A Two-Stage GCN-Based Deep Reinforcement Learning Framework for SFC Embedding in Multi-Datacenter Networks

Da Xiao, *Student Member, IEEE*, J. Andrew Zhang, *Senior Member, IEEE*, Xin Liu, *Fellow, IEEE*, Yiwen Qu, Wei Ni, *Senior Member, IEEE*, and Ren Ping Liu, *Senior Member, IEEE*

***Abstract*—Network Function Virtualization (NFV), which decouples network functions from hardware and transforms them into Virtual Network Functions (VNFs), is a crucial technology for data center (DC) networks. A service function chain (SFC) is composed of an ordered set of VNFs and virtual links (VLs) connecting them. To optimize the resource allocation in DC networks, we need to efficiently map SFCs onto the physical network. Nevertheless, the dynamics and diversity of SFC requests in multi-datacenter (MDC) networks pose a significant challenge in embedding SFCs. To overcome this challenge, we design a two-stage graph convolutional network (GCN) assisted deep reinforcement learning (DRL) scheme. This framework aims to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network. In the first stage, we propose a GCN-based DRL algorithm as a coarse granularity solution to the SFC embedding problem from the macro perspective. This solution outlines a local observation scope (LOS) for each agent in the multi-agent system of the second stage, where all agents simultaneously handle SFC requests from their respective DCs using a multi-agent framework from the micro perspective. Numerical evaluations show that, compared to state-of-the-art methods, the proposed scheme improves the acceptance ratio by approximately 13% compared with the Kolin algorithm and 18% compared with the DQN algorithm and saves the cost by around 28% compared with the Kolin and the DQN.**

***Index Terms*—Network Function Virtualization, Multi-Datacenter, Multi-Agent, Graph Convolutional Network**

I. Introduction

Nowadays, datacenters (DCs) have become a key component in supporting cloud computing, large-scale online services, and geographically distributed applications. To provide high-quality and non-disruptive network services to end-users, large enterprises such as Google, Facebook, and Amazon have created multiple DCs in different geographic regions and built inter-DC networks to interconnect them. A network service can be symbolized as a service function chain (SFC) or a virtual network functions forwarding graph (VNF-FG). An SFC is composed of an ordered set of VNFs and virtual links (VLs), realized by DC network resources (e.g., bandwidth (BW), central processing unit (CPU) cycles, and memory space) using network function virtualization (NFV). A successful embedding of an SFC requires that both the resource constraints of the infrastructure and the service level agreement (SLA) [1] of the network service be satisfied. To accommodate as many network services as possible with limited resources, efficient and automatic placement of SFCs becomes increasingly important.

A lot of research efforts have aimed to address the SFCs (the predefined VNFs and the VLs) placement problem. Some are heuristic-based methods that perform well in stationary systems but could have degraded performance in dynamic systems [2]–[7]; others are deep reinforcement learning (DRL)-based approaches, which are efficient when properly designed but inefficient when their state or action spaces, or reward functions are incorrectly designed [8]–[11]. Despite that these works demonstrate the potential of SFC placement optimization, research on this problem in multi-datacenter (MDC) networks is limited. Most of such works consider simple models by treating the MDC as a whole and assuming that the arrival of SFC requests follows the Poisson traffic model; thus, the load diversity of different DCs is generally neglected. In practice, the SFC requests load in an MDC may vary from DC to DC, e.g., some DCs may suffer traffic congestion while others are under-utilized. Hence, it is essential to design a load balancing algorithm to resolve the SFC requests load diversity issue in MDC networks.

In this paper, to address these issues, we propose a two-stage graph convolutional network (GCN)-based DRL framework to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network.

In the first stage, we propose a DRL algorithm – GCN-based Proximal Policy Optimization (PPO) to derive a coarse granularity solution to SFC embedding from the macro perspective. Specifically, in each DC, we treat the IT resources requested by all SFCs in the waiting queue of this DC as a whole; thus, we can abstract the placement of all the SFCs received by the MDC into a load balance problem. To balance loads in the MDC, we transfer some SFCs accumulated in high-load DCs to low-load DCs. We model the load transfer process as a Markov decision process (MDP) and represent the features of the MDC network as the MDP states and the possible DC-to-DC transfer options as the MDP actions. We use the reward function to capture the success/failure of an SFC transfer and the cost of network traffic caused by this transfer. At each time step of the MDP, the agent chooses an action based on our GCN-based PPO model and performs the chosen action in the MDC environment. In response to the action, an MDP

D. Xiao, J. A. Zhang, Y. Qu, and R. P. Liu are with the School of Electrical and Data Engineering, UTS, Sydney, Australia (e-mail: {Da.Xiao; Yiwen.Qu-1}@student.uts.edu.au; {Andrew.Zhang, RenPing.Liu}@uts.edu.au). X. Liu is with the Department of Computer Science, UC Davis, CA, USA (e-mail: xinliu@ucdavis.edu). W. Ni is with the Data61, CSIRO, Sydney, Australia (e-mail: Wei.Ni@data61.csiro.au).

reward, which will be used to train our model, is returned to the agent. Once trained, the model approximates the optimal solution for maximizing the overall acceptance ratio of SFC requests while minimizing the total cost. This stage aims to set a local observation scope (LOS) for each agent of the multi-agent framework in the second stage.

The second stage consists of two phases. During the first phase, to accommodate as many SFC requests as possible with limited resources, each agent embeds SFCs in its affiliated DC. The second phase commences once a high-load DC embeds SFCs in low-load DCs. In this phase, multiple agents of high-load DCs handle SFC requests simultaneously, with each embedding VNFs and VLs within its LOS step by step. Thus, we model the SFC embedding problem in this phase as a multi-agent MDP [12] and solve it using a MARL approach, GCN-based multi-agent PPO (MAPPO). A multi-agent MDP is defined by a set of states $\mathcal{S}$ that describes the possible global environmental features as well as a set of actions $\mathcal{A}_i$ $(i = 1, 2, \ldots, N)$ and a set of local observations $\mathcal{O}_i$ $(i = 1, 2, \ldots, N)$ for each agent. To choose actions, agent i feeds its observation o_i into the policy π_{θ_i} and gets a_i. The joint action of all agents produces the next global state according to the state transition function $\mathrm{T}(s; a_1, a_2, \ldots, a_N) \rightarrow s'$, where N is the number of agents. Meanwhile, agent i obtains rewards based on a function of the global state and the agent's action $\mathrm{R}_i(s, a_i) \rightarrow r_i$ and receives a private observation correlated with the state $s' \rightarrow o_i$. Each agent aims to maximize its own total expected return $\sum_{t=0}^{T} \gamma^t r_{i,t}$ $(i = 1, 2, \ldots, N)$, where γ is a discount factor and T is the time horizon. In our multi-agent MDP, for agent i, we represent the features of the local network that agent i can sense, the position of the VNF embedded at the previous time step, and the index of the agent's affiliated DC as the local observations. In addition, the possible placements of the current VNF and the VL connecting this VNF and the previous constitute the action space. The reward function R_i consists of an external part and an internal part. The former is designed to achieve the common goal of all agents – maximizing the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC environment. The latter is designed to accelerate the training. With training, each agent of the MAPPO model knows how to embed SFCs within its LOS to approximate the optimal solution.

The main contributions of this paper can be summarized as follows:

1) We propose a two-stage GCN-based DRL framework to handle service requests simultaneously from all DCs with the aim of maximizing the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC network. This work effectively addresses the limitation in existing research on embedding SFCs in MDC networks by considering SFC requests load diversity of different DCs.
2) We abstract the placement of all SFCs within a time slot into a load balance problem from the macro perspective in the first stage. To balance loads in the MDC environment, we model the load transfer process as an MDP and develop a coarse granularity solution to the placement of SFCs such that the acceptance ratio of SFC requests is maximized while the total cost is minimized. The solution provides an appropriate LOS for each agent in the multi-agent framework; thus, we can tailor the action space of each agent to a proper size, which speeds up the convergence rate of the algorithm in the second stage.
3) In the second stage, all agents handle SFCs received by their respective DCs simultaneously. In the initial phase, each agent places SFCs in its affiliated DC. In the second phase, we model the SFC embedding problem as a multi-agent MDP from the micro perspective and solve it using a MARL approach, GCN-based MAPPO. This multi-agent framework achieves better efficiency than single-agent RL algorithms in resolving the SFC requests load diversity issue in MDC networks.

The rest of the paper is organized as follows. In Section II, we discuss the related works. Then, we present the system model of the SFC embedding problem in an MDC network and formulate the problem in Section III. In Section IV, we present our algorithm. And in Section V, we numerically evaluate the performance of our algorithm. Finally, we conclude the paper in Section VI.

II. Related Works

Many studies have been devoted to the placement of network services techniques [13]–[15]. Since our objective is to deploy as many SFCs as possible with limited hardware resources in an MDC, we pay close attention to works researching the VNF-FG embedding issue in a single DC or an MDC. Also, we have studied several typical works regarding the VNF-FG placement problem. In general, they fall into the following two categories.

A. Heuristic-Based Approaches

Most techniques were proposed to formulate and solve optimization problems [3]–[5], [16]–[20]. Some works, [16], [17], and [18], are based on a strong assumption. In [16], the authors proposed a heuristic algorithm named LBA (LCS Based Algorithm) to realize efficient VNF service chaining in inter-DC elastic optical networks (EONs). Based on the Longest Common Subsequence (LCS) principle, this algorithm reuses the VNFs that have been deployed previously in multiple DCs to save IT cost. To reduce the inter-DC traffic cost, it places new instances of VNFs in DCs based on the nearby principle; thus, VNFs may concentrate in some DCs, while others are under-utilized. Consequently, some DCs might suffer traffic congestion or performance degradation. To resolve this issue, the authors of [17] proposed a resource balancing algorithm (RBA) for joint placement of service chains and routing/spectrum assignment (RSA) in inter-DC EONs. Also, in [18], a joint-optimization selection (JOS) algorithm was designed to select VNFs to achieve joint load balancing of IT and spectrum resources. In all these works, the VNF reuse solution improves resource utilization. This solution is based on the assumption that the capacity of a VNF is fixed, which is reasonable during the early stage of virtualization [21].

Nevertheless, when we walk into the Kubernetes era, VNFs are embedded in containers, for which CPU allocation can be as small as 1/1000 of the total CPU resources of a server [22]. Hence, the resource allocation for a VNF can be flexible, and we can initiate VNFs instances based on the number of virtual resources requested in the corresponding service requests. Furthermore, for security purposes, many customers are unwilling to share VNFs with others. Therefore, in our work, we consider dynamic and unshared VNFs.

Other papers, [3]–[5] and [19], focus on the convergence rate of their algorithms. In [19], the authors formulated the VNF placement as a binary integer programming model. They proposed the service function chains embedding approach (SFC-MAP) algorithm, in which the shortest path algorithm is iterated based on a multi-layer cost graph to acquire the optimal placement solution. The authors of [3] designed an eigendecomposition-based approach to address the VNF-FG placement issue. This matrix-based method reduces the complexity and accelerates the convergence. However, it does not fully explore the immense space of possible actions. A novel approach that combines Markov approximation with matching theory, sampling-based Markov approximation (SAMA), was proposed in [4] to minimize the joint operational and traffic cost. In [5], the authors formulated the VNF placement and flow routing problems as integer linear programming optimization problems and designed a set of heuristics to find near-optimal solutions. In our work, we accelerate the convergence rate partly by restricting the cooperation among multiple agents within a small scope and partly by inheriting the strengths of the PPO algorithm. Although the convergence rate is important, effectiveness is more important regarding our objective. Therefore, we tune the clipping ratio hyperparameter of our DRL framework to balance effectiveness and efficiency.

The authors in [20] readjusted the running SFCs when users move or SFCs requests change, which makes this paper very attractive and innovative. In our work, we consider SFCs' lifetime, and thus our scenarios could introduce more dynamics.

In summary, heuristics are efficient in stationary systems. Nevertheless, in scenarios where environmental characteristics, such as the arrival rate and the required resources of service requests, change dynamically, these approaches must be frequently re-triggered to obtain the optimal solutions for new environments. In comparison, our proposed GCN-based RL algorithm, whose agents interact with the environment during the training phase, can learn and adapt to environmental changes.

B. Reinforcement Learning-Based Approaches

Various techniques based on RL have also been developed [8], [23]–[26]. The ability to learn from experience makes these approaches noteworthy. The authors in [23] presented a DRL-based VNF-SC provisioning algorithm in inter-DC EONs. To solve the problem that the size of the state vector is not fixed because of the variable length of SFCs and the constantly changing number of available DCs, a feature matrix-based encoding scheme was introduced. This innovation makes RL-based algorithms feasible in network environments whose state sizes are prone to change. In [24], the authors proposed DeepRMSA, a DRL-based routing, modulation, and spectrum assignment (RMSA) framework for learning the optimal online RMSA policies in EONs. The authors in [25] developed a hybrid DRL-based framework for service function chaining across geo-distributed DCs. In [23], [24] and [25], the authors used deep neural networks (DNN) to extract the features of the complex EON states. However, for an environment that can be represented by a graph, Graph neural networks (GNN)-based encoding schemes might be more efficient in extracting its features. The authors in [26] designed a GNN-based hierarchical DRL algorithm for the VNF placement and RSA in EONs. They proposed a GNN-based encoder to abstract the features of the EON network states and the context of the pending VNFs. This GNN-based encoder advances the VNF-FG placement in network topologies that a graph can normally represent. However, as the authors in [8], [25], and [27] did, they decoupled the placement of VNFs and VLs and designed a routing module to select the optimal route given a VNF placement. This scheme reduces the action-space size and speeds up the convergence rate. Nevertheless, the greed for a high immediate reward might result in a disappointing long-term cumulative reward.

Most works on the MDC issue [16]–[19], [23]–[26] treated the MDC network as a whole and assumed that the arrival of SFC requests follows the Poisson distribution. However, in some real cases, the diversity of SFC requests in an MDC may pose an important challenge. Therefore, in our work, we design a two-stage GCN-based RL algorithm for the SFC embedding problem in an MDC network, where the SFC requests load varies from DC to DC.

III. System Model and Problem Formulation

A. System Architecture

According to 3rd Generation Partnership Project (3GPP) [28], [29], the NFV framework receives requests for allocating network services with specific characteristics. Accordingly, at the core of the NFV architecture, the management and orchestration (MANO) framework plans SFCs allocation using the NFV infrastructure.

Inspired by the multi-tier SDN-controller system proposed in [30], we design a multi-tier MANO system depicted in Fig.1. In the first stage, the upper-tier MANO cooperates with all lower-tier MANOs in setting a LOS for each of them. In the second stage, a discrete time-slot system is considered as in [8]. At the beginning of each time slot, the resources occupied by expired SFCs are released, and all lower-tier MANOs start to embed SFCs simultaneously. For each DC, based on its current state, the corresponding lower-tier MANO handles the SFCs received by this DC one after another. Once a DC has insufficient resources to accommodate the SFCs waiting in its queue, the lower-tier MANO in charge of this DC will place SFCs in low-load DCs within its LOS.

B. NFV Infrastructure

We model the MDC network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of DC nodes and $\mathcal{E}$ is the set of

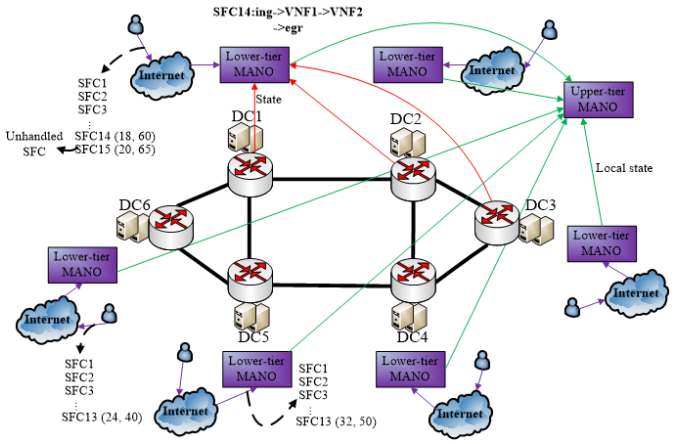

Fig. 1: Network infrastructure considered in this paper.

transmission links that interconnect these DC nodes. Hence, the numbers of DC nodes and physical links are $|\mathcal{V}|$ and $|\mathcal{E}|$, respectively. Furthermore, the CPU capacity and the remaining CPU resources of DC v are denoted by $\{C_v|C_v \geq 0; v \in \mathcal{V}\}$ and $\{c_v|c_v \geq 0; v \in \mathcal{V}\}$, respectively. The BW capacity and the remaining BW of link (v,u) are denoted by $\{B_{v,u}|B_{v,u} \geq 0, (v,u) \in \mathcal{E}\}$ and $\{b_{v,u}|b_{v,u} \geq 0, (v,u) \in \mathcal{E}\}$, respectively.

C. Characteristics of Service Requests

We denote M as the number of service requests to be handled within a time slot. Service request i can be defined as $\text{SR}_i = \{F_i, C_i, B_i\}$, where B_i indicates the BW requested by SFC i, $F_i = \{F_{i0}, F_{i1}, F_{i2}, ..., F_{iK_i}, F_{i(K_i+1)}\}$ (K_i is the number of VNFs in SFC i) denotes the ingress (F_{i0}), egress ($F_{iK_{i+1}}$), and VNFs ($\{F_{i1}, F_{i2}, ..., F_{iK_i}\}$) that are used to compose SFC i, and $C_i = \{C_{i1}, C_{i2}, ..., C_{iK_i}\}$ indicates the CPU requested by each VNF in SFC i. The VL between the j^{th} and $(j+1)^{\text{th}}$ nodes of SFC i is denoted by $(F_{ij}, F_{i(j+1)})$. In our scenario, SFCs come and go, and the life-time of an SFC can span over one or multiple sequential time slots.

D. Cost Model

1) Traffic Cost: In a service chain, one VNF must forward packets to the next through the VL connecting them. The VL can be embedded in one or multiple physical links, each of which might have a different hop distance depending on the network topology. As in [4], we define the traffic cost caused by a VL connecting two adjacent VNFs as the product of the overall distance between these two VNFs and the BW of the VL. Since the distance between two DCs is far longer than that between two servers within a DC, we only consider the traffic in the inter-DC network as in [23] and [26]. As a result, the total traffic cost can be written as

$$C_{\text{tr}} = \beta \sum_{i=1}^{M} \sum_{j=0}^{K_i} \sum_{(v,u)\in\mathcal{E}} z^{v,u}_{F_{ij},F_{i(j+1)}} \cdot B_i \cdot \text{Dis}\,(v,u)\,, \qquad (1)$$

where β is the price that converts the traffic cost to a monetary cost, binary $z^{v,u}_{F_{ij},F_{i(j+1)}}$ indicates whether the VL connecting the j^{th} and $(j+1)^{\text{th}}$ nodes of request i is embedded in physical link (v,u), and $\text{Dis}\,(v,u)$ is the distance between DC v and DC u.

2) Server Cost: Compared with memory and capacity, CPU's computational power is much more important. Therefore, as for server cost, we only consider CPU and the extension to multiple resource types is straightforward. The total server cost can be written as

$$C_{\text{se}} = \gamma \sum_{i=1}^{M} \sum_{j=1}^{K_i} \sum_{v\in\mathcal{V}} x^v_{i,j} \cdot C_{ij}, \qquad (2)$$

where γ is the price that converts the server cost to a monetary cost, and binary $x^v_{i,j}$ indicates whether the j^{th} VNF of request i nests in DC v.

E. Problem Formulation

In this part, we formulate the objective as an optimization problem. Our purpose is to maximize the overall acceptance ratio of SFC requests while minimizing the total cost regarding the infrastructure resource constraints. We first elaborate on the constraints of the SFC embedding problem. For any DC node, the computing resource constraint must be satisfied; thus, we have

$$\sum_{i=1}^{M} \sum_{j=1}^{K_i} x^v_{i,j} \cdot C_{ij} \leq C_v, \qquad \forall v \in \mathcal{V}. \qquad (3)$$

For any physical link, the BW constraint must be satisfied; therefore, we have

$$\sum_{i=1}^{M} B_i \sum_{j=0}^{K_i} z^{v,u}_{F_{ij},F_{i(j+1)}} \leq B_{v,u}, \qquad \forall (v,u) \in \mathcal{E}. \qquad (4)$$

Each VNF can be deployed at only one DC node; therefore, we have

$$\sum_{v\in\mathcal{V}} x^v_{i,j} \leq 1, \qquad (1 \leq i \leq M, 1 \leq j \leq K_i). \qquad (5)$$

To enforce a continuous substrate path connecting the j^{th} and $(j+1)^{\text{th}}$ nodes of request i, we have

$$\sum_{(v,u)\in\mathcal{O}(v)} z^{v,u}_{F_{ij},F_{i(j+1)}} - \sum_{(u',v)\in\mathcal{I}(v)} z^{u',v}_{F_{ij},F_{i(j+1)}} = x^v_{i,j} - x^v_{i,j+1},$$
$$\forall v \in \mathcal{V}, 1 \leq i \leq M, 0 \leq j \leq K_i, \qquad (6)$$

where $\mathcal{I}(v)$ and $\mathcal{O}(v)$ denote the sets of incoming links and outgoing links of DC v, respectively.

As in [5], we use a constraint to avoid a loop as follows:

$$\sum_{(v,u)\in\mathcal{O}(v)} z^{v,u}_{F_{ij},F_{i(j+1)}} + \sum_{(u',v)\in\mathcal{I}(v)} z^{u',v}_{F_{ij},F_{i(j+1)}} \leq 1,$$
$$\forall v \in \mathcal{V}, 1 \leq i \leq M, 0 \leq j \leq K_i. \qquad (7)$$

An SFC request is accepted if and only if all virtual elements in this SFC are successfully deployed; hence, we can denote the number of accepted requests as

$$A = \sum_{i=1}^{M} f((\sum_{j=1}^{K_i} \sum_{v\in\mathcal{V}} x^v_{i,j} - K_i) + (\sum_{j=0}^{K_i} \sum_{(v,u)\in\mathcal{E}} z^{v,u}_{F_{ij},F_{i(j+1)}} - (K_i+1))), \qquad (8)$$

where

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0. \end{cases} \tag{9}$$

Since our object is to maximize the overall acceptance ratio of SFC requests while minimizing the total cost, the optimization problem can be written as

$$(P1): \max_{x^v_{i,j}, z^{v,u}_{F_{ij},F_{i(j+1)}}} U(x^v_{i,j}, z^{v,u}_{F_{ij},F_{i(j+1)}}) = \alpha_1 \cdot A - \alpha_2 \cdot (C_{\text{tr}} + C_{\text{se}})$$
$$s.t. (3), (4), (5), (6), (7).$$

In the objective function above, α_1 and α_2 are the weights of two objectives. We normalize two objects before assigning weights to them. Since the objective of maximizing the acceptance ratio of SFC requests is more important, we assign a higher weight to it.

It can be challenging to solve this optimization problem using heuristic approaches because it is difficult to track the dynamics of the network. Alternatively, learning-based techniques can be utilized to learn the network dynamics, such as the arrival patterns and the resource requirements of various services, and solve such a problem. The learning technique aims to learn a policy that determines what action to take in each state. In the following, we introduce a two-stage GCN-based DRL for the SFC embedding problem in our MDC environment.

IV. Proposed Two-stage SFC Embedding Scheme

In this section, we present our proposed algorithm. First, the background of the proposed algorithm is provided. Then, we detail our two-stage GCN-based DRL algorithm for solving P1.

A. Background of the Proposed Algorithm

1) Model Our Environment as a Multi-agent MDP: The SFC embedding problem has been modeled as an MDP in many existing works. In our MDC environment, if we model this problem as an MDP, the agent needs to handle SFC requests received by all DCs; thus, the action space would be too large for the DRL algorithm to converge, especially in a large-scale MDC. Alternatively, we study a multi-agent MDP, which has already been introduced in Section I. The details of the state representation, action space, and reward function for each agent in the context of our problem are provided in Section IV-C.

2) Overview of Multi-agent DRL and Motivation for Using MAPPO: MARL algorithms have been widely adopted to solve the multi-agent MDP models. The authors in [31] proposed the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm for mixed cooperative-competitive environments. It is a simple extension of actor-critic policy gradient methods where the critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. In the MADDPG algorithm, each agent learns a policy network $\pi_i(a_i|o_i)$ and a centralized critic network $Q^\pi_i(\mathbf{x}, a_1, \ldots, a_N)$, where $\mathbf{x}$ consists of the observations of all agents: $\mathbf{x} = (o_1, \ldots, o_N)$. This algorithm follows the framework of centralized training with decentralized execution (CTDE). Although the DDPG, which follows the policy gradient to find the optimal actions, is more efficient than the value-based DRL algorithms, it does not consider the step size of each policy update. To resolve this issue, OpenAI released the PPO [32] algorithm in 2017. Its core idea is that the new policy should not be too far from the old one after an update. To achieve that, it uses a ratio to symbolize the difference between the new policy and the old one and clips this ratio within $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter. The authors in [33] investigated the MAPPO algorithm in three popular multi-agent testbeds: the particle-world environments, the Starcraft multi-agent challenge, and the Hanabi challenge. They found that the MAPPO achieves surprisingly strong performance while exhibiting comparable sample efficiency with the MADDPG.

3) Motivation for Using GCN: Our MDC network topology is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ indicates the set of DC nodes and $\mathcal{E}$ indicates the set of links. Since we model our environment as a multi-agent MDP, it is essential to design an MDP state space that comprehensively represents the features of the environment. Feeding the traditional DNN or Convolutional Neural Networks (CNN) with a flat feature vector containing all the state information is the most straightforward option. Nevertheless, this approach cannot scale well to complex topology structures, because it would be difficult for DNN or CNN to analyze the relations between every pair of nodes in a graph with handcrafted feature engineering. Therefore, we design an encoder for analyzing $\mathcal{G}$ based on GCN [34], which has been widely exploited in [35]–[37] to extract the features of undirected graphs through aggregating the characteristics of nodes and topologies.

4) Two-stage Design: In our MAPPO framework, each agent learns a local policy network and a centralized critic network. As in [31], for each agent, the critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. Here comes a question: how shall we set an observation scope for each agent? Let's consider the state of the environment depicted in Fig.1. If each agent (lower-tier MANO) can observe the information of all DCs, the MAPPO could converge to the optimum, but the convergence rate might be slow as each agent must cooperate with all the other agents during the training process; If we restrict the LOS of each agent within its one-hop neighbors' area, the MAPPO could converge much faster but might not be able to find the global optimum because of the restricted view of each agent. Therefore, to reach an ideal optimum as quickly as possible, we must design an appropriate LOS for each agent before running our multi-agent algorithm.

B. The First Stage

In this stage, we first model the load transfer process as an MDP. Then, we propose a GCN-based PPO to solve the MDP.

1) Model the Load Transfer Process as an MDP: The authors in [30] proposed a multi-tier SDN-controller system for

the space-air-ground integrated network. Similarly, we propose a multi-tier MANO system, in which an upper-tier MANO assembles local network states from all lower-tier MANOs and schedules resources for SFC requests globally. In this stage, the upper-tier MANO balances the SFC requests loads of multiple DCs from a macro perspective, aiming to define a LOS for each lower-tier MANO. In that case, each agent of the multi-agent framework in the second stage can only embed SFCs within its LOS; thus, unnecessary cooperation among multiple agents will be avoided. As a result, we speed up the convergence rate of the multi-agent algorithm in the second stage. Specifically, we define a DC node that has insufficient CPU resources to accommodate the SFC requests received by this DC as a 'poor' DC, and a DC node that has more CPU resources than those requested by all SFCs waiting in its queue as a 'rich' DC. During a data preprocessing phase, each lower-tier MANO fully exploits the CPU resources of its affiliated DC to accommodate SFCs received by this DC. When there are no SFCs in 'rich' DCs or resources in 'poor' DCs, the upper-level MANO transfers the unhandled SFCs from 'poor' DCs to 'rich' DCs step by step to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. Therefore, it is reasonable for us to model the load transfer process, which follows the data preprocessing phase, as an MDP, whose state representation, action space, state transition dynamics, and reward function are defined as follows:

State Representation: The state is represented by graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The features of a DC node $v \in \mathcal{V}$ include four parts: (i) the available CPU resources (c_v), (ii) the sum of BW defined as the total available BW of links associated with DC v, (iii) the requested CPU resources of each unhandled SFC, and (iv) the requested BW resources of each unhandled SFC. Considering the MDC topology depicted in Fig.1, the initial state is shown in Table I.

The graph-structured information of the state cannot be directly fed into a fully connected neural network. Hence, we must encode the state into a real-valued vector. We apply the GCN architecture proposed in [38], which involves two main steps shown below, to the state encoding process of our problem.

- Node-level encoding using GCN: Based on the natural features of each node in graph $\mathcal{G}$, we use GCN to aggregate neighborhood information of each node. After l layers of GCN, we obtain a matrix $\mathbf{N} \in \mathbb{R}^{|\mathcal{V}| \times D}$. In matrix $\mathbf{N}$, row vector $\mathbf{n}_v$ $(1 \leq v \leq |\mathcal{V}|)$ represents the encoding of node v and D is a hyperparameter representing the dimension of $\mathbf{n}_v$. We consider a multi-layer GCN with the typical layer-wise propagation rule:

$$\mathbf{H}^{(l+1)} = \sigma(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}). \tag{10}$$

  Here, $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_{|\mathcal{V}|}$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections, where $\mathbf{I}_{|\mathcal{V}|}$ is the identity matrix. $\mathbf{W}^{(l)}$ is a layer-specific trainable weight matrix, and $\sigma(\cdot)$ denotes an activation function such as the ReLU(·). $\mathbf{H}^{(l)}$ is the matrix of activations in the l^{th} layer; $\mathbf{H}^{(0)} = \mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times Y}$ is the natural feature matrix, with each row indicating the Y-dimensional feature vector for a DC node. In our case, $Y = 2 + 2\max(u(v), v = 1, \ldots, |\mathcal{V}|)$, where $u(v)$ indicates the number of unhandled SFCs in the waiting queue of DC v.
- Graph level encoding using attention layers: As in [38], we define the context of the MDC network as follows:

$$\mathbf{ct} = g((\frac{\sum_{v=1}^{|\mathcal{V}|}\mathbf{n}_v}{|\mathcal{V}|})\mathbf{W}), \tag{11}$$

  where $\mathbf{W}$ is a learnable weight matrix, $\mathbf{n}_v$ is the encoding of DC v, and $g(\cdot)$ is a nonlinear function. Similarly, we define the encoding of the MDC as a weighted sum of DC nodes encoding. As a consequence, the MDC network encoding $\mathbf{h}$, a real-valued vector, can be given by

$$\mathbf{h} = \sum_{v=1}^{|\mathcal{V}|} a_v \mathbf{n}_v, \tag{12}$$

  where a_v, the weight of DC v, equals the inner product of its node encoding and the context:

$$a_v = \mathbf{n}_v^T \mathbf{ct}. \tag{13}$$

Action Space: At time step t, the agent sequentially selects (i) a 'poor' DC and a 'rich' DC (ii) and the route from the 'poor' DC to the 'rich' DC. Thus, the action at time step t can be defined as follows:

$$\boldsymbol{a}_t = (s, d, i_{s,d}), \tag{14}$$

where s and d indicate the indexes of the 'poor' DC and the 'rich' DC, respectively, and $i_{s,d}$ indicates the index of the route from the 'poor' DC to the 'rich' DC. Considering the MDC topology depicted in Fig.1, the action space for the 'poor' DC (DC 1) and 'rich' DC (DC 2) pair is shown in Table II.

We denote the sets of 'poor' DCs and 'rich' DCs by $\mathcal{P}$ and $\mathcal{R}$, respectively. Therefore, the numbers of 'poor' DCs and 'rich' DCs are $|\mathcal{P}|$ and $|\mathcal{R}|$, respectively. Besides, we denote the number of routes between DC p and DC r as $K_{p,r}$. As in [24], rather than enumerating all possible candidate routes for each node pair, we set a small value of K (e.g., $K = 5$) to limit the number of routes because the gain from considering long-distance paths is limited. Consequently, the action-space size is $\sum_{p \in \mathcal{P}} \sum_{r \in \mathcal{R}} \min(K_{p,r}, K)$.

Action $\boldsymbol{a}_0 = [1, 2, 1]$ indicates that, at time step 1, the first unhandled SFC in the waiting queue of DC 1 will be placed in DC 2, and the agent will choose the first route between these two DCs (DC 1 $\rightarrow$ DC 2) to transfer this SFC.

State Transition Dynamics: At time step t, the agent observes the state s_t, chooses an action $\boldsymbol{a}_t = (s, d, i_{s,d})$, and performs the action in the MDC environment. Then, the MDP transits to a new state s_{t+1} in which the node features are updated as follows:

- If neither the BW constraint of any link along the chosen route nor the CPU constraint of the destination DC is violated, then the action is successful; thus, several related fields need to be updated. First, the remaining CPU of DC d is updated by subtracting the total requested CPU of

TABLE I: Initial state.

DC ID	Remaining CPU	Sum of BW	Req CPU of the first unhandled SFC	Req BW of the first unhandled SFC	Req CPU of the second unhandled SFC	Req BW of the second unhandled SFC
1	0	$b_{1,2}+b_{1,5}$ $+b_{1,6}$	18	60	20	65
2	$200-160$	$b_{2,1}+b_{2,3}$ $+b_{2,4}$	0	0	0	0
3	$200-120$	$b_{3,2}+b_{3,4}$	0	0	0	0
4	$200-100$	$b_{4,2}+b_{4,3}+b_{4,5}$	0	0	0	0
5	0	$b_{5,1}+b_{5,4}+b_{5,6}$	32	50	0	0
6	0	$b_{6,1}+b_{6,5}$	24	40	0	0

TABLE II: Action space for pair (DC 1, DC 2).

Source	Destination	Route
1	2	1→2
1	2	1→5→4→2
1	2	1→6→5→4→2
1	2	1→6→5→4→3→2

the first unhandled SFC in the waiting queue of DC s. Next, the BW of any link along route $i_{s,d}$ is updated by subtracting the requested BW of this SFC; thus, the BW feature of any DC will be updated if the DC is associated with an updated link. Last, for DC s, the first unhandled SFC has been settled, and the features of the $(n+1)^{\text{th}}$ unhandled SFC will be moved to the corresponding fields of the n^{th} unhandled SFC.

- Otherwise, this is a bad action, and thus the state remains unchanged.

Considering the case depicted in Fig.1. Under the initial state, the agent performs $\boldsymbol{a}_0$ in the environment. Consequently, the MDP transits to s_1, which can be symbolized by Table III.

Reward Design: Our objective is to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. For an SFC transfer, if no BW or CPU constraints violation exists, the agent receives a reward consisting of two parts. The first part indicates a successful SFC transfer, and the second is the network traffic cost caused by the transferred SFC. Otherwise, it gets a penalty. Accordingly, the reward function is given by

$$R_t = \begin{cases} 1-\delta \cdot B_t \cdot \text{Dis}(\boldsymbol{a}_t[0], \boldsymbol{a}_t[1], \boldsymbol{a}_t[2]), \\ \text{if no BW or CPU constraints violation} \\ -1, \text{ otherwise,} \end{cases}$$

where B_t is the requested bandwidth of the SFC to be handled at time step t, $\text{Dis}(\boldsymbol{a}_t[0], \boldsymbol{a}_t[1], \boldsymbol{a}_t[2])$ is the distance of the $(\boldsymbol{a}_t[2])^{\text{th}}$ route between $\boldsymbol{a}_t[0]$ and $\boldsymbol{a}_t[1]$, and δ is the weight of the cost. Since maximizing the acceptance ratio of SFC requests is our top priority, we assign a value to δ such that the second term in the first segment is always smaller than 1.

2) GCN-based PPO: The PPO algorithm is a policy gradient DRL algorithm that comprises two components: (i) sampling data through interactions with the environment (ii) and optimizing a 'surrogate' objective function using stochastic gradient ascent. Many existing works have shown it to be much more straightforward to implement and general than other policy-based DRL algorithms.

The policy gradient objective function of typical policy gradient algorithms is defined as follows:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t], \tag{15}$$

where $\hat{A}_t$ is an estimator of the advantage function at time step t and is given by

$$\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t). \tag{16}$$

If $\hat{A}_t > 0$, action a_t is better than the average of all the actions in state s_t. Therefore, by taking gradient ascent steps on the policy loss function $L^{PG}(\theta)$, we will push the agent to take actions that lead to higher rewards. However, it is challenging to select the step size. An inadequate small step size will result in a too slow training process, while with too large a step size, the algorithm may go astray and converge to a local maximum. Hence, OpenAI proposed the PPO algorithm, which improves the stability of the actor training by clipping the policy update in each training step. The clipped 'surrogate' objective of the PPO algorithm is the following:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{17}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{18}$$

In Eq.(18), $r_t(\theta)$ denotes the ratio between the probability of action a_t in state s_t under the current policy and that under the previous policy. If $r_t(\theta) > 1$, in state s_t, the action is more probable under the current policy than under the old policy; If $1 > r_t(\theta) > 0$, in state s_t, the action is less probable under the current policy than under the old one. In Eq.(17), we can see two probability ratios: one non-clipped and the other clipped within $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter (set to 0.2 in [33]) that defines the clip range. The minimum of the clipped and non-clipped terms ensures a modest policy update because the new policy cannot be too different from the old one.

A general overview of our proposed GCN-based PPO algorithm is shown in Fig.2. Its workflow is summarized below. The sample collection process is described in steps 1-5. The training procedures for the policy network and the critic network are described in steps 6-9 and step 10, respectively.

1) Step 1: The agent observes the current state (s_t) of the MDC network, represented by a graph.
2) Step 2: State s_t is fed into a GCN. Subsequently, a matrix $\mathbf{N}_t$, which represents the node-level encoding

TABLE III: State s_1.

DC ID	Remaining CPU	Sum of BW	Req CPU of the first unhandled SFC	Req BW of the first unhandled SFC
1	0	$b_{1,2}+b_{1,5}+b_{1,6}-60$	20	65
2	$200-160-18$	$b_{2,1}+b_{2,3}+b_{2,4}-60$	0	0
3	$200-120$	$b_{3,2}+b_{3,4}$	0	0
4	$200-100$	$b_{4,2}+b_{4,3}+b_{4,5}$	0	0
5	0	$b_{5,1}+b_{5,4}+b_{5,6}$	32	50
6	0	$b_{6,1}+b_{6,5}$	24	40

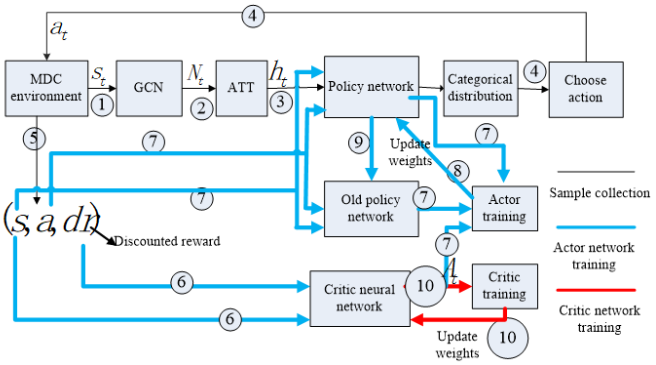

Fig. 2: A brief signal flow of our proposed PPO.

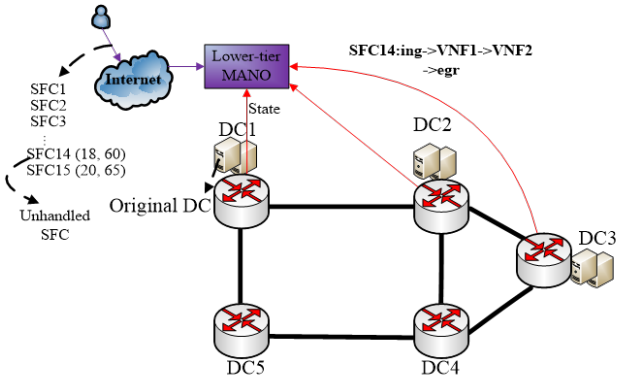

Fig. 3: The local observation scope of agent 1.

of the MDC network, is output after multiple layers of convolution.

3) Step 3: An attention mechanism is utilized to learn the weight of each DC node encoding, and the encoding of the MDC is defined as a weighted sum of DC nodes encoding. In this way, we encode the graph-structured information of the state into a real-valued vector $\mathbf{h}_t$, which is then fed into the current policy network.
4) Step 4: The outputs of the current policy network are fed into the SoftMax function to generate a Categorical distribution. Then, the agent samples data from the distribution to choose a corresponding action a_t, which will be performed in the environment.
5) Step 5: In response to action a_t, a reward r_t is returned to the agent. If time step t is the last step of an episode, the discounted reward for each step is calculated and the batch of samples collected in this episode (s_t, dr_t, a_t) $(t = 1, 2, \ldots, T)$ is stored into a buffer.
6) Step 6: Encoded s_t $(t = 1, 2, \ldots, T)$ is fed into the critic network $V_\phi(s_t)$. Then, the value $V_\phi(s_t)$ $(t = 1, 2, \ldots, T)$ and dr_t $(t = 1, 2, \ldots, T)$ are used to calculate the advantage $\hat{A}_t$ $(t = 1, 2, \ldots, T)$ based on Eq.(16).
7) Step 7: Encoded s_t $(t = 1, 2, \ldots, T)$ and a_t $(t = 1, 2, \ldots, T)$ are fed into the current policy network and the old one to get $\pi_\theta(a_t|s_t)$ and $\pi_{\theta_{\text{old}}}(a_t|s_t)$, respectively.
8) Step 8: The weights of the current policy network are updated by maximizing the policy loss function given by Eq.(17).
9) Step 9: Copying the weights of the current policy network to the old one.
10) Step 10: The weights of the critic network are updated by minimizing the value loss function given in line 14 of Algorithm 1.

The GCN-based PPO algorithm is summarized in Algorithm 1. At the beginning of each episode, we need to fulfill two tasks: (i) releasing the resources occupied by expired SFCs (line 3) (ii) and exploiting the resources thoroughly in all DCs to accommodate the SFCs waiting in their respective queues until there are no SFCs in 'rich' DCs or resources in 'poor' DCs (line 4). After that, the agent starts to transfer unhandled SFCs step by step (line 5) from 'poor' DCs to 'rich' DCs, aiming to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. When no SFCs left, the end of an episode is reached (line 6). Then, a batch of samples (s_t, dr_t, a_t) $(t = 1, 2, \ldots, T)$, which will be used to train the critic and the policy networks, is stored into a buffer. The batch of samples will be reused to train the critic and the actor for L epochs (lines 9-16). We train the neural networks episode by episode until their weights converge. Once the algorithm converges, for each 'poor' DC, we know to which 'rich' DCs and via which routes shall we transfer its unhandled SFCs; thus, we set an appropriate LOS for each agent in the multi-agent framework of the next stage. For instance, in the LOS of agent 1 depicted in Fig. 3, agent 1 has exhausted the IT resources in its affiliated (original) DC (DC 1); thus, it needs to take advantage of the IT resources of DC 2 and DC 3 to embed its unhandled SFCs, and the data streams of these SFCs might travel through the border router of DC 4 or DC 5. For the agent of each 'rich' DC, the LOS only includes its affiliated DC.

C. The Second Stage

In the first stage, we set a LOS for each lower-tier MANO; thus, we can efficiently design a multi-agent framework in this

Algorithm 1 GCN-based PPO Algorithm

Initialize neural network parameters θ and ϕ
for $episode = 1, 2, \ldots$ **do**
  Release the resources occupied by expired SFCs
  Preprocess SFCs received by each DC
  Run policy π_θ for T timesteps, collecting $\{s_t, a_t, dr_t\}$
  Break the timesteps loop if no requests left
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
  for $l = 1, \ldots, L$ **do**
    $L^{CLIP}(\theta) = \sum_{t=1}^{T} \min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t, \text{clip}(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t)$
    Update θ via a gradient method w.r.t. $L^{CLIP}(\theta)$
  end for
  for $l = 1, \ldots, L$ **do**
    $L^{VL}(\phi) = \sum_{t=1}^{T} (\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t))^2$
    Update ϕ via a gradient method w.r.t. $L^{VL}(\phi)$
  end for
end for

stage such that needless cooperation among multiple agents can be prevented. The multi-agent framework is composed of two phases. During the first phase, all agents simultaneously embed SFCs in their respective DCs. The second phase is initiated once the agent of a 'poor' DC starts to embed its unhandled SFCs in 'rich' DCs. We model the SFC embedding problem in phase two as a multi-agent MDP [12] and solve it using a MARL approach, GCN-based MAPPO.

1) The Components of the Multi-agent MDP: First, we define the observation representation, action space, and reward function for agent m at time step t as follows:

Observation Representation: The LOS of agent m can be modeled as a graph $\mathcal{G}_m = (\mathcal{V}_m, \mathcal{E}_m)$, a sub-graph of $\mathcal{G}$. The observation of agent m at time step t ($o_{m,t}$) consists of three parts: the graph-structured information of $\mathcal{G}_m$, the location (DC index) of the VNF embedded at the previous time step, and the index of the original DC. In $\mathcal{G}_m$, each DC node $v \in \mathcal{V}_m$ has four features: (i) the remaining CPU resources (c_v), (ii) the sum of BW defined as the total available BW of links associated with v, (iii) the requested CPU of the current VNF to be processed ($C_{r,m,t}$), (iv) and the requested BW of the VL to be processed ($B_{r,m,t}$).

To feed the graph-structured information into the policy network of agent m, we first reuse the two-step state encoding process as mentioned earlier to encode the sub-graph into a real-valued vector. Then, we concatenate the location of the VNF embedded at the previous time step, the index of the original DC, and the encoded vector to form a new vector $\boldsymbol{h}_{m,t}$, which will be fed into the policy network.

Action Space: Designing an actor network for the VNF-FG placement issue is challenging because the dimension of the output layer of a neural network is fixed, with each neuron symbolizing a possible action. Nevertheless, if we couple the placement of a VNF and a VL, the number of possible placements changes as the location of the previous embedded VNF changes. That is, the action-space size is not a constant under different states. Most existing works avoided this tricky problem by decoupling the placement of VNFs and VLs. They utilized the actor network to choose a VNF placement and adopted the Dijkstra algorithm to select the 'optimal' route given the VNF placement. This method reduces the action-space size because it only explores the placement of VNF; thus, it speeds up the convergence rate. However, given a VNF placement, the Dijkstra algorithm chooses the 'optimal' route for the VL placement and thus neglects all the other options. The greed for the 'optimal' route for some step might cause the failure placement of VLs in subsequent steps, which has a negative impact on the objective of maximizing the total acceptance ratio of all requests. To get a better global optimum, we devise an action set that extensively explores the possible placements of both VNFs and VLs. Specifically, we use a 3-column row vector $\boldsymbol{a}_{m,t}$ to symbolize the placement of the current VNF and the VL connecting it and its previous VNF. In $\boldsymbol{a}_{m,t}$, the first element indicates the source DC of the VL, the second indicates in which DC will the current VNF be embedded, and the third denotes the route index between the source DC and the DC that will accommodate the current VNF. We do not require that the source DC accommodate the previous VNF. Thus, the source DC can be any DC $v \in \mathcal{V}_m$, and the action-space size is fixed. We denote the number of routes between DC i and DC j by $K_{i,j}$. Because the benefit from enumerating long-distance routes is limited, as in [24] and [23], we restrict the number of routes between DC i and DC j and define it as $K^r_{i,j} = \min(K, K_{i,j})$, where K (e.g., $K = 5$) is the number of shortest routes between two DCs. Thus, the action-space size is $\sum_{i=1}^{|\mathcal{V}_m|} \sum_{j=1}^{|\mathcal{V}_m|} K^r_{i,j}$. Considering the sub-graph depicted in Fig.3, a small part of the action space is shown in Table IV.

Action $\boldsymbol{a}_{m,t} = [1, 2, 1]$ indicates that the current VNF is placed in DC 2, the source DC of the VL is DC 1, and the VL connecting DC 2 and DC 1 is embedded in the first route (DC 1 $\rightarrow$ DC 2) between these two DCs.

TABLE IV: Action space.

Source	Destination	Route
1	2	1→2
1	2	1→5→4→2
1	2	1→5→4→3→2

Reward: The reward function consists of an internal function and two external functions. The interval reward function, which only regulates the behavior of agent m and has no correlation with other agents, can be defined as

$$R^{\text{in}}(m,t) = \begin{cases} -1, & \text{if } (\boldsymbol{a}_{m,t}[0] \neq \boldsymbol{h}_{m,t}[0]) \\ & \text{or (egress and } \boldsymbol{a}_{m,t}[1] \neq \boldsymbol{h}_{m,t}[1]) \\ 1, & \text{otherwise,} \end{cases}$$

where $\boldsymbol{a}_{m,t}[0]$ and $\boldsymbol{a}_{m,t}[1]$ are the indexes of the source and destination DCs of the VL, $\boldsymbol{h}_{m,t}[0]$ is the index of the DC that the previous VNF is located in, and $\boldsymbol{h}_{m,t}[1]$ is the index of the original DC of agent m, at time step t. If the VL does not start at the previous VNF or end at the original DC for the last pair of VNF (egress) and VL in an SFC, the action is a bad action, and agent m receives a penalty.

The external reward functions guide all agents toward the common objective of maximizing the overall acceptance ratio of SFC requests while minimizing the total cost. The reward function regarding the overall acceptance ratio of SFC requests is defined as

$$R_{\mathrm{a}}^{\mathrm{ex}}(m,t)=\begin{cases} N_{a,t}-N_{c,t}, \text{ if } \exists v\in\mathcal{V},\ c_v\le\sum\limits_{m=1}^{|\mathcal{P}|}x_{m,t}^{v}C_{r,m,t} \\ \qquad\qquad \text{or } \exists(v,u)\in\mathcal{E},\ b_{v,u}\le\sum\limits_{m=1}^{|\mathcal{P}|}z_{m,t}^{v,u}B_{r,m,t} \\ N_{a,t}, \qquad\qquad \text{otherwise,} \end{cases}$$

where binary variable $x_{m,t}^{v}$ indicates whether the VNF to be processed at time step t by agent m nests in DC v, binary variable $z_{m,t}^{v,u}$ indicates whether the VL to be processed at time step t by agent m is embedded in link (v,u), $N_{a,t}$ is the number of agents that have SFCs to handle at time step t, and $N_{c,t}$ is number of agents that fail to accommodate their VNFs or VLs at time step t.

We define the reward function for the total cost as

$$\begin{aligned} R_{\mathrm{c}}^{\mathrm{ex}}(m,t)=&\theta_1\sum_{v\in\mathcal{V}}\sum_{m\in\mathcal{P}}x_{m,t}^{v}\cdot C_{r,m,t} \\ &+\theta_2\sum_{(v,u)\in\mathcal{E}}\sum_{m\in\mathcal{P}}z_{m,t}^{v,u}\cdot B_{r,m,t}\cdot \mathrm{Dis}\,(v,u)\,, \end{aligned}$$

where θ_1 and θ_2 are the weights of the CPU cost and the traffic cost, respectively.

Considering our objective, the reward function $R(m,t)$ is expressed as a weighted sum of the internal and external reward functions.

$$R(m,t)=\eta_1\cdot R^{\mathrm{in}}(m,t)+\eta_2\cdot R_{\mathrm{a}}^{\mathrm{ex}}(m,t)-\eta_3\cdot R_{\mathrm{c}}^{\mathrm{ex}}(m,t).$$

Regulating the interval behavior of an agent is an essential prerequisite for achieving the common objective; thus, η_1 is the greatest of the three weights. Maximizing the acceptance ratio is more important than minimizing the total cost; therefore, η_2 is greater than η_3.

Then, we define the global state s_t and action a_t as follows:

State: State (s_t) is the union of the local observation of each agent: $s_t=\{o_{1,t}\cup o_{2,t}\ldots\cup o_{|\mathcal{P}|,t}\}$.

Joint Action: We concatenate the action of each agent to generate the joint action: $a_t=\{a_{1,t},\ldots,a_{|\mathcal{P}|,t}\}$.

2) The Proposed Multi-agent Framework: As in [31], the MAPPO used in our scheme is based on the framework of CTDE. Its general overview is shown in Fig.4, where we detail the workflow of agent 1, which is the same as that of all the other agents of 'poor' DCs. Agent m $(1\le m\le|\mathcal{P}|)$ is equipped with a current policy network $\pi_\theta(a_{m,t}|o_{m,t})$, an old policy network $\pi_{\theta_{\text{old}}}(a_{m,t}|o_{m,t})$, and a critic network $V_{\phi_m}(s_t)$, where s_t is composed of the local observations of all agents: $s_t=\{o_{1,t}\cup o_{2,t}\ldots\cup o_{|\mathcal{P}|,t}\}$. The workflow of MAPPO from the perspective of agent m is summarized below.

1) Step 1: At time step t, agent m obtains its local observation $o_{m,t}$, which consists of the graph-structured information, the position (DC index) of the VNF embedded at time step $t-1$, and the index of the agent's original DC.
2) Steps 2-3: The graph-structured information is encoded into a real-valued vector, which is concatenated with the position of the previous embedded VNF and the index of the original DC to form a new vector $\boldsymbol{h}_{m,t}$. Then, $\boldsymbol{h}_{m,t}$, which can be viewed as an encoded $o_{m,t}$, is fed into the policy network $\pi_{\theta_m}(a_{m,t}|o_{m,t})$.
3) Step 4: An action $a_{m,t}$, the placement of the current VNF and the VL connecting this VNF and the previous, is chosen and performed in the environment.
4) Step 5: In response to the joint action of all agents, a reward $r_{m,t}$ and the global state s_t, which consists of the observations of all agents, are returned to agent m. If time step t is the last step of an episode, the discounted reward for each step is calculated and the batch of samples collected in this episode $(s_t,o_{m,t},dr_{m,t},a_{m,t})$ $(t=1,2,\ldots,T)$ is stored into buffer m.
5) Step 6: Encoded s_t $(t=1,2,\ldots,T)$ is fed into the critic network $V_{\phi_m}(s_t)$. Then, the value $V_{\phi_m}(s_t)$ $(t=1,2,\ldots,T)$ and $dr_{m,t}$ $(t=1,2,\ldots,T)$ are used to calculate the advantage $\hat{A}_{m,t}$ $(t=1,2,\ldots,T)$ based on the equation in line 15 of Algorithm 2.
6) Step 7: Encoded $o_{m,t}$ $(t=1,2,\ldots,T)$ and $a_{m,t}$ $(t=1,2,\ldots,T)$ are fed into the current policy network and the old one to get $\pi_\theta(a_{m,t}|o_{m,t})$ and $\pi_{\theta_{\text{old}}}(a_{m,t}|o_{m,t})$, respectively.
7) Steps 8-10: The same as the corresponding steps in PPO.

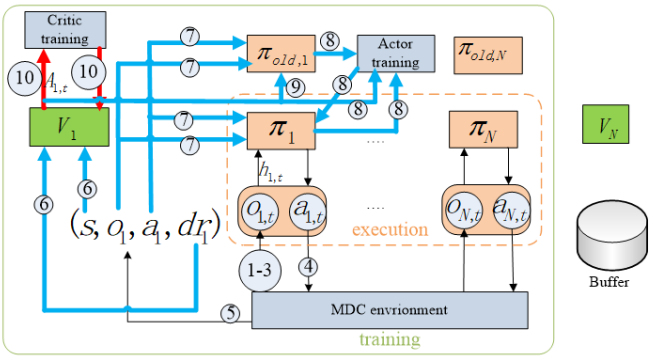

Fig. 4: A brief signal flow of our proposed MAPPO.

The proposed multi-agent algorithm is summarized in Algorithm 2. The agents of 'rich' DCs embed SFCs in their respective original DCs; thus, they do not need to optimize actions by training neural networks. In contrast, each agent of a 'poor' DC needs to embed SFCs in other DCs within its LOS when it exhausts the resources in its original DC; hence, each agent $m\in\mathcal{P}$ learns a local policy network and a centralized critic network. At the beginning of each episode, we release the resources occupied by expired SFCs (line 4). During the first phase (lines 5-9), all agents place SFCs in their respective DCs. During the second phase, multiple agents of 'poor' DCs simultaneously place their VNFs and VLs in 'rich' DCs (line 11), creating a multi-agent MDP environment. From the perspective of agent m, it embeds VNFs and VLs step by step in 'rich' DCs, aiming to maximize the overall acceptance ratio of SFC requests while minimizing the total cost. When no SFCs left, the end of an episode is reached (line 14). Then, a batch of samples $(s_{m,t},a_{m,t},o_{m,t},dr_{m,t})$ $(t=1,2,\ldots,T)$, which will be used to train the critic network $(V_{\phi_m}(s_t))$ and

the policy network ($\pi_{\theta_m}(a_{m,t}|o_{m,t})$), is stored into buffer m. The batch of samples will be reused to train the critic network and the policy network for L epochs (lines 17-24). We train the neural networks episode by episode until their weights converge. Once the algorithm converges, all agents of 'poor' DCs know how to cooperate with each other to maximize the overall acceptance ratio of SFC requests while minimizing the total cost.

Algorithm 2 The Proposed Multi-agent Algorithm

1: Create a thread for each agent
2: Initialize neural network parameters θ_m and ϕ_m for each agent that manages a poor DC
3: **for** $episode = 1, 2, \ldots$ **do**
4: Release the resources occupied by expired SFCs
5: **if** A rich DC **then**
6: Embeds SFCs in this DC
7: **else**
8: **if** A poor DC with sufficient resources to handle the current SFC **then**
9: Embeds SFCs in this DC
10: **else**
11: Run policy π_{θ_m} for T timesteps, embedding a pair of VNF and VL in each step and collecting $\{s_{m,t}, a_{m,t}, o_{m,t}, dr_{m,t}\}$
12: **end if**
13: **end if**
14: Break the timesteps loop if no requests left
15: Estimate advantages $\hat{A}_{m,t} = \sum_{t'>t} \gamma^{t'-t} r_{m,t'} - V_\phi(s_{m,t})$
16: $\pi_{\theta_{m,\text{old}}} \leftarrow \pi_{\theta_m}$
17: **for** $l = 1, \ldots, L$ **do**
18: $L^{CLIP}(\theta_m) = \sum_{t=1}^{T} \min(\frac{\pi_\theta(a_{m,t}|o_{m,t})}{\pi_{\theta_{\text{old}}}(a_{m,t}|o_{m,t})}\hat{A}_{m,t}, \text{clip}(\frac{\pi_{\theta_m}(a_{m,t}|o_{m,t})}{\pi_{\theta_{m,\text{old}}}(a_{m,t}|o_{m,t})}, 1-\epsilon, 1+\epsilon)\hat{A}_{m,t})$
19: Update θ_m via a gradient method w.r.t. $L^{CLIP}(\theta_m)$
20: **end for**
21: **for** $l = 1, \ldots, L$ **do**
22: $L^{VL}(\phi_m) = \sum_{t=1}^{T}(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_{m,t}))^2$
23: Update ϕ via a gradient method w.r.t. $L^{VL}(\phi_m)$
24: **end for**
25: **end for**

D. Complexity Analysis

For all DRL-based algorithms, the whole process consists of the offline training phase and online testing phase. For the training phase, the computational complexity is proportional to the number of training episodes and the number of time steps in each episode. We will showcase the superiority of our proposed algorithm in the simulation section. Here, we theoretically analyze the complexity of the testing phase of our proposed algorithm, which is related to the structure of neural networks. Since the algorithm follows the CTDE framework, we only need to analyze the complexity of the policy networks. As all agents have their respective policy networks, they have different computational complexities. Furthermore, they handle requests simultaneously; thus, the maximum complexity of all agents equals the computational complexity of the proposed algorithm. For each agent, the computational complexity should cover three parts: the GCN, the attention network, and its policy network. Let's suppose that agent m has a two-layer GCN, and $W^0 \in \mathbb{R}^{Y\times H}$ is the input-to-hidden weight matrix and $W^1 \in \mathbb{R}^{H\times D}$ is the hidden-to-output weight matrix. Y, H and D are the numbers of neurons of the input layer, hidden layer, and output layer. Based on [34], the computational complexity of the GCN is $\mathcal{O}(|\epsilon_m|YHD)$. As for the attention network, a weight matrix $W \in \mathbb{R}^{D\times D}$ is utilized to perform graph-level encoding. As for the policy network, the input layer receives the graph-level encoding vector; therefore, the number of its neurons is D. In the output layer, the number of neurons represents the dimension of actions; thus, there are 3 neurons in the output layer. In addition, we assume that the number of neurons in the hidden layer is I. Then, the computational complexity of the attention network and the policy network are $\mathcal{O}(D * D)$, and $\mathcal{O}(I(D + I + 3))$, respectively.

V. Simulation

A. Simulation Setup

In this Section, we evaluate the performance of the proposed two-stage GCN-based DRL algorithm regarding the admission ratio and the cost-effectiveness under different network sizes. Two scenarios, i.e., the 14-node NSFNET and the 11-node COST239 topologies [24] depicted in Figs.5 and 6, are utilized to assess our algorithm. The BW of each link is set to 1 Tbps, and the capacity of each DC node is set to 600 units. For the cost model, we set $\beta = 0.1$ \$/Tbps $\cdot$ Km and $\gamma = 0.1$ \$/unit. As for the weights of the two objectives, $\alpha_1 : \alpha_2 = 10 : 1$.

In [39], the authors studied the workload collected by Google and standardized the workload to decrease the fluctuation range. Then, they plotted a figure describing the aggregated task arrival rates of different types of service per minute throughout 29 days. In our simulation, we care about the SFC requests arrival rate of each DC in the MDC environment. Thus, we intercept 24 hours of data in [39] for each DC based on its time zone and pick out the 6 combinations of unbalanced traffic load for the MDC network. We scale down the numbers of SFC requests based on the capacity of our NFV environment. Besides, we assume that each SFC is composed of [2,4] VNFs, and its lifetime follows an exponential distribution with an average of one time slot. The requested CPU of a VNF is a discrete random variable uniformly distributed over the interval [4, 8] units, while the requested BW of an SFC is a random variable uniformly distributed over the interval [25, 100] Gbps.

Neural networks are set up with the following parameters. In the first stage, the number of GCN layers is set to 3. In the second stage, for each agent that manages a 'poor' DC, the number of GCN layers is determined by the scale of its LOS and can be set to 1 or 2 for the policy network. For the critic network, it is set to 3. And tanh(·) is used as the activation function. Two fully connected layers are created to represent the policy network and the critic network, respectively. Adam [40] is adopted to learn the neural network parameters. The

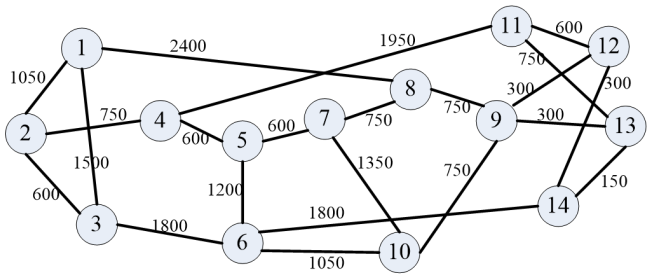

Fig. 5: 14-node NSFNET (link length in kilometers).

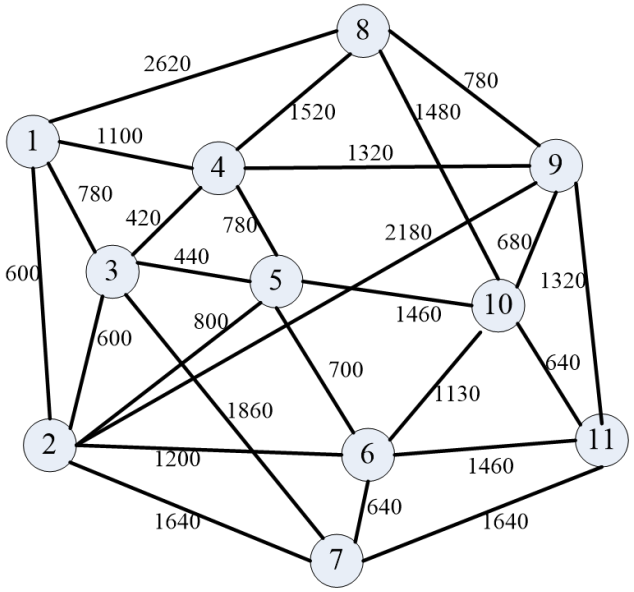

Fig. 6: 11-node COST239 (link length in kilometers).

learning rate is 10^{-4} for the policy network and 2×10^{-4} for the critic network, and the discount factor is 0.99. The Clipping ratio ϵ is set to 0.2, and the batch size is set to 100. Besides, each batch of samples will be reused for 10 epochs for training the policy and the critic networks.

During the training phase, one traffic combination is randomly chosen from the 6 combinations of unbalanced traffic load at the beginning of each episode. The system is trained episode by episode and the process ends when the algorithm converges. During the evaluation phase, we evaluate the performance of our proposed algorithm over every traffic combination. To acquire sufficient statistical accuracy, we obtain each performance metric by averaging the results from 10 simulation runs.

B. Algorithms Used for Comparison

1) MAPPO in a Partially Observable Environment (One-hop Neighbors): Each agent has its own LOS and can only allocate resources within its one-hop neighbors' scope. This partially observable environment is an extreme case for the LOS of each agent.

2) MAPPO in a Fully Observable Environment: All agents can observe the whole environment and may cooperate everywhere. This environment is another extreme case for the LOS of each agent.

3) Kolin: The authors in [27] proposed Kolin to embed VNF-FG in IoT networks. They devised a GNN-assisted DRL to place VNFs and generated VLs placement based on the Dijkstra algorithm (the shortest path in terms of the used bandwidth). To some extent, it achieves load balancing when embedding VLs. We modify Kolin to make it adaptive to our network scenario.

4) Deep Q-Network (DQN): The typical DQN algorithm [41] is adopted to select VNF placement actions, and the Dijkstra algorithm (the shortest path in terms of distance) is utilized to embed the VLs that connect adjacent VNFs.

5) First-Fit (FFT): This approach adopts the First-Fit algorithm to embed VNFs and the Dijkstra algorithm (the shortest path in terms of distance) to determine the placement of VLs. FFT allocates VNFs to a partition that is first sufficient from the top of the main memory address. In our MDC environment, the higher index a DC has, the higher priority it will be given to accommodate VNFs. Given a VNF placement, the Dijkstra algorithm is utilized to determine the route between adjacent VNFs. The FFT has been selected as one of the baselines to assess the performance in [8] and [24].

C. Simulation Results

We compare our proposed two-stage GCN-based RL algorithm, MAPPO in a partially observable environment (One-hop MAPPO), MAPPO in a fully observable environment (Fully-observable MAPPO), Kolin, DQN, and FFT with respect to the convergence rate, overall acceptance ratio, and average cost per SFC request.

In the first stage, we need to use an M-layer GCN to extract the features of the MDC network. A GCN with a large M requires excessive computational efforts, while a GCN with a small M may not be able to efficiently extract all features from the graph. To identify the best value of M, we evaluate the performance of the GCN-based PPO with different numbers of GCN layers. It is shown in Fig.7 that the GCN-based PPO with 3-layer GCN achieves a better result than that with 2 layers or 4 layers. Thus, we select the 3-layer GCN for our implementation.

The clipping ratio ϵ is a critical hyperparameter that we can tune to balance training stability and fast convergence. From Fig.8, we can see that the GCN-based PPO with $\epsilon = 0.2$ converges faster than that with $\epsilon = 0.15$. Meanwhile, they attain approximately the same long-term cumulative reward, which is better than that obtained by the GCN-based PPO with $\epsilon = 0.25$. Thus, we set $\epsilon = 0.2$ for our implementation.

In the second stage, for each agent in the MAPPO, the number of GCN layers is set to 1 or 2 for the policy network according to the scale of the agent's LOS. For the critic network, the number of GCN layers is set to 3 as we need to feed the encoded global state into it.

From Figs.9 and 10, we can see that our proposed algorithm outperforms other baselines. It achieves similar performance to the Fully-observable MAPPO because we apply the GCN-assisted encoding method and the VL-space exploration policy we designed for our proposed algorithm to the Fully-observable MAPPO. In addition, as our proposed algorithm

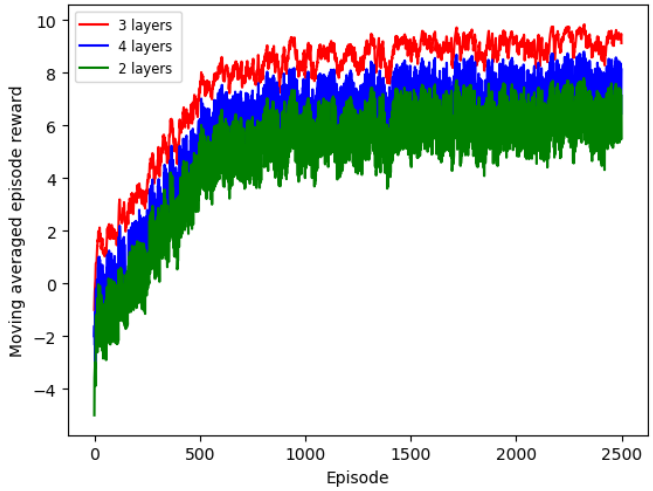

Fig. 7: The performance of our proposed GCN-based PPO with different numbers of GCN layers.

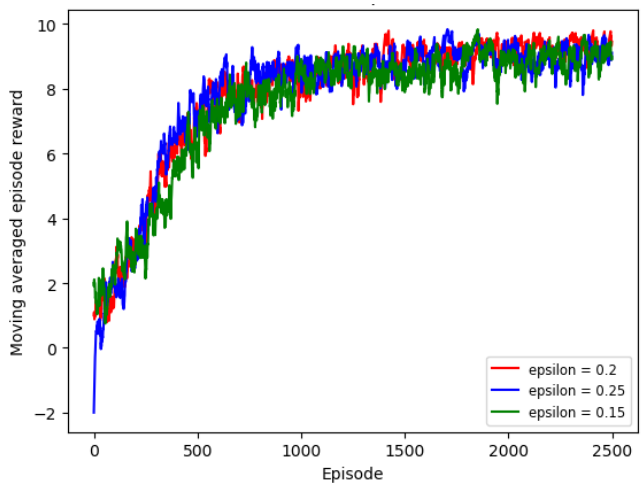

Fig. 8: The performance of our proposed GCN-based PPO with different clipping ratio hyperparameters.

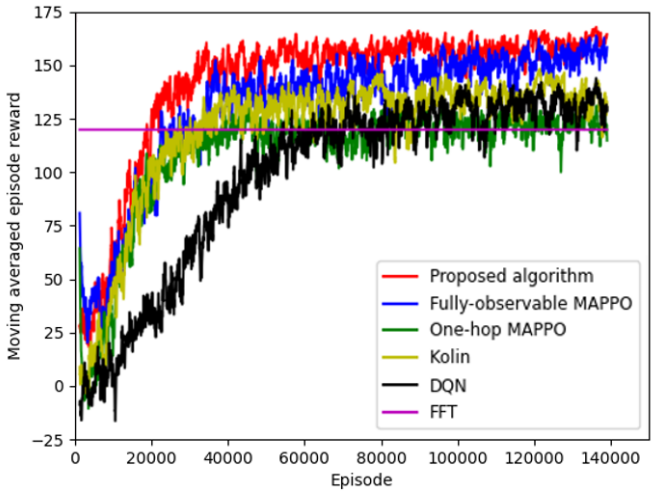

Fig. 9: The learning curve of six algorithms in 11-node NSFNET.

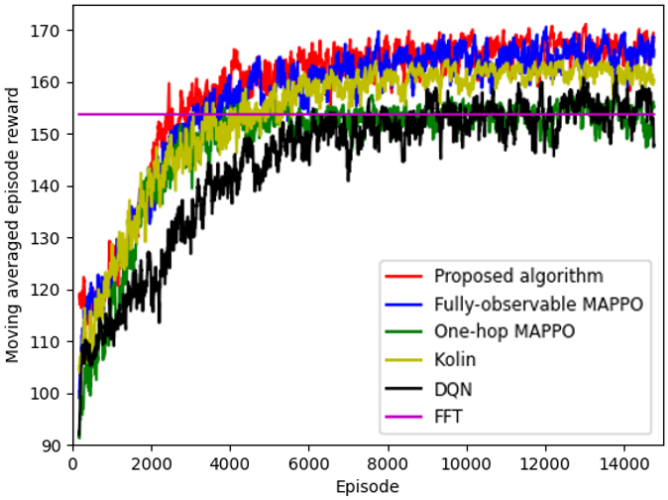

Fig. 10: The learning curve of six algorithms in 14-node COST239.

sets an appropriate LOS for each agent, it converges much faster than the Fully-observable MAPPO, whose agents can observe the whole environment. Specifically, each agent of the Fully-observable MAPPO can place VNFs or VLs in every corner of the whole environment, leading to unnecessary exploration that should have been avoided. In contrast, our proposed algorithm tailors an appropriate LOS for each agent, confining the exploration to the smallest scope. These two algorithms showcase better performance than all the others. The reason is that, given a VNF placement, they two choose a route for embedding a VL from K shortest candidates. In this way, they explore the environment better and thus acquire better long-term cumulative rewards than those algorithms that adopt the Dijkstra algorithm to select a short-sighted optimal immediate reward in each time step. The One-hop MAPPO converges fastest because each agent of this algorithm can only exploit the resources within its one-hop neighbors' scope. However, it achieves an unfavorable long-term cumulative reward because of the restricted view of each agent.

We train all DRL-based algorithms in two topologies. The training in the first topology can be viewed as the source task. For the training in the second topology, which can be viewed as the target task, we do not need to learn from scratch but from the trained model we acquire in the source task. We can see from Figs.9 and 10 that each of the DRL-based algorithms converges much faster and stands on a higher initial value in the second topology than in the first one.

The other three algorithms, which adopt the Dijkstra algorithm to determine the route option for a VL given a VNF placement, obtain worse accumulated rewards than our proposed algorithm and the Fully-observable MAPPO because of the greed for immediate rewards. The two DRL-based algorithms are better than the FFT. The reason is that the FFT allocates VNFs to a partition that is first sufficient from the top of the main memory address. In that case, the placement mode for VNFs is fixed. Given a VNF placement, it adopts the Dijkstra algorithm to embed a VL; thus, the placement

mode for VLs is also fixed. In contrast, the Kolin and DQN follow their logic to explore the environment, although they only consider the exploration of VNFs placement. Through interacting with the environment, they find the optimal actions to place VNFs and VLs, obtaining better long-term cumulative rewards than the FFT. Another finding is that, of the two DRL algorithms, the Kolin converges faster and achieves better long-term cumulative rewards than the DQN. Two reasons can account for this result. First, the DRL part of Kolin follows the actor-critic framework, which utilizes the policy gradient to find the optimal action given a state. In comparison, the DQN assigns a score indicating the maximum expected future reward to each possible action, drastically increasing the time complexity. Second, although they both adopt the Dijkstra algorithm to select routes for VLs, the Kolin chooses the shortest path in terms of used bandwidth while the DQN makes selections based on distance. Thus, the Kolin considers load balancing and can avoid traffic congestion on some critical paths that the DQN might meet when placing VLs.

During the evaluation phase, from Figs.11(a) and 12(a), we can see that our proposed algorithm achieves approximately the same acceptance ratio of SFC requests as the Fully-observable MAPPO. These two algorithms outperform all the other algorithms in terms of the acceptance ratio of SFC requests for the aforementioned reasons. When the capacity of the MDC infrastructure could not carry the total traffic load (traffic combinations $4, 5$, and 6), the advantages of our proposal over the others are more obvious. In Fig.12(a), compared with the $80.46\%, 76.39\%$, and 69.06% acceptance ratio accomplished by Kolin, DQN, and FFT, respectively, our proposed algorithm achieves 91.3% acceptance ratio. In Fig.11(a), compared with the $83.52\%, 78.39\%$, and 71.06% acceptance ratio accomplished by Kolin, DQN, and FFT, respectively, our proposed algorithm achieves 94.66% acceptance ratio. Thus, on average, our proposed algorithm improves the acceptance ratio by approximately 13% compared with the Kolin and 18% compared with the DQN.

From Figs.11(b) and 12(b), we can see that the One-hop MAPPO has the least average cost because each agent can only exploit resources within its one-hop neighbors' scope, resulting in the minimum traffic cost. Nevertheless, it consumes the least average BW per SFC request at the sacrifice of the most important objective: the overall acceptance ratio of the SFC requests. Thus, its great performance in saving BW cost is meaningless. Of all the other algorithms, our proposed algorithm and the Fully-observable MAPPO use the minimum average traffic cost for accommodating an SFC request. The aforementioned VLs embedding scheme plays a major role in accounting for this result. Furthermore, PPO-based algorithms tune a clip ratio, which controls the step size of policy updates, balancing training stability and fast convergence. This innovation ensures that PPO-based algorithms have better long-term cumulative rewards than the Kolin and DQN. Let's see the overloaded scenarios under traffic combinations $4, 5$, and 6 again. We can see that in Fig.12(b), compared with the $17.54\$, 17.42\$$, and $20.3\$$ average cost spent by Kolin, DQN, and FFT, respectively, our proposed algorithm has an average cost of $12.53\$$. In Fig.12(a), compared with the $19.16\$, 19.1\$$, and $22.3\$$ average cost spent by Kolin, DQN, and FFT, respectively, our proposed algorithm has an average cost of $15.3\$$. Thus, on average, our proposed algorithm reduces the average cost by around 28.6% compared with the Kolin and 28.2% compared with the DQN.

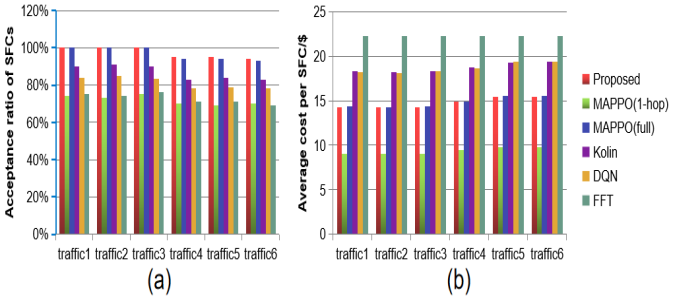

Fig. 11: Performance evaluation in 14-node NSFNET.

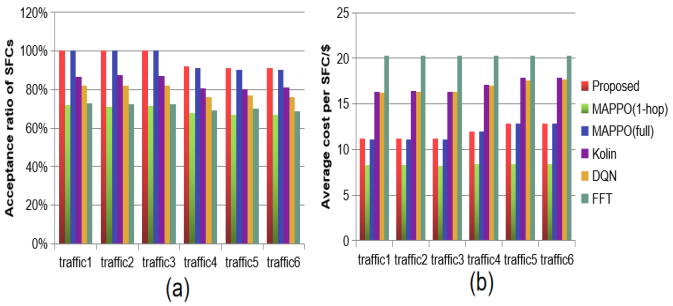

Fig. 12: Performance evaluation in 11-node COST239.

VI. Conclusion

In this paper, we propose a two-stage GCN-based DRL algorithm to maximize the overall acceptance ratio of SFC requests while minimizing the total cost in an MDC environment. We consider an MDC scenario where the arrival rates of SFC requests vary from DC to DC and develop an intelligent policy to place the VNFs and VLs requested by SFCs. To achieve the objective, we design the GCN-based PPO framework in the first stage and the multi-agent framework in the second stage. Simulation results demonstrate that our proposed algorithm outperforms other baselines. It achieves similar performance to the fully-observable MAPPO while converging much faster. In summary, compared to state-of-the-art methods, our proposed scheme improves the overall acceptance ratio of SFC requests and cost-effectiveness. As for future work, we will design a multi-agent DRL for the SFC embedding issue in the inter-DC EONs. Since the spectrum assignment scheme of the lightpath should comply with the spectrum contiguous, non-overlapping, and continuous constraints, the new algorithm could be different from the current one. Another field of future interest could be the multi-agent DRL framework. We will design a distributed training framework and compare it with the CTDE framework used in this paper.

REFERENCES

[1] S. Garg and A. Misra, "Service level agreements for cloud infrastructures," in *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, Mar. 2019, pp. 276–279.

[2] A. Baumgartner, V. S. Reddy, and T. Bauschert, "Combined virtual mobile core network function placement and topology optimization with latency bounds," in *2015 Fourth European Workshop on Software Defined Networks*, Bilbao, Spain, Sep.-Oct. 2015, pp. 97–102.

[3] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[4] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach," *IEEE Transactions on Services Computing*, vol. 13, no. 1, pp. 172–185, Jan.-Feb. 2020.

[5] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, Mar. 2019.

[6] M. Dieye *et al.*, "Cpvnf: Cost-efficient proactive vnf placement and chaining for value-added services in content delivery networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 774–786, Jun. 2018.

[7] A. Varasteh, B. Madiwalar, A. V. Bemten, W. Kellerer, and C. Mas-Machuca, "Holu: Power-aware and delay-constrained vnf placement and chaining," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1524–1539, Jun. 2021.

[8] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, Dec. 2019.

[9] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, F. De Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, Krakow, Poland, May 2014, pp. 1–9.

[10] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-end performance-based autonomous vnf placement with adopted reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 534–547, Jun. 2020.

[11] A. Rkhami, Y. Hadjadj-Aoul, and A. Outtagarts, "Learn to improve: A novel deep reinforcement learning approach for beyond 5g network slicing," in *IEEE Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan. 2021, pp. 1–9.

[12] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," Mar. 2021. [Online]. Available: https://arxiv.org/abs/2011.00583

[13] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, third quarter 2018.

[14] J. Gil Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[15] S. Mostafavi, V. Hakami, and M. Sanaei, "Quality of service provisioning in network function virtualization: A survey," *Computing*, vol. 103, pp. 917–991, Mar. 2021.

[16] W. Fang, M. Zeng, X. Liu, W. Lu, and Z. Zhu, "Joint spectrum and it resource allocation for efficient vnf service chaining in inter-datacenter elastic optical networks," *IEEE Communications Letters*, vol. 20, no. 8, pp. 1539–1542, Aug. 2016.

[17] A. Khatiri and G. Mirjalily, "Resource balanced service chaining in nfv-enabled inter-datacenter elastic optical networks," in *2020 12th International Conference on Knowledge and Smart Technology (KST)*, Pattaya, Thailand, Jan.-Feb. 2020, pp. 168–171.

[18] Y. Li *et al.*, "Joint balancing of it and spectrum resources for selecting virtualized network function in inter-datacenter elastic optical networks," *Opt Express*, vol. 27, no. 11, pp. 15 116–15 128, May 2019.

[19] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2179–2192, Oct. 2019.

[20] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 543–553, Sep. 2017.

[21] Open Science Data Cloud, "Virtual machines (vms)," 2014. [Online]. Available: https://www.openscidatacloud.org/support/instances.html

[22] Kubernetes Documentation, "Resource management for pods and containers," Jun. 2022. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

[23] M. Zhu, Q. Chen, J. Gu, and P. Gu, "Deep reinforcement learning for provisioning virtualized network function in inter-datacenter elastic optical networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 3341–3351, May 2022.

[24] X. Chen, B. Li, R. Proietti, Z. Z. H. Lu, and S. J. B. Yoo, "Deeprmsa: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4155–4163, Aug. 2019.

[25] T. Tang, B. Wu, and G. Hu, "A hybrid learning framework for service function chaining across geo-distributed data centersa hybrid learning framework for service function chaining across geo-distributed data centers," *IEEE Access*, vol. 8, pp. 170 225–170 236, Sep. 2020.

[26] B. Li and Z. Zhu, "Gnn-based hierarchical deep reinforcement learning for nfv-oriented online resource orchestration in elastic optical dcis," *Journal of Lightwave Technology*, vol. 40, no. 4, pp. 935–946, Feb. 2022.

[27] Y. Xie *et al.*, "Virtualized network function forwarding graph placing in sdn and nfv-enabled iot networks: A graph neural network assisted deep reinforcement learning method," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 524–537, Mar. 2022.

[28] T. Tovinger, "Management, orchestration and charging for 5g networks," Mar. 2018. [Online]. Available: https://www.3gpp.org/news-events/1951-sa5_5g

[29] 3GPP, "3gpp tr 28.801 v15.1.0," Oct. 2018. [Online]. Available: https://www.3gpp.org/ftp/TSG_SA/WG5_TM/TSGS5_116/SA_78/28801-f10.doc

[30] N. Zhang, S. Zhang, P. Yang, O. Alhussein, W. Zhuang, and X. S. Shen, "Software defined space-air-ground integrated vehicular networks: Challenges and solutions," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 101–109, Jul. 2017.

[31] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," Mar. 2020. [Online]. Available: https://arxiv.org/abs/1706.02275

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," Aug. 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[33] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," Jul. 2021. [Online]. Available: https://arxiv.org/abs/2103.01955

[34] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," Feb. 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[35] P. Pan, Q. Fan, S. Wang, X. Li, J. Li, and W. Shi, "Gcn-td: A learning-based approach for service function chain deployment on the fly," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Taipei, Taiwan, Dec. 2020, pp. 1–6.

[36] 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 *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, Montreal, QC, Canada, Oct. 2020, pp. 1–6.

[37] S. Qi, S. Li, S. Lin, M. Y. Saidi, and K. Chen, "Energy-efficient vnf deployment for graph-structured sfc based on graph neural network and constrained deep reinforcement learning," in *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Tainan, Taiwan, Sep. 2021, pp. 348–353.

[38] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," Mar. 2020. [Online]. Available: https://arxiv.org/abs/1808.05689

[39] J. Bai, H. Yuan, L. Zhang, and J. Zhang, "Sgw-scn: An integrated machine learning approach for workload forecasting in geo-distributed cloud data centers," *Information Sciences*, vol. 481, pp. 57–68, May 2019.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Jan. 2017. [Online]. Available: https://arxiv.org/abs/1412.6980v9

[41] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," Sep. 2015. [Online]. Available: https://arxiv.org/abs/1509.06461