Virtual Network Embedding Based on Hierarchical Cooperative Multiagent Reinforcement Learning

Hyun-Kyo Lim, Ihsan Ullah, Ju-Bong Kim, and Youn-Hee Han, *Member, IEEE*

***Abstract*—Virtual network embedding (VNE) is a promising technique enabling 5G networks to satisfy the given requirements of each service via network virtualization (NV). For better performance of the embedding algorithm, it is necessary to automatically detect the network status and provide an optimal embedding decision. However, existing VNE algorithms disregard the long-term effect by focusing on selecting only one virtual network request (VNR) from the waiting queue, without considering all waiting virtual network requests concurrently. In this study, we propose a hierarchical cooperative multiagent reinforcement learning (MARL) algorithm to optimize the VNE problem by maximizing average revenue, minimizing average cost, and also improving the request acceptance ratio. The proposed algorithm applies two RL algorithms: 1) two-level hierarchical RL (HRL) to efficiently solve the problem by dividing it into subproblems and 2) multiagent-based cooperative RL to improve algorithm performance through the cooperation of multiple agents. In order to evaluate and analyze the proposed scheme from the long-term perspective, four performance parameters are evaluated: 1) revenue; 2) cost; 3) revenue-to-cost ratio; and 4) acceptance ratio. The simulation results demonstrate that the proposed VNE algorithm based on hierarchical and MARL outperforms the existing RL-based approaches.**

***Index Terms*—Hierarchical reinforcement learning (HRL), multiagent reinforcement learning (MARL), virtual network embedding (VNE).**

I. Introduction

IN RECENT years, the commercialization of 5G networks has had a significant impact on the general network management systems. In particular, the rapid growth of the IoT device market has led to significant challenges for current network infrastructures in meeting the increasing resource demands [1]. Due to limited network resources, it is impossible to meet the requirements of all services offered by service providers (SPs) or expand the infrastructure resources based on user needs. Moreover, the current network system architecture is rigid and unyielding to change, which makes it challenging to incorporate emerging technologies and services due to network ossification [2].

Network slicing (NS) and network virtualization (NV) technology is a potential solution to the problem of network ossification [3], [4]. It partitions a static physical network and configures custom virtual networks to lease resources from the Internet provider (InP) according to the needs of the end users. Efficient SP and InP management techniques increase the effectiveness of the substrate network and boost revenue for both providers. In the context of 5G networks, virtual network embedding (VNE) is important, especially with regard to NS and NV [5]. As 5G technology enables dynamic NS and NV, VNE plays a pivotal role in efficiently allocating virtual resources across diverse slices for quality service delivery. VNE refers to the process of efficiently allocating substrate network resources to meet the constraints of virtual network requests (VNRs) with respect to virtual nodes and links. network service providers (NSPs) are responsible for mapping VNRs onto the underlying substrate network.

However, the VNE problem is NP-hard due to the extensive search space, even with single VNRs or single virtual nodes, and is further compounded by the NP-hard characteristic of mapping VNRs onto substrate networks [6]. So, many VNE algorithms exist to facilitate the efficient allocation of network resources for embedding VNRs into the underlying substrate network. Several heuristic-based algorithms [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] and bio-inspired algorithms [16], [17], [18], [19], [20] have been proposed to solve VNE problems where several subgoals are sometimes incorporated in different ways. While heuristic-based algorithms often use greedy optimization and may not yield optimal solutions, bio-inspired methods, such as ant colony optimization, genetic algorithms, and particle swarm optimization, have shown effectiveness in solving VNE problems. However, these bio-inspired methods require significant solution search time, especially when dealing with large or dynamic networks.

In recent years, several intelligent networks have been proposed by applying machine learning techniques to network management problems. One of such techniques is reinforcement learning (RL) [21], [22]. An RL agent learns to find the optimal VNE solution through interaction with the network environment, and it obtains a policy model that maximizes a predefined reward function. Several RL algorithms, including simple Q-learning, deep Q-learning (DQN), advantage actor–critic (A2C), asynchronous advantage actor–critic (A3C), proximal policy optimization (PPO), and deep

Manuscript received 13 July 2023; revised 8 September 2023; accepted 21 September 2023. Date of publication 26 September 2023; date of current version 21 February 2024. This work was supported by two Basic Science Research Programs through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2023R1A2C1003143 and Grant NRF-2018R1A6A1A03025526. *(Corresponding author: Youn-Hee Han.)*

Hyun-Kyo Lim, Ju-Bong Kim, and Youn-Hee Han are with the Future Convergence Engineering, Korea University of Technology and Education, Cheonan 31253, South Korea (e-mail: glenn89@koreatech.ac.kr; rlawnqhd@koreatech.ac.kr; yhhan@koreatech.ac.kr).

Ihsan Ullah is with the Advanced Technology Research Center, Korea University of Technology and Education, Cheonan 31253, South Korea (e-mail: ihsan@koreatech.ac.kr).

Digital Object Identifier 10.1109/JIOT.2023.3319542

deterministic policy gradient (DDPG), have been utilized to improve the performance of VNE algorithms [23], [24], [25], [26], [27], [28], [29], [30], [31], [32]. Most existing methods solve the VNE problem by first embedding all nodes without considering link embedding information. However, such approaches can lead to suboptimal link embeddings due to decisions made without explicitly considering the requirements and characteristics of the links. So, recently, graph neural network models-based RL methods have been introduced to address these limitations by considering both nodes and links information within the substrate network topology, enabling more effective node embedding [33], [34], [35], [36], [37]. Moreover, the existing RL methods to the VNE problem often overlook the long-term perspective of revenue, cost, and acceptance ratio. To address these issues, a hierarchical RL (HRL)-based technique [37] has been proposed. The HRL-based technique involves decomposing the VNE problem into smaller subproblems and learning policies for each subproblem. By using HRL, the technique tries to optimize revenue, reduce cost, and improve acceptance ratio over a longer time horizon. However, a limitation of the technique is that it assigns only one VNR per time step throughout the entire RL time horizon, and does not consider a group of VNRs and their resource demand systematically. This may result in suboptimal resource allocation and placement, leading to lower revenue and acceptance ratio in the long-term perspective.

In the context of VNE problem, it is common to have multiple VNRs waiting for resources simultaneously. Considering all waiting VNRs concurrently is crucial for RL algorithms to achieve optimal resource allocation and VNR placement. It ensures that all available resources are utilized efficiently, which can result in improved revenue and acceptance ratio over a longer time horizon. The decision-making process involved in selecting which VNRs to accept or reject during embedding is a complex and challenging problem. To overcome this challenge, a novel coordinated approach is required to optimize the performance of VNE.

To address the complex problem of selection in multiple VNRs, a multiagent RL (MARL) approach can be employed. One potential approach is to create a logical link between each VNR and an agent, allowing for cooperative decision making among agents. This approach can help to optimize VNR placement and resource allocation, resulting in improved revenue and acceptance ratio. In particular, MARL has gained significant attention for network resource allocation recently. Most of the existing research applying MARL has primarily focused on resource allocation in wireless environments [38], [39]. However, designing an effective MARL algorithm for VNE requires addressing how to facilitate efficient cooperation among agents. Particularly, with the advancement of 5G networks and the rapid growth of the IoT device market, there is a growing demand for a sophisticated and efficient algorithm that can autonomously embed VNRs to support diverse services. The automated VNE algorithm should have the capacity to concurrently handle multiple VNRs, enhance the revenue and acceptance ratio for NSP, and efficiently embed network resources at a reduced cost for SP. This study proposes a hierarchical and cooperative solution for VNE problem by incorporating HRL with MARL, named HCMARL-VNE. It employs HRL for efficient long-term exploration and MARL to accelerate VNE solutions via agent collaboration. To achieve an optimal link embedding solution, our approach also generates an augmented graph that encompasses the substrate network, virtual network nodes, and link information. GCNs model [33] is employed to extract network features and information by identifying the relationships between each node and link. Typical RL methods often require much exploration to solve problems, which can be computationally expensive. In contrast, HRL can break down problems into smaller and more manageable subproblems.

The proposed HRL algorithm divides the embedding task into two levels. At the high level (HL), MARL is applied to evaluate the VNRs in the waiting queue. The evaluation determines whether each waiting VNR should be embedded immediately or postponed for later embedding based on their long-term revenue potential. The collaboration of multiple agents in MARL leads to the maximization of long-term revenue. At the low level (LL), single-agent RL is used to select the best substrate node for embedding the virtual nodes of the VNRs, that have been chosen by the HL, based on the short-term embedding cost as well as the short-term revenue.

The main contributions of this study are summarized as follows.

1) To the best of our knowledge, this is the first study that MARL has been applied to the VNE problem to promote collaboration between individual agents and to improve the overall VNE performance.
2) The proposed algorithm divides the VNE problem into subproblems and utilizes HRL to enable agents at different levels to focus on enhancing their performance.
3) To automatically extract features, the proposed algorithm incorporates substrate and virtual network information into a newly defined augmented graph. The GCN is then employed to identify relationships between nodes and links in the network.

The remainder of this article is organized as follows. Section II describes the research status of RL-based VNE algorithms and the preliminaries of our work. In Section III, the substrate network and VNR models are described and our VNE problem is formalized. Section IV describes the details of the proposed algorithm. In Section V, the performance evaluation and comparison results are presented. Finally, the conclusions and future work are described in Section VI.

II. Related Work

A. Heuristic and Bio-Inspired Algorithms for VNE

In [7], the PageRank algorithm [40] is referenced to measure the relative importance of nodes by considering the topological attributes of substrate network components. The topology-aware node ranking is a method to measure the relative importance of nodes in a network based on their resource and topological attributes. It uses a Markov random walk model to compute the node rank, which reflects the CPU and bandwidth resources of the node and its neighbors, as well as the connectivity between them. In [14], node weight is

determined by factoring in node degree within the network topology as well as the available resources at each substrate node. An ego-network is then established around the substrate node with the highest weight, followed by link mapping where neighboring nodes meeting the VNR's demand constraint are selected. In [17], a genetic algorithm employing parallel computation and a novel fitness function is employed to discover nearly optimal solutions for the link mapping phase. Initially, a path pool is generated using the k-shortest path algorithm, considering only the substrate nodes complying with the VNR's virtual node resource constraints. Subsequently, the genetic algorithm undergoes iterative evaluation, selection, crossover, and mutation processes to ascertain a suitable path for link mapping. However, many existing heuristics and bio-inspired algorithms often yield suboptimal solutions due to their reliance on approximation techniques or simplified optimization strategies. This can lead to inefficient resource utilization and revenue loss for service providers. Therefore, recently, RL-based VNE algorithms have been proposed to overcome resource utilization, load balancing, revenue, cost and acceptance ratio.

B. Reinforcement Learning Algorithms for VNE

In [21] and [22], RL is used to generate the possibility of training abstractions on a high-dimensional state space; however, exploration tasks using sparse feedback remain a major challenge. To this end, Boltzmann search and Thomson sampling [41], [42] use a rudimentary level algorithm to outperform the ϵ-greedy algorithm. Recently, RL has gained significant attention for efficiently allocating limited resources of substrate networks with various VNRs [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [34], [35], [36].

Wang et al. [29] proposed an RL-based VNE algorithm using a pointer network model [43]. The algorithm employs the attention mechanism to focus on a specific substrate node, and the pointer network model comprises an encoder–decoder that takes the substrate node's features as input. The decoder outputs the probability of attention to the substrate nodes, which correspond to the virtual nodes of the VNR. The RL agent selects a substrate node by sampling the final outcome obtained by multiplying the attention probability with the masking information generated by a rule-based function. However, existing RL-based algorithms lack sufficient consideration for link embedding when the RL agent makes node selections.

The VNE algorithm introduced in [36] combines GCN and RL algorithms to solve the VNE problem. By utilizing GCN model to comprehend the relationships between nodes, node selection can be performed more efficiently. It combines the information in the substrate node and virtual node features for the input of GCN. Subsequently, the output of GCN is directly used to select the substrate node. The algorithm is developed based on the actor–critic concept, which means that the actor selects a substrate node depending on the current policy, and the critic evaluates the currently selected action, that is, the selected substrate node, whereby, the critic's state value is used to update the model.

Yan et al. [34] adopted an advanced deep RL technique, using the A3C policy gradient method [44], to solve the VNE problem. To speed up the training procedure and generate the training experience more efficiently, they trained the policy-generation algorithm using A3C. To extract spatial features from network information (raw state) more efficiently, they designed a 3-ordered layer GCN inside the training agent. While using GCN for node embedding takes into account link embedding as well, it exists an issue where the dimension of the state–action space that a single agent needs to consider becomes too large.

In [37], the presented solution for the complex VNE problem employs HRL and GCN. While GCN is used for feature extraction based on node and edge relationships, the approach involves an HL agent for ordering waiting VNRs and an LL RL agent for selecting suitable substrate nodes. However, this approach focuses solely on the VNRs at the current step without considering future rewards and revenue implications sufficiently. This lack of future-oriented decision making can lead to inefficient resource allocation, potentially resulting in reduced long-term revenue and acceptance rates. In addition, the most significant contribution of our scheme lies in utilizing a MARL framework to allow multiple agents to simultaneously choose actions in a cooperate way, and also in introducing a "postponing" action in the HL decision-making process, which deliberately defers the embedding of certain VNRs. They empower multiple agents for efficient VNR embedding, a critical factor for accommodating multiple VNRs concurrently, and its distributed architecture enhances scalability, making it ideal for complex real-world networks.

C. Hierarchical Reinforcement Learning

HRL algorithm is a new approach in the field of RL that aims to enhance the efficiency and effectiveness of learning in complex environments that provide sparse rewards and complex tasks [45], [46], [47]. It introduces a hierarchical structure in which multiple levels of agents or controllers work together to solve a problem.

Particularly in [45], an HRL-based approach is used to solve a problem by hierarchically dividing the main objective into subgoals, with HRL agent learning options to explore complex and sparse reward issues. This study proposes a novel approach for important concepts in RL: temporal abstraction. Temporal abstraction refers to the ability of an RL agent to reason and make decisions at different levels of time scales or levels of abstraction. The proposed method uses a hierarchical architecture where the agent learns at different levels of abstraction, allowing it to make decisions based on long-term goals and short-term actions.

D. Cooperative Multiagent Reinforcement Learning

Using MARL can provide several advantages compared to single-agent RL [48], [49], [50]. One advantage is that it allows multiple agents to learn and interact in the same environment, leading to better coordination and cooperation between agents. This can lead to more efficient use of substrate resources and better overall performance. Additionally, MARL

can handle complex and dynamic tasks, where the behavior of one agent may impact the behavior of other agents. By considering the actions of multiple agents, MARL can generate more robust and adaptive policies.

Challenges arise when using single-agent RL with a centralized method or fully decentralized method for multiple agents. The former faces difficulty in finding an optimal solution, while the latter struggles to learn the desired cooperative or competitive behavior through MARL. To address these issues, MARL methods have been developed to enable organic cooperation among agents for efficient actions. One popular approach is centralized training with decentralized execution (CTDE) [51]. Representative models of MARL include value-decomposition networks (VDN) [52] and Q-value mixing (QMIX) [53], based on state–action values. QMIX is a representative MARL algorithm designed to learn decentralized policies for cooperative tasks in partially observable settings. The core idea of QMIX is to generate a joint action-value function by mixing the individual agents' action-value functions (Q-values). To do this, QMIX configured with a hypernetwork [54] to generate weights and biases for the mixing network. The hypernetwork takes the global state as input and produces the weights and biases for the mixing network, which in turn combines the agents' Q-values to obtain the joint action-value (Q-total) through an average pooling process called readout.

In this study, multiple agents are created, equal to the number of waiting VNRs, with each VNR being associated with an agent. At each time step, these agents need to cooperatively decide whether to embed their corresponding VNRs or postpone them to optimize the overall network performance and resource allocation. The QMIX algorithm is employed in this context to learn a decentralized decision policy for the VNE problem. By using QMIX, the agents can learn to work together and make coordinated decisions on VNR embedding, taking into account the constraints and resource availability of the substrate network.

TABLE I
NOTATIONS OF NETWORK MODELING

Notation	Description
G^S	Substrate network
N^S	The set of substrate nodes
E^S	The set of substrate links
$c(n^s)$	Each substrate node's CPU capacity ($n^s \in N^S$)
$c(e^s)$	Each substrate link's bandwidth capacity ($e^s \in E^S$)
$\sigma(n^s)$	The nodal cost per CPU unit capacity
$\sigma(e^s)$	The link cost per bandwidth unit capacity
G^V	Virtual network request
N^V	The set of virtual nodes
E^V	The set of virtual links
$d(n^v)$	Each virtual node's CPU demand ($n^v \in N^V$)
$d(e^v)$	Each virtual link's bandwidth demand ($e^v \in E^V$)
d_t	VNR arrival time
d_d	Maximum delay time on the waiting queue for VNR
d_s	VNR service duration time

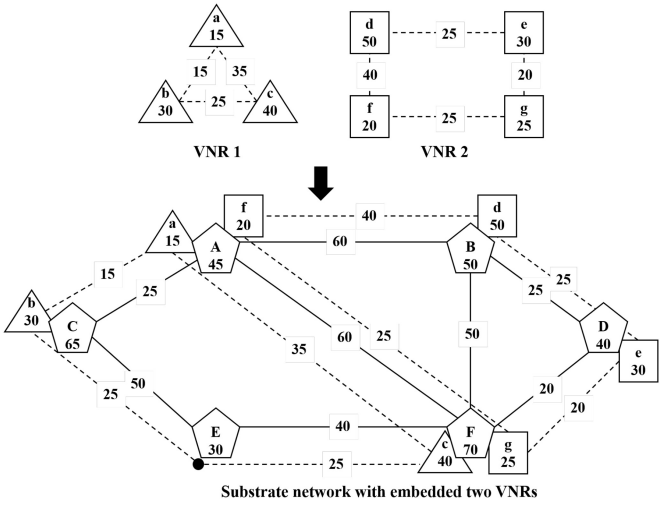


Fig. 1. Example of VNE. Triangles represent the nodes of VNR 1, squares represent the nodes of VNR 2, and pentagons represent substrate nodes. Triangles and squares attached near pentagons indicate that virtual nodes have been embedded on this substrate node, while solid lines close to dashed lines represent embedded virtual links.

III. FORMALIZATION OF THE VNE PROBLEM

In this section, we describe the system models for the VNE problem and provide the definition of the problem objective. The major notations used in the proposed system model are listed in Table I.

A. Substrate Network Modeling

The substrate network S is a physical network managed by an InP. It is typically modeled as an undirected graph $G^S = (N^S, E^S, c(\cdot), \sigma(\cdot))$, where N^S and E^S refer to the sets of substrate nodes and links, respectively. The substrate node $n^s (\in N^S)$ and link $e^s (\in E^S)$ are associated with the capacities of $c(n^s)$ and $c(e^s)$, respectively. $c(n^s)$ represents the maximum amount of computational resources, such as CPU or memory, that can be allocated to the node, and $c(e^s)$ the maximum amount of data that can be transmitted through the physical link, typically referred to as the bandwidth capacity. In addition, each substrate node and link has a per-unit capacity cost, denoted by $\sigma(n^s)$ and $\sigma(e^s)$, respectively. These costs represent the cost per unit of capacity that the InP or NSP must pay to provide and maintain the physical infrastructure.

B. Virtual Network Modeling

In VNE, a virtual network V is created by an SP to allocate network resources to the InP. The VNR is typically modeled as an undirected graph, denoted by $G^V = (N^V, E^V, d(\cdot), d_t, d_d, d_s)$. N^V and E^V denote the sets of virtual nodes and links, respectively, where $n^v (\in N^V)$ and $e^v (\in E^V)$ represent the virtual nodes and links associated with the resource demand, denoted by $d(n^v)$ and $d(e^v)$, respectively. Additionally, d_t represents the VNR arrival time. Furthermore, d_d represents the maximum delay time in the waiting queue for the VNR, while d_s denotes the service duration time of the VNR. By modeling the VNR as an undirected graph with these parameters, we can more accurately represent the demands and constraints of virtual networks in the process of embedding them onto the substrate network.

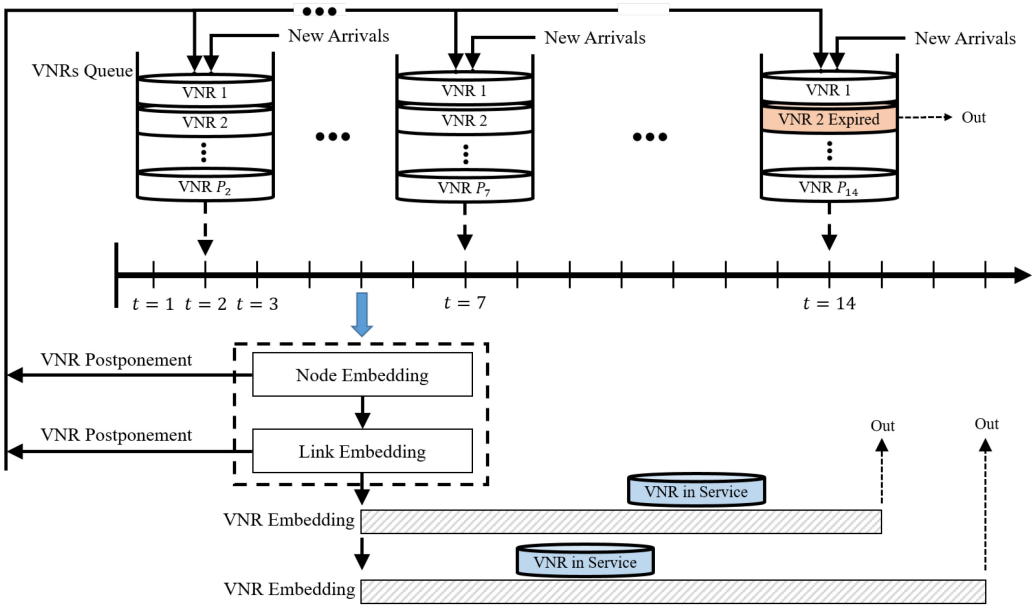


Fig. 2. In-depth view of the VNE procedure.

C. VNE Procedure

Fig. 1 provides an example of substrate network onto which VNRs are mapped. Multiple virtual nodes from different VNRs can be assigned to a substrate node concurrently (e.g., the virtual nodes "a" and "f" are assigned to the same substrate node "A"). Similarly, multiple virtual links can be assigned to one substrate link, or a virtual link can be mapped to multiple substrate links (e.g., the virtual link of two nodes "b" and "c" are mapped onto substrate links C-E-F). However, virtual nodes and links from the same VNR cannot be assigned to the same substrate node and link, respectively [23]. This constraint can improve network reliability and assists in preventing potential congestion or resource bottleneck.

There are multiple VNRs in the queue waiting to be serviced, and at each step the RL agent selects the most appropriate VNRs for embedding. In the VNE problem, the embedding consists of two main parts, node embedding and link embedding, as explained in Fig. 2. For a selected VNR, node embedding maps a virtual node to a substrate node, while link embedding maps a virtual link to a path in the substrate network. Once both the node embedding and link embedding are successful, the corresponding VNR service is activated and the resource gets occupied based on the VNR's demand. Once the VNR's service time d_s has elapsed, the occupied resources are released and restored. In case the node or link embedding fails, the VNR is postponed and placed again in the waiting queue for reallocation in the next step. In some cases, a VNR may be postponed multiple times and remain in the queue for an extended period. However, if the maximum delay time d_d for a VNR expires, it is rejected and removed from the queue.

D. Formal Definition of Our VNE Problem

M_N^V and M_E^V refer to the functions that define how the nodes and links of a virtual network G^V are embedded onto a given substrate network G^S, respectively. The node mapping is denoted as $M_N^V : N^V \rightarrow N^S$. On the other hand, the link mapping is denoted by $M_E^V : E^V \rightarrow Power(E^S)$, where $Power(E^S)$ represents all subsets of E^S. In our algorithm, the virtual nodes and virtual links are limited to only having CPU and bandwidth requirements, respectively. A virtual node $n^v \in N^V$ can be embedded into a substrate node $n^s \in N^S$ if the remaining capacity of n^s is greater than or equal to the CPU demand $d(n^v)$ of the virtual node, i.e., $c(M_N^V(n^v)) \geq d(n^v)$. A virtual link can be embedded into the links composed of a loop-free substrate path if the substrate links have a larger capacity than its required bandwidth, i.e., $c(e^s) \geq d(e^v)$ for each $e^s \in M_E^V(e^v)$.

In this study, the main objective of the VNE problem is to accept as many VNRs as possible while maximizing long-term average revenue and minimizing long-term average cost. To measure the effectiveness of a VNE algorithm, we use four metrics: 1) revenue; 2) cost; 3) revenue-to-cost ratio (R/C ratio); and 4) acceptance ratio. Revenue is the primary metric for evaluating the profit earned by InP or NSP from SP through VNR embeddings, and it increases with more VNRs embedded. Revenue at each time step t can be expressed as

$$\mathrm{REV}_t = \sum_{i=1}^{P_t} \left(\sum_{n^v \in N^{V_i}} d\left(n^v\right) + \sum_{e^v \in E^{V_i}} d\left(e^v\right) \right) \tag{1}$$

where P_t represents the total number of the serving VNRs and the new VNRs embedded at time step t. REV_t represents the sum of the CPU demand of the virtual nodes and the bandwidth demand of the virtual links of all the VNRs. The overall performance evaluation of a VNE algorithm is based on the long-term average revenue, as follows:

$$\lim_{T \rightarrow \infty} \frac{\sum_{t=1}^{T} \mathrm{REV}_t}{T}. \tag{2}$$

The cost refers to the amount of resources, i.e., CPU and bandwidth, consumed during the embedding process. When

a virtual node n^v with CPU demand $d(n^v)$ is embedded into the substrate node n^s with CPU unit capacity of $\sigma(n^s)$, the total node embedding cost is $\sigma(n^s) \cdot d(n^v)$. When a virtual link e^v with bandwidth demand $d(e^v)$ is embedded into the substrate path $(e_1^s, e_2^s, \ldots, e_K^s)$, where each bandwidth unit capacity $\sigma(e_1^s), \sigma(e_2^s), \ldots, \sigma(e_K^s)$, the total link embedding cost is $\sum_{k=1}^{K} \sigma(e_k^s) \cdot d(e^v)$. The cost incurred at each time step t can be defined as

$$\mathrm{COST}_t = \sum_{i=1}^{P_t} \left(\sum_{\substack{n^v \in N^{V_i}, \\ n^s = M_N^{V_i}(n^v)}} \sigma\left(n^s\right) \cdot d\left(n^v\right) + \sum_{\substack{e^v \in E^{V_i},\, e_k^s \in M_E^{V_i}(e^v), \\ K = \left|M_E^{V_i}(e^v)\right|}} \sum_{k=1}^{K} \sigma\left(e_k^s\right) \cdot d\left(e^v\right) \right). \quad (3)$$

On the other hand, the long-term objective is to minimize the following average cost:

$$\lim_{T \to \infty} \frac{\sum_{t=1}^{T} \mathrm{COST}_t}{T}. \quad (4)$$

The R/C ratio takes into account both the revenue and cost for the VNR embeddings. The R/C ratio at time step t has a range of 0 to 1, and defined as follows:

$$\alpha_t = \frac{\mathrm{REV}_t}{\mathrm{COST}_t} \quad (5)$$

and the long-term average R/C ratio is calculated as

$$\lim_{T \to \infty} \frac{\sum_{t=1}^{T} \alpha_t}{T}. \quad (6)$$

The acceptance ratio is a metric that expresses the ratio of embedded VNRs to waiting VNRs in the queue at each time t, and it can be defined as

$$\beta_t = \frac{\overline{\mathrm{NUM}}_t^V}{\mathrm{NUM}_t^V} \quad (7)$$

where $\overline{\mathrm{NUM}}_t^V$ is the number of embedded VNRs at time step t and NUM_t^V is the number of waiting VNRs in the queue. Finally, the long-term average acceptance ratio is

$$\lim_{T \to \infty} \frac{\sum_{t=1}^{T} \beta_t}{T}. \quad (8)$$

IV. Proposed Hierarchical Cooperative Multiagent Reinforcement Learning

In this section, the proposed hierarchical cooperative multiagent RL called HCMARL-VNE is explained in detail.

A. Designing Process

In this section, we explain the overall design of the proposed HCMARL-VNE. We divide the embedding decision into two levels, i.e., the HL and the LL, and HRL is utilized to solve our VNE problem. In our HRL, multiple HL agents collectively determine the subgoal to be achieved. The subgoal is to decide which VNRs to select for embedding and which VNRs to postpone, among the waiting VNRs in the queue. The LL agent makes decisions to solve the VNE problem in the given subgoal. That is, the LL agent learns a policy to decide and select a substrate node for each virtual node in VNRs that have been selected by the multiple HL agents.

Here, is a more exploration on the design process. At each time step, the multiple HL agents are created, matching the number of VNRs in the queue. The agents then select the appropriate subgoal, i.e., VNRs to be embedded, based on the current state of the environment. They share a single Q-network for a proper selection of action. During the training of the Q-network, the QMIX framework is utilized. It involves a mixing network based on the Q-values of the actions selected by each agent. Based on the multiple agents' Q-values, the mixing network calculates the Q-total value and updates the shared Q-network in the HL. Then, the LL agent executes its policy for that subgoal until it attempts to embed all virtual nodes of the selected VNRs from HL. Once the subgoal is achieved, the LL agent reports back to the multiple HL agents, who then select the next subgoal and assign it to the LL agent. The process continues until the predefined number of time steps is reached. Throughout the process, the multiple HL agents learn their policies that select subgoals based on the current state, i.e., the information of the overall substrate network and VNRs. We design the action chosen in the multiple HL agents to be critical to maximizing long-term revenue. The LL agent also learns a policy for each subgoal. We design the action chosen in the LL agent to be critical to maximizing the short-term R/C ratio over the duration of the subgoal. By breaking down the VNE problem into smaller subgoals and assigning them to an LL agent, the overall task becomes more manageable and can be achieved more efficiently and effectively. The hierarchical structure also allows the agents to adapt to changes in the information of the overall substrate network and VNRs, and to leverage prior knowledge and experience.

Fig. 3 demonstrates an example of steps involved in the proposed HCMARL-VNE algorithm. It employs an augmented substrate graph (ASG) composer function that creates states and observations using the overall current information of the substrate network and waiting VNRs. Further details on the ASG composer are provided in Section IV-C. First of all, multiple HL agents simultaneously select VNRs #1 and #3 to be embedded into the substrate network, while VNR #2 has been decided to be postponed for embedding at a later time. At the LL steps, the LL agent prioritizes the embedding of the VNR with the highest revenue when choosing between VNR #1 and VNR #3. A single agent selects a proper substrate node for each virtual node, and continues this process until it attempts to embed all virtual nodes of the selected VNRs.

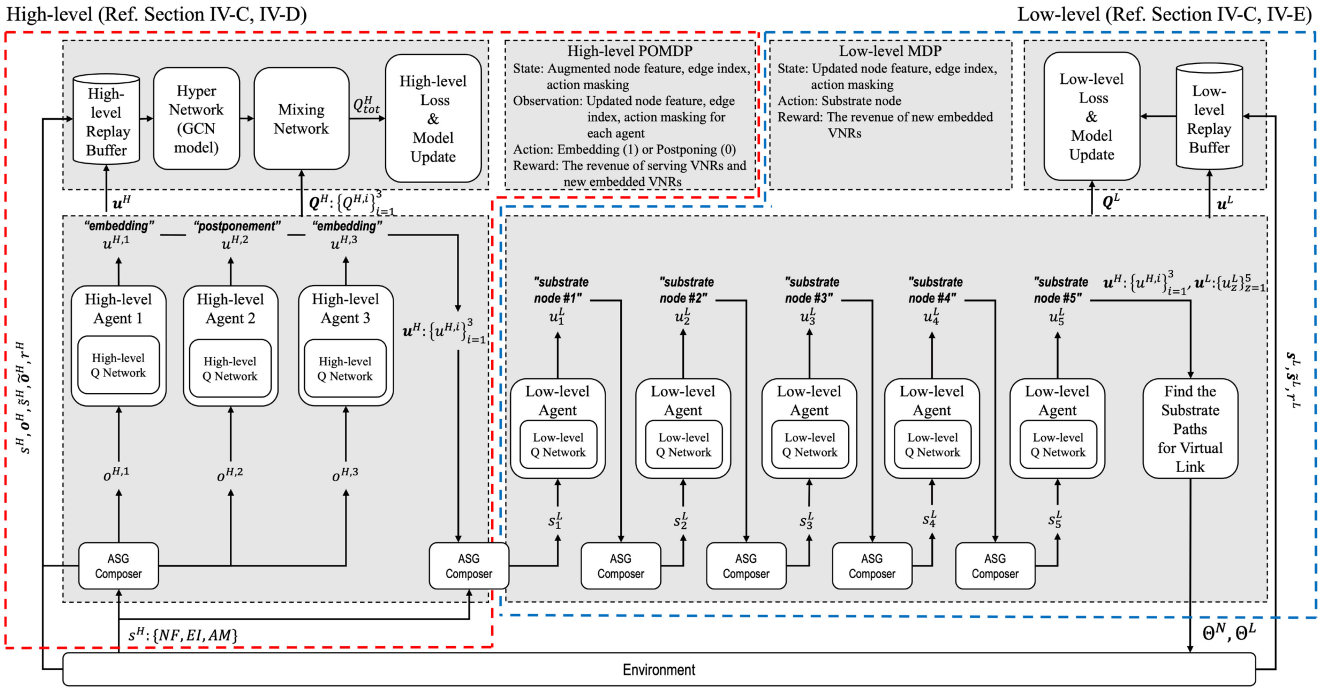


Fig. 3. Example of steps involved in the proposed HCMARL-VNE procedure at a time step (Θ^N and Θ^L represent the final node and link embedding information).

B. Markov Decision Process for HCMARL-VNE

In the proposed algorithm, the VNE problem is formulated as a hierarchical decentralized partially observable Markov decision process (HDec-POMDP), which consists of HL-MDP and LL-MDP. With fully cooperative multiagents, HL-MDP can be expressed as $\langle S^H, N^H, O^H, U^H, t^H, r^H, y^H\rangle$, where $s^H \in S^H$ is the true state of the environment and is common to the HL agents, N^H is the number of HL agents and equals the number of waiting VNRs for embedding, $o^{H,i} \in O^H$ is the observation for the HL agent i, $u^{H,i} \in U^H$ is an action for the HL agent i, t^H is the HL state transition probability which the multiple HL agents do not know, r^H is the HL reward function, and y^H is the discount factor.

The LL-MDP is based on single-agent RL which for selecting appropriate substrate nodes for the virtual nodes of the VNRs. The number of internal steps in the LL-MDP is equal to the number of virtual nodes in VNRs chosen by multiple HL agents. Therefore, the LL-MDP can be defined as a general MDP, unlike the HL Dec-POMDP. The LL-MDP can be defined as $\langle S^L, U^L, t^L, r^L, y^L\rangle$, where $s^L \in S^L$ is a state for the LL agent, $u^L \in U^L$ is an action for the LL agent, t^L is the LL state transition probability which the LL agent does not know, r^L is the LL reward function, and y^L is the LL discount factor.

C. States, Actions, and Rewards

In our algorithm, the state representation includes information about the overall substrate network and VNRs at each time step. The ASG composer serves as a crucial component in constructing the ASG model. The ASG model captures the resources and connectivity of the substrate network, and is further augmented with information about serving VNRs (or embedding VNRs) and waiting VNRs. By utilizing ASG in the state construction, several critical factors are taken into account when the multiple HL agents select VNRs in the waiting queue, or the LL agent chooses appropriate substrate nodes for the selected VNRs. These factors include 1) the availability of resources and connectivity within the substrate network; 2) the allocation status of currently serving VNRs; and 3) the status of waiting VNRs in the queue.

At time step t, ASG is defined as $G_t^A = (N_t^A, E_t^A)$ where N_t^A is the set of ASG nodes and E_t^A is the set of ASG links. N_t^A is defined as follows:

$$N_t^A = N^S \cup \bigcup_{V_s \in \Omega_t^V} N^{V_s} \cup \bigcup_{V_w \in \Phi_t^V} N^{V_w} \tag{9}$$

where N^S is the set of substrate nodes, Ω_t^V is the set of currently serving VNRs, and Φ_t^V is the set of waiting VNRs in the queue.

On the other hand, E_t^A comprises different types of links, including substrate links and virtual links of waiting VNRs in the queue. Additionally, augmented links are added to E_t^A to establish the relationship between VNRs and the substrate network. The augmented links serve two primary functions.

1) They connect the virtual node of currently serving VNRs to the embedded substrate node, reporting the current mapping and allocation of network resources.
2) They connect the virtual nodes of waiting VNRs to the candidate substrate nodes that can embed these virtual nodes, facilitating the selection of appropriate resources for the pending VNRs.

Accordingly, E_t^A is defined as follows:

$$E_t^A = E^S \cup \bigcup_{V_w \in \Phi_t^V} E^{V_w}$$

$$\cup\left\{\left(M_N^V(n^v), n^v\right)\middle| M_N^V(n^v)\in N^S, n^v\in \bigcup_{V_s\in\Omega_t^V} N^{V_s}\right\}$$
$$\cup\left\{\left(n^s, n^v\right)\middle| n^s\in N^S, n^v\in \bigcup_{V_w\in\Phi_t^V} N^{V_w}, c(n^s)\geq d(n^v)\right\}. \tag{10}$$

1) States: For a time step t, a state s_t^H configured by the ASG composer consists of ASG node features NF_t, ASG edge index EI_t, and action masking AM_t. For a node in AGS, NF_t consists of seven features denoted by $[T, A, B, C, D, E, X]$.

1) T represents whether a node is a substrate node (denoted by 0), a serving VNR node (denoted by 1), or a waiting VNR node (2).
2) A represents the remaining or demanded resource of the corresponding node, depending on whether it is a substrate or a virtual node.
3) B is the node degree.
4) C is the number of postponements if this node is a virtual node in a waiting VNR.
5) D is the remaining delay time if this node is a virtual node in a waiting VNR.
6) E is the service duration time if this node is a virtual node in a waiting VNR.
7) X can take on values of 0 or 1 if this node is a virtual node in a waiting VNR. The value varies based on the agent type and will be described in more detail later.

It is noted that C, D, E, and X are set to -1 for a substrate node or a virtual node in a serving VNR. On the other hand, EI_t simply represents the adjacency of each node, and it is formed using E_t^A at every step t.

AM_t refers to the binary action masking information for candidate substrate nodes that are suitable for embedding each virtual node of a VNR. For a virtual node, candidate substrate nodes are masked when they have been already mapped to other virtual nodes of a VNR, and the amount of remaining CPU in the candidate substrate node is less than the demanded CPU for the virtual node. This AM helps to ensure that unmasked candidate nodes have sufficient remaining CPU and memory are available for use, resulting in better RL learning performance.

As shown in Fig. 3, the ASG composer configures the observation $o_t^{H,i}$ for each HL agent i by using the current identical state s_t^H. Each $o_t^{H,i}$ consists of NF_t and EI_t included in s_t^H, but the last feature X of NF_t is set to one for virtual nodes in the VNR associated with the HL agent i, and set to zero to other virtual nodes. Therefore, each HL agent uses unique observation as its input, allowing it to make independent decisions. This observation enables the HL agents to consider various features, such as the specific VNR's requirements and the substrate network's current status, when deciding whether to embed or postpone the associated VNR.

The ASG composer then configures the state s_z^L for each internal step z of the LL agent. For each internal step z, the LL agent attempts to sequentially embed a virtual node of the VNRs selected by multiple HL agents. For an HL time step t, s_z^L is configured by utilizing NF_t, EI_t, and AM_t included in s_t^H, as well as by incorporating the VNRs selected by the HL agents (i.e., the action information of the HL agents). However, the last feature X in NF_t is set to one for the virtual node being targeted for embedding by the LL agent at internal step z, and is set to zero for all other virtual nodes. For each internal step z, NF_t, EI_t, and AM_t are denoted by NF_t^z, EI_t^z, and AM_t^z, respectively, and changed by the ASG composer to reflect the remaining resources of substrate nodes and links, which have been reduced due to virtual node embedding in the previous internal steps.

2) Actions: At time step t, each HL agent i makes a decision on embedding (1) or postponing (0) the VNR associated with the HL agent based on its policy, and takes the corresponding action $u_t^{H,i}$. Then, $\boldsymbol{u}_t^H$ is formed by collecting the individual actions $\{u_t^{H,i}\}_{i=1}^{N^H}$ of multiple HL agents. $\boldsymbol{u}_t^H$ is also used for the ASG composer to configure the state for the LL agent.

On the other hand, the LL agent chooses a substrate node n^s for a virtual node being targeted for embedding at an internal step z for the time step of the HL agents, and takes the corresponding action u_z^L. The virtual nodes targeted for embedding are selected in order of their resource demand, so that the LL agent first selects the virtual node with the highest demand. If the LL agent is unable to find an appropriate substrate node for a virtual node in a VNR, the entire VNR is postponed and added to the queue of waiting VNRs. Then, $\boldsymbol{u}_t^L$ is formed by collecting the individual actions $\{u_z^L\}_{k=1}^{N^L}$.

The HL and LL agents interact with the environment by executing both $\boldsymbol{u}_t^H$ and $\boldsymbol{u}_t^L$ actions, and receiving feedback in the form of rewards. After taking actions and receiving rewards, the agents update their policies to improve future decision making.

3) Rewards: We use HL agents to embed high-revenue VNRs while postponing lower revenue ones to maximize long-term cumulative revenue. Therefore, the HL agent's reward is based on the revenue generated by both the VNRs currently being served and the revenue of any new VNRs embedded during the current time step t. If the LL agent fails to embed a VNR that was selected by an HL agent, then the revenue generated by that VNR is not added to the HL agent's reward. The reward function for the HL agents' action $\boldsymbol{u}_t^H$ is defined as follows:

$$r_t^H = \sum_{V_s\in\Omega_t^V}\left(\sum_{n^v\in N^{V_s}} d(n^v) + \sum_{e^v\in E^{V_s}} d(e^v)\right) + \sum_{V_n\in\pi_t^V}\left(\sum_{n^v\in N^{V_n}} d(n^v) + \sum_{e^v\in E^{V_n}} d(e^v)\right) \tag{11}$$

where Ω_t^V is the set of currently serving VNRs and π_t^V is the set of new VNRs embedded at time step t.

For a virtual node of VNRs chosen by the HL agents, we use the LL agent to find a high-revenue and low-cost substrate node for each of the virtual nodes. This means that the LL agent will try to avoid long paths when embedding links between the virtual nodes, as longer paths tend to have higher costs. Because the LL agent's decisions are made based on the

HL agents' long-term goal, the LL agent needs to consider the short-term goals and make decisions. For a time step of the HL agents, the reward function for the LL agent's action at an internal step z (i.e., the reward for the LL agent's action u_z^L) is defined as follows:

$$r_z^L = \sum_{V_n \in \pi_t^V} \left(\sum_{n^v \in N^{V_n}} d(n^v) + \sum_{e^v \in E^{V_n}} d(e^v) \right) - \alpha \sum_{V_n \in \pi_t^V} \left(\sum_{\substack{n^v \in N^{V_n} \\ n^s = M_N^{V_n}(n^v)}} \sigma(n^s) \cdot d(n^v) + \sum_{\substack{e^v \in E^{V_n}, e_k^s \in M_E^{V_n}(e^v), \\ K = |M_E^{V_n}(e^v)|}} \sum_{k=1}^{K} \sigma(e_k^s) \cdot d(e^v) \right) \quad (12)$$

where α determines how much weight is given to the cost against the reward. It should be noted that the reward r_z^L is calculated based on both the node embedding and the link embedding.

D. High-Level Mixed Q-Network Model and Training

Our proposed algorithm aims to maximize long-term revenue by training multiple HL agents using the QMIX algorithm. As shown in Fig. 4, the Q-network of each HL agent is used to select an action based on the ASG node features (NF_t) and ASG edge index EI_t configured by the ASG composer. The proposed HL agent Q-network consists of GCNs and the gated recurrent units (GRUs) [55]. In VNE problems, it is important for RL agents to recognize the spatial characteristics of the substrate network and virtual networks. GCNs are used to extract the features based on the relationship between nodes and edges in the configured ASG. As explained in Section IV-C, the ASG's augmented nodes and links represent the diverse relations among the serving VNRs, the pending VNRs, and the substrate nodes, so that the features extracted by the GCN model can be useful in selecting appropriate VNRs for embedding. In the proposed HL agent's Q-network, GRUs are used to model the temporal dependencies of the agents' actions and states, which is particularly useful in partially observable environments. The final output layer of Q-network is configured by a fully connected layer. The LL agent's Q-network produces Q-values for all possible actions from the final output layer. These Q-values estimate the future rewards for each action. During the training phase, the HL agent selects the action to take using an ϵ-greedy method based on the Q-values.

The proposed GCN-based HL mixing network, as shown in Fig. 5, is designed to enhance the QMIX algorithm by incorporating GCNs for better representation learning of the global state, which is also modeled as the ASG. It utilizes a hypernetwork [54] to generate weights and biases for the mixing

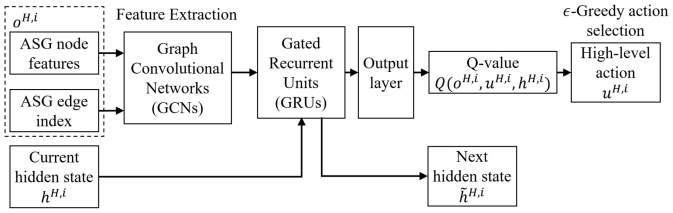


Fig. 4. Q-network structure for the HL agent i.

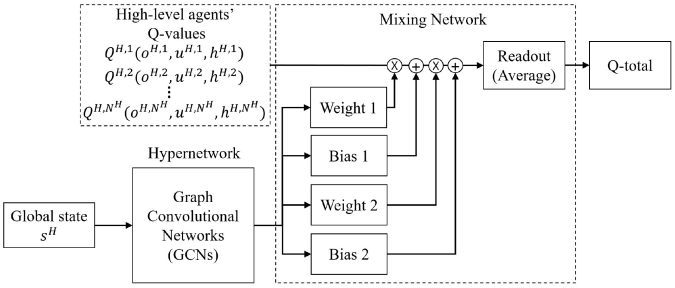


Fig. 5. Mixing network structure for the cooperation of the multiple HL agents.

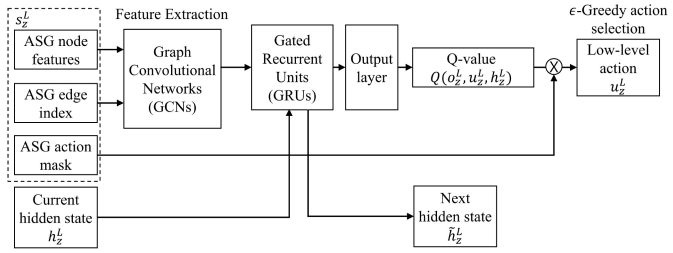


Fig. 6. Q-network structure for the LL agent at internal step z.

network based on the global state. These parameters are then employed by the mixing network to combine agents' Q-values and compute the joint action-value (Q-total) using an average pooling process known as readout.

During the training of HL agents, for each time step t, the following pieces of information are collected.

1) s_t^H *and* $\tilde{s}_t^H$*:* The global state and the next global state shared by all HL agents and configured by using NF_t and EI_t.
2) $o_t^{H,i}$ *and* $\tilde{o}_t^{H,i}$*:* The HL agent i's observation and the agent's next observation configured by using NF_t and EI_t.
3) $h_t^{H,i}$ *and* $\tilde{h}_t^{H,i}$*:* The HL agent i's hidden information and the agent's next hidden information.
4) $u_t^{H,i}$*:* The HL agent i's action.
5) r_t^H*:* The HL reward.

It is noted that s_t^H, $\tilde{s}_t^H$, $o_t^{H,i}$, and $\tilde{o}_t^{H,i}$ are configured by using NF_t and EI_t. The collected information for each time step and each HL agent is referred to as an HL transition. The HL agents' transitions from all time steps are stored in the replay memory for later use in training both the HL mixing Q-network and the Q-network of each HL agent. In QMIX, the replay memory (also known as experience replay buffer) plays a crucial role in stabilizing and improving the learning process. The replay memory allows the QMIX algorithm to reuse past

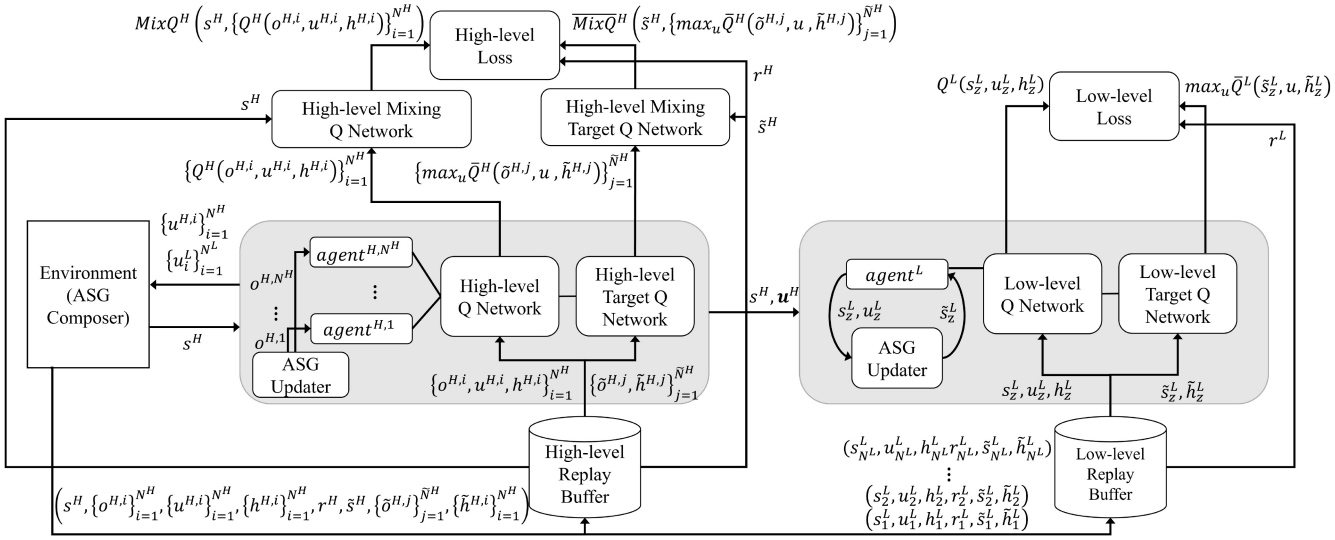


Fig. 7. Overall training procedure for the proposed models.

experiences multiple times. This helps improve the sample efficiency and accelerate the learning process. During training, the QMIX algorithm samples mini-batches of HL transitions from the replay memory to update the agents' Q-networks and the mixing network.

For a mini-batch sampled from the replay memory and the index τ representing an HL transition within the mini-batch, they are trained together in an end-to-end manner to minimize the following temporal-difference (TD) error:

$$\text{Loss}^H = \sum_{\tau=1}^{B}\left(r_\tau^H + \gamma^H \cdot \overline{\text{Mix}Q}^H\left(\tilde{s}_\tau^H, \left\{\max_u \overline{Q}^H\left(\tilde{o}_\tau^{H,i}, u, \tilde{h}_\tau^{H,j}\right)\right\}_{j=1}^{\tilde{N}^H}\right) - \text{Mix}Q^H\left(s_\tau^H, \left\{Q^H\left(o_\tau^{H,i}, u_\tau^{H,i}, h_\tau^{H,i}\right)\right\}_{i=1}^{N^H}\right)\right)^2 \quad (13)$$

where B represents the batch size, and other notations are defined as follows.

1) Q^H: The Q-value of each HL agent.
2) $MixQ^H$: The mixing network's total Q-value.
3) $\overline{Q}^H$: The target Q-value of each HL agent.
4) $\overline{MixQ}^H$: The mixing network's target Q-value.
5) N^H: The number of HL agents τ.
6) $\tilde{N}^H$: The number of HL agents at the next time step.

By configuring the loss function in (13), the QMIX algorithm can effectively train the HL agents' Q-networks and the mixing network. Fig. 7 illustrates the overall training process of the model from Section IV, starting from Section IV-B to Section IV-F. The environment's state, denoted as s^H, is provided as input to the MARL-based HL agents through the ASG composer, generating observations $o^{H,i}$. Subsequently, the HL agents employ their Q-networks to select Q-values and actions, denoted as $\boldsymbol{u}^H$. The selected actions $\boldsymbol{u}^H$ from the HL agents, along with state s^L, are then input into the LL Q-networks, determining Q-values and u^L. Following this, both the HL and LL agents update their respective replay buffers. In the case of the HL agents, they undergo additional mixing network operations for cooperation before updating.

The procedure depicted in Fig. 7 involves training the HL agent Q-network and mixing network, which allows them to learn a decentralized decision policy. This policy involves determining whether to embed or postpone the VNR associated with each HL agent. By repeating this procedure, the agents can work together to learn and improve their decision-making abilities in a collaborative manner.

E. Low-Level Q-Network Model and Training

The proposed LL agent Q-network utilizes a similar architecture to the Q-network used by the HL agent. Specifically, the LL agent Q-network is composed of GCNs and GRUs, as shown in Fig. 6. GRU uses the hidden state h_z^L to receive additional input to recognize information about the substrate node previously selected by the LL agent. Moreover, the LL agent Q-network operates in a similar manner to the HL agent Q-network. However, action masking is employed by the LL agent to filter out substrate nodes that have already been embedded with virtual nodes, as well as those that do not have sufficient remaining CPU and memory to support the virtual node at the internal step. By using action masking in this way, the LL agent can focus on selecting only those substrate nodes that are viable candidates for mapping virtual nodes, which can ultimately lead to more efficient and effective resource allocation.

During the training of LL agents, for each internal step z, the following pieces of information are collected.

1) s_z^L: The LL agent's internal state.
2) $\tilde{s}_z^L$: The LL agent's next internal state.
3) h_z^L: The LL agent's hidden information.
4) $\tilde{h}_z^L$: The LL agent's next hidden information.
5) u_z^L: The LL agent action.
6) r_z^L: The LL reward.

It is noted that s_z^L and $\tilde{s}_z^L$ are configured by using AM_t^z as well as NF_t^z and EI_t^z. As described in Section IV-C3, it is important to note that the LL reward r_z^L is calculated based on both the node embedding and the link embedding. The link

embedding process involves finding a suitable substrate path by using the k-shortest path algorithm [6]. It helps ensure that the virtual links are mapped to substrate paths that can support their bandwidth requirements, and also helps to minimize bandwidth consumption by the link embedding. By incorporating both node and link embedding into the reward calculation, the LL agent can make more informed decisions for VNE.

The collected information for each time step is referred to as an LL transition. The LL transitions from all internal steps are stored in the replay memory for later use in training the Q-network of the LL agent. This is accomplished through the DQN training method, which involves updating the Q-network by minimizing the TD-error between the predicted Q-values and the target Q-values. The target Q-network is updated periodically (less frequently than the Q-network) by copying the parameters from the Q-network. This helps stabilize the target Q-value estimates.

For a mini-batch sampled from the replay memory and the index τ representing a transition within the mini-batch, the Q-network is trained to minimize the following TD error:

$$\text{Loss}^L = \sum_{\tau=1}^{B} \left(r_\tau^L + \gamma^L \cdot \max_u \overline{Q}^L\left(\widetilde{s}_\tau^L, u, \widetilde{h}_\tau^L\right) - Q^L\left(s_\tau^L, u_\tau^L, h_\tau^L\right) \right)^2 \tag{14}$$

where B represents the batch size, Q^L is the Q-value of the LL agent, and $\overline{Q}^L$ is the target Q-value of the LL agent. The procedure depicted in Fig. 7 also involves training the LL agent Q-network. By repeating this procedure, the LL agent can learn and improve its decision-making ability to select the best possible substrate nodes for the virtual nodes.

F. HCMARL-VNE Procedure With Trained Models

After training all the network models of the HL and LL agents, they can be used for VNE using Algorithm 1 called HCMARL-VNE. For each time step, the HL agents to select the proper VNRs for embedding based on their learned decision policy (lines 7–12). Once the VNR selection is complete, the LL agent performs the node and link embedding procedure (lines 13–40). If a suitable substrate node is not identified for a virtual node through the action mask, the corresponding VNR embedding is deferred to the next time step (lines 20–22). Only when suitable substrate nodes are successfully identified for all virtual nodes in a VNR (lines 24–28), the LL agent initiates the link embedding process to find a suitable substrate path for each virtual link of the VNR (lines 30–33). This process involves using a k-shortest path algorithm to ensure that the virtual links are mapped to the most suitable substrate paths possible. If a suitable substrate path is not found for a particular virtual link, the corresponding VNR embedding is also postponed until the next time step (lines 35 and 36). If the node and link embedding for a VNR are successfully performed, the embedding information is stored to the final embedding information at the current time step. After the node and link embeddings have been carried out for all VNRs, the actual embeddings are performed using the final embedding information (line 41). On the other hand, the VNRs that are classified as postponed ones are placed into the queue of waiting VNRs for the next time step (line 42).

Algorithm 1: Proposed Overall Procedure of HCMARL-VNE at a Time Step t

1 $\Phi_t^V \leftarrow$ the waiting VNRs
2 $NF_t, EI_t, AM_t \leftarrow$ the current ASG
3 $\boldsymbol{u}_t^H \leftarrow \emptyset$ /* HL agent's action */
4 $\Theta_t^N \leftarrow \emptyset$ /* final node embeddings */
5 $\Theta_t^L \leftarrow \emptyset$ /* final link embeddings */
6 $\Gamma_t \leftarrow \emptyset$ /* VNRs decided to be postponed */
7 **foreach** $V_i \in \Phi_t^V$ **do**
8 Update NF_t, EI_t, AM_t
9 Gather $s_t^{H,i}$, $o_t^{H,i}$, and $h^{H,i}$
10 $u_t^{H,i} \leftarrow max_u Q^H(o_t^{H,i}, u, h_t^{H,i})$
11 $\boldsymbol{u}_t^H \leftarrow \boldsymbol{u}_t^H \cup \{u_t^{H,i}\}$
12 Save $h^{H,i}$ for next time step
13 **foreach** $u_t^{H,i} \in \boldsymbol{u}_t^H$ **do**
14 $V_i \leftarrow$ the corresponding VNR
15 $z \leftarrow 0$ /* internal step */
16 **if** $u_t^{H,i}$ *is 'Embedding'* **then**
17 $\Theta_t^{N_i} \leftarrow \emptyset$ and $\Theta_t^{L_i} \leftarrow \emptyset$
18 **for** $n \in N^{V_i}$ **do**
19 Update NF_t^z, EI_t^z, AM_t^z
20 **if** *no available substrate node in* AM_t^z **then**
21 $\Gamma_t \leftarrow \Gamma_t \cup \{V_i\}$
22 break
23 **else**
24 Gather the updated s_z^L and h_z^L
25 $u_z^L \leftarrow max_u Q^L(s_z^L, u, h_z^L)$
26 $\Theta_t^{N_i} \leftarrow \Theta_t^{N_i} \cup \{(n \mapsto u_z^L)\}$
27 Save h_z^L for next internal step
28 $z \leftarrow z + 1$
29 **if** *all nodes* $\in N^{V_i}$ *are embedded* **then**
30 **for** $e \in E^{V_i}$ **do**
31 Find the substrate path p using the k-shortest paths while increasing k
32 **if** p *is found* **then**
33 $\Theta_t^{L_i} \leftarrow \Theta_t^{L_i} \cup \{(e \mapsto p)\}$
34 **else**
35 $\Gamma_t \leftarrow \Gamma_t \cup \{V_i\}$
36 break
37 **if** *all links* $\in E^{V_i}$ *are embedded* **then**
38 $\Theta_t^N \leftarrow \Theta_t^N \cup \Theta_t^{N_i}$ and $\Theta_t^L \leftarrow \Theta_t^L \cup \Theta_t^{L_i}$
39 **else**
40 $\Gamma_t \leftarrow \Gamma_t \cup \{V_i\}$
41 Perform the embeddings Θ_t^N and Θ_t^L to the substrate networks
42 $\Phi_{t+1}^V \leftarrow \Phi_{t+1}^V \cup \Gamma_t$ for the procedure at the next time step $t+1$

In Algorithm 1, the time complexity of the k-shortest path algorithm is $\mathcal{O}(m^s + n^s \log n^s + k)$ [37], where m^s, n^s, and m^v refer to the number of substrate nodes, substrate links and virtual links, respectively. Therefore, the overall time

TABLE II
PARAMETER SETTINGS OF THREE SIMULATED NETWORKS

	Didactic network I	**Didactic network II**	**ISP network**
N^S	3	4	100
E^S	2	5	500
$c(n^s)$	10, 10, and 10 units for each of three substrate nodes	13, 16, 11, and 15 units for each of four substrate nodes	50∼100 units for all substrate nodes (Uniform distribution)
$c(e^s)$	10 and 10 units for each of two substrate edges	8, 10, 6, 10, and 13 units for each of five substrate edges	50∼100 units for all substrate edges (Uniform distribution)
$\sigma(n^s)$	1	1	1
$\sigma(e^s)$	1	1	1
N^V	VNR #1: 3 nodes VNR #2: 3 nodes VNR #3: 3 nodes	VNR #1: 3 nodes VNR #2: 3 nodes VNR #3: 3 nodes VNR #4: 3 nodes VNR #5: 3 nodes	5∼10 nodes for each VNRs (Uniform distribution)
E^V	VNR #1: 2 edges VNR #2: 2 edges VNR #3: 2 edges	VNR #1: 2 edges VNR #2: 3 edges VNR #3: 2 edges VNR #4: 3 edges VNR #5: 2 edges	0.5 (Link connection probability for each pair of nodes)
$d(n^v)$	VNR #1: [6, 6, 6] VNR #2: [1, 1, 1] VNR #3: [4, 4, 4]	VNR #1: [3, 4, 7] VNR #2: [3, 5, 2] VNR #3: [3, 4, 1] VNR #4: [3, 5, 2] VNR #5: [3, 4, 7]	10∼30 units (Uniform distribution)
$d(e^v)$	VNR #1: [1, 1] VNR #2: [1, 1] VNR #3: [1, 1]	VNR #1: [1, 5] VNR #2: [3, 5, 1] VNR #3: [1, 3] VNR #4: [3, 5, 1] VNR #5: [1, 5]	10∼30 units (Uniform distribution)
d_t	1, 2, and 3 time epochs for each of 3 VNRs	1, 2, 3, 4, and 5 time epochs for each of 5 VNRs	$\lambda = 1/20$ per time unit (Poisson distribution, about 2800 VNRs)
d_d	5, 5, and 5 time units for each of 3 VNRs	5, 5, 5, 5, and 5 time units for each of 5 VNRs	200 time units for all VNRs
d_s	5, 5, and 5 time units for each of 3 VNRs	10, 9, 8, 7, and 6 time units for each of 5 VNRs	$\lambda = 500$ time units (Exponential distribution)

complexity of the proposed HCMARL-VNE is denoted as $\mathcal{O}(m^v(n + m^s + n^s \log n^s + k))$.

V. PERFORMANCE EVALUATION

This section describes the performance evaluation of the proposed algorithm. The simulation environment is set up using an i9-9900K CPU with 64-GB RAM and an RTX 3090 GPU with a Linux Ubuntu 20.04 LTS. The proposed algorithm is implemented with Python 3.8 and PyTorch 1.9.0 where the NetworkX 2.5.1 Library is utilized to configure the network environments. In addition, Torch-geometric 1.7.2 is used to construct the GCN model. The parameters of three simulated network environments are described in detail in Table II. The parameters of the HCMARL-VNE algorithm are also described in detail in Table III. The parameters of the HCMARL-VNE algorithm are selected based on our experiments to achieve the best results and appropriate experimental runtime.

TABLE III
LEARNING PARAMETER SETTING

Parameters	**Values**
GRU hidden dimension	64
Output dense layer dimension	32
QMIX hidden dimension	32
Hypernetwork hidden dimension	64
Replay Buffer Capacity	5000
Target update training step cycle	100
Gradient clipping	10
Discount factor (γ)	0.99
Learning rate	0.0005
Batch size (B)	128
Epsilon (ϵ)	1 (initial), 0.05 (final)
The number of the train steps	1

A. Simulation Environment

1) Didactic Network I: Didactic network I refers to a simplistic and easy-to-understand environment used for representing our algorithm's superiority. The experiment demonstrates that a better episode reward can be achieved by postponing the second VNR. This means that delaying the embedding of the second VNR allows for more efficient resource allocation.

As described in Table II, three VNRs arrive sequentially at time epoch 1, 2, and 3, where their CPU resource demands are [6, 6, 6], [1, 1, 1], and [3, 3, 3] for all three virtual nodes. In the substrate network, there are a total of three substrate nodes, each with a CPU capacity of 10. Given these constraints, it becomes infeasible to embed the virtual nodes of all three VNRs into the substrate nodes. In the specified didactic network, the substrate network's capacity is limited to embedding a maximum of two VNRs. The length of an episode is equivalent to five time steps. Embedding the first and second VNRs consumes seven CPU units, and thus it is impossible to accommodate the third VNR. Consequently, this approach does not yield optimal long-term revenue. To achieve better long-term revenue, it is better to embed the first and third VNRs while postponing the second VNR.

2) Didactic Network II: In the didactic network II, the substrate network's scale is expanded and the number of incoming VNRs has increased to five. In the substrate network, the substrate nodes and links have diverse CPU capacities and bandwidth resources. Additionally, the five VNRs possess varying CPU and bandwidth demands for their three virtual nodes and two or three virtual links. They arrive sequentially to the waiting queue at time epoch 1, 2, 3, 4, and 5. For the VNRs, the maximum delay times on the waiting queue are consistently five time units, but their service duration times after successful embedding differ, with values of 10, 9, 8, 7, and 6, respectively. The length of an episode is equivalent to 10 time steps. The combination of heterogeneous substrate resource capacities and diverse VNR demands add complexity to the VNE problem. In addition, it is crucial to carefully consider the maximum delay time and the service time when making decisions related to the problem. All these factors underscore the necessity of developing an

TABLE IV
EXISTING VNE ALGORITHMS FOR PERFORMANCE COMPARISON

VNE algorithms	Description
Heuristic baseline [6]	To begin the embedding process, the VNR with the highest revenue among those waiting in the queue is greedily selected. Virtual nodes are then allocated resources on the substrate nodes based on the available resources, with priority with more remaining resources. The k-shortest path algorithm is employed for link embedding.
Topology-Aware [7]	Topology-aware node ranking evaluates node significance within a network by considering resource and topological attributes. Utilizing a Markov random walk model, it calculates node ranks based on CPU, bandwidth, and neighbor resources, along with their inter-connectivity.
A3C+GCN [34]	GCN is used to automatically extract spatial features in a substrate network. A3C is also used to ensure efficient sampling of the training experiences and optimization of the learning agent.
Pointer networks [29]	Pointer network and Dijkstra algorithm are used to determine embedding policies for virtual nodes and links, respectively. Additionally, the attention mechanism is incorporated to emphasize the substrate nodes that are more probable to be chosen.
HRL-VNE [37]	This considers both the long-term and short-term effects of VNR embedding. The process involves a HL agent selecting a feasible VNR in the waiting queue with the highest long-term revenue, followed by a low-level agent embedding the VNR on the substrate node.

effective VNE algorithm to optimize embedding performance and efficiently manage network resources.

3) ISP Network: In alignment with a specification for medium-sized ISPs, we have configured the substrate network to comprise 100 nodes and 500 links. The CPU resources allocated to the substrate nodes are uniformly distributed within the range of 50–100 units. Similarly, the bandwidth resources assigned to the substrate links are also uniformly distributed within the range of 50–100 units. The arrival rate of VNRs aligns with a Poisson distribution, with an average occurrence of one VNR every 20 time units. The delay expiration time at the waiting queue is set to 200 time units, while the serving duration follows an exponential distribution with a mean of 500 time units. The number of virtual nodes for the VNR is uniformly distributed within the range of 5–15 nodes. The probability of a connection between two nodes is 0.5. The CPU and bandwidth demands for virtual nodes and links are uniformly distributed between 10 and 30 units. The overall time span extends to a duration of 56 000 time steps, indicating that the length of an episode is equivalent to 56 000 time steps.

B. Performance of Proposed Algorithm

The proposed HCMARL-VNE algorithm is compared with four existing VNE algorithms, namely: 1) the heuristic baseline (called Baseline) [6]; 2) RL algorithm using A3C and GCN (called A3C+GCN) [34]; 3) the attention mechanism using pointer networks (called PN) [29]; and 4) HRL-based algorithm using DQN (called HRL-VNE) [37]. Table IV describes the four VNE algorithms.

Fig. 8 represents the results of a comparison between the proposed algorithm and existing VNE algorithms in the didactic network I. The proposed HCMARL-VNE algorithm

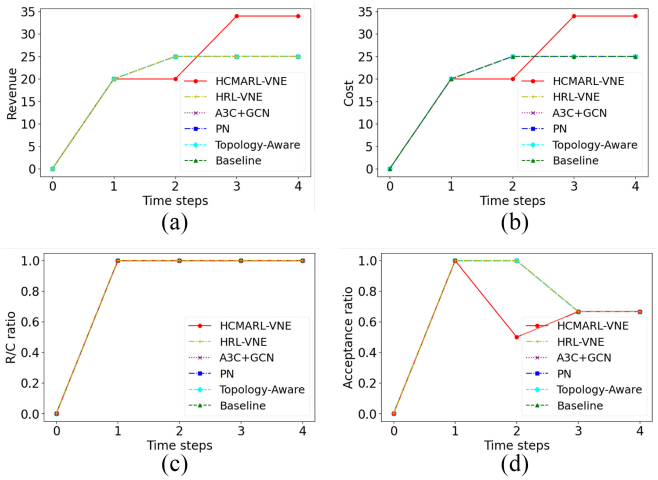


Fig. 8. Effectiveness of the proposed algorithm in the didactic network I. (a) Revenue results. (b) Cost results. (c) R/C ratio results. (a) Acceptance ratio results.

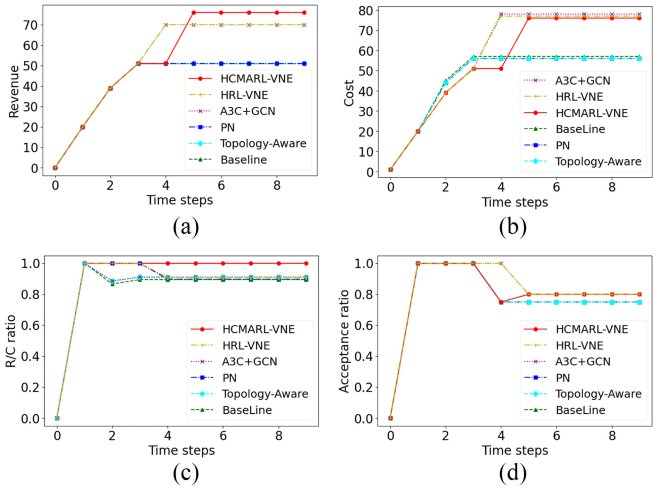


Fig. 9. Effectiveness of the proposed algorithm in the didactic network II. (a) Revenue results. (b) Cost results. (c) R/C ratio results. (a) Acceptance ratio results.

demonstrates lower revenue, cost, and acceptance ratio than other VNE algorithms until the time step 2. This result can be attributed to an HL agent's choice of "Postponing" action for the second VNR. However, by embedding the third VNR in the time step 3, the proposed algorithm achieved the highest revenue (34 revenue) compared with the one (25 revenue) of the other VNE algorithms. Although the cost increases due to the embedding of the third VNR, the R/C ratio and the acceptance ratio remain unchanged. The proposed HCMARL-VNE algorithm is expected to achieve similar performance to HRL-VNE if all VNRs are added to the queue at time step $t = 1$. However, in the cases where VNRs are accumulated step by step, such as in the didactic network I, HRL-VNE would embed VNR #2 as it lacks the "Postponing" action. In contrast, the proposed HCMARL-VNE would not embed, but choose to postpone VNR #2, because it is trained to maximize the overall long-term revenue.

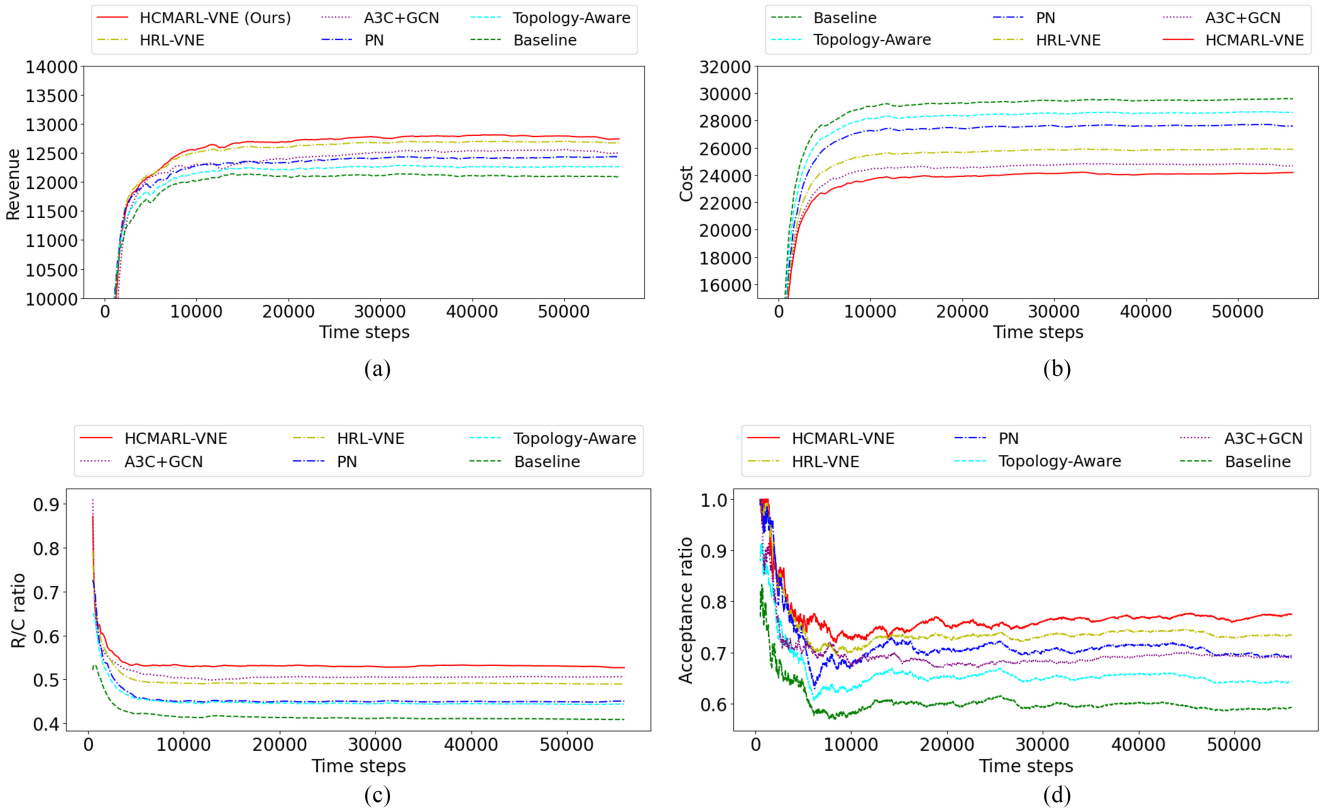


Fig. 10. Effectiveness of the proposed algorithm in the ISP network. (a) Long-term average revenue results. (b) Long-term average cost results. (c) Long-term average R/C ratio results. (d) Long-term average acceptance ratio results.

The experimental results in the didactic network II are shown in Fig. 9. In this particular network, the embedded VNRs for each respective algorithm are as follows.

1) *HCMARL-VNE:* VNR #1, VNR #2, VNR #3, VNR #5.
2) *HRL-VNE:* VNR #1, VNR #2, VNR #3, VNR #4.
3) *A3C+GCN:* VNR #1, VNR #2, VNR #3, VNR #4.
4) *PN:* VNR #1, VNR #2, VNR #3.
5) *Baseline:* VNR #1, VNR #2, VNR #3.

Therefore, the performance of HCMARL-VNE, HRL-VNE, and A3C+GCN surpasses that of both PN and Baseline in terms of generated revenue. Moreover, the highest revenue is achieved by the proposed HCMARL-VNE (76 revenue) in comparison to HRL-VNE (70 revenue) and A3C+GCN (70 revenue). This outcome can be attributed to HCMARL-VNE's strategic decision to postpone the low-revenue VNR (the 4th VNR) while embedding the high-revenue VNR (the fifth VNR). Also, with respect to the cost, HCMARL-VNE (76 cost) is more efficient than HRL-VNE (76 cost) and A3C+GCN (78 cost) because it postpones the high-cost VNR (the 4th VNR) while embedding the low-cost VNR (the fifth VNR). As a result, it can be confirmed that HCMARL-VNE has the best R/C ratio. Hence, the proposed HCMARL-VNE demonstrates the optimal results in terms of most performance metrics.

In contrast to the results presented in Figs. 8 and Fig. 9, Fig. 10 presents the long-term outcomes of a proposed algorithm and provides a comparison with existing VNE algorithms in the ISP network. Fig. 10(a) and (b) presents the long-term average revenue and cost obtained up to 56 000 time steps. The HCMARL-VNE algorithm, as proposed, outperforms all other compared VNE algorithms in terms of both revenue generation and cost efficiency. Specifically, the algorithm achieves the highest revenue among all compared algorithms, while also recording the lowest cost. In contrast, the baseline algorithm performs the worst in terms of revenue generation, while also having the highest cost among all compared algorithms. The PN and A3C+GCN algorithms exhibit similar revenue performance. However, A3C+GCN demonstrates superior cost performance compared to PN. This suggests that A3C+GCN may be a more cost-efficient solution than PN, while still achieving comparable revenue outcomes. To summarize once again, the proposed HCMARL-VNE algorithm efficiently manages the allocation of physical resources to VNRs while embedding virtual nodes using potentially shorter paths. These results obtained can be also validated at Fig. 10(c), which provides a long-term perspective of the R/C ratio for the four VNE algorithms. The efficient use of CPU and bandwidth resources makes the proposed HCMARL-VNE algorithm the best choice in terms of R/C ratio.

In Fig. 10(d), we also compare the long-term acceptance ratio of the proposed algorithm with existing VNE algorithms. For all the algorithms under consideration, the initial acceptance ratio is initially high, as the substrate network has sufficient resources available to embed a large number of VNRs. However, as time goes by, it often happens that VNRs waiting in the queue cannot be embedded due to insufficient

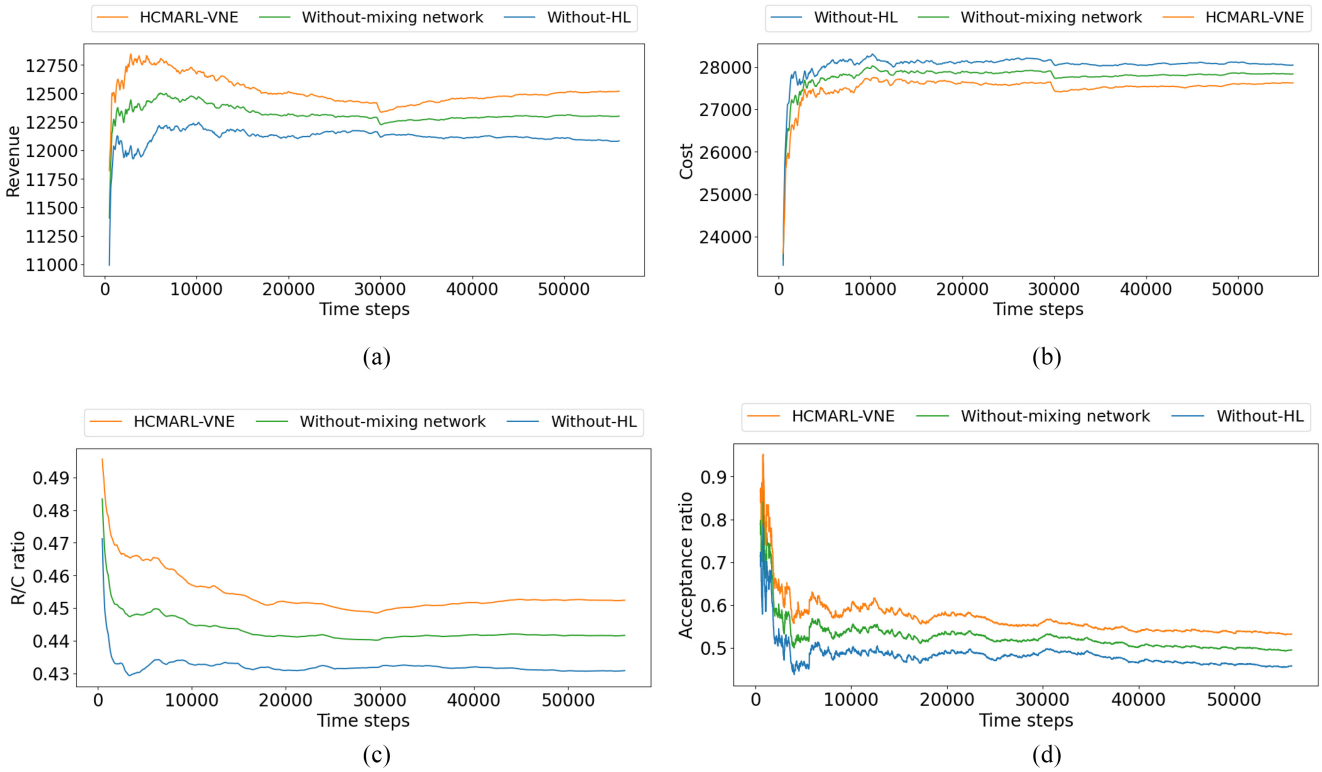


Fig. 11. Results of ablation studies of without-HL, without-mixing network, and HCMARL-VNE. (a) Long-term average revenue results. (b) Long-term average cost results. (c) Long-term average R/C ratio results. (d) Long-term average acceptance ratio results.

resources, so the acceptance ratio gradually decreases, and after some time, it converges to a certain value based on the performance of each algorithm. As also shown in the figure, the proposed HCMARL-VNE algorithm exhibits the best performance in terms of the acceptance ratio. Because it allocates the CPU and bandwidth resources efficiently, it can embed more VNRs than others VNE algorithms. We assert that the reason for this outcome is attributed to the effective learning of the HL agent, enabling it to strategically perform the "Postponing" action for a waiting VNR. Furthermore, the cooperation between multiple HL agents in selecting VNRs to be embedded also contributes to achieving optimal results.

C. Ablation Study

In this section, we show how effective the HRL and MARL used in HCMARL-VNE are in improving the performance of the VNE algorithm in an explicit manner. We implement two comparative versions of HCMARL-VNE. The first one is "without-HL," which just uses the LL single agent. This algorithm treats all VNRs equally and allows agents to process VNRs with the highest revenue first, ignoring the long-term impact of each VNR embedding. The second one is a "without-mixing network," which still uses the HL agents, but does not utilize the mixing network designed for cooperation in the training of the HL agents. This removes the part where each agent comprehensively evaluates the actions selected by others through a mixing network for cooperation during training.

The results are shown in Fig. 11, which clearly demonstrates that the proposed scheme outperforms the two comparative versions of it in terms of all performance metrics. Furthermore, it indicates that the performance is significantly lower when using only the LL single agent. Additionally, it shows the beneficial impact of having multiple HL agents on overall performance, even in the absence of the mixing network.

VI. Conclusion

The study proposes a new approach called HCMARL-VNE to efficiently solve the VNE problem through hierarchical cooperative MARL. This approach is novel and aims to improve upon existing methods for addressing the VNE problem. It divides the VNR embedding task into two levels of agents: 1) HL agent and 2) LL agent. The HL agent's role is to choose the most feasible VNR from the waiting queue with the highest long-term revenue potential. The LL agent's responsibility is to embed the selected VNR onto the substrate network while taking into account the embedding cost. This hierarchical approach aims to improve the efficiency and effectiveness of VNR embedding. The simulation results demonstrate that HCMARL-VNE outperforms existing VNE algorithms in terms of long-term revenue, R/C ratio, acceptance ratio, and reduced long-term cost. These results suggest that HCMARL-VNE has the potential to provide significant improvements in VNR embedding efficiency and cost-effectiveness. Therefore, the proposed HCMARL-VNE has the potential to efficiently allocate network resources

for various VNRs from SPs. This could be a significant contribution to NS and NV, especially in the context of the expanding 5G network and IoT market. As a future research direction, HCMARL-VNE could be extended by incorporating an explicit RL-based optimal path algorithm to further enhance the performance of link embedding. Additionally, we will further enhance the performance of the algorithm and address the issue of long learning time.

REFERENCES

[1] J. A. Stankovic, "Research directions for the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014.

[2] I. Alam et al., "A survey of network virtualization techniques for Internet of Things using SDN and NFV," *ACM Comput. Surveys*, vol. 53, no. 2, pp. 1–40, 2020.

[3] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 9, no. 4, pp. 373–392, Dec. 2012.

[4] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[5] X. Li, C. Guo, J. Xu, L. Gupta, and R. Jain, "Towards efficiently provisioning 5G core network slice based on resource and topology attributes," *Appl. Sci.*, vol. 9, no. 20, p. 4361, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/20/4361

[6] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, 2008.

[7] X. Cheng et al., "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 38–47, 2011.

[8] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on the degree and clustering coefficient information," *IEEE Access*, vol. 4, pp. 8572–8580, 2016.

[9] M. Feng, L. Zhang, X. Zhu, J. Wang, Q. Qi, and J. Liao, "Topology-aware virtual network embedding through the degree," in *Proc. Nat. Doctoral Acad. Forum Inf. Commun. Technol.*, 2013, pp. 1–6.

[10] Z. Wang, Y. Han, T. Lin, H. Tang, and S. Ci, "Virtual network embedding by exploiting topological information," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2012, pp. 2603–2608.

[11] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 1–9.

[12] H. Cao, Y. Zhu, L. Yang, and G. Zheng, "A efficient mapping algorithm with novel node-ranking approach for embedding virtual networks," *IEEE Access*, vol. 5, pp. 22054–22066, 2017.

[13] S. Haeri and L. Trajković, "Virtual network embedding via monte carlo tree search," *IEEE Trans. Cybern.*, vol. 48, no. 2, pp. 510–521, Feb. 2018.

[14] I. Ullah, H.-K. Lim, and Y.-H. Han, "Ego network-based virtual network embedding scheme for revenue maximization," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)* 2021, pp. 155–160.

[15] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[16] L. Wang, H. Qu, J. Zhao, and Y. Guo, "Virtual network embedding with discrete particle swarm optimisation," *Electron. Lett.*, vol. 50, no. 4, pp. 285–286, 2014.

[17] K. T. Nguyen and C. Huang, "Distributed parallel genetic algorithm for online virtual network embedding," *Int. J. Commun. Syst.*, vol. 34, no. 4, 2021, Art. no. e4691.

[18] P. Zhang, Y. Hong, X. Pang, and C. Jiang, "VNE-HPSO: Virtual network embedding algorithm based on hybrid particle swarm optimization," *IEEE Access*, vol. 8, pp. 213389–213400, 2020.

[19] L. Boyang, W. Muqing, and Z. Haosen, "Virtual network embedding based on hybrid adaptive genetic algorithm," in *Proc. IEEE 5th Int. Conf. Comput. Commun. (ICCC)*, 2019, pp. 1197–1202.

[20] A. Song, W.-N. Chen, T. Gu, H. Yuan, S. Kwong, and J. Zhang, "Distributed virtual network embedding system with historical archives and set-based particle swarm optimization," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 2, pp. 927–942, Feb. 2021.

[21] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[22] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.

[24] M. He, L. Zhuang, S. Tian, G. Wang, and K. Zhang, "Multi-objective virtual network embedding algorithm based on Q-learning and curiosity-driven," *EURASIP J. Wireless Commun. Netw.*, vol. 2018, no. 1, pp. 1–12, 2018.

[25] S. Wang, J. Bi, J. Wu, A. V. Vasilakos, and Q. Fan, "VNE-TD: A virtual network embedding algorithm based on temporal-difference learning," *Comput. Netw.*, vol. 161, pp. 251–263, Oct. 2019.

[26] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "Evolutionary actor-multi-critic model for VNF-FG embedding," in *Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC)* 2020, pp. 1–6.

[27] Y. Yuan, Z. Tian, C. Wang, F. Zheng, and Y. Lv, "A Q-learning-based approach for virtual network embedding in data center," *Neural Comput. Appl.*, vol. 32, no. 7, pp. 1995–2004, 2020.

[28] H. Afifi and H. Karl, "Reinforcement learning for virtual network embedding in wireless sensor networks," in *Proc. 16th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, 2020, pp. 123–128.

[29] C. Wang et al., "Modeling on virtual network embedding using reinforcement learning," *Concurr. Comput. Pract. Exp.*, vol. 32, no. 23, 2020, Art. no. e6020.

[30] D. Andreoletti, T. Velichkova, G. Verticale, M. Tornatore, and S. Giordano, "A privacy-preserving reinforcement learning algorithm for multi-domain virtual network embedding," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2291–2304, Dec. 2020.

[31] S. Zhang, C. Wang, J. Zhang, Y. Duan, X. You, and P. Zhang, "Network resource allocation strategy based on deep reinforcement learning," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 86–94, 2020.

[32] M. Elkael, M. A. Aba, A. Araldo, H. Castel-Taleb, and B. Jouaber, "Monkey business: Reinforcement learning meets neighborhood search for virtual network embedding," *Comput. Netw.*, vol. 216, Oct. 2022, Art. no. 109204.

[33] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[34] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020.

[35] A. Rkhami, T. A. Q. Pham, Y. Hadjadj-Aoul, A. Outtagarts, and G. Rubino, "On the use of graph neural networks for virtual network embedding," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, 2020, pp. 1–6.

[36] P. Zhang, C. Wang, N. Kumar, W. Zhang, and L. Liu, "Dynamic virtual network embedding algorithm based on graph convolution neural network and reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9389–9398, Jun. 2022.

[37] J. Cheng, Y. Wu, Y. Lin, E. Yuepeng, F. Tang, and J. Ge, "VNE-HRL: A proactive virtual network embedding algorithm based on hierarchical reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4075–4087, Dec. 2021.

[38] H. Zhou, M. Elsayed, and M. Erol-Kantarci, "RAN resource slicing in 5G using multi-agent correlated Q-learning," in *Proc. IEEE 32nd Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, 2021, pp. 1179–1184.

[39] M. Sulaiman, A. Moayyedi, M. Ahmadi, M. A. Salahuddin, R. Boutaba, and A. Saleh, "Coordinated slicing and admission control using multi-agent deep reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 2, pp. 1110–1124, Jun. 2023.

[40] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Comput. Netw.*, vol. 30, pp. 107–117, Apr. 1998. [Online]. Available: http://www-db.stanford.edu/ backrub/google.html

[41] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–18.

[42] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," 2015, *arXiv:1507.00814*.

[43] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," 2015, *arXiv:1506.03134*.

[44] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," 2016. *arXiv:1602.01783*.
[45] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–9.
[46] Y. Ma et al., "A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds. Red Hook, NY, USA: Curran Assoc., 2021.
[47] A. S. Vezhnevets et al., "FeUdal networks for hierarchical reinforcement learning," 2017, *arXiv:1703.01161*.
[48] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of Reinforcement Learning and Control. Studies in Systems, Decision and Control*. Cham, Switzerland: Springer, 2021, pp. 321–384.
[49] A. Oroojlooy and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *Appl. Intell.*, vol. 53, pp. 13677–13722, Oct. 2022.
[50] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor–critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6382–6393.
[51] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 2145–2153.
[52] P. Sunehag et al., "Value-decomposition networks for cooperative multi-agent learning," 2017, *arXiv:1706.05296*.
[53] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 7234–7284, 2020.
[54] D. Ha, A. M. Dai, and Q. V. Le, "HyperNetworks," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–18.
[55] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.

Hyun-Kyo Lim received the B.S. degree in computer science and engineering and the M.S. degree in computer science engineering from Korea University of Technology and Education, Cheonan, South Korea, in 2015 and 2017, respectively, and the Ph.D. degree from the Department of Interdisciplinary Program in Creative Engineering, Korea University of Technology and Education, in 2022.

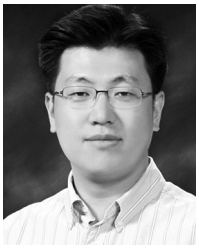
Since 2022, he has been a Postdoctoral Researcher with the Future Convergence Engineering, Korea University of Technology and Education. He studied mobility management during his master's course and he especially researched distributed mobility management in software-defined networking. He is studying deep learning and reinforcement learning during his doctoral and postdoctoral studies. He is also exploring ways to apply deep learning and reinforcement learning to the network and is working on applying deep learning and reinforcement learning to a variety of applications.

Ihsan Ullah received the B.S. and M.S. degrees in computer science from the University of Peshawar, Peshawar, Pakistan, in 2001 and 2004, respectively, and the Ph.D. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2019.

From September 2019 to August 2020, he was a Postdoctoral Research Fellow with the Ubiquitous Computing Technology Research Institute, Sungkyunkwan University. Since 2020, he has been a Research Professor with the School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, South Korea. His research interests include data aggregation, data fusion, virtual network embedding, network slicing (5G), Internet of Things, artificial intelligence, deep reinforcement learning, and cloud computing.

Ju-Bong Kim received the B.S. degree, the M.S. degree, and the Ph.D. degree in computer engineering from Korea University of Technology and Education, Cheonan, South Korea, in 2017, 2019, and 2022, respectively.

Since 2022, he has been a Postdoctoral Researcher of Computer Engineering with Korea University of Technology and Education. He has been continuously conducting research applying deep reinforcement learning to various applications, such as machine autonomous control, financial portfolios, and smart factory since 2017. Particularly, during his doctoral studies, he focused on research aimed at enhancing the exploration performance of agents in multiagent reinforcement learning.



Youn-Hee Han (Member, IEEE) received the B.S. degree in mathematics and the M.S. and Ph.D. degrees in computer science and engineering from Korea University, Seoul, South Korea, in 1996, 1998, and 2002, respectively.

From 2002 to 2006, he was a Senior Researcher with the Next Generation Network Group, Samsung Advanced Institute of Technology, Suwon, South Korea. Since 2006, he has been a Professor with the School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, South Korea. He also served as a Visiting Professor with the Department of Computer Science, State University of New York at Albany, Albany, NY, USA, from September 2013 to January 2015. Since 2002, his activities have focused on mobility management, media-independent handover, and cross-layer optimization for efficient mobility support. He has published approximately 270 research articles on the theory and application of mobile computing and has led 45 patents regarding the information and communication technology domain. He is currently very interested in artificial intelligence technology, especially reinforcement learning, and he has been in charge of many research projects regarding improving the performance of reinforcement learning algorithms for various fields, such as intelligent networking on 5G and 6G, Internet of Things, economics, and financial engineering. He has made several contributions to IETF and IEEE standardization. He actively participated in the IEEE 802.21 Working Group and was also an author of IETF RFC 5181, RFC 5270, and RFC 7864. His current research interests include the theory and application of computer networks, including protocol design and mathematical analysis, mobile sensor/actuator networks, social network analysis, machine learning, deep learning, and reinforcement learning.