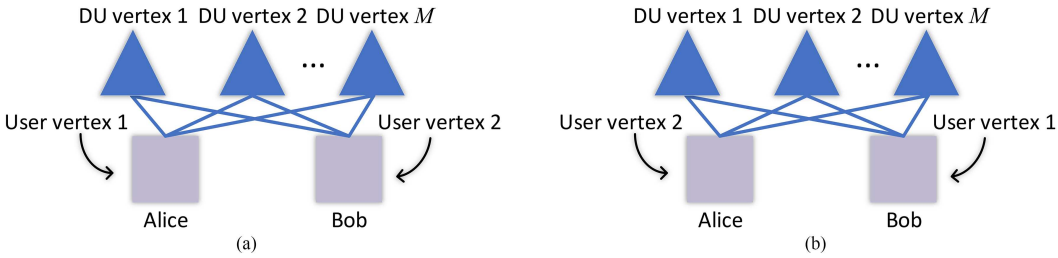

Figure 2 (Color online) Permuting the indices of two user vertices in the state graph constructed for the predictive power allocation problem. (a) Alice is user vertex 1; (b) Bob is user vertex 1.

4.3 Graph DDPG algorithm for predictive rate allocation

In this section, we design a graph DDPG algorithm that is with the permutation properties of predictive rate allocation. To satisfy the permutation properties of the policy function and the action-value function, both the GNN and the output layer in the graph DDPG algorithm need to be designed.

Let $\mathrm{GNN_a}(\cdot)$ and $F_{\mathrm{a}}(\cdot)$ denote the GNN and the output layer of the actor network, respectively, and $\mathrm{GNN_c}(\cdot)$ and $F_{\mathrm{c}}(\cdot)$ denote the GNN and the output layer of the critic network, respectively. The following proposition provides the conditions of the GNNs and output layers in the actor and critic networks to satisfy the properties in (9).

Proposition 2. If $\mathrm{GNN_a}(\cdot)$, $F_{\mathrm{a}}(\cdot)$, $\mathrm{GNN_c}(\cdot)$, and $F_{\mathrm{c}}(\cdot)$, respectively, satisfy the following properties, then the actor network satisfies the permutation property in (9a) and the critic network satisfies the permutation property in (9b):

$$\boldsymbol{\Pi}_1\boldsymbol{U}_t^{(L)}, \boldsymbol{\Pi}_2\boldsymbol{B}_t^{(L)} = \mathrm{GNN_a}(\boldsymbol{\Pi}_1\boldsymbol{U}_t, \boldsymbol{\Pi}_1\boldsymbol{E}_t\boldsymbol{\Pi}_2^{\mathsf{T}}, \tilde{\boldsymbol{\Pi}}_1\boldsymbol{A}\tilde{\boldsymbol{\Pi}}_1^{\mathsf{T}}), \qquad \boldsymbol{\Pi}_1\boldsymbol{a}_t = F_{\mathrm{a}}(\boldsymbol{\Pi}_1\boldsymbol{U}_t^{(L)}, \boldsymbol{\Pi}_2\boldsymbol{B}_t^{(L)}), \tag{10a}$$

$$\boldsymbol{\Pi}_1\widetilde{\boldsymbol{U}}_t^{(L)}, \boldsymbol{\Pi}_2\widetilde{\boldsymbol{B}}_t^{(L)} = \mathrm{GNN_c}(\boldsymbol{\Pi}_1\widetilde{\boldsymbol{U}}_t, \boldsymbol{\Pi}_1\boldsymbol{E}_t\boldsymbol{\Pi}_2^{\mathsf{T}}, \tilde{\boldsymbol{\Pi}}_1\boldsymbol{A}\tilde{\boldsymbol{\Pi}}_1^{\mathsf{T}}), \qquad q_t = F_{\mathrm{c}}(\boldsymbol{\Pi}_1\widetilde{\boldsymbol{U}}_t^{(L)}, \boldsymbol{\Pi}_2\widetilde{\boldsymbol{B}}_t^{(L)}), \tag{10b}$$

where $\boldsymbol{U}_t^{(L)} \triangleq [\boldsymbol{u}_{t,1}^{(L)}, \ldots, \boldsymbol{u}_{t,K}^{(L)}]^{\mathsf{T}}$, $\boldsymbol{B}_t^{(L)} \triangleq [\boldsymbol{b}_{t,1}^{(L)}, \ldots, \boldsymbol{b}_{t,M}^{(L)}]^{\mathsf{T}}$, $\widetilde{\boldsymbol{U}}_t^{(L)} \triangleq [\widetilde{\boldsymbol{u}}_{t,1}^{(L)}, \ldots, \widetilde{\boldsymbol{u}}_{t,K}^{(L)}]^{\mathsf{T}}$, $\widetilde{\boldsymbol{B}}_t^{(L)} \triangleq [\widetilde{\boldsymbol{b}}_{t,1}^{(L)}, \ldots, \widetilde{\boldsymbol{b}}_{t,M}^{(L)}]^{\mathsf{T}}$, $\boldsymbol{u}_{t,k}^{(l)}$ and $\boldsymbol{b}_{t,m}^{(l)}$ respectively denote the hidden representations of the kth user vertex and the mth DU vertex in the lth layer of the actor network at time step t, $\widetilde{\boldsymbol{u}}_{t,k}^{(l)}$ and $\widetilde{\boldsymbol{b}}_{t,m}^{(l)}$ respectively denote the hidden representations of the kth user vertex and the mth DU vertex in the lth layer of the critic network at time step t, and L is the number of layers of GNNs.

Proof. See Appendix B.

In the following, we respectively design the GNNs and output layers in the actor and critic networks to satisfy the permutation properties in (10).

4.3.1 *Designing GNNs to satisfy the permutation properties in* (10)

By substituting (7) into (8), we can obtain the updating equation of the vanilla GNN. The GNN with such an updating equation can satisfy the 1D-PE, 1D-PI, joint PE, joint-PI, and their combinational properties, which however cannot satisfy the properties of GNNs in (10) that include 2D-PE property.

By comparing (10a) and (10b), we can see that $\mathrm{GNN_a}(\cdot)$ and $\mathrm{GNN_c}(\cdot)$ exhibit the same permutation property, and their only difference lies in the input graph. In the sequel, we only design $\mathrm{GNN_a}(\cdot)$ to satisfy the property in (10a). The design of $\mathrm{GNN_c}(\cdot)$ is the same.

For easy understanding, we replace $\boldsymbol{d}_v^{(l-1)}$ in (7) and (8) by $\boldsymbol{b}_{t,m}^{(l-1)}$ and $\boldsymbol{u}_{t,k}^{(l-1)}$, and replace $\boldsymbol{e}_v^n$ in (7) and (8) by $\boldsymbol{e}_{t,m}^k$ (i.e., the feature of the edge between the mth DU vertex and the kth user vertex at time step t). We add subscript t since the feature matrices of the state graph and the state-action graph in (6) are time-varying.

Different from (7) and (8) that use the same processor and combiner for all vertices, we use different processors and combiners for user vertices and DU vertices. Specifically, the processor and combiner for DU vertices are respectively $q_{\mathtt{DU}}(\boldsymbol{u}_{t,k}^{(l-1)}, \boldsymbol{e}_{t,m}^k; \boldsymbol{\phi}_{\mathtt{DU}}^{(l)}) = \boldsymbol{V}_1^{(l)}\boldsymbol{u}_{t,k}^{(l-1)} + \boldsymbol{V}_2^{(l)}\boldsymbol{e}_{t,m}^k$ and $\mathrm{CB}_{\mathtt{DU}}(\boldsymbol{b}_{t,m}^{(l-1)}, \boldsymbol{d}_m^{'(l)}; \boldsymbol{\theta}_{\mathtt{DU}}^{(l)}) = \sigma(\boldsymbol{V}_3^{(l)}\boldsymbol{b}_{t,m}^{(l-1)} + \boldsymbol{d}_m^{'(l)})$, where $\sigma(\cdot)$ is the activation function, $\boldsymbol{V}_1^{(l)}$, $\boldsymbol{V}_2^{(l)}$, and $\boldsymbol{V}_3^{(l)}$ are model parameters. The processor and combiner for user vertices are respectively $q_{\mathtt{user}}(\boldsymbol{b}_{t,m}^{(l-1)}, \boldsymbol{e}_{t,m}^k; \boldsymbol{\phi}_{\mathtt{user}}^{(l)}) = \boldsymbol{Y}_1^{(l)}\boldsymbol{b}_{t,m}^{(l-1)} + \boldsymbol{Y}_2^{(l)}\boldsymbol{e}_{t,m}^k$ and $\mathrm{CB}_{\mathtt{user}}(\boldsymbol{u}_{t,k}^{(l-1)}, \boldsymbol{d}_k^{'(l)}; \boldsymbol{\theta}_{\mathtt{user}}^{(l)}) = \sigma(\boldsymbol{Y}_3^{(l)}\boldsymbol{u}_{t,k}^{(l-1)} + \boldsymbol{d}_k^{'(l)})$, where $\boldsymbol{Y}_1^{(l)}$, $\boldsymbol{Y}_2^{(l)}$, and $\boldsymbol{Y}_3^{(l)}$ are model parameters.

Then, the aggregation and combination of the designed GNN for predictive power allocation, which can be applied to both $\mathrm{GNN_a}(\cdot)$ and $\mathrm{GNN_c}(\cdot)$, are as follows:

• DU vertices aggregating information from user vertices

$$\begin{aligned}&\mathsf{Aggregation}: \boldsymbol{b}'^{(l)}_{t,m} = \frac{1}{K}\sum_{k=1}^{K}\left(\boldsymbol{V}_1^{(l)}\boldsymbol{u}_{t,k}^{(l-1)} + \boldsymbol{V}_2^{(l)}\boldsymbol{e}_{t,m}^{k}\right),\\ &\mathsf{Combination}: \boldsymbol{b}_{t,m}^{(l)} = \sigma\left(\boldsymbol{V}_3^{(l)}\boldsymbol{b}_{t,m}^{(l-1)} + \boldsymbol{b}'^{(l)}_{t,m}\right), m = 1,\ldots,M;\end{aligned} \tag{11a}$$

• User vertices aggregating information from DU vertices

$$\begin{aligned}&\mathsf{Aggregation}: \boldsymbol{u}'^{(l)}_{t,k} = \frac{1}{M}\sum\nolimits_{m=1}^{M}\left(\boldsymbol{Y}_1^{(l)}\boldsymbol{b}_{t,m}^{(l-1)} + \boldsymbol{Y}_2^{(l)}\boldsymbol{e}_{t,m}^{k}\right),\\ &\mathsf{Combination}: \boldsymbol{u}_{t,k}^{(l)} = \sigma\left(\boldsymbol{Y}_3^{(l)}\boldsymbol{u}_{t,k}^{(l-1)} + \boldsymbol{u}'^{(l)}_{t,k}\right), k = 1,\ldots,K,\end{aligned} \tag{11b}$$

where the pooling function for all DU and user vertices are mean function, $\boldsymbol{b}_{t,m}^{(0)} = 0, \forall t, m$ since DU vertices have no features.

Similar to the proof in [15], we can prove that the GNN with the aggregation and combination in (11) satisfies the permutation properties of $\mathrm{GNN_a}(\cdot)$ and $\mathrm{GNN_c}(\cdot)$ in (10).

Remark 2. In (11), each user vertex or DU vertex aggregates information from all DU vertices or user vertices. This is because the features of edges between user vertices and DU vertices contain the historical channels that are useful for predictive power allocation, as explained in Remark 1.

4.3.2 *Designing the output layer to satisfy the permutation properties in* (10)

For the actor network, the output layer needs to satisfy $\boldsymbol{\Pi}_1\boldsymbol{a}_t = F_{\mathrm{a}}(\boldsymbol{\Pi}_1\boldsymbol{U}_t^{(L)}, \boldsymbol{\Pi}_2\,\boldsymbol{B}_t^{(L)})$, as shown in (10a). Considering that the output layer satisfying this property is not unique, we select a simple one, which is $\boldsymbol{a}_t = F_{\mathrm{a}}(\boldsymbol{U}_t^{(L)}, \boldsymbol{B}_t^{(L)}) = \boldsymbol{U}_t^{(L)}$. Since $\boldsymbol{a}_t$ is a vector but $\boldsymbol{U}_t^{(L)}$ is a matrix, all dimensions of $\boldsymbol{u}_{t,1}^{(L)},\ldots,\boldsymbol{u}_{t,K}^{(L)}$ should be set as one.

For the critic network, the output layer needs to satisfy $q = F_{\mathrm{c}}(\boldsymbol{\Pi}_1\boldsymbol{U}_t^{(L)}, \boldsymbol{\Pi}_2\boldsymbol{B}_t^{(L)})$, as shown in (10b). Again, we select a simple function that satisfies the property, which is $q = \lambda_1\sum_{k=1}^{K}\boldsymbol{u}_{t,k}^{(L)} + \lambda_2\sum_{m=1}^{M}\boldsymbol{b}_{t,m}^{(L)}$, where λ_1 and λ_2 are hyper-parameters. Since q is a scalar, the dimension of every vector $\boldsymbol{u}_{t,1}^{(L)},\ldots,\boldsymbol{u}_{t,K}^{(L)},\boldsymbol{b}_{t,1}^{(L)},\ldots,\boldsymbol{b}_{t,M}^{(L)}$ should be set as one.

5 Time complexity of graph-based DDPG and FNN-DDPG

In the following, we analyze the time complexity of training the graph-based DDPG (GNN-DDPG for short) and FNN-DDPG algorithms for predictive power allocation. The FNN-DDPG uses two FNNs to learn the policy and action-value functions [49], with the matrix-based MDP established in Subsection 2.2.

We use the number of multiplications as the metric of time complexity [38]. The batch size for training the GNN-DDPG and FNN-DDPG algorithms is defined as B.

5.1 Time complexity of training the GNN-DDPG

The dimension of the hidden representation of each vertex in each layer is defined as d, and the dimension of the feature vector of each edge is $(T+1)$, i.e., $\boldsymbol{e}_{t,m}^{k} \in \mathbb{R}^{1\times(T+1)}$. Suppose that both the actor and critic networks have L_1 layers.

The gradient with respect to $\boldsymbol{V}_1^{(l)}$ and $\boldsymbol{Y}_1^{(l)}$ can be derived from (11) as

$$\frac{\partial q_t}{\partial \boldsymbol{V}_1^{(l)}} = \sum_{m=1}^{M}\frac{\partial q_t}{\partial \boldsymbol{b}_{t,m}^{(l)}}\odot\sigma'\left(\boldsymbol{P}_{t,m}^{(l)}\right)\cdot\left(\sum_{k=1}^{K}\boldsymbol{u}_{t,k}^{(l-1)}\right)^{\mathsf{T}},\ \frac{\partial q_t}{\partial \boldsymbol{Y}_1^{(l)}} = \sum_{k=1}^{K}\frac{\partial q_t}{\partial \boldsymbol{u}_{t,k}^{(l)}}\odot\sigma'\left(\boldsymbol{Q}_{t,k}^{(l)}\right)\cdot\left(\sum_{m=1}^{M}\boldsymbol{b}_{t,m}^{(l-1)}\right)^{\mathsf{T}}, \tag{13a}$$

$$\frac{\partial q_t}{\partial \boldsymbol{b}_{t,m}^{(l-1)}} = (\boldsymbol{V}_3^{(l)})^{\mathsf{T}} \frac{\partial q_t}{\partial \boldsymbol{b}_{t,m}^{(l)}} \odot \sigma'\left(\boldsymbol{P}_{t,m}^{(l)}\right), m = 1, \ldots, M, \ \frac{\partial q_t}{\partial \boldsymbol{u}_{t,k}^{(l-1)}} = (\boldsymbol{Y}_3^{(l)})^{\mathsf{T}} \frac{\partial q_t}{\partial \boldsymbol{u}_{t,k}^{(l)}} \odot \sigma'\left(\boldsymbol{Q}_{t,k}^{(l)}\right), k = 1, \ldots, K, \tag{13b}$$

$$\boldsymbol{P}_{t,m}^{(l)} \triangleq \boldsymbol{V}_3^{(l)} \boldsymbol{b}_{t,m}^{(l-1)} + \frac{1}{K} \sum_{k=1}^{K} (\boldsymbol{V}_1^{(l)} \boldsymbol{u}_{t,k}^{(l-1)} + \boldsymbol{V}_2^{(l)} \boldsymbol{e}_{t,m}^{k}), \ \boldsymbol{Q}_{t,k}^{(l)} \triangleq \boldsymbol{Y}_3^{(l)} \boldsymbol{u}_{t,k}^{(l-1)} + \frac{1}{M} \sum_{m=1}^{M} (\boldsymbol{Y}_1^{(l)} \boldsymbol{b}_{t,m}^{(l-1)} + \boldsymbol{Y}_2^{(l)} \boldsymbol{e}_{t,m}^{k}), \tag{13c}$$

where $\odot$ is element-wise product, Eq. (13b) is used for computing $\partial q_t / \partial \boldsymbol{V}_1^{(l-1)}$ and $\partial q_t / \partial \boldsymbol{Y}_1^{(l-1)}$, $\boldsymbol{P}_{t,m}^{(l)}$ and $\boldsymbol{Q}_{t,k}^{(l)}$ in (13c) are defined for simplifying the formula, and $\sigma'(\cdot)$ denotes the derivative of the activation function $\sigma(\cdot)$. The gradients $\partial q_t / \partial \boldsymbol{V}_2^{(l)}$, $\partial q_t / \partial \boldsymbol{V}_3^{(l)}$, $\partial q_t / \partial \boldsymbol{Y}_2^{(l)}$, $\partial q_t / \partial \boldsymbol{Y}_3^{(l)}$ can be derived similarly.

From (13a), we can derive that $M(K+2)d^2 + M(3K+1)d \sim \mathcal{O}(MKd^2)$ multiplications are required to compute $\partial q_t / \partial \boldsymbol{V}_1^{(l)}$ with one experience. The number of multiplications for computing $\partial q_t / \partial \boldsymbol{V}_1^{(l)}$ with a batch of B experiences is $\mathcal{O}(BMKd^2)$. After regular manipulations, the numbers of multiplications for computing $\partial q_t / \partial \boldsymbol{V}_2^{(l)}, \partial q_t / \partial \boldsymbol{V}_3^{(l)}, \partial q_t / \partial \boldsymbol{Y}_1^{(l)}$, $\partial q_t / \partial \boldsymbol{Y}_2^{(l)}, \partial q_t / \partial \boldsymbol{Y}_3^{(l)}$, $\partial q_t / \partial \boldsymbol{b}_{t,m}^{(l-1)}, m = 1, \ldots, M$ and $\partial q_t / \partial \boldsymbol{u}_{t,k}^{(l-1)}, k = 1, \ldots, K$ can be approximated as $\mathcal{O}(BMKd^2)$. Then, the number of multiplications for computing gradients in each layer is $(M + K + 6) \times \mathcal{O}(BMKd^2) \sim \mathcal{O}(BMK^2d^2 + BM^2Kd^2)$, and the time complexity for updating the model parameters of the actor or critic network in each time step is $\mathcal{O}[(MK^2 + M^2K)BL_1d^2]$. The time complexity for updating the model parameters of GNN-DDPG once is $2 \times \mathcal{O}[(MK^2 + M^2K)BL_1d^2] \sim \mathcal{O}[(MK^2 + M^2K)BL_1d^2]$.

Let c_1 denote the number of time steps required by training the actor and critic networks to achieve an expected performance. Then, the time complexity for training GNN-DDPG is $\mathcal{O}[c_1(MK^2+M^2K)BL_1d^2]$, where the time complexity in forward propagation is ignored since it is much lower than the complexity of computing gradients.

5.2 Time complexity of training the FNN-DDPG

The dimension of the hidden output of each layer is D. Suppose that both the actor and critic networks have L_2 layers. The output of the lth layer in both actor and critic networks can be expressed as $\boldsymbol{y}^{(l)} = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)})$, where $\boldsymbol{y}^{(l)}$ is the hidden output in the lth layer, $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are, respectively, the weight matrix and bias in the lth layer.

The gradients in the lth layer can be derived as

$$\frac{\partial q_t}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial q_t}{\partial \boldsymbol{y}^{(l)}} \odot \sigma'(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}) \cdot (\boldsymbol{y}^{(l-1)})^{\mathsf{T}}, \quad \frac{\partial q_t}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial q_t}{\partial \boldsymbol{y}^{(l)}} \odot \sigma'(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}),$$
$$\frac{\partial q_t}{\partial \boldsymbol{y}^{(l-1)}} = \boldsymbol{W}^{(l)\mathsf{T}} \cdot \frac{\partial q_t}{\partial \boldsymbol{y}^{(l)}} \odot \sigma'(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}),$$

from which we can obtain that computing the gradients of q_t with respect to $\boldsymbol{W}^{(l)}$, $\boldsymbol{b}^{(l)}$, and $\boldsymbol{y}^{(l-1)}$ with one experience requires $\mathcal{O}(D^2)$ multiplications.

Let c_1 denote the number of time steps required by training the actor and critic networks to achieve an expected performance. Then, the time complexity for training FNN-DDPG is $\mathcal{O}(c_2BL_2D^2)$, where the time complexity of forward propagation is ignored.

From the expressions of big $\mathcal{O}$, it seems that the time complexity of GNN-DDPG is higher than FNN-DDPG, which however is not true. This is because the values of c_1, L_1, and d of GNN-DDPG are, respectively, smaller than c_2, L_2, and D of FNN-DDPG, as to be seen in Section 6.

6 Simulation results

In this section, we evaluate the performance of predictive power allocation learned by the DDPG algorithms. To show the impact of satisfying the desired permutation property, we compare the proposed method (with legend "GNN-DDPG") designed in Subsections 3.2 and 4.3 that optimizes predictive rate allocation in each frame using the graph DDPG algorithm and optimizes the power allocation in each time slot using $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$ with the following methods.

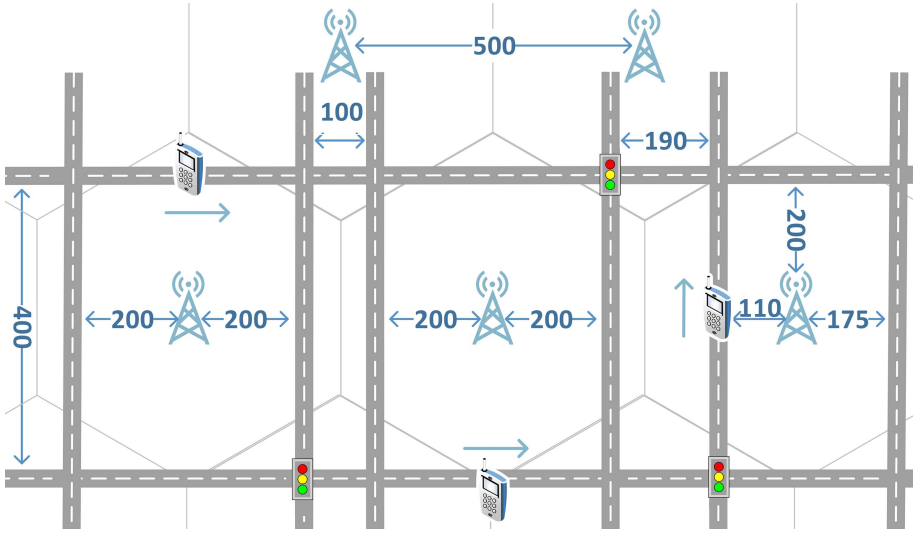

Figure 3 (Color online) Simulation scenario, where K users move along roads in five cells, and the integers indicate distances in meters.

Optimal. This method uses the numerical algorithm in [6] to find the optimal power allocation from P1 by assuming perfect future large-scale channel gains, which can serve as the performance upper bound for all the learning-based solutions.

Non-predictive. This is a traditional method of transmission without using predicted information, which maintains a constant average rate for each user in each frame [41]. To satisfy the QoS constraints, the average rate for the kth user in the tth frame is set as $\bar{R}_t^k = d_{n_t+1}^k/(N_f\Delta T)$, where n_t is the index of the segment played in the tth frame, and $d_{n_t+1}^k$ is the number of bits of the (n_t+1)th segment in the video to be played by the kth user. For a fair comparison, the power allocation in each time slot is obtained with $\pi_{\mathrm{P3}}(\cdot;\boldsymbol{\phi})$.

LSTM & **Optimize.** This method solves problem P2 by treating the predicted large-scale channels as the real channels [8]. The large-scale channel gains in future $(N_v-1)N_f$ frames are first predicted with a long short-term memory network (LSTM), with which the future powers are obtained with the numerical optimization algorithm in [6].

GNN-DDPG+FNN. This DRL algorithm differs from "GNN-DDPG" only in the output layers, which are FNNs analogous to the graph DQN in [24]. As a result, the actor and critic networks do not satisfy the PE/PI properties in (9).

FNN-DDPG. This is the original DDPG algorithm in [49], as mentioned in Section 5. The actor and critic networks do not satisfy any permutation properties. Since this algorithm uses matrix-based MDP in Subsection 2.2, the policy function and action-value function to be learned are respectively $\boldsymbol{a}_t = \mu'(\boldsymbol{S}_t)$ and $q_t = Q'([\boldsymbol{S}_t, \boldsymbol{a}_t])$, which are different from the two functions defined on graphs as introduced in Subsection 4.2.2, where $\mu'(\cdot)$ and $Q'(\cdot)$ are two functions.

PE/PINN-DDPG. The only difference of this DRL algorithm from "FNN-DDPG" is that the two FNNs are replaced by PE/PINNs that are designed with the following permutation properties using the method in [51]. The permutation properties of the policy function $\mu'(\cdot)$ and action-value function $Q'(\cdot)$ are respectively $\boldsymbol{\Pi a}_t = \mu'(\boldsymbol{\Pi S}_t)$ and $q_t = Q'(\boldsymbol{\Pi}[\boldsymbol{S}_t, \boldsymbol{a}_t])$, where $\boldsymbol{\Pi}$ represents the permutation of indices of users. Hence, the permutation prior is exploited but the topology prior is not harnessed.

6.1 Simulation setup and fine-tuned hyper-parameters

Consider a scenario shown in Figure 3, where K users move along roads across cells. The trajectory of each user is generated with Bonnmotion software [52] (the trajectory follows the Gauss-Markov model), which is commonly applied for vehicular networks. The initial velocity of the users is set as 16 m/s, and the minimal and maximal velocities are 12 and 20 m/s, respectively. Users may encounter traffic lights, which are red or green with 50% probability. A user stops for 0–30 s if the traffic light is red.

The playback duration of each video and each segment are 150 and 10 s, respectively. Each segment is with the size of 8 Mbps [53]. Each time frame is with duration of $\Delta T = 1$ s, and each time slot is with duration of $\tau = 1$ ms (i.e., $N_s = 1000$).

Table 1 Hyper-parameters of each DRL algorithm.

		FNN-DDPG	GNN-DDPG	GNN-DDPG+FNN	PE/PINN-DDPG
Learning rate of actor network		0.001	0.001	0.001	0.001
Learning rate of critic network		0.0001	0.0001	0.001	0.0001
Number of neurons in each layer and number of layers	$K=2$	[200, 200]	[128, 128]	[128, 128], [500]	[200, 200]
	$K=5$	[400, 400, 400]	[128, 128]	[128, 128], [500]	[400, 400]
	$K=8$	[800, 800, 800]	[128, 128]	[128, 128], [500]	[800, 800, 800]

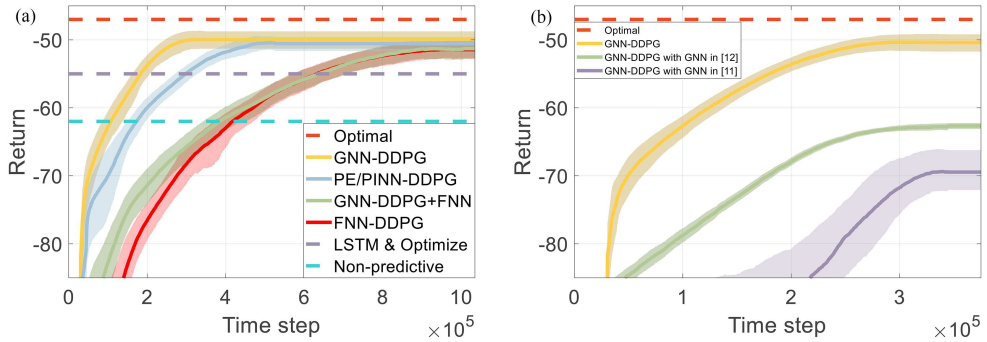

Figure 4 (Color online) Learning curves, averaged over 10 times of training and 100 successive episodes, $K=5$. (a) Impact of two relational priors and channel prediction; (b) impact of GNNs with unmatched permutation properties.

The maximal transmit power and the number of antennas of each DU are 200 W and 64, respectively [54]. The bandwidth assigned for each user is 2 MHz. The path loss is modeled as $13.54+39.08\log_{10}(d)+20\log_{10}(f_c)$ in dB, where d is the distance between user and DU in meters, f_c is the carrier frequency and is set as 3.5 GHz [55]. The small-scale channels follow Rayleigh fading. The noise spectral density is -174 dBm/Hz.

We set $T=2$ in the state, and $\lambda=0.1$ in the reward. The discount factor $\gamma=1$. The replay memory size is 10^6, and the mini-batch size for gradient descent is 1024. The noise term for exploration follows Gaussian distribution with zero mean and variance decreased linearly from 0.1 to 0. We use the Adam algorithm [56] to optimize the model parameters. For "GNN-DDPG", we set $\lambda_1=1$ and $\lambda_2=0$ in the critic network. Other fine-tuned hyper-parameters are listed in Table 1.

6.2 Performance evaluation

In Figure 4, we show the learning curves of the DRL algorithms. The shadow near each curve is the maximal and minimal returns of the 10 times of independent training. For each time of training, both the trajectories of users and the weight matrices in the DNNs are randomly generated.

As shown in Figure 4(a), "GNN-DDPG" converges faster than all the other algorithms. Due to not satisfying the permutation properties, "GNN-DDPG+FNN" converges much slower than "GNN-DDPG" and only converges slightly faster than "FNN-DDPG". Since "PE/PINN-DDPG" does not exploit the topology prior, it converges slower than "GNN-DDPG". Due to not using future information, "Non-predictive" is inferior to all the DRL algorithms that can predict channels implicitly by including historical channels in the states. Due to the large prediction errors in the multi-step prediction, "LSTM & Optimize" is also inferior to all the DRL algorithms that are with an implicit single-step prediction. After convergence, none of the DRL algorithms can achieve the upper bound provided by "Optimal". This is due to the randomly generated trajectories, including the random initial locations of users, random velocities, random turn in crossings in the road, and random stopping time for red traffic lights that are unpredictable[1)].

We also compare with other DDPG algorithms in Figure 4(b), where the actor and critic networks are the GNNs designed in [11, 12], which perform worse due to the unmatched permutation properties with (9a) and (9b).

In Table 2, we compare the training complexity of several DDPG algorithms to achieve an expected performance, where the time complexity is evaluated as the running time in hours, which is another widely used metric [38]. The results are obtained on a computer with Intel CoreTM i9-10940X CPU (3.30 GHz) and NVIDIA GeForce RTX 3080 GPU. The expected performance is set as 90% of the upper bound.

1) Our results show that the DRL algorithms can achieve the upper bound if the users move in a constant velocity, which is not provided due to the space limitation.

Table 2 Training complexity comparison.

	Training complexity	FNN-DDPG	GNN-DDPG	GNN-DDPG+FNN	PE/PINN-DDPG
$K=2$	Sample complexity	0.21M	0.09M	0.21M	0.18M
	Time complexity	1.85	0.91	3.48	0.90
	Space complexity	178k	401k	914k	90k
$K=5$	Sample complexity	0.90M	0.09M	0.84M	0.24M
	Time complexity	8.67	0.96	11.53	1.68
	Space complexity	1369k	401k	1686k	60k
$K=8$	Sample complexity	1.50M	0.12M	1.08M	0.60M
	Time complexity	18.27	1.29	17.10	9.28
	Space complexity	3092k	401k	2456k	169k

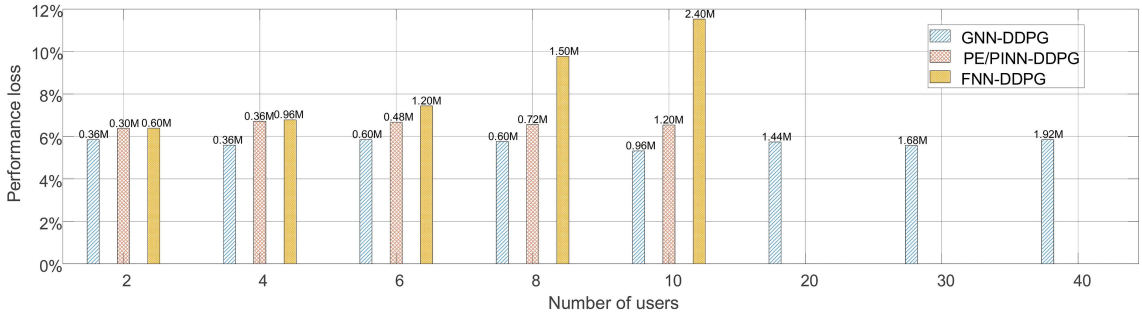

Figure 5 (Color online) Scalability of the DDPG algorithms, averaged over 10 times of training.

According to the definition of sample complexity for reinforcement learning, lower sample complexity indicates faster convergence.

We can see from Table 2 that the gain of exploiting relational priors in terms of reducing training complexity is large. When $K=8$, "GNN-DDPG" converges 12 times faster than "FNN-DDPG", and five times faster than "PE/PINN-DDPG". The running time for training "GNN-DDPG" is 1/14 of "FNN-DDPG", and 1/7 of "PE/PINN-DDPG". Recall that the required numbers of multiplications of training "GNN-DDPG" and "FNN-DDPG" to achieve an expected performance are, respectively, $\mathcal{O}[c_1(MK^2+M^2K)BL_1d^2]$ and $\mathcal{O}(c_2BL_2D^2)$. When $K=8$ and the expected performance is 90% of the upper bound, we can obtain from Tables 1 and 2 that $c_1=0.12$ M, $L_1=2$, and $d=128$ of "GNN-DDPG", $c_2=1.50$ M, $L_2=3$, and $D=800$ of "FNN-DDPG". For both "GNN-DDPG" and "FNN-DDPG", $M=5$ and $B=1024$. Then, the required numbers of multiplications of training "GNN-DDPG" and "FNN-DDPG" are, respectively, 2×10^{15} and 3×10^{15}. The ratio of the two numbers is much less than the ratio of the running time in the table. This is because the structure of GNNs allows parallel computing, which leads to shorter running time on GPU.

In Figure 5, we show the scalability of the DDPG algorithms, which are trained for each value of K. The definition of scalability that we use is the same as that in [57]; i.e., if a DNN well-trained in a small scale system can still perform well in a large scale system after re-training with acceptable training complexity, then the DNN is scalable. The results for different numbers of users are obtained after convergence (the numbers of required time steps are shown on the top of the bars). Since the performance upper bound and the return depend on channels and the number of users, to compare the algorithms in different setups, we consider the performance loss from the upper bound, defined as $|\frac{|\text{Return}_\text{c}|-\text{Energy}_\text{op}}{\text{Energy}_\text{op}}|$, where Return_c is the return achieved by the DDPG algorithms after convergence, and Energy_op is the energy achieved by "Optimal". Since training "FNN-DDPG" and "PE/PINN-DDPG" is too time-consuming, their performance losses for $K>10$ are not provided. We can see that the performance losses of "GNN-DDPG" and "PE/PINN-DDPG" remain unchanged when K increases (i.e., they are scalable), but "FNN-DDPG" performs much worse for more users even with significantly more time steps than "GNN-DDPG".

Reinforcement learning interacts with environment, which incurs signaling overhead between DUs and CU. In each time step, each DU sends the state vectors and the summation of consumed energy and penalty of the associated users to the CU in the uplink, and the CU sends the action vector to the DUs in the downlink. After gathering the state vectors of all users, the CU can obtain the adjacency and feature matrices using the method in Section 3. Hence, the signaling overhead in each time step of the matrix-based and graph-based MDPs is identical.

Table 3 Signaling overhead comparison, $K = 8$ and $M = 5$.

	FNN-DDPG	GNN-DDPG	GNN-DDPG+FNN	PE/PINN-DDPG
Signaling overhead (MB)	869.8	69.6	626.2	347.9

Since the signaling overhead of downlink is much lower than uplink, we only consider uplink. For K users when $T = 2$, the state matrix contains $(3M + 3) \cdot K$ decimals, and the action vector contains K decimals. Therefore, the DUs send $(3M + 4) \cdot K$ decimals to the CU in each time step. The signaling overhead for DDPG algorithms is $(3M + 4) \cdot K \cdot b_s \cdot T_{\text{total}}$, where T_{total} is the number of time steps for converging to an expected performance, and b_s is the number of bits to quantize each decimal. In Table 3, we compare the uplink signaling overhead of the DDPG algorithms in Mbyte (MB) to achieve the 90% performance of the upper bound, where $b_s = 32$ bits, $K = 8$, and $M = 5$. We can see that "GNN-DDPG" incurs much lower signaling overhead than other algorithms. By comparing with the sample complexity in Table 2 when $K = 8$, we can see that an algorithm with lower sample complexity needs lower signaling overhead. This indicates that faster convergence amounts to lower signaling overhead.

7 Conclusion

In this paper, we investigated how to harness two kinds of relational priors for reducing the training complexity of DRL algorithms. We provided a method for converting the matrix-based MDP into graph-based MDP, and proposed a graph DDPG algorithm, where the actor and critic networks can satisfy the permutation properties of the policy and action-value functions by designing the GNNs and the output layers. We derived the time complexity of GNN-based DDPG and FNN-based DDPG for training. Simulation results for the predictive resource allocation problem showed that the designed graph-based DDPG algorithm converges much faster and needs much lower time and space complexity than existing DDPG algorithms, especially for more users.

Acknowledgements This work was supported in part by National Key R&D Program of China (Grant No. 2022YFB2902002) and National Natural Science Foundation of China (Grant No. 62271024).

References

1 Alwarafy A, Abdallah M, Ciftler B S, et al. The frontiers of deep reinforcement learning for resource management in future wireless HetNets: techniques, challenges, and research directions. IEEE Open J Commun Soc, 2022, 3: 322–365
2 Huang K, Luo Z Z, Liang L, et al. Fast spectrum sharing in vehicular networks: a meta reinforcement learning approach. In: Proceedings of the IEEE Vehicular Technology Conference, London, 2022. 1–5
3 He Y, Wang Y W, Lin Q Z, et al. Meta-hierarchical reinforcement learning (MHRL)-based dynamic resource allocation for dynamic vehicular networks. IEEE Trans Veh Technol, 2022, 71: 3495–3506
4 Yang T J, Chen Y H, Emer J, et al. A method to estimate the energy consumption of deep neural networks. In: Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, 2017. 1916–1920
5 Battaglia P W, Hamrick J B, Bapst V, et al. Relational inductive biases, deep learning, and graph networks. 2018. ArXiv:1806.01261
6 She C Y, Yang C Y. Energy efficient resource allocation for hybrid services with future channel gains. IEEE Trans Green Commun Netw, 2020, 4: 165–179
7 Index C. Global mobile data traffic forecast update, 2017–2022. Cisco white paper. 2019. http://media.mediapost.com/uploads/CiscoForecast.pdf
8 Bui N, Widmer J. Data-driven evaluation of anticipatory networking in LTE networks. IEEE Trans Mobile Comput, 2018, 17: 2252–2265
9 Zhang C Z, Guo J, Yang C Y. When the gain of predictive resource allocation for content delivery is large? Sci China Inf Sci, 2023, 66: 222302
10 Liu D, Zhao J J, Yang C Y, et al. Accelerating deep reinforcement learning with the aid of partial model: energy-efficient predictive video streaming. IEEE Trans Wireless Commun, 2021, 20: 3734–3748
11 Eisen M, Ribeiro A. Optimal wireless resource allocation with random edge graph neural networks. IEEE Trans Signal Process, 2020, 68: 2977–2991
12 Shen Y F, Shi Y M, Zhang J, et al. Graph neural networks for scalable radio resource management: architecture design and theoretical analysis. IEEE J Sel Areas Commun, 2021, 39: 101–115
13 Lee M Y, Yu G D, Li G Y. Graph embedding-based wireless link scheduling with few training samples. IEEE Trans Wireless Commun, 2021, 20: 2282–2294
14 Jiang T, Cheng H V, Yu W. Learning to reflect and to beamform for intelligent reflecting surface with implicit channel estimation. IEEE J Sel Areas Commun, 2021, 39: 1931–1945
15 Guo J, Yang C Y. Learning power allocation for multi-cell-multi-user systems with heterogeneous graph neural networks. IEEE Trans Wireless Commun, 2022, 21: 884–897
16 Chen T R, Zhang X R, You M L, et al. A GNN-based supervised learning framework for resource allocation in wireless IoT networks. IEEE Internet Things J, 2022, 9: 1712–1724
17 Kosasih A, Onasis V, Miloslavskaya V, et al. Graph neural network aided MU-MIMO detectors. IEEE J Sel Areas Commun, 2022, 40: 2540–2555
18 He S W, Xiong S W, Zhang W, et al. GBLinks: GNN-based beam selection and link activation for ultra-dense D2D mmWave networks. IEEE Trans Commun, 2022, 70: 3451–3466
19 Li Y H, Lu Y, Zhang R C, et al. Deep learning for energy efficient beamforming in MU-MISO networks: a GAT-based approach. IEEE Wireless Commun Lett, 2023, 12: 1264–1268
20 Kim J, Lee H, Hong S E, et al. A bipartite graph neural network approach for scalable beamforming optimization. IEEE Trans Wireless Commun, 2023, 22: 333–347

21 Li B N, Verma G, Segarra S. Graph-based algorithm unfolding for energy-aware power allocation in wireless networks. IEEE Trans Wireless Commun, 2023, 22: 1359–1373
22 Liu S J, Guo J, Yang C Y. Multidimensional graph neural networks for wireless communications. IEEE Trans Wireless Commun, 2024, 23: 3057–3073
23 Zhao B C, Guo J, Yang C Y. Understanding the performance of learning precoding policies with graph and convolutional neural networks. IEEE Trans Commun, 2024, 72: 5657–5673
24 Nakashima K, Kamiya S, Ohtsu K, et al. Deep reinforcement learning-based channel allocation for wireless LANs with graph convolutional networks. IEEE Access, 2020, 8: 31823–31834
25 Xu C H, Song W. An adaptive data uploading scheme for mobile crowdsensing via deep reinforcement learning with graph neural network. IEEE Internet Things J, 2022, 9: 18064–18078
26 Sun Z C, Mo Y J, Yu C. Graph-reinforcement-learning-based task offloading for multiaccess edge computing. IEEE Internet Things J, 2023, 10: 3138–3150
27 Liu Z, Huang L F, Gao Z B, et al. GA-DRL: graph neural network-augmented deep reinforcement learning for DAG task scheduling over dynamic vehicular clouds. IEEE Trans Netw Serv Manage, 2024, 21: 4226–4242
28 Zhang X C, Zhao H T, Xiong J, et al. Decentralized routing and radio resource allocation in wireless ad hoc networks via graph reinforcement learning. IEEE Trans Cogn Commun Netw, 2024, 10: 1146–1159
29 Wu J J, Sun C J, Yang C Y. On the size generalizibility of graph neural networks for learning resource allocation. Sci China Inf Sci, 2024, 67: 142301
30 Li Y H, Lu Y, Ai B, et al. GNN-based beamforming for sum-rate maximization in MU-MISO networks. IEEE Trans Wireless Commun, 2024, 23: 9251–9264
31 Wu Z H, Pan S R, Chen F W, et al. A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst, 2021, 32: 4–24
32 Luo J, Chen Q B, Tang L, et al. Adaptive resource allocation considering power-consumption outage: a deep reinforcement learning approach. IEEE Trans Veh Technol, 2023, 72: 8111–8116
33 Peng T, Guo Y C, Wang Y C, et al. An interference-oriented 5G radio resource allocation framework for ultradense networks. IEEE Internet Things J, 2022, 9: 22618–22630
34 Xiao Y, Song Y Q, Liu J. Collaborative multi-agent deep reinforcement learning for energy-efficient resource allocation in heterogeneous mobile edge computing networks. IEEE Trans Wireless Commun, 2024, 23: 6653–6668
35 Zhang Y C, Lu Y, Zhang R C, et al. Deep reinforcement learning for secrecy energy efficiency maximization in RIS-Assisted networks. IEEE Trans Veh Technol, 2023, 72: 12413–12418
36 Wang Y, Shang F J, Lei J J. Reliability optimization for channel resource allocation in multihop wireless network: a multigranularity deep reinforcement learning approach. IEEE Internet Things J, 2022, 9: 19971–19987
37 Wang D, Li R, Huang C, et al. User association and power allocation for user-centric smart-duplex networks via tree-structured deep reinforcement learning. IEEE Internet Things J, 2023, 10: 20216–20229
38 Blondel V D, Tsitsiklis J N. A survey of computational complexity results in systems and control. Automatica, 2000, 36: 1249–1274
39 Guo Z Y, Chen Z Y, Liu P, et al. Multi-agent reinforcement learning-based distributed channel access for next generation wireless networks. IEEE J Sel Areas Commun, 2022, 40: 1587–1599
40 Tuong V D, Noh W, Cho S. Sparse CNN and deep reinforcement learning-based D2D scheduling in UAV-assisted industrial IoT networks. IEEE Trans Ind Inf, 2024, 20: 213–223
41 Rosario J M D, Fox G. Constant bit rate network transmission of variable bit rate continuous media in video-on-demand servers. Multimed Tools Appl, 1996, 2: 215–232
42 Wang S Y, Bi S Z, Zhang Y J A. Deep reinforcement learning with communication transformer for adaptive live streaming in wireless edge networks. IEEE J Sel Areas Commun, 2022, 40: 308–322
43 Lan Q, Lv B J, Wang R, et al. Adaptive video streaming for massive MIMO networks via approximate MDP and reinforcement learning. IEEE Trans Wireless Commun, 2020, 19: 5716–5731
44 Lu Z, de Veciana G. Optimizing stored video delivery for wireless networks: the value of knowing the future. IEEE Trans Multimedia, 2019, 21: 197–210
45 Atawia R, Hassanein H S, Ali N A, et al. Utilization of stochastic modeling for green predictive video delivery under network uncertainties. IEEE Trans Green Commun Netw, 2018, 2: 556–569
46 Sutton R S, Barto A G. Reinforcement Learning: An Introduction. Cambridge: MIT Press, 2018
47 Fathy H K, Bortoff S A, Copeland G S, et al. Nested optimization of an elevator and its gain-scheduled LQG controller. In: Proceedings of the ASME International Mechanical Engineering Congress and Exposition, Louisiana, 2002. 119–126
48 Dalal G, Dvijotham K, Vecerik M, et al. Safe exploration in continuous action spaces. 2018. ArXiv:1801.08757
49 Lillicrap L P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning. In: Proceedings of the International Conference on Learning Representations, San Juan, 2016. 1–14
50 Kirkpatrick D. Determining graph properties from matrix representations. In: Proceedings of the Annual ACM Symposium on Theory of Computing, Washington, 1974. 84–90
51 Ravanbakhsh S, Schneider J, Poczos B. Equivariance through parameter-sharing. In: Proceedings of the International Conference on Machine Learning, Sydney, 2017. 2892–2901
52 Aschenbruck N, Ernst R, Gerhards-Padilla E, et al. BonnMotion: a mobility scenario generation and analysis tool. In: Proceedings of the International ICST Conference on Simulation Tools and Techniques, Malaga, 2010. 1–10
53 Youtube. Recommended upload encoding setting. 2021. https://support.google.com/youtube/answer/1722171
54 Tan R M, Shi Y, Wu T, et al. Electromagnetic field safety compliance assessments for 5G wireless networks. In: Proceedings of the IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity, Reno, 2020. 659–662
55 ESTI. 5G; Study on channel model for frequencies from 0.5 to 100 GHz. TR 28.901 version 16.1.0 Release 16. https://www.etsi.org/deliver/etsi_tr/138900_138999/138901/16.01.00_60/tr_138901v160100p.pdf
56 Kingma D P, Ba J. Adam: a method for stochastic optimization. In: Proceedings of the International Conference on Learning Representations, San Diego, 2015. 1–15
57 Bondi A B. Characteristics of scalability and their impact on performance. In: Proceedings of the International Workshop on Software and Performance, Ottawa, 2000. 195–203

Appendix A Proof of Proposition 1

Let $\pi^*(\cdot,\cdot,\cdot)$ denote the optimal solution obtained from P3, which minimizes the total average energy consumption to achieve a given average data rate $\bar{R}_t^k$ under the maximal power constraint of each DU. In what follows, we prove that $\pi^*(\cdot,\cdot,\cdot)$ can minimize the objective function of P1 and satisfy the two constraints in P1.

(a) Prove that $\pi^*(\cdot,\cdot,\cdot)$ can minimize the objective function of P1. Let $\pi'(\cdot,\cdot,\cdot)$ be an arbitrary solution different from $\pi^*(\cdot,\cdot,\cdot)$, which consumes the average energy of $\bar{E}_t^m$ to achieve $\bar{R}_t^k$. Then, to achieve the same average rate $\bar{R}_t^k$, the average

energy consumed by $\pi^*(\cdot,\cdot,\cdot)$ will be less than $\bar{E}_t^m$. Since $\pi^*(\cdot,\cdot,\cdot)$ can use less average power than arbitrary policies to achieve the same average rate, it can minimize the objective function of P2, i.e., $\sum_{t=1}^{(N_v-1)N_f}\sum_{m=1}^{M}\bar{E}_t^m$.

The objective function of P2 is the same as that of P1, i.e.,

$$\begin{aligned}\sum_{t=1}^{(N_v-1)N_f}\sum_{m=1}^{M}\bar{E}_t^m &= \sum_{t=1}^{(N_v-1)N_f}\sum_{m=1}^{M}\mathbb{E}_g\left[\sum_{j=1}^{N_s}\sum_{k\in\mathcal{K}_{t,m}}\tau p_{tj}^k\right]=\mathbb{E}_g\left[\sum_{m=1}^{M}\sum_{k\in\mathcal{K}_{t,m}}\sum_{t=1}^{(N_v-1)N_f}\sum_{j=1}^{N_s}\tau p_{tj}^k\right]\\ &= \mathbb{E}_g\left[\sum_{k=1}^{K}\sum_{t=1}^{(N_v-1)N_f}\sum_{j=1}^{N_s}\tau p_{tj}^k\right].\end{aligned}$$

Hence, $\pi^*(\cdot,\cdot,\cdot)$ can minimize the objective function of P1.

(b) Prove that $\pi^*(\cdot,\cdot,\cdot)$ can satisfy the two constraints in P1. Since P1 and P3 are with the same power constraints and $\pi^*(\cdot,\cdot,\cdot)$ can satisfy the power constraints in P3, $\pi^*(\cdot,\cdot,\cdot)$ can satisfy the power constraints in P1.

Using $\pi^*(\cdot,\cdot,\cdot)$, the average rate optimized from P2 can be achieved, hence $\pi^*(\cdot,\cdot,\cdot)$ can satisfy the QoS constraints in P2. Since $\Delta T=\tau N_s$, we have $\sum_{j=1}^{N_s}\tau R_{tj}^k=\Delta T\sum_{j=1}^{N_s}\frac{R_{tj}^k}{N_s}$. Considering that $\tau\ll\Delta T$, the value of N_s is very large. Hence, $\sum_{j=1}^{N_s}\frac{R_{tj}^k}{N_s}\approx\bar{R}_t^k$ and the approximation is very accurate. This indicates that the QoS constraints in P2 are approximately the same as the constraints in P1; i.e., $\pi^*(\cdot,\cdot,\cdot)$ can satisfy the QoS constraints in P1.

Since $\pi^*(\cdot,\cdot,\cdot)$ can minimize the objective function of P1 meanwhile satisfy the constraints, it is the optimal solution of P1.

This completes the proof.

Appendix B Proof of Proposition 2

According to (10a), we can obtain $\boldsymbol{\Pi}_1\boldsymbol{a}_t = F_{\mathrm{a}}(\mathrm{GNN}_{\mathrm{a}}(\boldsymbol{\Pi}_1\boldsymbol{U}_t,\boldsymbol{\Pi}_1\boldsymbol{E}_t\boldsymbol{\Pi}_2^{\mathsf{T}},\tilde{\boldsymbol{\Pi}}_1\boldsymbol{A}_{\mathrm{PRA}}\tilde{\boldsymbol{\Pi}}_1^{\mathsf{T}}))$, which indicates that the actor network $F_{\mathrm{a}}(\mathrm{GNN}_{\mathrm{a}}(\cdot))$ satisfies the permutation property of policy function in (9a).

According to (10b), we can obtain $q = F_{\mathrm{c}}(\mathrm{GNN}_{\mathrm{c}}(\boldsymbol{\Pi}_1\widetilde{\boldsymbol{U}}_t,\boldsymbol{\Pi}_1\boldsymbol{E}_t\boldsymbol{\Pi}_2^{\mathsf{T}},\tilde{\boldsymbol{\Pi}}_1\boldsymbol{A}_{\mathrm{PRA}}\tilde{\boldsymbol{\Pi}}_1^{\mathsf{T}}))$, which indicates that the critic network $F_{\mathrm{c}}(\mathrm{GNN}_{\mathrm{c}}(\cdot))$ satisfies the permutation property of action-value function in (9b).

This completes the proof.