Deep Reinforcement Learning with Graph Neural Networks for Capacitated Shortest Path Tour based Service Chaining

Takanori Hara and Masahiro Sasabe
Graduate School of Science and Technology, Nara Institute of Science and Technology,
8916-5 Takayama-cho, Ikoma, Nara 630-0192, Japan.
Email: hara@ieee.org, m-sasabe@ieee.org

***Abstract*—Network functions virtualization (NFV) realizes diverse and flexible network services by executing network functions on generic hardware as virtual network functions (VNFs). A certain network service is regarded as a sequence of VNFs, called service chain. The service chaining (SC) problem aims at finding an appropriate service path from an origin node to a destination node while executing the VNFs at the intermediate nodes in the required order under resource constraints on nodes and links. The SC problem belongs to the complexity class NP-hard. In our previous work, we modeled the SC problem as an integer linear program (ILP) based on the capacitated shortest path tour problem (CSPTP) where the CSPTP is an extended version of the SPTP with the node and link capacity constraints. We also developed the Lagrangian heuristics to achieve the balance between optimality and computational complexity. In this paper, we further propose a deep reinforcement learning (DRL) framework with the graph neural network (GNN) to realize the CSPTP-based SC adaptive to changes in service demand and/or network topology. Numerical results show that (1) the proposed framework achieves almost the same optimality as the ILP for the CSPTP-based SC and (2) it also works well without retraining even when the service demand changes or the network is partly damaged.**

***Index Terms*—Network functions virtualization (NFV), service chaining, capacitated shortest path tour problem (CSPTP), deep reinforcement learning (DRL), graph neural networks (GNNs).**

I. Introduction

With rapidly spreading smartphones and internet of things (IoT) devices, diverse services have constantly been created and the network traffic has exponentially been increasing. To achieve sustainable networking, automatic network operations based on network functions virtualization (NFV) have attracted many network service providers. NFV can decouple network functions from dedicated hardware and execute them as virtual network functions (VNFs) on generic hardware [1]–[3]. As a result, it can deploy network services with agility and flexibility as well as reducing capital expenditure (CAPEX) and operating expenditure (OPEX).

In NFV networks, a certain network service can be interpreted as a sequence of VNFs, called a *service (function) chain*. Given a *service chain request* (SCR), an NFV orchestrator has to solve a service chaining (SC) problem, which aims at finding a special path (i.e., *service path*) from an origin node to a destination node while executing the VNFs at the intermediate nodes in required order under resource constraints [1]. It is well known that the SC problem belongs to the complexity class **NP**-hard [1].

Several existing studies pointed out the similarity between SC and shortest path tour problem (SPTP) [4], [5]. SPTP is a variant of shortest path problem and aims at finding the shortest path from an origin node to a destination node while visiting at least one node from given disjoint node subsets, $\mathcal{T}_1, \dots, \mathcal{T}_k$, in this order. Focusing on this similarity, Bhat and Rouskas proposed an algorithm called depth first tour search (DFTS) to efficiently solve SPTP [4]. In our previous work [5], we modeled the SC problem as the capacitated SPTP (CSPTP) and formulated an integer linear program (ILP) for the CSPTP-based SC. CSPTP is an extension of SPTP with constraints on node and link capacities with real values. We also proposed the Lagrangian heuristics to solve the online CSPTP-based SC by taking account of the balance between optimality and computational complexity [6]. This approach, however, may not sufficiently work under dynamical demand change and/or network dynamics (e.g., temporal link failures).

Machine learning (ML) techniques have been applied to various domains in networking to solve diverse and complex optimization problems under uncertain environments [7]. Specifically, graph neural networks (GNNs) have been one of the promising approaches to explore hidden representation in networks from the complex relationship between network traffic and topologies [8]–[13]. In recent years, there are several studies applying ML and reinforcement learning (RL) techniques to SC [14]–[17]. They, however, did not sufficiently consider the following issues of CSPTP-related SC: (1) allowing the use of identical links as many times as required, (2) ensuring the service chain requirements, (3) meeting resource constraints, and (4) achieving resource allocation adaptive to demand and topology changes.

In this paper, we propose a deep RL (DRL) framework with the GNN for the online CSPTP-based SC, where the NFV orchestrator immediately serves a new SCR arriving at the NFV network. More specifically, the proposed framework aims at realizing resource allocation adaptive to changes in service demand and/or network topology (e.g., link failures).

This work was supported in part by the JSPS Research Activity Start-up under Grant 21K21288 and JSPS KAKENHI B under Grant 22H03586, Japan.

Through numerical results, we demonstrate the fundamental characteristics of the proposed framework as well as its generalization capabilities against demand/topology changes.

The rest of the manuscript is organized as follows. Section II gives the related work. In Section III, we introduce the some preliminaries, i.e., CSPTP-based SC, DRL, and GNN. In Section IV, we propose the DRL framework with the GNN for CSPTP-based SC. Section V shows the fundamental characteristics of the proposal. Finally, Section VI gives the conclusion and future work.

II. Related Work

A. Service Chaining Problem

Bhat and Rouskas first pointed out the similarity between SC and SPTP, and proposed the DFTS algorithm to find the shortest path tour [4]. DFTS, however, does not consider the resource constraints. The SPTP aims at finding the shortest path from an origin to a destination while visiting at least one intermediate node from given disjoint node subsets in required order [18]. Focusing on this similarity and the resource constraints on the NFV network, we modeled the SC problem as the CSPTP-based SC, where the CSPTP is an extended version of SPTP supporting general constraints on node and link capacities with real values [5]. In addition, we formulated the CSPTP-based SC as an ILP using a special network model called augmented network. In what follows, we refer to it as CSPTP-based ILP. The augmented network model can efficiently and agilely handle the SC problem under the online processing, compared with other existing network models, i.e., layered graph and expanded network models [19], [20].

To overcome the computational complexity, we further proposed a simple greedy-based heuristic algorithm [5] and a more sophisticated Lagrangian heuristics [6]. In [21], Gao and Rouskas applied the game-theoretic approach to SPTP-realted traffic steering for service chaining. However, these approaches respond the SCR one by one in a myopic manner and lack the adaptability to demand/network dynamics. In particular, the Lagrangian heuristics requires environment-dependent parameter tuning. In this paper, we propose a DRL framework to achieve the CSPTP-based SC that can achieve effective resource allocation in response to the demand trend.

B. Machine Learning for Networking

ML techniques have been applied to various domains in networking and expected to realize automated network optimization even under uncertain environments [7]. Specifically, GNNs have been one of the promising approaches to explore the hidden representation of network traffic and topologies [15], [16], [22]. GNNs for networking can find complex relationship among network traffic features, traffic steering, and topologies to estimate performance metrics [15], [16], [22]. There are several studies applying ML and RL techniques to SC [14]–[17]. Pei et al. proposed deep learning based two-phase VNF selection and chaining algorithms for networks with software defined networking (SDN) and NFV support [17]. Chen et al. proposed quality of service (QoS) and quality of experience (QoE) aware SC based on RL in SDN and NFV enabled slices [14]. Rafiq et al. proposed GNN-based SC in SDN to realize the delay-aware traffic steering [15]. Heo et al. proposed GNN-based SC using the encoder-decoder model to minimize the total delay of service path and extended this model by applying RL algorithms [16]. Almasan et al. applied message passing neural networks (MPNNs) to the DRL framework to solve the minimum cost flow problem in optical networks and showed the generalization capabilities of MPNN based GNN over different topologies [22].

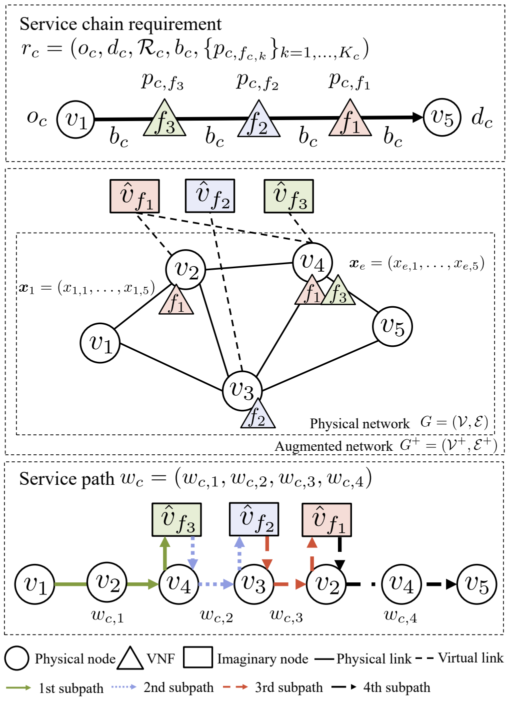
Service chain requirement
$r_c = (o_c, d_c, \mathcal{R}_c, b_c, \{p_{c,f_{c,k}}\}_{k=1,\dots,K_c})$
p_{c,f_3}
p_{c,f_2}
p_{c,f_1}
o_c
v_1
f_3
f_2
f_1
v_5
d_c
b_c
$\hat{v}_{f_1}$
$\hat{v}_{f_2}$
$\hat{v}_{f_3}$
$\boldsymbol{x}_1 = (x_{1,1}, \dots, x_{1,5})$
$\boldsymbol{x}_e = (x_{e,1}, \dots, x_{e,5})$
v_2
v_4
v_3
Physical network $G = (\mathcal{V}, \mathcal{E})$
Augmented network $G^+ = (\mathcal{V}^+, \mathcal{E}^+)$
Service path $w_c = (w_{c,1}, w_{c,2}, w_{c,3}, w_{c,4})$
$w_{c,1}$
$w_{c,2}$
$w_{c,3}$
$w_{c,4}$
Physical node
VNF
Imaginary node
Physical link
Virtual link
1st subpath
2nd subpath
3rd subpath
4th subpath

Fig. 1: Overview of CSPTP-based SC.

Inspired by the approach in [22], we propose the DRL with GNN framework to solve SC, which is more difficult than the conventional routing problem considered in [22]. The proposed framework aims at realizing (1) adaptive resource allocation based on the learning of demand trend and (2) generalization capabilities against topology changes due to link failures, thanks to both the DRL and GNN.

III. Preliminaries

In this section, we briefly introduce the preliminaries of the proposed framework from the viewpoint of CSPTP-based SC, DRL, and GNN, respectively.

A. CSPTP-based Service Chaining

In this paper, we consider the system model used in [5]. Fig. 1 illustrates the overview of the CSPTP-based SC.

1) Service Chain Request: We assume the online SC where the NFV orchestrator immediately serves a new SCR c arriving at the NFV network. As shown in the top layer of Fig. 1, each SCR c has the service chain requirements

$r_c = (o_c, d_c, \mathcal{R}_c, b_c, \{p_{c,f_{c,m}}\}_{k=1,\ldots,M_c})$ where o_c and d_c represent an origin node and a destination node, respectively. Let $\mathcal{R}_c$ be a sequence $(f_{c,1}, \ldots, f_{c,M_c})$ of M_c *functions* in required order. b_c and $p_{c,f_{c,m}}$ denote the required bit rate and the processing resources required for executing the kth function $f_{c,m}$ at a physical node, respectively.

2) Physical Network: A *physical network* is defined as a directed graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{V}$ (resp. $\mathcal{E}$) denotes a set of *physical nodes (resp. links)*. Let $\mathcal{X}$ denote a set of *features* of physical links, i.e., $\mathcal{X} = \{\boldsymbol{x}_e\}_{\forall e \in \mathcal{E}}$, and $\boldsymbol{x}_e$ represents a vector of $D > 0$ features of physical link e, i.e., $\boldsymbol{x}_e = (x_{e,1}, \ldots, x_{e,D})$. The NFV network supports a set of F distinct functions, $\mathcal{F} = \{f_1, \ldots, f_F\}$, and consists of two types of the physical nodes: *VNF-enabled nodes* $\mathcal{V}_{\mathrm{VNF}}$ and conventional ones (i.e., routers and switches). Each VNF-enabled node $i \in \mathcal{V}_{\mathrm{VNF}}$ can have one or more functions $\mathcal{F}_i \subseteq \mathcal{F}$ where each function $f \in \mathcal{F}$ is possessed by part of VNF-enabled nodes, i.e., $\mathcal{V}_f \subseteq \mathcal{V}_{\mathrm{VNF}}$.

3) Augmented Network: To cope with CSPTP-based SC, the *augmented network* $G^+ = (\mathcal{V}^+, \mathcal{E}^+, \mathcal{X}^+)$ is constructed by extending the physical network G with *imaginary nodes* $\hat{\mathcal{V}}$ and *virtual links* $\hat{\mathcal{E}}^{\mathrm{in}} \cup \hat{\mathcal{E}}^{\mathrm{out}}$ where $\mathcal{V}^+ = \mathcal{V} \cup \hat{\mathcal{V}}$, $\mathcal{E}^+ = \mathcal{E} \cup \hat{\mathcal{E}}^{\mathrm{in}} \cup \hat{\mathcal{E}}^{\mathrm{out}}$. $\mathcal{X}^+$ denotes a set of features on physical and virtual links, i.e., $\mathcal{X}^+ = \{\boldsymbol{x}_e\}_{e \in \mathcal{E}^+}$. An imaginary node $\hat{v}_{c,f_{c,m}} \in \hat{\mathcal{V}}$ is responsible for function $f_{c,m}$ and is connected to VNF-enabled node(s) having $f_{c,m}$. Links incoming to (resp. outgoing from) imaginary node $\hat{v}_f$, called virtual links, are defined as $\hat{\mathcal{E}}^{\mathrm{in}}$ (resp. $\hat{\mathcal{E}}^{\mathrm{out}}$). Note that $\hat{\mathcal{E}}^{\mathrm{in}} = \{(v, \hat{v}_f) \mid v \in \mathcal{V}_f, \hat{v}_f \in \hat{\mathcal{V}}, f \in \mathcal{F}_v\}$ (resp. $\hat{\mathcal{E}}^{\mathrm{out}} = \{(\hat{v}_f, v) \mid v \in \mathcal{V}_f, \hat{v}_f \in \hat{\mathcal{V}}, f \in \mathcal{F}_v\}$). The virtual link $(\hat{v}_f, v) \in \hat{\mathcal{E}}^{\mathrm{out}}$ indicates that the VNF-enabled node $v \in \mathcal{V}_{f_{c,m}}$ has the function $f_{c,m}$. Each virtual link $(\hat{v}_f, v)$ (resp. physical link (i,j)) has the residual processing capacity $P_{\hat{v}_f, v}$ of physical node v for executing function f (resp. residual link capacity $B_{i,j}$) at the arrival of SCR c. The middle layer of Fig. 1 illustrates an example of the augmented network.

4) Service Path: With the help of the augmented network, the *service path* w_c with origin o_c, destination d_c, and required functions $\mathcal{R}_c$ can be decomposed into a sequence of $M_c + 1$ subpaths, i.e., $w_c = (w_{c,1}, \ldots, w_{c,M_c+1})$. The pair $(o_{c,m}, d_{c,m})$ of origin and destination nodes of the mth subpath $w_{c,m}$ is given by $(o_c, \hat{v}_{f_{c,1}})$ for $m = 1$, $(\hat{v}_{f_{c,m-1}}, \hat{v}_{f_{c,m}})$ for $m = 2, \ldots, M_c$, and $(\hat{v}_{f_{c,M_c}}, d_c)$ for $m = M_c + 1$. Note that selecting the virtual link in the service path determines the physical node on which the corresponding function is conducted. Each subpath does not contain any loop while the entire service path may have loop(s). As a result, a certain link may be used more than once in the service path. Let $\mathcal{E}^+_{w_c} \subseteq \mathcal{E}^+$ be a set of links included in the service path w_c i.e., $\mathcal{E}^+_{w_c} = \mathcal{E}_{w_c} \cup \hat{\mathcal{E}}^{\mathrm{in}}_{w_c} \cup \hat{\mathcal{E}}^{\mathrm{out}}_{w_c}$, where $\mathcal{E}_{w_c}$ is a set of physical links included in w_c and $\hat{\mathcal{E}}^{\mathrm{in}}_{w_c}$ (resp. $\hat{\mathcal{E}}^{\mathrm{out}}_{w_c}$) is a set of incoming (resp. outgoing) virtual links included in w_c. The bottom layer of Fig. 1 shows an example of the service path.

B. Deep Reinforcement Learning

RL aims at learning a long-term strategy (i.e., policy) to solve an optimization problem under a certain environment, which is defined by a set $\mathcal{S}$ of states [23]. Q-learning is an RL algorithm making the agent learn an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$. It maintains a table with the size of $|\mathcal{S}| \times |\mathcal{A}|$, where (s,a)th element is initialized as zero or a random value and updated with a q-value for the combination of state s and action a. Given a state $s \in \mathcal{S}$, an agent takes an action a according to the current policy π, which yields the reward r and the transition to a next state s' with a probability $\Pr(s', r|s, a)$. Here, the q-value $Q(s,a)$ is updated according to the rule based on the Bellman equation [24]:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')),$$

where γ $(0 \le \gamma \le 1)$ denotes a discount rate indicating the importance of the future reward and α $(0 < \alpha \le 1)$ is a learning rate.

Deep Q-network (DQN) can solve one of the potential drawbacks of Q-learning, i.e., the scalability against the size of state and action space, by approximating the q-value function using a deep neural network (DNN) and learning it through observed states and actions [25]. The state transition information $\{s, a, r, s'\}$ is stored in a memory called an experience replay buffer, which is used for training DNNs.

C. Graph Neural Network

GNNs are deep learning based methods to operate the graph domain [8]–[13]. Given the graph structure and node feature information as inputs, a GNN outputs the node, edge, or graph-level representation by graph convolution operation in the spectral or spatial domain. Message passing neural networks (MPNNs) are a well-known type of GNNs, which is a unified framework for the graph convolution operations (i.e., aggregation, update, and readout) in the spatial domain [13]. In MPNNs, each node in the graph initially has its own features. Then, it collects the features from the neighbors and aggregates them into a message. It further combines the message with its own features and updates its features as the hidden embedding. These operations are repeated along with multiple layers of MPNNs. The output of the final layer defines the node-level representation, i.e., embedding of each node, and it can generate a graph-level representation through the readout operation.

Graph convolutional networks (GCNs) are one of the most popular baseline GNN models and employ the first-order neighboring aggregation and the self-loop update [9]. GCN with the renormalization trick can be defined as the following layer-wise aggregation and update operations:

$$\mathbf{X}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^{(l)}\mathbf{\Theta}^{(l)}\right). \quad (1)$$

Here, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} \in \mathbb{R}^{N \times N}$ is the adjacent matrix with self loops where $\mathbf{A}$ is the original adjacent matrix of the undirected graph G with N nodes and $\mathbf{I}$ is the identity matrix. $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$ is the degree matrix of $\tilde{\mathbf{A}}$. $\mathbf{X}^{(l)} \in \mathbb{R}^{N \times D}$ represents a feature

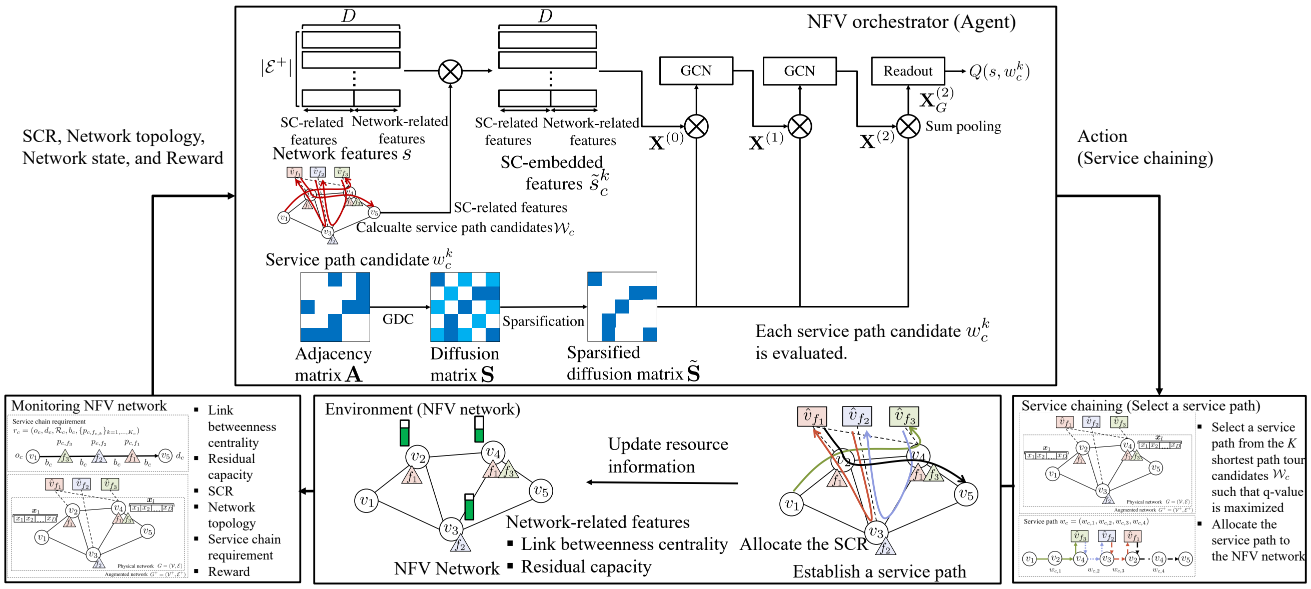
NFV orchestrator (Agent)
SCR, Network topology, Network state, and Reward
Action (Service chaining)
SC-related features
Network-related features
Network features s
SC-embedded features $\tilde{s}_c^k$
GCN
Readout
$Q(s, w_c^k)$
Sum pooling
Calcualte service path candidates $\mathcal{W}_c$
Service path candidate w_c^k
GDC
Sparsification
Adjacency matrix $\mathbf{A}$
Diffusion matrix $\mathbf{S}$
Sparsified diffusion matrix $\tilde{\mathbf{S}}$
Each service path candidate w_c^k is evaluated.
Monitoring NFV network
Link betweenness centrality
Residual capacity
SCR
Network topology
Service chain requirement
Reward
Environment (NFV network)
Update resource information
Network-related features
Link betweenness centrality
Residual capacity
NFV Network
Allocate the SCR
Establish a service path
Service chaining (Select a service path)
Select a service path from the K shortest path tour candidates $\mathcal{W}_c$ such that q-value is maximized
Allocate the service path to the NFV network

Fig. 2: DRL framework with GNN for the CSPTP-based SC.

matrix at the lth GCN layer, where $\mathbf{X}^{(0)} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\mathsf{T}$ indicates an original feature matrix. $\mathbf{\Theta}^{(l)}$ indicates a learnable weight matrix for the lth layer, and $\sigma(\cdot)$ is a general element-wise nonlinear activation function, e.g., rectified linear unit function (ReLU) [26].

To cope with more information derived from a graph, graph diffusion convolution (GDC) was proposed, which generalizes the graph convolution by removing the restriction of using only the direct neighbors [10]. GDC replaces the adjacency matrix $\mathbf{A}$ with the following diffusion matrix $\mathbf{S}$:

$$\mathbf{S} = \sum_{n=0}^{\infty} \eta_n \mathbf{T}^n, \tag{2}$$

where $\mathbf{T} \in \mathbb{R}^{N \times N}$ is a transition matrix whose (i, j)th element means the transition probability from node i to node j. $\mathbf{T}^n$ gives the n-step transition probabilities and $\eta_n > 0$ is the weighting coefficient for $\mathbf{T}^n$. If the diffusion matrix $\mathbf{S}$ is dense, the sparsified diffusion matrix $\tilde{\mathbf{S}}$ was used to obtain the spatial locality by removing links with small values of $\mathbf{S}$ in a simple manner, e.g., top-k-based sparsification or threshold-based sparsification.

IV. Proposed Scheme

A. Overview

In this paper, inspired by the DRL with GNN architecture for network routing problems [22], we propose the DRL based framework with a GNN for the CSPTP-based SC. The SC problem as CSPTP is more challenging than the conventional routing problem as the shortest path problem. The agent, i.e., the NFV orchestrator, aims at accepting as many SCRs as possible, which will be achieved by the minimization of the overall physical and virtual link utilization in the NFV network. The proposed DRL agent is realized by the double-DQN algorithm [27], where the q-value function is modeled by a GNN. (We will give the DRL agent design in Section IV-C and the GNN architecture in Section IV-C3.)

Fig. 2 illustrates the overview of the DRL based framework with a GNN for the CSPTP-based SC. At each time step, the agent (i.e., NFV orchestrator) monitors the environment, i.e., NFV network (the bottom center in Fig. 2) and obtains a network state and an SCR as inputs from the environment (the bottom left in Fig. 2). More specifically, the network state is represented by the features of each link in the augmented network, which will be described in Section IV-B. Next, the agents finds the service path candidates $\mathcal{W}_c$, i.e., an action set $\mathcal{A}$ (the top layer in Fig. 2). We will show the details of the action set in Section IV-C2. For each service path candidate (i.e., action), it generates an SC-embedded state from the current state $s \in \mathcal{S}$ by concatenating the network-related features and SC-related ones and then computes the q-value of the anction-embedded state, with the help of GCN and GDC (the top layer in Fig. 2). The details of the agent operation will be described in Section IV-C1. Note that the existing work [22] adopts a different GNN approach, i.e., MPNN. Finally it performs an appropriate action $a \in \mathcal{A}$, i.e., selecting an appropriate service path, according to the policy π (the bottom right in Fig. 2), and then obtains the reward r, the next SCR c', and the next state $s' \in \mathcal{S}$ from the environment (the bottom center in Fig. 2).

B. Environment

In this paper, we consider the environment as the augmented network with link features, as shown in the middle layer of Fig. 1. More specifically, the network state s is defined as the feature matrix $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|\mathcal{E}^+|}]^\mathsf{T}$ where $\boldsymbol{x}_e$ is a $D = 5$ dimensional feature vector of physical/virtual link $e \in \mathcal{E}^+$, i.e., $\boldsymbol{x}_e = (x_{e,1}, \ldots, x_{e,5})$. The feature vector $\boldsymbol{x}_e$ is composed of the SC-related features, i.e., $x_{e,1}$, $x_{e,2}$, and $x_{e,3}$, and the

Algorithm 1 Agent operation.

```
Require: Agent $agent$, environment $env$, the number $K$ of actions, training iteration id $\tau$, training interval $M$.
 1: $s, c, r_c \leftarrow$ INIT($env$)
 2: $R \leftarrow 0$
 3: do
 4:     $\mathcal{W}_c \leftarrow$ K-DFTS($K, r_c$)
 5:     $\mathcal{Q} \leftarrow \emptyset$
 6:     for $w_c^k \in \mathcal{W}_c$ do
 7:         $\tilde{s}_c^k \leftarrow$ ALLOCATE-SCR($env, c, r_c, w_c^k$)
 8:         $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(w_c^k,$ GET-Q-VAL-FROM-GNN$(\tilde{s}_c^k, w_c^k)\}$
 9:     $w_c^{k'} \leftarrow$ EPSILON-GREEDY($\mathcal{Q}, \varepsilon$)
10:     $r, allocated, s', c', r_c' \leftarrow$ STEP($env, s, w_c^{k'}$)
11:     $R \leftarrow R + r$
12:     MEMORIZE($agent, \{s, a, r, s'\}$)
13:     if $\tau \mod I = 0$ then
14:         TRAIN-GNN-USING-REPLAY-BUFFER($agent$)
15:     $s, c, r_c \leftarrow s', c', r_c'$
16: while $allocated =$ true
```

network-related features, i.e., $x_{e,4}$ and $x_{e,5}$. The SC-related features are calculated per service path candidate $w_c \in \mathcal{W}_c$ to evaluate its deployment cost in terms of resource usage. $x_{e,1}$ is the number of times that the link e is used in w_c. Please note that $x_{e,3}$ can be more than one if the service path candidate w_c has loop(s), which makes the problem more difficult than the conventional routing problem [22]. $x_{e,2}$ is SCR c's bandwidth requirement b_c (resp. processing capacity requirement $p_{c,f_{c,m}}$) for the physical (resp. virtual) link e. $x_{e,3}$ is the link e's utilization $u_{c,e}$ required to support w_c. Focusing on the SC-related features only for the links used in w_c, we set $x_{e,1}$, $x_{e,2}$, and $x_{e,3}$ to be zero for the links unused in w_c. (Similar assumption is also used in [22].)

The network-related features are used to evaluate the overall utilization of NFV network, which will contribute to saving the network resources for future requests. For this purpose, we apply the link betweenness centrality [28] and the residual capacity of link e as the network-related features $x_{e,4}$ and $x_{e,5}$, respectively. Note that these features are also used in [22].

C. Agent Design

1) Agent Operation: The agent operates through the interactions with the environment. We assume that the agent learns the optimal policy through $T \geq 1$ *training iterations*, each of which consists of $L \geq 1$ *episodes*. Algorithm 1 presents a pseudocode describing the proposed agent behavior in one episode of the τth training iteration ($\tau = 1, \ldots, T$). At the beginning of the episode, the environment env is initialized by calling the INIT() function, which also generates a new SCR c with the service chain requirements r_c (line 1). At the same time, the cumulative reward R is set to be zero (line 2).

Algorithm 1 executes the following procedures as long as the agent succeeds in allocating a service path to a new SCR c, i.e., the corresponding binary flag *allocated* is true (lines 3–16). Since considering all possible service path candidates will result in a highly dimensional action space, the action set is limited to K service path candidates as in [22]. The agent calculates the set of K service path candidates, $\mathcal{W}_c = \{w_c^1, \ldots, w_c^k\}$, by calling the K-DFTS() function (line 4). We will describe the details of K-DFTS() function in Section IV-C2. Note that symbols $\mathcal{A}$ and $\mathcal{W}_c$ will be used interchangeably. We also initialize a set $\mathcal{Q}$ of each pair of action a and its yielding q-value $Q(s,a)$ to an empty set (line 5).

For each service path candidate $w_c^k \in \mathcal{W}_c$, the agent computes the corresponding q-value using the GNN (lines 6–8). More specifically, the agent first generates the SC-embedded state $\tilde{s}_c^k$ using the ALLOCATE-SCR() function (line 7). As described in Section IV-B, $\tilde{s}_c^k$ is represented by the feature vector $\boldsymbol{x}_e$ of each link e. With the state $\tilde{s}_c^k$ as the input, the agent then computes the corresponding q-value $Q(\tilde{s}_c^k, w_c^k)$ by calling the GET-Q-VAL-FROM-GNN() function and adds the new element $(w_c^k, Q(\tilde{s}_c^k, w_c^k))$ to $\mathcal{Q}$ (line 8). The details of the GET-Q-VAL-FROM-GNN() function will be explained in Section IV-C3.

Next, the agent selects a service path candidate $w_c^{k'}$ from $\mathcal{W}_c$ according to $\mathcal{Q}$ and the ε-greedy exploration strategy [23] by calling the EPSILON-GREEDY() function (line 9). In the STEP() function, the agent tries to apply the service path candidate $w_c^{k'}$ to the NFV network and then obtains the reward r, the binary flag *allocated*, the next state s', and the next SCR c' with r_c' from the environment (line 10). Here, we design the reward r after selecting $w_c^{k'}$ such that it should be nonnegative and becomes large in case of low utilization of network resources:

$$r = \omega_1 \exp\left(-\sum_{e \in \mathcal{E}_{w_c^{k'}}} u_{c,e}\right) + \omega_2 \exp\left(-\sum_{\hat{e} \in \hat{\mathcal{E}}^{\text{out}}_{w_c^{k'}}} u_{c,\hat{e}}\right),$$

where the first (resp. second) term is related to the usage degree of physical links (resp. virtual links) in $w_c^{k'}$, $\omega_1 > 0$ (resp. $\omega_2 > 0$) is the corresponding weighting parameter, and $\mathcal{E}_{w_c^{k'}}$ (resp. $\hat{\mathcal{E}}^{\text{out}}_{w_c^{k'}}$) is a set of physical links (resp. outgoing virtual links) in $w_c^{k'}$. Note that the cumulative reward R is defined as the sum of reward r during one episode.

The agent updates the cumulative reward R (line 11) and stores the transition (experience), i.e., $\{s, a, r, s'\}$, into the experience replay buffer (line 12). The stored transition will be used to train the GNN by executing the TRAIN-GNN-USING-REPLAY-BUFFER() function every $I \geq 1$ training iterations (lines 13–14). The GNN model is trained such that a loss function $L(\boldsymbol{\Theta})$ with the learnable weight matrix $\boldsymbol{\Theta}$ approaches to zero by using the samples $\mathcal{Z}$ randomly chosen from the experience reply buffer. $L(\boldsymbol{\Theta})$ is defined as follows:

$$L(\boldsymbol{\Theta}) = \sum_{\{s,a,r,s'\} \in \mathcal{Z}} \left(Q(s, a \mid \boldsymbol{\Theta}) - (r + \gamma \max_{a' \in \mathcal{A}} Q(s', a' \mid \boldsymbol{\Theta}))\right)^2 + \lambda E_{\text{L1}}(\boldsymbol{\Theta}),$$

where the first term is the mean squared error between the estimated q-value and observed one. The second term indicates L1 regularization penalty to prevent overfitting, where $E_{\mathrm{L1}}(\Theta)$ is the L1 regularization and $\lambda > 0$ is a weighting parameter.

2) Action Set: To obtain the action set, i.e., K service path candidates $\mathcal{W}_c$, we propose a K-DFTS algorithm by extending the DFTS algorithm [4]. K is expected to be a moderate value, e.g., 5, to hold the balance between computational complexity and flexibility of steering traffic. To derive the exclusive service path candidates as much as possible, we design the K-DFTS algorithm as follows. The agent calculates the first service path candidate under r_c by using the DFTS algorithm [4]. Since saving the network resources leads to accepting more SCRs in future, the cost of each link (i,j) is defined as $b_c/B_{i,j}$, $p_{c,f}/P_{i,j}$, and zero in case of $(i,j) \in \mathcal{E}$, $(i,j) \in \hat{\mathcal{E}}^{\mathrm{out}}$, and $(i,j) \in \hat{\mathcal{E}}^{\mathrm{in}}$, respectively. Then, it updates the NFV network by removing a virtual link with the highest utilization in the service path candidates selected so far, and calculates the next service path candidate in the same way. It continues this procedure to obtain K service path candidates.

3) GNN architecture: With the graph structure and link feature information as inputs, the GNN model outputs the q-value by the following procedures. To deal with link features and neighborhood links, we first transform the augmented network by treating links as nodes. Note that two nodes in the transformed augmented network are connected if the corresponding two links in the original augmented network are connected to the same node. Let $\mathbf{A}$ denote the adjacency matrix of the transformed augmented network. As a result, we can interpret the link features of the original augmented network as the node features of the transformed one.

Next, to extract the hidden representation in the graph domain, we apply the topological augmentation to $\mathbf{A}$ and obtain the diffusion matrix $\mathbf{S}$, which is given by Eq. (2), according to the extended version of GDC [10]. The extended version of GDC applies the weighted PageRank [29] into GDC to derive the transition matrix $\mathbf{T}$, where the weight of a link (e_i, e_j) in the transformed augmented network is defined as the minimum of the normalized residual capacities of links e_i and e_j in the original augmented network. We further calculate the sparsified diffusion matrix $\tilde{\mathbf{S}}$ by using the threshold-based sparsification. Then, a two-layer GCN is applied to the sparsified diffusion matrix $\tilde{\mathbf{S}}$ and link feature matrix $\mathbf{X}$ to derive the hidden representation $\mathbf{X}^{(l)}$ according to Eq. (1). Next, the graph-level features $\mathbf{X}_G^{(l)} \in \mathbb{R}^D$ are obtained by applying the sum-pooling to the feature matrix $\mathbf{X}^{(l)} \in \mathbb{R}^{N \times D}$ across nodes. Finally, the readout function modeled by DNNs computes the q-value from $\mathbf{X}_G^{(l)}$.

V. Numerical Results

A. Evaluation Settings

The NSFNET topology with 14 nodes and 21 links is used for the evaluation, which is available at the Internet topology zoo [31]. The original capacity of each physical link (i,j) is set to be identical, $\hat{B}_{i,j} = 1\,\mathrm{Gbps}$. As for each virtual link

TABLE I: Service chain demand and requirements (NAT: Network Address Translator, FW: Firewall, TM: Traffic Monitor, WOC: WAN Optimization Controller, IDPS: Intrusion Detection Prevention System, and VOC: Video Optimization Controller).

Service	Sequence of functions	Demand	b_c
Web service	NAT-FW-TM-WOC-IDPS	18.2%	1 Mbps
VoIP	NAT-FW-TM-FW-NAT	11.8%	4 Mbps
Video streaming	NAT-FW-TM-VOC-IDPS	69.9%	16 Mbps
Online gaming	NAT-FW-VOC-WOC-IDPS	0.1%	32 Mbps

TABLE II: Relationship between function type and processing requirements per SCR [30].

Function type	NAT	FW	TM	IDPS	VOC	WOC
$p_{c,f_c,m}$	0.00092	0.0009	0.0133	0.0107	0.0054	0.0054

$(\hat{v}_f, v)$, the original capacity $\hat{P}_{\hat{v}_f,v}$ is set to be $1/|\mathcal{F}_v|$ such that the physical node v equally divides and distributes the processing resource of one CPU to its supporting functions $\mathcal{F}_v$. Each function $f \in \mathcal{F}$ is assigned to $V_f = 3$ VNF-enabled nodes randomly selected. Table II gives the processing requirement $p_{c,f,k}$ for executing each function $f \in \mathcal{F}$.

An event-driven simulator is developed according to Algorithm 1. The SC scenario in one episode is as follows. A new SCR c with a random o–d pair occurs in the NFV network (i.e., environment) according to the demand distribution in Table I. Next, the NFV orchestrator (i.e., agent) allocates the resources to the SCR c according to the ε-greedy exploration strategy. To examine how many SCRs the NFV orchestrator can simultaneously support, we assume that each established service path holds until the end of simulation. If the NFV orchestrator fails to allocate resources to the SCR c, the simulation is terminated. The set of accepted SCRs is defined as $\mathcal{C}_{\mathrm{accept}}$. These procedures are repeated TL times where T and L are the number of iterations and that of episodes in one iteration, respectively. $(T, L) = (100, 50)$ is used for both the training and testing phases.

The DRL+GNN agent is implemented by using Pytorch and Pytorch geometric libraries [32], [33]. In the training phase, we use the Adam optimizer [34] with the initial learning rate of 10^{-4} and the discount rate $\gamma = 0.95$. We train the model every $I = 2$ training iterations by using 5 batches with 32 samples randomly chosen from the experience replay buffer. The experience replay buffer has the size of 5000 samples with the first-in first-out (FIFO) updating policy. As for the ε-greedy exploration strategy, in the training phase, ε initially takes one, keeps the value during the first ten iterations, then exponentially decays with the base of 0.99 every two episodes, and approaches asymptotically to 0.01. On the other hand, in the testing phase, ε is fixed to be zero in order to apply the learned GNN model and lines 12–14 in Algorithm 1 are skipped.

As for the evaluation metric, we use the average number of SCRs that are successfully allocated per episode, i.e., $C_{\mathrm{accept}} = |\mathcal{C}_{\mathrm{accept}}|$. Note that we have confirmed that the

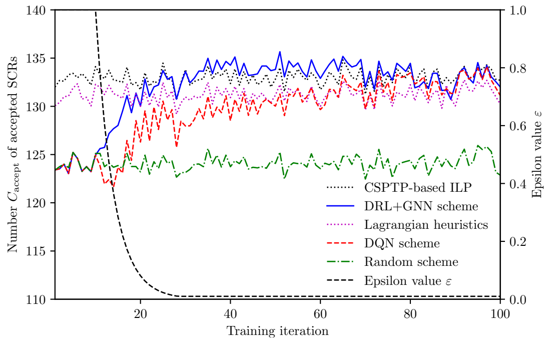

Fig. 3: Evolution of number C_{accept} of accepted SCRs in training phase.

TABLE III: Service demand distribution of each demand trend scenario for the testing phase.

Demand trend	Web service	VoIP	Video streaming	Online gaming	Cosine similarity θ
Base	18.2%	11.8%	69.9%	0.1%	1
Different 1	24.2%	17.8%	51.9%	6.1%	0.96
Different 2	30.2%	23.8%	33.9%	12.1%	0.82
Different 3	36.2%	29.8%	15.9%	18.1%	0.54

cumulative reward R shows the similar tendency to C_{accept} in the following results. In terms of the efficient resource allocation, we also use the total amount of incoming traffic among accepted SCRs, i.e., $B_{\text{accept}} = \sum_{c \in \mathcal{C}_{\text{accept}}} b_c$. From the viewpoint of the computational complexity, we adopt the *computation time*, which is the average time required to calculate a service path in the testing phase. We compare the DRL+GNN scheme with the following three schemes: (a) a random scheme where the agent randomly selects an action regardless of the state, (b) a DQN scheme where the agent computes the q-value based on the DNN only with graph pooling, (c) the Lagrangian heuristics for CSPTP-based SC [6], and (d) the online CSPTP-based ILP [5] where the objective function is modified to minimize the overall physical and virtual link utilization. To cope with the topological change, the DQN scheme first obtains the graph-level features $\mathbf{X}_G \in \mathbb{R}^D$ by applying the sum-pooling to the feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ across nodes and then computes the q-value by applying the two-layer neural networks to $\mathbf{X}_G$. To solve the online CSPTP-based ILP, we use the existing solver CPLEX 12.8 [35] with the parallel optimization parameter (i.e., the number of threads) of 32. In the calculation, we use the server with Intel Xeon Gold 6226R 16 core, 196 GB memory, and an NVIDIA GeForce RTX 3090 GPU.

B. Fundamental Characteristics

We train the DRL+GNN scheme and the DQN scheme under the NSFNET topology. As for the Lagrangian heuristics, we use it with appropriate parameter tuning. Fig. 3 illustrates the evolution of the number C_{accept} of accepted SCRs averaged over $L = 50$ episodes per iteration during the training phase. Since all the schemes except both CSPTP-based ILP and Lagrangian heuristics randomly adopt a service path candidate per SCR during the first 10 iterations, due to the ε-greedy exploration strategy with $\varepsilon = 1$, they show almost the same behavior. On the other hand, they show different behavior after the 11th iteration. Since the random scheme continues the random selection, it cannot improve C_{accept}. On the contrary, the DRL+GNN scheme increases C_{accept} with iteration, which is confirmed as the learning effect with the decay of ε, and becomes competitive with the online CSPTP-based ILP. Someone might wonder why the DQN+GNN scheme sometimes overcomes the online CSPTP-based ILP. This is because the online CSPTP-based ILP gives the optimal solution per SCR but does not guarantee the optimality in the long-term perspective. The DQN scheme also shows the learning effect but has the slower convergence rate than the DRL+GNN scheme.

We also observe that the performance improvement between the DRL+GNN scheme and the random scheme is not so large, i.e., less than 8.8%. This comes from the fact that both schemes share the same K-DFTS algorithm with a moderate value of K (i.e., $K = 5$) for calculating the action set, as mentioned in Section IV-C2. Larger K value would improve the performance of the DRL+GNN scheme at expense of the computational complexity. In future work, we will investigate how much K affects the performance and computational complexity.

C. Adaptability to Different Service Demand Trend

Next, we evaluate the trained models in terms of the adaptability to demand trend through the evaluations under the following scenarios. We first prepare the *base* (demand trend) scenario, which is the same environment in the training phase except for the random seed value. Then, we prepare the three different demand trend scenarios (i.e., *different 1*, *different 2*, and *different 3*) in descending order of its cosine similarity θ to the base service demand trend. More specifically, we make these scenarios by modifying the base scenario as follows: We reduce a certain amount of the service demand of video streaming and equally dividing it among the others. Table III shows the service demand distribution in each scenario with its cosine similarity θ to the base scenario.

Fig. 4 (resp. Fig. 5) depicts the box-and-whisker plot of C_{accept} (resp. B_{accept}) for each scheme in the base and different demand trend scenarios. The box-and-whisker plot consists of three parts, i.e, box, two whisker lines, and outliers. The box has the height ranging in $[Q_1, Q_3]$ where Q_1 (resp. Q_3) is the first (resp. third) quartile and includes a horizontal line as the median. The upper (resp. lower) whisker line is connected between Q_3 (resp. Q_1) and the upper (resp. lower) bound, over (resp. under) which the data samples are regarded as outliers, denoted by points. The length of whisker line is given by $1.5(Q_3 - Q_1)$.

We first focus on C_{accept} of each scheme in the base demand trend scenario. As we expect, the performance of each scheme has almost the same as that achieved at the end of the training phase in Fig. 3. As a result, the DRL+GNN scheme exhibits

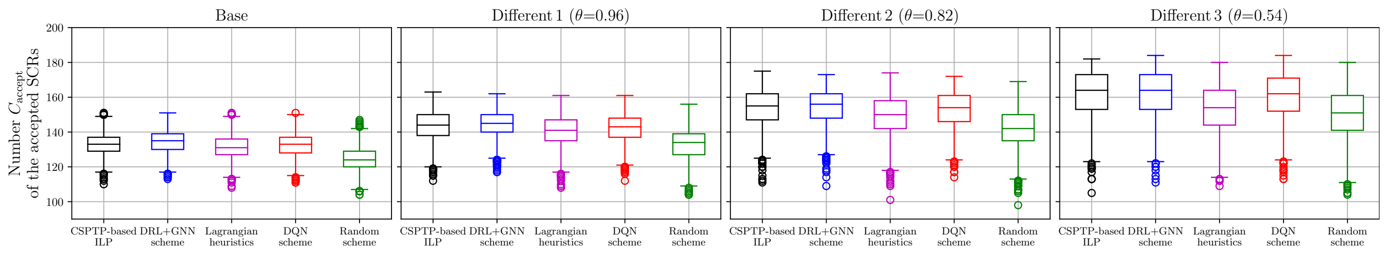

Fig. 4: The number C_{accept} of the accepted SCRs for five schemes in the testing phase.

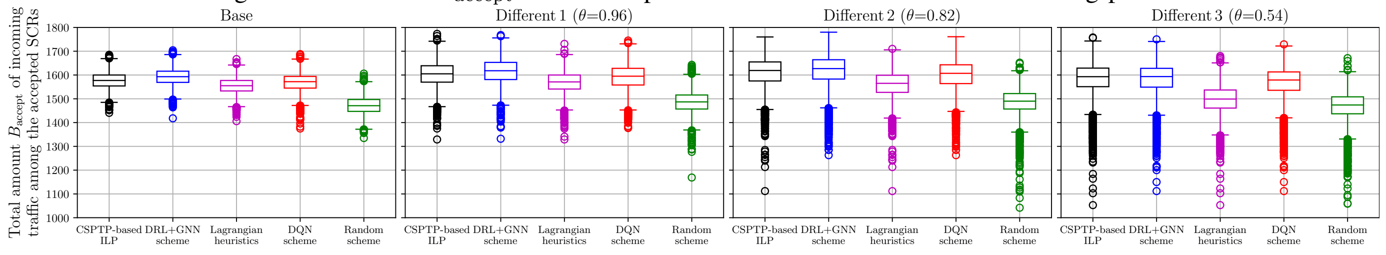

Fig. 5: Total amount B_{accept} of incomming traffic among accepted SCRs for five schemes in the testing phase.

TABLE IV: Copmarison of computation time.

Scheme	Computation time per SCR [ms]
CSPTP-based ILP	142
DRL+GNN scheme	81
Lagrangian heuristics	61
DQN scheme	77
Random scheme	65

the slightly higher performance compared with the online CSPTP-based ILP. B_{accept} in Fig. 5 shows the same tendency as C_{accept} in Fig. 4.

Next, we compare the results among the four scenarios. At first, someone might wonder why C_{accept} of each scheme increases with decrease of the cosine similarity θ (from the left to right in Fig. 4). This is because the different bandwidth requirement b_c among services as shown in Table I. More specifically, in the preparation of the three different scenarios, we reduce β% service demand of video streaming and add $\beta/3$% service demand to each remaining service, which reduces the bandwidth requirement in proportion to $16\beta-(1+4+32)\beta/3 \simeq 3.67\beta$. Since the amount of network resource is identical among all scenarios, such increasing trend does not arise in terms of B_{accept}, as shown in Fig. 5.

We observe that both the DRL+GNN scheme and DQN scheme have competitive C_{accepts} and B_{accepts} with the online CSPTP-based ILP in all scenario, thanks to their generalization capabilities. Note that the DQN scheme requires more training iterations as shown in Fig. 3. On the other hand, the Lagrangian heuristics gradually degrades the performance with decrease of θ and consequently exhibits almost the same performance as the random scheme. This indicates that the Lagrangian heuristics fine-tuned for the base scenario cannot adapt to the demand change.

Finally, we evaluate the computational complexity under the base scenario. Table IV presents the computation time of the five schemes. We observe that the DRL+GNN scheme

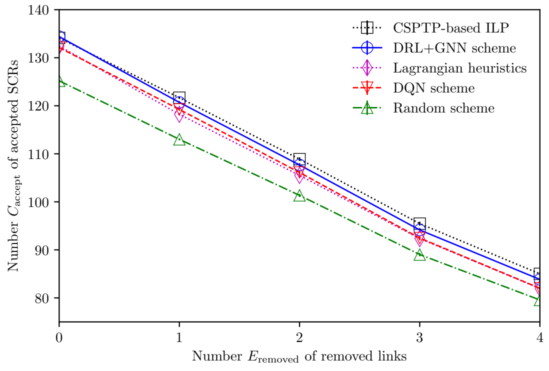

Fig. 6: Impact of the number E_{removed} of removed links on the number C_{accept} of accepted SCRs.

can reduce the computation time by 57% compared with the CSPTP-based ILP while suppressing its increase in the range of 5–32% compared with other schemes.

D. Adaptability to Topology Change with Link Failures

In actual systems, some of the links may be temporarily down, due to equipment failures, which changes the network topology. In this section, we evaluate how much the DRL+GNN scheme learned through the original NSFNET topology can work well even under the NSFNET topology with link failure(s). In the testing phase, we prepare link failure scenarios by changing the number E_{removed} of links removed from the NSFNET topology from 1 to 4. More specifically, for each E_{removed}, we randomly remove E_{removed} link(s) from the original NSFNET topology at the beginning of an episode.

Fig. 6 depicts the relationship between E_{removed} and C_{accept} for the five schemes. In this figure, we show the average with 95% confidence interval. We observe that all the schemes decrease C_{accept} with E_{removed}. Comparing the results of the DRL+GNN scheme, the DQN scheme, Lagrangian heuristics, and random scheme with those of the online CSPTP-based ILP, we confirm that the performance degradation becomes 0.8%–1.4%, 1.5%–3.5%, 1.3%–3.6%, and 6.4%–7.1%, respectively. The smaller performance degradation of the DRL+GNN scheme comes from the capabilities of the GNN and the similarity between the original topology and modified topology. The DQN scheme (resp. Lagrangian heuristics) decreases its performance due to the lack of graph-feature learning (resp. insufficient parameter tuning).

VI. Conclusion

In this paper, we have proposed the deep reinforcement learning (DRL) framework with the graph neural network (GNN) for the capacitated shortest path tour problem (CSPTP)-based service chaining (SC). The proposed framework adopts the GNN architecture for computing the q-values, which consists of the graph convolutional network and graph diffusion convolution. Through the numerical results, we have shown that the proposed framework is competitive with the online CSPTP-based ILP and achieves resource allocation adaptive to the demand trend and the topology changes due to link failures, thanks to both the DRL and GNN. In future work, we plan to examine to what extent the proposed framework can realize the generalization capabilities through evaluations over other real-world network topologies.

References

[1] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent Advances of Resource Allocation in Network Function Virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295–314, Feb. 2021.

[2] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A Survey on Service Function Chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, Nov. 2016.

[3] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A Comprehensive Survey of Network Function Virtualization," *Computer Networks*, vol. 133, pp. 212–262, Mar. 2018.

[4] S. Bhat and G. N. Rouskas, "Service-Concatenation Routing with Applications to Network Functions Virtualization," in *Proc. of ICCCN*, Jul. 2017, pp. 1–9.

[5] M. Sasabe and T. Hara, "Capacitated Shortest Path Tour Problem Based Integer Linear Programming for Service Chaining and Function Placement in NFV Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 104–117, Mar. 2021.

[6] T. Hara and M. Sasabe, "Lagrangian Heuristics for Capacitated Shortest Path Tour Problem Based Online Service Chaining," in *Proc. of the IEEE/IFIP NOMS*, Apr. 2022, pp. 1–8.

[7] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 16, pp. 1–99, Jun. 2018.

[8] W. L. Hamilton, *Graph Representation Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool, Sep. 2020, vol. 14.

[9] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv preprint arXiv:1609.02907*, pp. 1–14, Feb. 2017.

[10] J. Klicpera, S. Weiß enberger, and S. Günnemann, "Diffusion Improves Graph Learning," in *Proc. of Advances in Neural Information Processing Systems*, vol. 32, Dec. 2019, pp. 13 333–13 345.

[11] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *arXiv preprint arXiv:1812.04202*, pp. 1–24, Mar. 2020.

[12] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *AI Open*, vol. 1, pp. 57–81, Jan. 2020.

[13] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," *arXiv preprint arXiv:1704.01212*, pp. 1–14, Jun. 2017.

[14] X. Chen, Z. Li, Y. Zhang, R. Long, H. Yu, X. Du, and M. Guizani, "Reinforcement Learning–based QoS/QoE-aware Service Function Chaining in Software-Driven 5G Slices," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, pp. 1–18, Jul. 2018.

[15] A. Rafiq, T. A. Khan, M. Afaq, and W.-C. Song, "Service Function Chaining and Traffic Steering in SDN using Graph Neural Network," in *Proc. of International Conference on ICTC*, Oct. 2020, pp. 500–505.

[16] D. Heo, D. Lee, H.-G. Kim, S. Park, and H. Choi, "Reinforcement Learning of Graph Neural Networks for Service Function Chaining," *arXiv preprint arXiv:2011.08406*, pp. 1–5, Nov. 2020.

[17] J. Pei, P. Hong, K. Xue, D. Li, D. S. L. Wei, and F. Wu, "Two-Phase Virtual Network Function Selection and Chaining Algorithm Based on Deep Learning in SDN/NFV-Enabled Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1102–1117, Jun. 2020.

[18] P. Festa, "The Shortest Path Tour Problem: Problem Definition, Modeling, and Optimization," in *Proc. of INOC*, Apr. 2009, pp. 1–7.

[19] N. Huin, B. Jaumard, and F. Giroire, "Optimal Network Service Chain Provisioning," *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1320–1333, Jun. 2018.

[20] T. Nguyen, A. Girard, C. Rosenberg, and S. Fdida, "Routing via Functions in Virtual Networks: The Curse of Choices," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1192–1205, Jun. 2019.

[21] L. Gao and G. N. Rouskas, "Congestion Minimization for Service Chain Routing Problems With Path Length Considerations," *IEEE/ACM Transactions on Networking*, vol. 28, no. 6, pp. 2643–2656, Dec. 2020.

[22] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep Reinforcement Learning Meets Graph Neural Networks: Exploring a Routing Optimization Use Case," *arXiv preprint arXiv:1910.07421*, pp. 1–11, Feb. 2020.

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, Nov. 2018.

[24] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34–37, Jul. 1966.

[25] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[26] M. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. Hinton, "On Rectified Linear Units for Speech Processing," in *Proc. of IEEE ICASSP*, May 2013, pp. 3517–3521.

[27] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proc. of the AAAI Conference on Artificial Intelligence*, Feb. 2016, p. 2094–2100.

[28] U. Brandes, "On Variants of Shortest-Path Betweenness Centrality and Their Generic Computation," *Social Networks*, vol. 30, no. 2, pp. 136–145, May 2008.

[29] W. Xing and A. Ghorbani, "Weighted PageRank Algorithm," in *Proc. of Second Annual Conference on Communication Networks and Services Research, 2004.*, May 2004, pp. 305–314.

[30] M. Savi, M. Tornatore, and G. Verticale, "Impact of Processing-Resource Sharing on the Placement of Chained Virtual Network Functions," *IEEE Transactions on Cloud Computing*, pp. 1–14, May 2019.

[31] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. of Advances in Neural Information Processing Systems*, Dec. 2019, pp. 8024–8035.

[33] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," in *Proc. of ICLR Workshop on Representation Learning on Graphs and Manifolds*, May 2019, pp. 1–9.

[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980 [cs]*, Jan. 2017.

[35] ILOG, "IBM ILOG CPLEX Optimizer," https://www.ibm.com/products/ilog-cplex-optimization-studio, 2022, Accessed 14 Jun. 2022.