Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Combining graph neural network with deep reinforcement learning for resource allocation in computing force networks*

Xueying HAN†1, Mingxi XIE[2], Ke YU†‡2, Xiaohong HUANG[1], Zongpeng DU[3], Huijuan YAO[3]

[1]*School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China*

[2]*School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing 100876, China*

[3]*Department of Infrastructure Network Technology Research, China Mobile Research Institute, Beijing 100032, China*

†E-mail: hanxueying@bupt.edu.cn; yuke@bupt.edu.cn

Received Jan. 5, 2023; Revision accepted Apr. 24, 2023; Crosschecked Apr. 24, 2024

Abstract: Fueled by the explosive growth of ultra-low-latency and real-time applications with specific computing and network performance requirements, the computing force network (CFN) has become a hot research subject. The primary CFN challenge is to leverage network resources and computing resources. Although recent advances in deep reinforcement learning (DRL) have brought significant improvement in network optimization, these methods still suffer from topology changes and fail to generalize for those topologies not seen in training. This paper proposes a graph neural network (GNN) based DRL framework to accommodate network traffic and computing resources jointly and efficiently. By taking advantage of the generalization capability in GNN, the proposed method can operate over variable topologies and obtain higher performance than the other DRL methods.

Key words: Computing force network; Routing optimization; Deep learning; Graph neural network; Resource allocation

https://doi.org/10.1631/FITEE.2300009 **CLC number:** TP393

1 Introduction

In the era of cloud computing, Internet service providers (ISPs) typically transfer data to a centralized cloud computing center to process massive data. The powerful storage and computing capabilities of cloud computing fulfill the requirements of traditional network businesses (Kiani and Ansari, 2018). However, with the development of fifth-generation wireless communication (5G) and artificial intelligence, tremendous real-time applications are emerging, and the amount of data generated is exploding in the network, bringing new requirements for network latency, bandwidth, and computing response time. To address these needs, multi-access edge computing (MEC) has emerged. MEC schedules requested service to local edge computing nodes, which alleviates the pressure on cloud computing servers (Mao et al., 2017; Tran et al., 2017). However, deploying MEC sites often comes with considerable constraints, and the edge nodes and cloud computing center are not well coordinated in the network.

Predictably, in the future digital society, a significant number of computing devices will be dispersed at different sites close to users, and provide them with various personalized services through global networks. Users will therefore be able to access ubiquitous computing resources on demand

‡ Corresponding author

* Project supported by the Beijing University of Posts and Telecommunications - China Mobile Research Institute Joint Innovation Center

ORCID: Ke YU, https://orcid.org/0000-0002-1158-1483

anytime and anywhere (Barbarossa et al., 2014). As the essential computing resource, computing power has been defined by William D. NORDHAUS, winner of the 2018 Nobel Prize in Economics, as "the amount of information and data processed per second" (Nordhaus, 2001). In other words, computing power refers to the computing capability of a device to output specific results by processing data.

Motivated by these challenges, the network industry has proposed the computing force network (CFN) as a promising paradigm that leverages ubiquitous computing resources distributed in the network (China Mobile and Huawei Technologies, 2019; Du et al., 2022; Yao et al., 2022). In CFN, the network layer can schedule service requests to eligible computing nodes while simultaneously optimizing network resources. CFN serves as the bridge connecting the data sources and computing nodes, and is required to coordinate the network resources (e.g., delay, jitter, and link bandwidth) and computing resources (e.g., computing nodes and computing power). Therefore, it is indispensable to implement a resource allocation mechanism in CFN to leverage computing and network resources.

Traditional optimization methods require accurate mathematical modeling, cannot solve the optimization problem in real time, and therefore are not applicable in the CFN resource allocation scenario. Recent advances in deep reinforcement learning have dramatically improved the real-time decision-making process, but still suffer from the generalization problem. The deep reinforcement learning (DRL) framework requires data be represented in a Euclidean domain (e.g., images in a two-dimensional grid and text in a one-dimensional sequence), but in network resource allocation tasks, network topology components (including links, nodes, and their relations) are non-Euclidean graph-structure data. These data contain rich information about the relationships (edges) between nodes that are naturally suitable to graph neural networks (GNNs).

In this paper, we present a GNN-based DRL architecture that copes with the three challenges in the CFN scenario. The main contributions of the paper are summarized as follows: First, we leverage the model-free DRL framework to devise a multi-dimensional reward function that accommodates network traffic and computing resources. Second, the proposed method can make a real-time decision after the training phase, so the agent can obtain the optimization result in a real-time response especially for time-sensitive computing applications. Finally, existing algorithms based on DRL face the generalization problem; when the network topology is updated, previous models need to be retrained. By taking advantage of GNN, we integrate the message passing neural network (MPNN) in the internal DRL neural network, and experimental results show that the proposed method can operate and generalize over diverse network topologies, even if the structure changes.

2 Related works

In this section, we present an overview of the CFN architecture and some typical examples of resource allocation methods used in relevant cloud–edge computing networks, reveal some challenges in CFN, and briefly introduce the graph-based learning models as the theoretical basis of our solution.

2.1 Overview of CFN

CFN, also known as the computing first network, is an emerging paradigm proposed by the network community (China Mobile and Huawei Technologies, 2019; Du et al., 2022; Yao et al., 2022). It aims to connect the ubiquitous resources distributed in the terminal, edge, and cloud to provide ultra-low-latency and real-time computing services for diverse applications across the network. Fig. 1 demonstrates the architecture of CFN, which consists of five entities: clients, computing nodes, ingress nodes, egress nodes, and intermediate nodes. The service ID concept is introduced in CFN when a computing service or resource is registered; a unique service ID is assigned according to the service type, so that clients can include the service ID when initiating a computing service request of a particular type.

The functionalities of each entity are described as follows:

Clients: Clients are the nodes that initiate computing service requests. The requests include the computing resource demand (i.e., computing power) and network resource demand (e.g., delay, bandwidth, and packet loss), and must specify a service type using the service ID.

Computing nodes: Computing nodes are the nodes that possess computing resources, provide

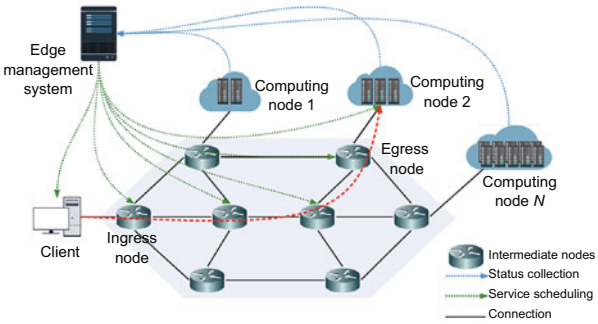

Fig. 1 Architecture of the computing force network (CFN)

particular computing services, and handle the computation results. The supported services are identified as a series of service IDs.

Ingress nodes: These nodes are the first-hop routers nearest the client. As the entry point of the network, ingress nodes have particular functionality to work as the decision-maker in the centralized CFN. When requests arrive, ingress nodes are responsible for dispatching services and path selection based on computing resources and network status. Ingress nodes maintain computing-augmented routing information (i.e., routing information base, or RIB) to dynamically schedule service requests to eligible computing nodes through the optimal path. In software-defined networking (SDN) environments, the ingress node executes the forwarding function according to the policy from the control plane.

Egress nodes: Egress nodes are the first-hop routers that are directly connected to computing nodes. They are responsible for collecting the computing status and registering computing services with the edge management system. In a decentralized CFN, the egress node generates a computing status advertisement and propagates it through the network. This enables all ingress nodes to retrieve the advertisement and calculate the computing-augmented routing information table.

Intermediate nodes: Intermediate nodes serve as a bridge between clients and computing nodes, exchanging computing status information and forwarding this information to their peers.

In CFN, application requests for computing services will no longer be limited to the capacity of a specific service node. Still, they will be routed to the optimal computing node for processing based on the computing resource demand and network resources (e.g., delay, bandwidth, and jitter) combined with the available network path.

Under the dual constraints of network resources and application computing demand, CFN can categorize and decompose application services based on network and computing requirements. CFN routes traffic to appropriate computing nodes to provide accurate and differentiated computing services while optimizing application service quality and network-wide resource utilization.

2.2 Resource allocation in distributed computing

In the present study of resource allocation in edge computing, most research has been carried out on either computing resources (e.g., computing power and storage) or network resources (e.g., delay, bandwidth, jitter, and packet loss).

Yang et al. (2018) proposed a resource allocation framework for MEC to tackle resource waste and the computation congestion problem by selecting an MEC node and network path according to certain rules and mechanisms in a static network, but their scheme achieved low efficiency when the network environment was updated; such results are unsatisfactory in dynamic environments. Wang et al. (2021) investigated a computing force proactive IP-optical integrated network by restructuring an architecture with an adaptive topology algorithm to provide on-demand services for the dynamic computing service demands in edge networks. Results indicated that the average network resource utilization was greatly improved. However, the architecture works only in MEC networks where the algorithm breaks the separation status between the IP layer and the optical layer, and thus is not realistic in CFN. Moreira et al. (2018) proposed an approach for mitigating a globally managed network and computing resources for multimedia applications. They focused on deploying flexible multimedia applications provided in the cloud, with a control plane entity capable of orchestrating computing and network resources for the multimedia application scenario that relies on session initiation protocol (SIP) control messages. Experiments demonstrated that the proposed approach brings quality-of-service (QoS) enhancement to the user through resiliency, load balancing, packet inspection, scaling on demand, and decoupling control and data planes, but the deployment is restricted in the SDN

environment and is not applicable in the current IP infrastructure.

Although the computing network considers the dual constraints of network resources and application computing demand jointly, previous allocation methods cannot be applied in CFN directly. Liu et al. (2021) proposed the CFN-dyncast protocol, a promising distributed technique that dispatches client computing demands to an optimal site according to the load of each computing site and the network status, but they did not provide a feasible solution that solves the practical challenge in implementation. Traditional optimization methods require accurate modeling of the network, and because the IP topology optimized for a particular traffic demand may not have the same performance for different computing demands, the above research has a scalability problem and lack of real-time responses in dynamic network environments.

With the advantage of recent breakthroughs in deep neural networks applied to reinforcement learning, many researchers have attempted to leverage the emerging DRL technique to enable model-free control in network optimization.

Xu et al. (2018) initially adopted the deep deterministic policy gradient (DDPG) algorithm to implement an experience-driven approach for resource allocation in communication networks, and presented the first DRL-based traffic engineering algorithm that does not require complex modeling and generates real-time results where the network changes frequently. After this, an increasing amount of literature on DRL-based resource allocation started to thrive. Ren et al. (2021) used DRL to schedule service offloading and mitigation for delay-sensitive services in vehicular edge computing (VEC) networks. Guo et al. (2020) proposed DRL-driven service scheduling and orchestration in the trusted cloud–edge network, to realize trusted resource sharing and allocation. Yu et al. (2021) combined federated learning with the DRL method to achieve stable, reliable, and real-time interactions between edge nodes and their serving edge computing nodes in the 5G ultradense network. Li et al. (2019) proposed a DRL-based approach for joint optimization of network and computing resources in machine-to-machine (M2M) networks, focusing mainly on Internet of Things (IoT) devices. Sun et al. (2021) combined the advantage of DRL and GNN to design a virtual network function (VNF) placement scheme called DeepOpt, to deal with the VNF placement problem with multiple resource types in the network. These methods can deal with the scalability problem and obtain decisions in real time, providing a feasible plan for time-sensitive applications, but they still suffer from the dilemma where all the nodes in CFN are updated and the topology may vary in dynamic environments. In the above-mentioned studies, the structure of the network is not fully used because most state-of-the-art deep neural networks are designed for Euclidean structure data (e.g., image, video, and text).

In summary, three challenges presented in previous studies are vital and need to be solved in CFN:

1. The primary challenge in CFN is to accommodate network traffic and computing resources jointly and efficiently, because the key performance indicators (KPIs) in CFN are multi-dimensional.

2. The method should obtain optimization results in a timely manner, because most of the applications require a real-time response in an ultra-low-latency environment.

3. Existing algorithms based on DRL have the generalization problem. When the network topology is updated, previous models need to be retrained.

2.3 Graph neural networks

Previous studies of DRL-based techniques cannot deal with the generalization problem, especially for those topologies not seen during the training phase. The reason behind this phenomenon is that the previous DRL methods cannot learn from the topology where links and nodes are all structured as graph data. To amend this shortcoming, graph-based deep learning, represented by GNN models, has gained massive attention in the network community and is designed particularly to achieve relational reasoning on non-Euclidean graph data (Ruiz et al., 2021; Wu et al., 2021; Zhang et al., 2022). GNNs are suitable for optimization problems in computer networks. Because of their strong learning capabilities, they can capture spatial information hidden in the network topology and generalize for invisible topologies when the network dynamically changes.

By using GNNs to model networks, the estimation of various network metrics or KPIs (e.g., delay, bandwidth, jitter, and packet loss) was addressed in previous studies. Provided by network

topology, routing algorithm, and traffic matrices, most research was conducted in a supervised (Geyer, 2017; Badia-Sampera et al., 2019; Rusek et al., 2019; Suárez-Varela et al., 2019; Ferriol-Galmés et al., 2020) or semi-supervised (Suzuki et al., 2020) way. Numerous GNN variants have been used for network modeling, including GNNs (Ferriol-Galmés et al., 2020), graph convolutional networks (GCNs) (Sun et al., 2021; Almasan et al., 2022; Xie et al., 2023), and MPNNs (Badia-Sampera et al., 2019; Rusek et al., 2019; Suárez-Varela et al., 2019). An example of MPNN is shown in Fig. 2.

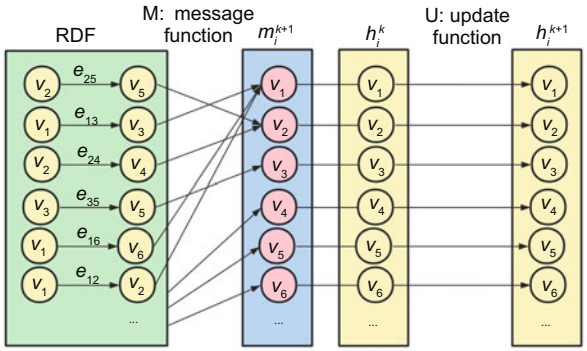

Fig. 2 An example of the message passing neural network

3 GNN combined with DRL in CFN

In this section, we present our GNN-based DRL architecture for resource allocation in CFN, which is aimed at optimizing network bandwidth and service scheduling. We propose the GNN-based DRL architecture, discuss the essential components, and outline the overall workflow of the proposed method.

3.1 Major components in the GNN-based DRL framework

The overview of the proposed architecture is shown in Fig. 3. The standard DRL framework consists of five major components: (1) environment, (2) agent, (3) state, (4) action, and (5) reward. In the learning process, the agent serves as the brain, collects the current states from the environment, takes an action according to its policy, and receives rewards as feedback. These components are described as follows:

Environment: The environment in our framework refers to the data plane in CFN, where the agent can learn meaningful information and relationships between entities (i.e., links and nodes) by interacting with the environment.

Agent: The agent serves as the decision-maker, interacts with the environment, and observes the consequences by trial and error. As the agent learns different actions and receives instantaneous rewards and new environment states, the agent evaluates each state–action pair's expected reward and updates its internal neural networks. In a centralized CFN, the control plane can work as the agent to collect states, make decisions, and deploy an action to the data plane.

State: The state refers to the state of links and nodes. The state of links contains information about the paths that go through the link, the connected node pair, link capacity, delay, and other network metrics. The state of nodes involves nodal features, their neighborhood, and the computing status (e.g., computing types, loads, and computing power) in each computing node.

Action: In the learning phase, the agent maps the state space into the action space and aims to make an optimal decision by choosing the action with the highest expected reward. The action space in the CFN scenario involves path selection, service scheduling (choosing eligible computing nodes), and allocation of bandwidth resources for the traffic.

Reward: By modeling networks using GNN, the estimation of various network metrics or KPIs (e.g., delay, bandwidth, jitter, and packet loss) was concerned with in previous studies. The reward for taking an action represents the feedback from the observed environment, and the purpose of the reward is to give the agent a target to learn a long-term strategy by maximizing the reward function.

As shown in Fig. 3, we implement GNN in the original convolutional neural network (CNN) to transfer the success of GNN on graph data. CNN models fail to operate in the network topology that is not seen during training, because CNN is designed to understand the spatial structure. In contrast, computer networks are fundamentally represented as graphs with relational features on nodes and links.

3.2 GNN and the message passing process

MPNN, as a general graph-based framework, was first introduced in chemistry for message passing and information aggregation (Gilmer et al.,

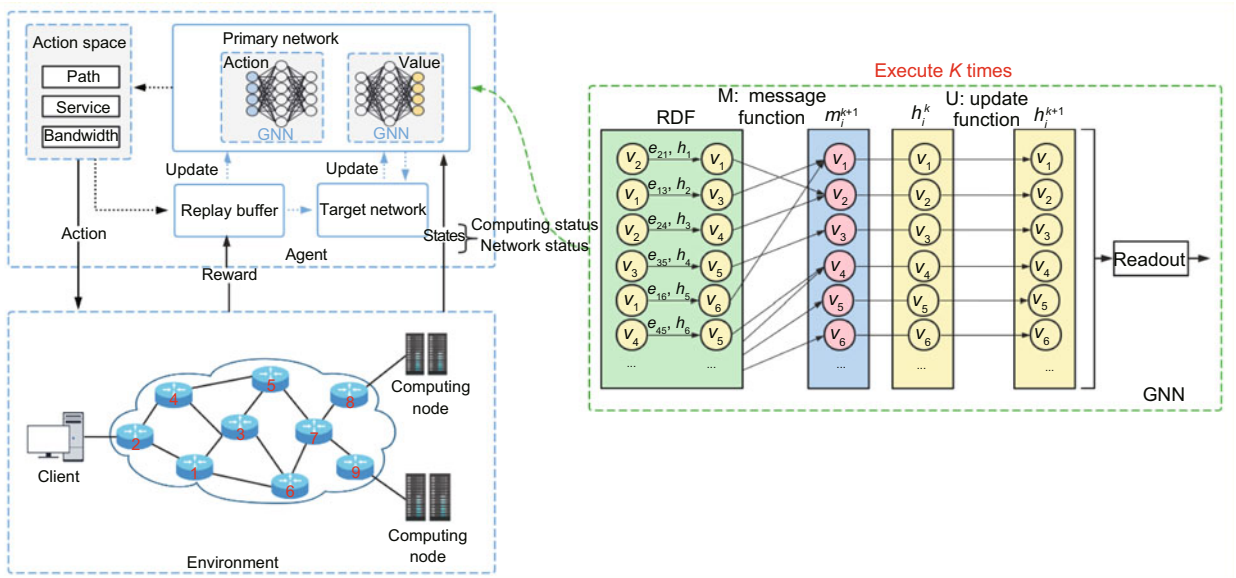

Fig. 3 Overview of the graph neural network (GNN) based deep reinforcement learning (DRL) framework in the computing force network (CFN)

2017). Fig. 2 demonstrates an example of MPNN. In network modeling, a topology can be defined as $G = (V, E)$, where V represents the set of nodes and E is the set of edges between connected nodes. Both nodes and edges are associated with features or attributes, which are encoded using a resource description framework (RDF). MPNN adopts a message passing function to propagate information between the nodes, and aggregate and update the state of each element; the process will be iteratively executed K times. The general process of MPNN involves two phases, a message passing phase and a readout phase.

Algorithm 1 illustrates that the message passing phase operates for K time steps, and each node i receives messages from all neighbors in terms of message function M^k and update function U^k. In Eq. (1), $N(v_i)$ denotes the set of neighbors of node i, h_i^k and h_j^k represent the hidden states at nodes i and j respectively, and e_{ij} denotes the edge features on edge $< i, j >$. All hidden states will be updated according to message m_i^{k+1} in Eq. (1) and update function U^k in Eq. (2):

$$m_i^{k+1} = \sum_{v_j \in N(v_i)} M^k \left(h_i^k, h_j^k, e_{ij}\right), \tag{1}$$

$$h_i^{k+1} = U^k \left(h_i^k, m_i^{k+1}\right). \tag{2}$$

After message passing, the feature vector can be computed in the readout phase, which is noted as the readout function R according to

$$\hat{y} = R\left(\left\{h_i^k \mid i \in G\right\}\right). \tag{3}$$

Note that MPNN can also learn edge features by introducing hidden states for all edges in the graph, and the structure of MPNN can be adopted flexibly to reason different relationships in links and nodes (Gilmer et al., 2017). The message function M^k, update function U^k, and readout function R may vary depending on certain scenarios.

Algorithm 1 Message passing process

Input: x_l
Output: h_i^k, $\hat{y}$, q
for $l \in \mathcal{L}$ **do**
  $h_l^0 \leftarrow [x_l, 0, \ldots, 0]$
end for
for $k = 1$ to K **do**
  for $l \in \mathcal{L}$ **do**
    Aggregate $m_i^{k+1} = \sum_{v_j \in N(v_i)} M^k \left(h_i^k, h_j^k, e_{ij}\right)$
    Update $h_i^{k+1} = U^k \left(h_i^k, m_i^{k+1}\right)$
  end for
end for
$r \leftarrow \sum_{l \in \mathcal{L}} h_l$
$q \leftarrow R(r)$
return q

3.3 GNN-based DRL agent design

In this study, we explore the potential of a deep Q-learning network (DQN) agent combined

with GNN to operate in CFN by choosing optimal computing nodes and allocating network resources (i.e., bandwidth). With the advantages of GNN, the proposed approach can operate and generalize over the dynamic environment where the topology may change.

3.3.1 Link states in MPNN

DQN is a DRL algorithm that explores all the possible combinations of states and actions by creating a q-table. The table maps each state–action pair to its corresponding q-value using neural networks. MPNN, as a general framework of GNN, can be adopted as the internal neural network in the DQN agent. MPNN components were demonstrated in Section 2.3. The message passing phase involves two essential functions: the message passing function in Eq. (1) and the update function in Eq. (2). Apart from the nodes and links in both network topologies, we encode hidden states in MPNN, denoted as $h_i = (x_1, x_2, ..., x_N)$; the states are described in Table 1. The values x_1 to x_3 represent network states and the action of bandwidth allocation. Link capacity specifies the threshold of available bandwidth. Link betweenness represents the number of paths that flow through the link divided by the total number of paths, and reflects the priority of the link. With greater link betweenness, the link is more likely to be the critical link (bottleneck) in the topology. The computing resources are stored in x_4 to x_6, including the computing type, load, and node assigned for the service request. For the readout phase in Eq. (3), MPNN receives link features x_i and outputs the q-value to estimate the expected reward for the DQN agent.

Table 1 Notations of hidden states

Symbol	Meaning
x_1	Link available capacity
x_2	Link betweenness
x_3	Action vector (bandwidth allocated)
x_4	Node Com_states_type
x_5	Node Com_states_load
x_6	Action vector (node assigned)
x_7–x_N	Zero padding

3.3.2 Reward function

The reward function in the DRL agent represents the objective function that the agent learns to maximize. In the CFN scenario, the agent learns to choose eligible computing nodes while allocating bandwidth through the network path.

Computing power refers to the amount of information and data processed per second (Nordhaus, 2001). To quantify computing power, we refer to the service offloading in MEC, where computing power can be defined as the number of data processes per second. Therefore, the computing time for a single task can be calculated as the size of the computing task (in bits) divided by the computing power (in bits/s).

When we consider optimizing both network resources (bandwidth) and computing resources (computing power), we devise our reward function as

$$\text{reward} = \alpha \cdot \text{Bandwidth}\left(p_{m,n}\right) - \beta \cdot \text{Time}_n, \quad (4)$$

where $\text{Bandwidth}\left(p_{m,n}\right)$ denotes the bandwidth allocation from client m to computing node n through path p, and Time_n refers to the time consumed for the requested task on computing node n. α and β are the normalization factors to ensure that these indicators can be calculated in the same dimension.

To reduce the complexity, we adopt a preprocessing step in path selection, in which the candidate path set is calculated, and the size is set to three. Because network cost is economically vital to the decision-making process, the weighted Dijkstra algorithms are used to calculate the least-cost path, where the metric is the weighted sum of the link cost, and the weight of each link is defined to be inversely proportional to the link capacity.

3.3.3 GNN-based DRL agent operation

The DRL agent learns to take an action by interacting with the environment. Algorithm 2 illustrates the learning process. In the beginning stage, we initialize the environment and experience replay buffer. In the meantime, the environment generates a service request defined by $\{\text{src}, \text{SID}, \text{cp}, \text{bw}\}$, where src represents the client, SID specifies the computing service type, cp represents the computing power demand, and bw represents the bandwidth demand. Also, the reward is set to zero and the weights of the neural networks are randomly initialized.

After that, we execute a loop where the agent selects an action a_t, observes reward r_t, and receives a new state s_{t+1} from the data plane. To reduce

the complexity, we adopt three candidate network paths and three eligible computing nodes for service scheduling. The neural network stores the transition $(s, \text{src}, \text{SID}, \text{cp}, \text{bw}, a, r, s', \text{src}', \text{SID}', \text{cp}', \text{bw}')$ in the experience replay buffer and samples them in minibatches. Experience replay is able to update the network parameters using the q-value and stored information from previously taken actions. The above process is executed multiple times for convergence.

Algorithm 2 DRL agent learning process

$s, \text{src}, \text{SID}, \text{cp}, \text{bw} \leftarrow$ env.init_env()
reward $\leftarrow 0$
memory $\leftarrow \{\}$
for each episode **do**
  candidate_paths $\leftarrow$ compute_paths(src, dst)
  for $k = 1$ to K **do**
    $p' \leftarrow$ get_path(k, candidate_paths)
    $s' \leftarrow$ allocate(s, p', src, dst, cp_dem, bw_dem)
    Compute q_values based on Algorithm 1
  end for
  q_value $\leftarrow$ epsilon_greedy(q_values, ϵ)
  $a \leftarrow$ get_action(q_values, candidate_paths, s)
  $r, s', \text{src}', \text{SID}', \text{cp}', \text{bw}' \leftarrow \text{env}(s, a)$
  memory $\leftarrow (s, \text{src}, \text{SID}, \text{cp}, \text{bw}, a, r, s', \text{src}', \text{SID}', \text{cp}', \text{bw}')$
  reward $\leftarrow$ reward $+ r$
  Sample from memory
  src $\leftarrow$ src$'$; SID $\leftarrow$ SID$'$; cp $\leftarrow$ cp$'$; bw $\leftarrow$ bw$'$; $s \leftarrow s'$
end for

4 Experimental results

In this section, we train and evaluate our GNN-based DRL agent in several commonly used network datasets from the open-source Survivable Network Design library, or SNDlib (Orlowski et al., 2010). The experiment aims to verify whether the agent can efficiently allocate bandwidth demand and schedule computing services to eligible computing nodes by maximizing the reward. Also, the generalization capability of the proposed method is presented and evaluated.

4.1 Experiment settings

The experiments are conducted and implemented using TensorFlow on a personal laptop with Core i7-11800 (2.3 GHz) with 16 GB memory, and the GPU is NVIDIA 3060/LHR.

The GNN-based DRL agent is trained on several topologies from SNDlib: Germany50 (50 nodes in total), Geant2 (23 nodes), Cost266 (37 nodes), and India35 (35 nodes); the topologies are shown in Fig. 4. In each topology, several nodes (red nodes) are randomly selected as clients, several (green nodes) as computing nodes, and blue nodes represent CFN nodes (including ingress, egress, and intermediate nodes). For evaluation, the agent is trained during 1000 episodes with 50 iterations in each episode, and then 50 episodes are conducted in the testing phase. We conduct many experiments to verify the convergence and performance, based on the current model; 50 iterations are enough to obtain a valid and stable result. The metric used is the cumulative reward gained in each episode, which can be calculated as in Eq. (4). The metric can be seen as the performance indicator of both network resources and computing resources during the experiments.

4.2 Performance evaluation

We compare the proposed GNN-based DQN agent with another DQN model (called Net_First) and a load-balancing routing scheme (called Com_First) in a series of topologies and datasets. Fig. 4 presents the topology structure of each dataset. In our experiments, computing nodes and client nodes are randomly set in the topology. Because the number of these nodes is proportional to the topology size, the remaining nodes serve as CFN nodes for forwarding. The service request from the client node (source node) contains bandwidth demand, service ID (SID), and computing power demand. The link capacity of each link in our topology is set according to real data in datasets. The bandwidth demand and computing power demand are randomly generated following a normal distribution. Our agent aims to select eligible computing nodes for each request while properly allocating the bandwidth for the source–destination pair. By the definition of the reward function in Eq. (4), the action taken by the agent reaches a higher reward when the bandwidth allocated is as large as possible, and the time consumed for computing service remains small, which represents a challenging optimization scenario in CFN.

We compare our proposed GNN-based DRL method (referred to as the GNN+DQN agent) with two baseline solutions in terms of cumulative reward.

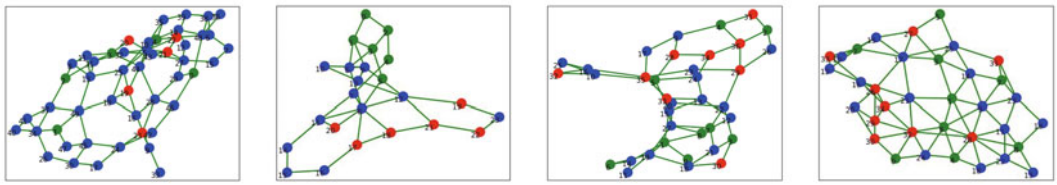

Fig. 4 Topologies of datasets from SNDlib: (a) Germany50; (b) Geant2; (c) Cost266; (d) India35. Several nodes (red nodes) are randomly selected as clients, several (green nodes) as computing nodes, and blue nodes represent CFN nodes (including ingress, egress, and intermediate nodes). References to color refer to the online version of this figure

A load-balancing routing scheme (Com_First) and the original DQN model (Net_First) are selected as baselines for performance evaluation. Com_First emphasizes the computing resource, selecting the computing node with the shortest computing time and then allocating bandwidth in multiple candidate paths in a load-balancing way. Net_First is the original DQN algorithm that focuses on network resources. The Net_First agent also decides the action like the GNN+DQN agent does, including computing node selection, path selection, and bandwidth allocation, but it is embedded with the normal CNN. The metric on which we focus is the cumulative reward after 50 iterations in each testing episode. With the definition in Eq. (4), the cumulative reward (normalized as Score) represents the overall performance considering both network resources and computing resources, which indicates that the bandwidth allocated is as large as possible and that the time consumed for the computing service remains small. The performance comparison of each method in different topologies is presented in Fig. 5. The mean values are given by dashed lines, from which we can see that the performance of the GNN+DQN agent is superior to those of the baseline solutions in each topology. In Germany50, our proposed GNN+DQN agent can reach a mean value of 0.51, while the baseline values are 0.37 for Net_First and 0.14 for Com_First. In the smallest topology, Geant2, with only 23 nodes in total, the GNN+DQN method achieves 0.55 on average, while Com_First and Net_First achieve only 0.39 and 0.42, respectively. In the 37-node topology Cost266, the mean values of our proposed method, Com_First, and Net_First are 0.53, 0.24, and 0.26, respectively. In the last topology, India35, the average reward is 0.48 for our method, while 0.21 and 0.26 for the two baseline solutions.

4.3 Generalization capability

To verify the effectiveness and generalization capability of our GNN+DQN agent, we conduct four experiments using the India35 topology. In these experiments, we randomly delete 1, 3, and 5 nodes to generate a dynamic topology structure, so the environment changes while the agent is trained through the original topology with 35 nodes in total. Fig. 6 illustrates the numerical results; when 1 or 3 nodes are randomly changed, the cumulative reward of our agent keeps at the same level, which indicates that the proposed method can operate and generalize over diverse network topologies. The intuition behind this phenomenon is that when the GNN-based DRL agent learns the graph relationships among nodes and links, it can reason and generalize the decision even if the topology changes. However, the numerical result of deleting 5 nodes is reduced to 0.064 after several experiments, which we believe is because the fives nodes randomly deleted may involve the computing nodes that provide the computing service, which will lead to service degradation and significant increase in the computing time, resulting in a degraded reward value.

The original DQN agent, Net_First, is also compared with the GNN+DQN agent in the India35 topology. In the experiment, we randomly delete three nodes in the India35 topology to verify whether GNN+DQN obtains better performance than Net_First. The results are presented in Fig. 7, where "GNN+DQN original" and "Net_First original" represent the cumulative rewards on the complete India35 topology with 35 nodes, and "GNN+DQN–3 nodes" and "Net_First–3 nodes" represent the experimental results when randomly deleting three nodes. From the results, we can

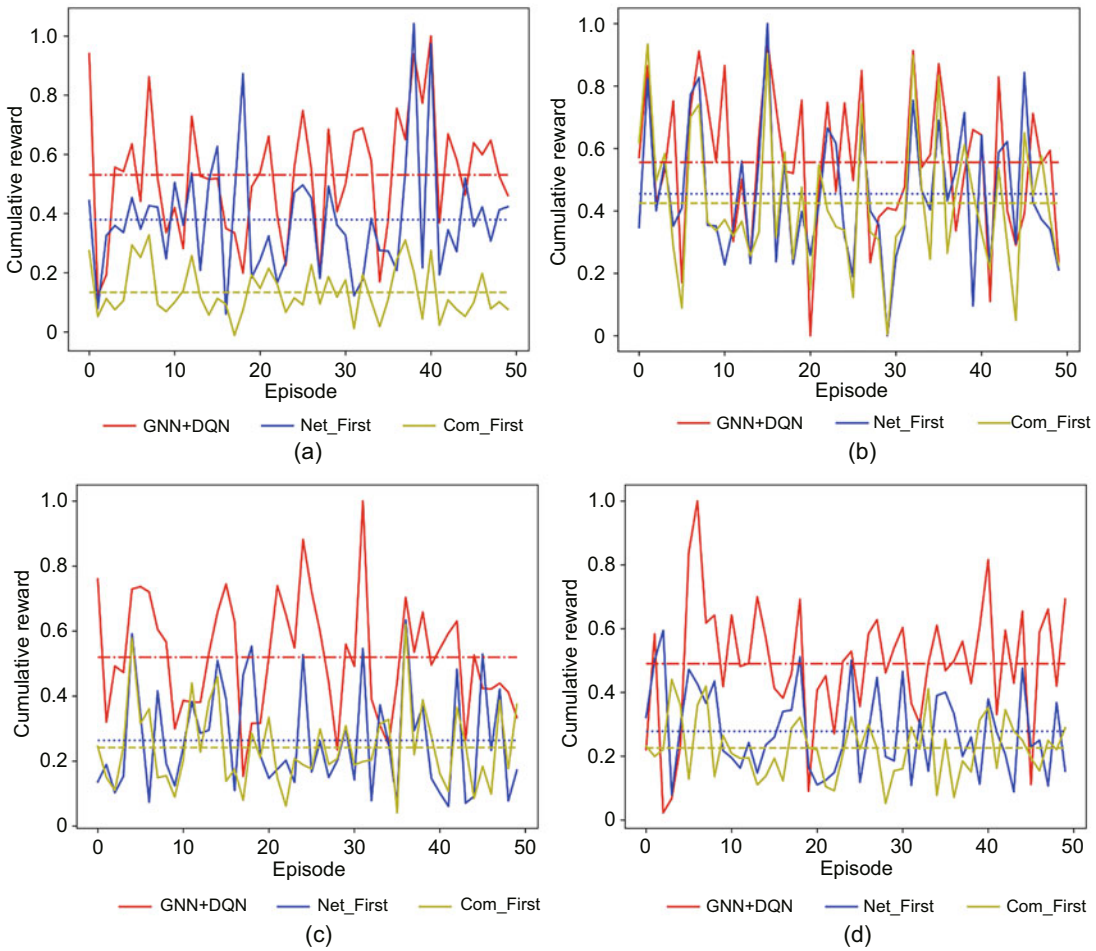

Fig. 5 Performance comparison of the cumulative reward with different methods: (a) Germany50; (b) Geant2; (c) Cost266; (d) India35. References to color refer to the online version of this figure

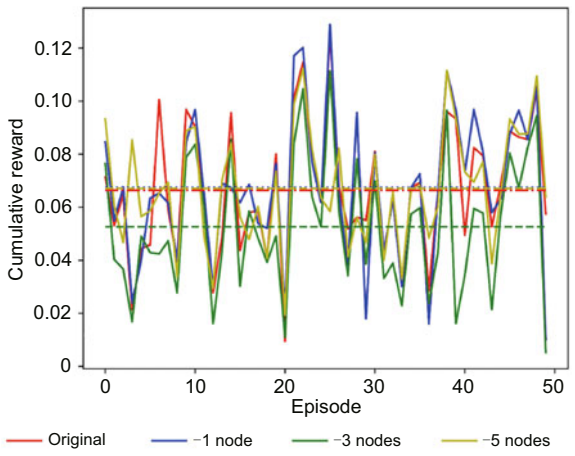

Fig. 6 Performance of the GNN+DQN agent when randomly deleting nodes in the India35 topology. References to color refer to the online version of this figure

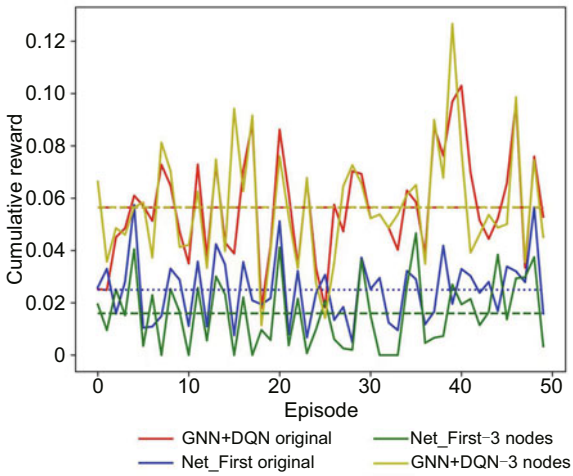

Fig. 7 Performance comparison when randomly deleting nodes in the India35 topology. References to color refer to the online version of this figure

conclude that the cumulative reward of Net_First decreases from 0.022 to 0.017, whereas GNN+DQN keeps at the same reward level.

5 Summary

In the emerging CFN scenario, the network can schedule service requests to eligible computing nodes while optimizing network resources. In this paper, we present a GNN-based DRL architecture that copes with the three challenges in the CFN scenario. First, we leverage the model-free DRL framework to devise a multi-dimensional reward function that accommodates network traffic and computing resources jointly. Second, once the proposed method finishes its training phase, the agent can obtain optimized decisions in real time. Finally, existing algorithms based on DRL face the generalization problem, in which previous models need to be retrained when the topology is updated. By taking advantage of GNN, we implement MPNN in the internal DQN neural network. Experimental results show that the proposed method can operate and generalize over diverse network topologies, even if the structure changes.

Contributors

Xueying HAN and Mingxi XIE designed the research. Mingxi XIE processed the data. Xueying HAN drafted the paper. Ke YU and Xiaohong HUANG helped organize the paper. Xueying HAN, Ke YU, Zongpeng DU, and Huijuan YAO revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Almasan P, Suárez-Varela J, Rusek K, et al., 2022. Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case. https://arxiv.org/abs/1910.07421

Badia-Sampera A, Suárez-Varela J, Almasan P, et al., 2019. Towards more realistic network models based on graph neural networks. Proc 15th Int Conf on Emerging Networking Experiments and Technologies, p.14-16. https://doi.org/10.1145/3360468.3366773

Barbarossa S, Sardellitti S, Lorenzo PD, 2014. Communicating while computing: distributed mobile cloud computing over 5G heterogeneous networks. *IEEE Signal Process Mag*, 31(6):45-55. https://doi.org/10.1109/MSP.2014.2334709

China Mobile, Huawei Technologies, 2019. Technical White Paper on Computing-Aware Networking.

Du ZP, Li ZQ, Duan XD, et al., 2022. Service information informing in computing aware networking. Proc Int Conf on Service Science, p.125-130. https://doi.org/10.1109/ICSS55994.2022.00027

Ferriol-Galmés M, Suárez-Varela J, Barlet-Ros P, et al., 2020. Applying graph-based deep learning to realistic network scenarios. https://arxiv.org/abs/2010.06686

Geyer F, 2017. Performance evaluation of network topologies using graph-based deep learning. Proc 11th EAI Int Conf on Performance Evaluation Methodologies and Tools, p.20-27. https://doi.org/10.1145/3150928.3150941

Gilmer J, Schoenholz SS, Riley PF, et al., 2017. Neural message passing for quantum chemistry. Proc 34th Int Conf on Machine Learning, p.1263-1272.

Guo SY, Dai Y, Xu SY, et al., 2020. Trusted cloud-edge network resource management: DRL-driven service function chain orchestration for IoT. *IEEE Int Things J*, 7(7):6010-6022. https://doi.org/10.1109/JIOT.2019.2951593

Kiani A, Ansari N, 2018. Edge computing aware NOMA for 5G networks. *IEEE Int Things J*, 5(2):1299-1306. https://doi.org/10.1109/JIOT.2018.2796542

Li M, Yang L, Yu FR, et al., 2019. Joint optimization of networking and computing resources for green M2M communications based on DRL. Proc IEEE Global Communications Conf, p.1-6. https://doi.org/10.1109/GLOBECOM38437.2019.9013366

Liu B, Mao JW, Xu L, et al., 2021. CFN-dyncast: load balancing the edges via the network. Proc IEEE Wireless Communications and Networking Conf Workshops, p.1-6. https://doi.org/10.1109/WCNCW49093.2021.9420028

Mao YY, You CS, Zhang J, et al., 2017. A survey on mobile edge computing: the communication perspective. *IEEE Commun Surv Tutor*, 19(4):2322-2358. https://doi.org/10.1109/COMST.2017.2745201

Moreira R, Silva F, Frosi P, et al., 2018. A flexible network and compute-aware orchestrator to enhance QoS in NFV-based multimedia services. Proc IEEE 32nd Int Conf on Advanced Information Networking and Applications, p.512-519. https://doi.org/10.1109/AINA.2018.00081

Nordhaus WD, 2001. The Progress of Computing. Cowles Foundation Discussion Papers.

Orlowski S, Wessäly R, Pióro M, et al., 2010. SNDlib 1.0—survivable network design library. *Networks*, 55(3):276-286. https://doi.org/10.1002/net.20371

Ren YL, Chen XY, Guo S, et al., 2021. Blockchain-based VEC network trust management: a DRL algorithm for vehicular service offloading and migration. *IEEE Trans Veh Technol*, 70(8):8148-8160. https://doi.org/10.1109/TVT.2021.3092346

Ruiz L, Gama F, Ribeiro A, 2021. Graph neural networks: architectures, stability, and transferability. *Proc IEEE*, 109(5):660-682. https://doi.org/10.1109/JPROC.2021.3055400

Rusek K, Suárez-Varela J, Mestres A, et al., 2019. Unveiling the potential of graph neural networks for network modeling and optimization in SDN. Proc ACM Symp on SDN Research, p.140-151. https://doi.org/10.1145/3314148.3314357

Suárez-Varela J, Carol-Bosch S, Rusek K, et al., 2019. Challenging the generalization capabilities of graph neural networks for network modeling. Proc ACM SIGCOMM Conf Posters and Demos, p.114-115. https://doi.org/10.1145/3342280.3342327

Sun PH, Lan JL, Li JF, et al., 2021. Combining deep reinforcement learning with graph neural networks for optimal VNF placement. *IEEE Commun Lett*, 25(1):176-180. https://doi.org/10.1109/LCOMM.2020.3025298

Suzuki T, Yasuda Y, Nakamura R, et al., 2020. On estimating communication delays using graph convolutional networks with semi-supervised learning. Proc Int Conf on Information Networking, p.481-486. https://doi.org/10.1109/ICOIN48656.2020.9016603

Tran TX, Hajisami A, Pandey P, et al., 2017. Collaborative mobile edge computing in 5G networks: new paradigms, scenarios, and challenges. *IEEE Commun Mag*, 55(4):54-61. https://doi.org/10.1109/MCOM.2017.1600863

Wang LN, Gu RT, Li ZK, et al., 2021. Computing-aware proactive IP-optical integrated network restructuring for edge computing. Proc 19th Int Conf on Optical Communications and Networks, p.1-3. https://doi.org/10.1109/ICOCN53177.2021.9563767

Wu ZH, Pan SR, Chen FW, et al., 2021. A comprehensive survey on graph neural networks. *IEEE Trans Neur Netw Learn Syst*, 32(1):4-24. https://doi.org/10.1109/TNNLS.2020.2978386

Xie M, Yu K, Wu X, 2023. Adaptive routing of task in computing force network by integrating graph convolutional network and deep Q-network. Proc 8th IEEE Int Conf on Network Intelligence and Digital Content, p.242-247. https://doi.org/10.1109/IC-NIDC59918.2023.10390731

Xu ZY, Jian T, Meng JS, et al., 2018. Experience-driven networking: a deep reinforcement learning based approach. Proc IEEE Conf on Computer Communications, p.1871-1879. https://doi.org/10.1109/INFOCOM.2018.8485853

Yang BX, Chai WK, Xu ZC, et al., 2018. Cost-efficient NFV-enabled mobile edge-cloud for low latency mobile applications. *IEEE Trans Netw Serv Manag*, 15(1):475-488. https://doi.org/10.1109/TNSM.2018.2790081

Yao H, Duan X, Fu Y, 2022. Computing-aware routing protocol for computing force network. Int Conf on Service Science, p.137-141 https://doi.org/10.1109/ICSS55994.2022.00029

Yu S, Chen X, Zhou Z, et al., 2021. When deep reinforcement learning meets federated learning: intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network. *IEEE Int Things J*, 8(4):2238-2251. https://doi.org/10.1109/JIOT.2020.3026589

Zhang ZW, Cui P, Zhu WW, 2022. Deep learning on graphs: a survey. *IEEE Trans Knowl Data Eng*, 34(1):249-270. https://doi.org/10.1109/TKDE.2020.2981333