# Graph Neural Network-based Virtual Network Function Management

Hee-Gon Kim*, Suhyun Park*, Stanislav Lange†, Doyeong Lee*, Dongnyeong Heo‡, Heeyoul Choi‡, Jae-Hyoung Yoo*, James Won-Ki Hong*

*Computer Science and Engineering, Pohang University of Science and Technology, Pohang, South Korea
{sinjint, sh.park11, dylee90, jwkhong, jhyoo78}@postech.ac.kr

†Information Security and Communication Technology, Norwegian University of Science and Technology, Trondheim, Norway
stanislav.lange@ntnu.no

‡Handong Global University, Pohang, South Korea
{21931011, hchoi}@handong.edu

*Abstract*—**Software-Defined Networking (SDN) and Network Function Virtualization (NFV) help reduce OPEX and CAPEX as well as increase network flexibility and agility. But at the same time, operators have to cope with the increased complexity of managing virtual networks and machines, which are more dynamic and heterogeneous than before. Since this complexity is paired with strict time requirements for making management decisions, traditional mechanisms that rely on, e.g., Integer Linear Programming (ILP) models are no longer feasible. Machine learning has emerged as a possible solution to address network management problems to get near-optimal solutions in a short time. In this paper, we propose a Graph Neural Network (GNN) based algorithm to manage VNFs. The proposed model solves the complex VNF management problem in a short time and gets near-optimal solutions.**

*Index Terms*—**Virtual Network Function, Machine Learning, Graph Neural Network.**

## I. Introduction

Software-Defined Networking (SDN) and Network Function Virtualization (NFV) enable more efficient network management by allowing administrators to manage the network centrally and dynamically. The SDN controller and NFV manager provide global views of the network and NFV environment, and operators can use those to manage the network and service orchestration. Also, they can prevent the over-provisioning of resources and provide high availability by scaling and optimizing Virtual Network Functions (VNFs). However, although SDN/NFV enable efficient network management, they do not provide the optimal management solutions we need.

Integer Linear Programming (ILP) is one of the optimization methods for network management. ILP aims to minimize a linear cost function and uses a set of linear equality and inequality constraints to get an optimal solution. However, finding the optimal solution based on ILP takes a relatively long time, making it unsuitable for real-time network management.

Recently, Machine Learning (ML) is emerging as a new paradigm to solve various networking problems and to automate network management. **ML provides models that automatically learn and improve from experiences without being explicitly programmed** [1]. ML takes some time to learn, but takes little time after learning. Also, ML is **more effective in learning wide and dynamically changing data than statistical methods** [2]. However, while ML has these advantages, it is not easy to apply ML to network management.

In order to apply ML to network management, it is necessary to provide sufficient network data and corresponding optimal management data as label data. Of course, well-designed ML models are also needed. Currently, there are only dozens of data available for network management, and these data target a small range of network management. Also, most of the studies use simple ML models for network management, and these models cannot understand the network structures or topologies [3]. Thus, the current studies are limited to solving simple problems and they do not show enough merit of ML compared to the other ML research areas.

In this paper, we generate dynamic and diverse network data, and represent the network states as label data. Also, we proposed to use Graph Neural Network (GNN) for VNF management. Our model uses network data represented by a graph and learns the state embedding [4] of nodes. This model effectively learns the network structure and generates near-optimal solutions in a short time.

## II. Related Work

In this section, we introduce several studies that are relevant to our study. The orchestration of VNF is derived from [5]. It points out the execution time problem of ILP and proposes a heuristic method. The proposed heuristic method shows fast execution time, but it generates a sub-optimal solution. ML-based VNF management is presented in [6]–[9]. These papers target only simple problems such as optimal VNF instance number [6], [7] and VNF chaining [8], [9]. There is an approach to graph-based ML in [10]. The approach is a little similar to GNN based network management, but it focuses more on graph theory and introduces core concepts only. The GNN based VNF resource prediction is proposed in [11]. It predicts resource usage using GNN and scales resources sub-optimally. Besides, the proposed ML-based resource scaling method is validated by comparing it with the human manual scaling in terms of the latency measured in post-scaling. However, this study lacks to validate whether the result is optimal because the proposed method does not compare itself with the optimal scaling method. Although GNN can represent graph data well, the method is not adequate to predict future data without recurrent models. The models simply follow historical resource data rather than predict future data.

Our model uses GNN to better represent network information. Our proposed model produces more specific and more realistic VNF management policies rather than merely predicting the overall number of VNF instances or network resource usage. We created three VNF policy classes (*add, remove, none*) and predicted policies for all network servers and VNF types.

## III. Problem Definition

### A. Physical Network

We represent the physical network as an undirected graph $G = (N, E)$, where $N$ and $E$ denotes the set of nodes and links. We classify nodes into the server $s \in N$ that can deploy VNFs and the switch that cannot deploy VNFs. Supposing $c_s \in \mathbb{R}^+$ is the number of CPU cores and $V_s$ is the deployed VNFs on $s$, server data $D$ has $D_s = (c_s, D_{V_s})$ for all $s$, where $D_{V_s}$ is the deployed VNF data on server $s$.

The physical links $E$ have $L$ and $C$ denoted by link data and connection data. For $(i, j) \in E$, the link has $L_{ij} \in L$ defined as $(m_{ij}, b_{ij}, d_{ij})$, where $m_{ij} \in \mathbb{R}^+$, $b_{ij} \in \mathbb{R}$ and $d_{ij} \in \mathbb{R}^+$ is the maximum bandwidth, available bandwidth and delay between node $i$ and $j$. The link also has $C_{ij} \in C$, the indicator whether the link is between the nodes.

$$C_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or there is a link between node } i \text{ and } j, \\ 0 & \text{otherwise.} \end{cases}$$

### B. Virtual Network Function

Supposing $\mathcal{T}$ is the set of VNF types, each VNF has different VNF type $t \in \mathcal{T}$. The VNF type decides the number of the required CPU cores, processing capacity, processing delay, and deployment cost represented by $\mathcal{E}_t \in \mathbb{R}^+, \tau_t$ (in Mbps), $\delta_t$ (in ms), $\mathcal{F}_t \in \mathbb{R}$, respectively. We define deployed $t$ type VNF as $\mathcal{V}_{st} \in V_s$. $\mathcal{V}_{st}$ has dedicated data $D_{st}$. We assume that $\mathcal{I}_{st} \in \mathbb{R}^+$ is the VNF instance number and $\tau_t$ (in Mbps) , $\kappa_{st}$ (in Mbps) is the maximum capacity and used capacity. We represent the deployed VNF type data $D_{st}$ as follows:

$$D_{st} = \begin{cases} (\mathcal{I}_{st}, \tau_t, \kappa_{st}) & \text{if } V_s \text{ has type } t, \\ 0 & \text{otherwise.} \end{cases}$$

Now, the deployed VNF data $D_{V_s}$ can be expressed as $\bigcup_{t \in \mathcal{T}} D_{st}$.

### C. Service Request

Let the network receives many different service requests by users. $\Psi$ is the set of services, and a service is $\nu \in \Psi$ represented by $(w_\nu, u_\nu, d_\nu, \phi_\nu, p_\nu, \beta_\nu, \gamma_\nu)$. $w_\nu, u_\nu \in N$ are the ingress and egress switch, respectively. $d_\nu \in \mathbb{R}^+$ is the running time of the service and $p_\nu \in \mathbb{R}$ is the penalty cost of Service Level Agreement (SLA) violation. $\phi_\nu$ is the service request type that represents ordered VNF sequence. $\beta_\nu$ (in Mbps) , $\gamma_\nu$ (in ms) are the bandwidth demand of the traffic and max latency of SLA violation, respectively.

### D. VNF Management

The objective of VNF management is to reduce network OPEX while guaranteeing service requirements [5]. To achieve this objective, the optimal number of VNF instances should be deployed on optimal locations (servers) while considering several requirements, i.e., service constraints and the physical network. The VNF management policy can be expressed as *add* or *remove* specific type of VNFs on specific servers. We regard this policy as a classification for all servers and VNF types.

## IV. Graph Neural Network

### A. Graph Neural Network

ML is a promising technique in many research fields such as natural language processing and computer vision. Most of these fields use Euclidean domain data, and Convolutional Neural Network (CNN) [12] and Feed Forward Neural Network (FNN) are usually used to learn the data. Recently, ML is used in chemistry and biology, but CNN and FNN cannot learn their data because their data is usually non-Euclidean graph data. This non-Euclidean graph data contains rich relational information between each pair of neighboring elements and represents many kinds of graph structure data, i.e., social networks, physical systems [13]. Thus, the graph data gets attention and motivates to derive GNN.

GNN is the generalized model of CNN. GNN uses graph data as input, but it shares many things with CNN. While CNN uses the local connection between data and shares weights to reduce the computational cost [14], GNN uses the connection in the graph and share weights [13]. GNN is also motivated by graph embedding [4] that learns to represent graph information about nodes, edges, and sub-graphs. However, graph embedding does not share weights and it has a generalization problem [15].

The objective of GNN [16] is to learn state embeddings [4] and obtain outputs. Let $x$ and $h$ denotes the input features and hidden states. When $co[v]$ is the set of edges connected to $v$ and $ne[v]$ is set of neighbors of node $v$, we can define state embedding and output using $h_v$ and $o_v$ as follows:

$$h_v = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]})$$
$$o_v = g(h_v, x_v),$$

$x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]}$ is the features of $v$, feature of edges connected to $v$, the states of neighborhood nodes of $v$, features of the neighborhood nodes of $v$. $f$ is transition function and $g$ is output function.

$\boldsymbol{H}, \boldsymbol{O}, \boldsymbol{X}, \boldsymbol{X_D}$ are stacking variable of all the states, outputs, graph features and node features, respectively. $\boldsymbol{F}, \boldsymbol{G}$ are global transition function and global output function. The $\boldsymbol{H}$ is the banach's fixed point [17] and uniquely defined with contraction map $\boldsymbol{F}$.

$$\boldsymbol{H} = \boldsymbol{F}(\boldsymbol{H}, \boldsymbol{X})$$
$$\boldsymbol{O} = \boldsymbol{G}(\boldsymbol{H}, \boldsymbol{X_D}),$$

The function $f$ and $g$ can be a neural network (e.g, FNN, CNN, RNN), and the model updates $\boldsymbol{H}$ until the learning process is finished. We define target information and output of node $i$ as $r_i$ and $o_i$, respectively. Then, the loss can be represented as follows :

$$loss = \sum_{i=1}(r_i - o_i)$$

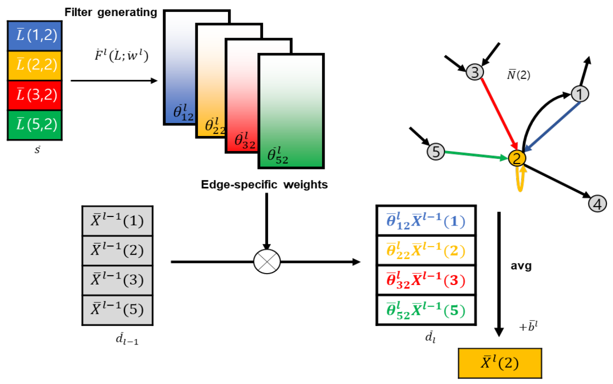


Fig. 1. Edge-conditioned filtered Graph Convolutional Neural Network

### B. Edge-conditioned Filtered Graph Convolutional Neural Network

We use Edge-conditioned Filtered Graph Convolutional Neural Network [18] to get a near-optimal solution for VNF management. Supposing an undirected graph $\bar{G} = (\bar{N}, \bar{E})$, we define a set of nodes and edges as $\bar{N}$ with $|\bar{N}| = \bar{n}$ and $\bar{E} \subseteq \bar{N} \times \bar{N}$ with $|\bar{E}| = \bar{m}$. We assume $l \in \{0, .., l_p\}$ as FNN layer in each learning iteration. All nodes and edges have labeled data that are defined as labeling function $\bar{X}^l$ and $\bar{L}$, respectively. Supposing $\bar{d}_l$ and $\bar{s}$ are the number of node features and edge features respectively, $\bar{X}^l : \bar{N} \mapsto \mathbb{R}^{\bar{d}_l}$ assigns nodes to labels (node features) and $\bar{L} : \bar{E} \mapsto \mathbb{R}^{\bar{s}}$ assigns edges to labels (edge features). We consider $\bar{X}^0$ as the first input features. The neighborhood nodes of node $\bar{i}$ are represented by $\eta(\bar{i}) = \{\bar{j} | (\bar{i}, \bar{j}) \subseteq \bar{E}\} \cup \{\bar{i}\}$, where $\bar{i}$, $\bar{j} \in \bar{N}$.

GNN model usually uses neighborhood information to make new states for nodes. One of the easiest ways to make new states is the weighted sum method. The weighted sum method does not need to consider the order and sizes of nodes. However, this method has the disadvantage of smoothing structural information [18]. To overcome the disadvantage, Edge-conditioned Filtered Graph Convolutional Neural Network generates edge filters. Filter generation function $F^l : \mathbb{R}^{\bar{s}} \mapsto \mathbb{R}^{\bar{d}_l \times \bar{d}_{l-1}}$ is FNN, and it generates filter matrix $\bar{\theta}$ that is multiplied by the feature sets of neighborhoods. By averaging the multiplied values, a new node label is generated. Supposing $F^l$ is parameterized by weight parameter $\bar{w}^l$, $\bar{X}(\bar{i})$ can be defined as follows with bias parameter $\bar{b}^l$:

$$\bar{X}^l(\bar{i}) = \frac{\bar{i}}{|\bar{N}|} \sum_{\bar{j} \in \bar{N}} \bar{F}^l(\bar{L}(\bar{j}, \bar{i}); \bar{w}^l) \bar{X}^{l-1}(\bar{j}) + \bar{b}^l$$
$$= \frac{\bar{i}}{|\bar{N}|} \sum_{\bar{j} \in \bar{N}} \bar{\theta}^l_{\bar{j}\bar{i}}(\bar{j}) \bar{X}^{l-1}(\bar{j}) + \bar{b}^l$$

We can regard $\bar{X}^l(\bar{i})$ as the state from the $l$th transition function. The process of the obtaining $\bar{X}^l(\bar{2})$ is shown in the Fig 1. After obtaining the states, we can get the output using output transition function.

### C. GNN for VNF Management

Some studies try to use ML to manage VNFs. However, they treat the network data as numeric data rather than the graph data [6]–[8]. They just make one long table and put all network data into the table. However, this table data is just numeric values of network components and cannot fully represent network structure. The data does not have connectivity information among network components, and this deficiency is one of the factors of the performance bottlenecks. In this paper, we use GNN and treat network data as a graph. The graph has connectivity information [13] and it can provide the network structure data [13]. Compared to the other VNF management studies [6]–[8], our VNF management problem is much more difficult to solve. We decide the number of instances and locations for all nodes and all VNFs types instead of deriving the only optimal number of VNF instances. This is not an easy task and requires many network information and useful learning model. However, we can get the solution by using GNN. Also, our model can make VNFs management decisions more specific and realistic.

TABLE I
SERVICE CATALOG [6]

| Service id | Type ($\phi$) | Proportion |
|---|---|---|
| 1 | NAT - Firewall - IDS | 0.6 |
| 2 | NAT - Proxy | 0.1 |
| 3 | NAT - WANO | 0.2 |
| 4 | NAT - Firewall - WANO - IDS | 0.1 |

## V. Data Generation

### A. Service Request Generation

We generate service requests and request lists. Whenever a new service request is generated, the list is also generated. The request lists contain several service requests whose service time $d$ is not expired yet. These lists suppose a realistic network situation where multiple services are provided simultaneously. Supposing $\nu_{\text{new}}$ is the newest generated request and $\mu$ is the request list id and $\hat{a}$ is an arrival time of request, we can represent the request list as follows:

$$\pi_{\mu+1} = \{\nu_{\text{new}}\} \cup \pi_{\mu} \cap \{x : d_{\nu_x} > \hat{a}_{\nu_{\text{new}}} - \hat{a}_{\nu_x}\}$$

We use internet2 traffic pattern for generating service requests. Fig 2 is the one week traffic pattern that generated from real networks [19]. We generate 3 requests per minutes according to proportion $P$ of the request type $\phi$ in Table I. Also, we discard $3 * (1 -$ **normalized traffic volume** in Fig 2) requests per minutes to follow the traffic pattern. Each request has normalized arrival time (mean = $1/6P_{\phi}$, s.d = 7.5) and service time $d$ as $|100 + \mho|$, where $\mho$ is a random value in normalized value (mean = 50, s.d = 25).

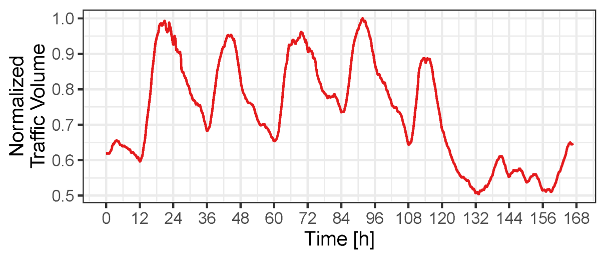


Fig. 2. Normalized traffic pattern of Internet2 network

Each request has a different max latency of SLA violation that is randomly determined between 700ms and 750ms. The bandwidth of the service is randomly chosen between 40000 Mbps and 50000 Mbps. We suppose all services have the same SLA penalty as 1E-7.

## B. Network Data Generation

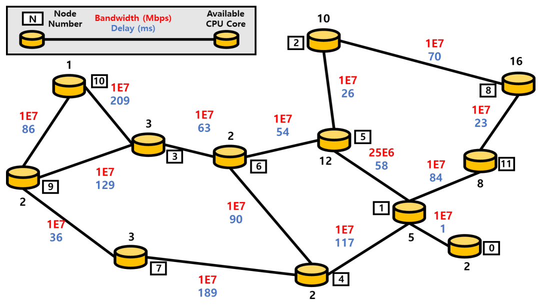


Fig. 3. Internet2 network topology

We use internet2 network topology in Fig 3. The topology consists of 12 nodes and 15 edges. We suppose all nodes are servers, and all nodes can be a source and destination of services. Each server and edge has information of available CPU cores, bandwidth, and delay. All servers have the same specification in Table II. We also set the energy consumption cost and transition cost in the Table II and Table III shows the VNF catalog we used.

Whenever a new request list is generated, we collect current network data $(D, L, C)$. Then, we use ILP to get optimal VNF polices for the current network with the request list. When ILP finds the optimal VNF polices, we change the current network configuration to optimal network configuration. This process is repeated until all of the generated data is handled.

TABLE II
SERVER AND COST INFORMATION

| Server specification [5] | | |
|---|---|---|
| Idle Energy Consumption ($e_{\text{ie}}$) | Peak Energy Consumption ($e_{\text{pk}}$) | CPU Cores ($C_{\text{spec}}$) |
| 80.5W | 2735W | 160 |
| Energy consumption cost ($\lambda_{energy}$) | | |
| 0.1 | | |
| Transition cost per bit ($\lambda_{transit}$) | | |
| 3.62E-7 | | |

TABLE III
VNF CATALOG [20]

| Network Function | CPU Required | Processing Capacity | Processing Delay |
|---|---|---|---|
| Firewall | 4 | 900Mbps | 45ms |
| Proxy | 4 | 900Mbps | 40ms |
| IDS* | 8 | 600Mbps | 1ms |
| NAT | 1 | 900Mbps | 10ms |
| WANO** | 4 | 400Mbps | 5ms |

IDS* : Intrusion Detection System
WANO** : Wide Area Network Optimizer

## C. ILP Calculation

We use specific ILP equations to get optimal VNF policy. This ILP solution reduces OPEX and gets optimal VNF instance numbers and locations [5]. It calculates the VNF deployment cost, energy cost, traffic forwarding cost, SLO violation cost and resource fragmentation cost. As we set all VNF deployment cost as zero, we only consider four costs.

**Energy cost:**

$$\mathbb{E} = \sum_{n \in N} \sum_{t \in \mathcal{T}} \mathcal{I}_{st}(e_{ie} + (e_{pk} - e_{ie})\frac{\mathcal{E}_t}{C_{spec}})\lambda_{energy}$$

**Traffic forwarding cost:**

$$\mathcal{W}^{\nu t_1 t_2}_{s_1 s_2} = \begin{cases} 1 & \text{if } \mathcal{V}_{s_1 t_1} \text{ has traffic between } \mathcal{V}_{s_2 t_2} \text{ for } \nu \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathcal{J}^{\mu t_1 t_2}_{s_1 s_2} = \sum_{\nu \in \pi_\mu} \sum_{t_1, t_2 \in \mathcal{T}} \mathcal{W}^{\nu t_1 t_2}_{s_1 s_2} \beta_\nu$$

$$\mathbb{T} = \sum_{s_1 \in N} \sum_{\substack{s_2 \in \eta(s_1), \\ s_2 < s_1}} (\mathcal{J}^{\mu t_1 t_2}_{s_1 s_2} - \mathcal{J}^{(\mu-1) t_1 t_2}_{s_1 s_2})\lambda_{transit}$$

**SLO violation cost:**

$$\hat{\phi_\nu} \text{ is the VNFs set of } \phi_v.$$

$$\mathbb{S} = \sum_{\nu\in\pi_\mu} \textbf{max}(\sum_{t\in\hat{\phi_\nu}} \delta_t + \sum_{\substack{t_1t_2\in\mathcal{T}\\ s_1\in N}} \sum_{\substack{s_2\in\eta(s_1),\\ s_2<s_1}} \mathcal{W}^{\nu t_1 t_2}_{s_1 s_2} d_{s_1 s_2} - \gamma_\nu, 0) p_\nu$$

**Resource fragmentation cost:**

$$\mathbb{F} = \sum_{\substack{s\in N,\\ V_s\neq 0}} (C_{\text{spec}} - \sum_{t\in\mathcal{T}} \mathcal{I}_{st}\mathcal{E}_{st})\lambda_{\text{core}}$$

$$+ \sum_{s_1\in N} \sum_{s_2\in\eta(s_1)} \frac{\textbf{max}(\mathcal{J}^{\mu t_1 t_2}_{s_1 s_2}, 0)}{\mathcal{J}^{\mu t_1 t_2}_{s_1 s_2}} (m_{s_1 s_2} - \mathcal{J}^{\mu t_1 t_2}_{s_1 s_2})\lambda_{\text{band}}$$

($\lambda_{\text{core}}$ : cost of CPU cores, $\lambda_{\text{band}}$ : cost of bandwidth)

The objective of ILP is minimizing $(\acute{a}\mathbb{E}+\acute{b}\mathbb{T}+\acute{c}\mathbb{S}+\acute{d}\mathbb{F})$ [5]. $\acute{a}$, $\acute{b}$, $\acute{c}$, $\acute{d}$ is a weighting factor.

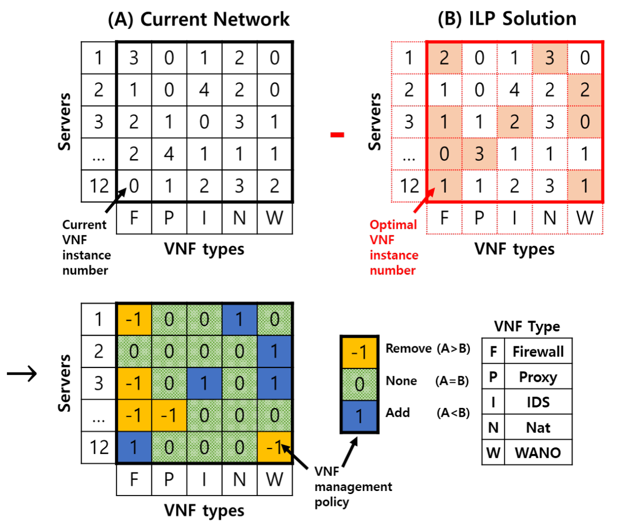


Fig. 4. Generation of the label data

## D. Learning Data Set Generation

We generate the learning data set for ML. Whenever a new request is generated, the feature data is generated from the network data $(D, L, C)$ and service request $(\nu)$. At the same time, the label data is also generated by classifying the difference between the current deployed VNFs and the ILP-based VNF deployment solution as Fig 4. The three management policies are used for classification.

The learning data consists of numeric data and categorical data together. We normalize all numeric data and apply one-hot encoding for categorical data. The network data should be represented as a graph. We convert $D$ as matrix that size is **node number** $\times$ **node feature number**. $C$, $L$ is also converted as **node number** $\times$ **node number** size matrix.

## VI. Learning Process

We implement an ML model using tensorflow backend. The learning process is shown in Fig 5. The network data is converted as graph data, and GNN uses these data as input. GNN learns the state embedding of the nodes, and then we concatenate the state embedding with a service list. FNN

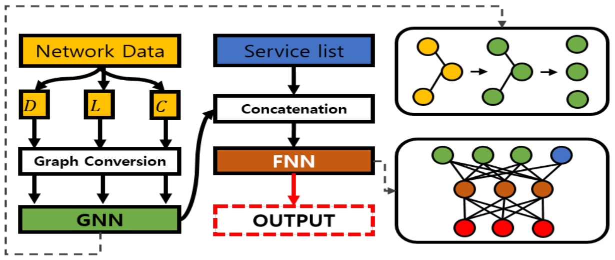


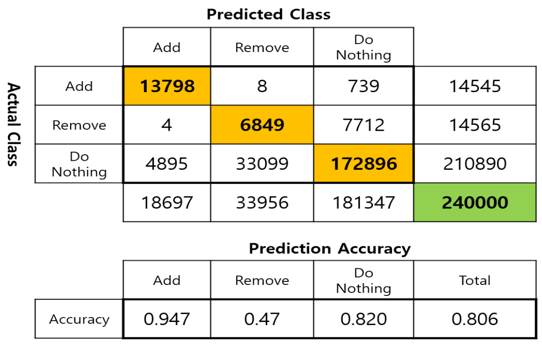
Fig. 5. GNN-based VNF management policy learning model

learns the concatenated data and makes the output. We use three FNN layers and apply a batch normalization and dropout (0.5) to each layer.

Because each data set is used to classify 60 policies into the three classes of VNF management policy as shown in Fig 4, the last FNN layer has 180 output. For each policy, we apply softmax and use cross-entropy as a sub-loss function. The objective loss function is the sum of the sub-loss function multiplied class weight. The class weight is the reciprocal of each class ratio of the data. The VNF policies are usually imbalance and this imbalance can degrade the performance of learning. Thus, we multiply the sub-loss functions with class weights following the label class as below:

$$\textbf{loss function} = \sum_{\substack{s\in N\\ t\in\mathcal{T}}} \sum_{i}^{3} l_i^{st} log(o_i^{st}) \frac{A_1^{st} + A_2^{st} + A_3^{st}}{A_i^{st}}$$

$(l_i^{st}$, $o_i^{st}$, $A_i^{st}$ : label, output, number of class when policy is $i$ and server is $s$ and VNF type is $t)$



Predicted Class

| Actual Class | Add | Remove | Do Nothing | |
|---|---|---|---|---|
| Add | 13798 | 8 | 739 | 14545 |
| Remove | 4 | 6849 | 7712 | 14565 |
| Do Nothing | 4895 | 33099 | 172896 | 210890 |
| | 18697 | 33956 | 181347 | 240000 |

Prediction Accuracy

| | Add | Remove | Do Nothing | Total |
|---|---|---|---|---|
| Accuracy | 0.947 | 0.47 | 0.820 | 0.806 |

Fig. 6. Confusion matrix of the prediction results

## VII. Experiment

We use 20,000 data set described in V. We split the data into 12800, 3200, 4000 as training, validation, and test, respectively. We use Stochastic Gradient Descent (SGD) optimizer and the learning rate is 0.01. Our GNN has 15 dimension for each node output state and three FNN layer has 700, 500, 180 hidden states. The total number of parameters is 678K.

The prediction result of the test data set is shown in Fig 6. The total accuracy of prediction is 0.806 and *add,*

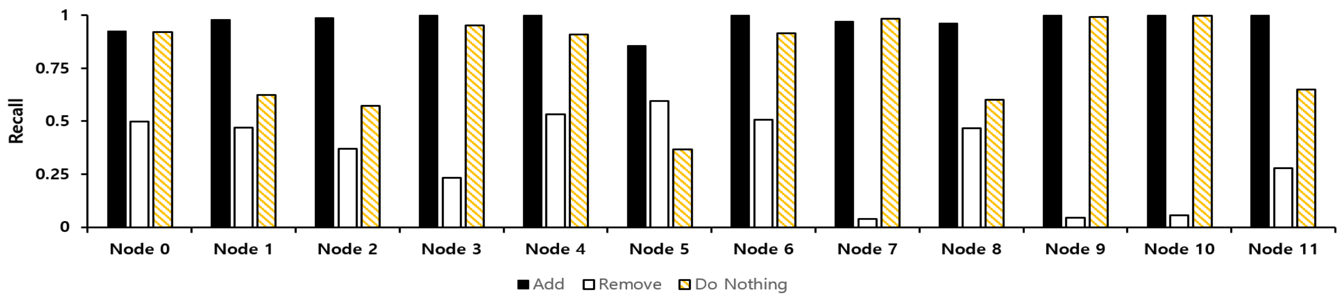


Fig. 7. Recall of add, remove, none

*remove, none* class have 0.947, 0.470, 0.820, respectively. Our model has a high prediction accuracy for *add* class, while the prediction accuracy for *remove* class is low. Interestingly, we can find the model tended to predict the remove class as *none* class. The reason for this phenomenon is that the ILP solution is used for generating label data. ILP weights more on SLA violation cost than the energy cost, and resource fragmentation cost. Thus, the model tends to be conservative to remove VNF that can make SLA cost.

The recall result of each node is shown in Fig 7. The recall of *add* class is almost the same for all nodes, but the recall of *remove* class is low for node 7, 9, 10. These nodes have less than 4 CPU cores and located outward. It means that these nodes cannot deploy VNFs excluding NAT and cannot be included in the routing routes many times due to delay constraint. Thus, these nodes do not remove NAT to deploy other VNFs or to change routes. Thus, the model has difficulty learning to *remove* class for these nodes.

## VIII. Conclusion

In this paper, we proposed a method of managing VNF using Graph Neural Networks (GNNs). Our model uses network data as graph data and learns the state embedding of each node. Then, we concatenate states of node and service requests to find optimal VNF management action. Our model solves complex VNF management problems. The model considers many network constraints that include limitations of physical network and requirements of service requests and VNFs. In the experiment, our model provides near-optimal VNF management policy within a reliable time.

## IX. Acknowledgment

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP, Development of virtual network management technology based on artificial intelligence) under Grant 2018-0-00749.

## References

[1] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[2] Altman N. Krzywinski M. Bzdok, D. Statistics versus machine learning. *Nat Methods*, 15:233—-234, 2018.

[3] Raouf Boutaba, Mohammad A Salahuddin, et al. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, 2018.

[4] Hongyun Cai, Vincent Zheng, et al. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[5] Faizul Bari, Shihabur Rahman Chowdhury, et al. Orchestrating virtualized network functions. *IEEE Transactions on Network and Service Management*, 13(4):725–739, 2016.

[6] Stanislav Lange, Hee-Gon Kim, et al. Predicting VNF Deployment Decisions under Dynamically Changing Network Conditions. In *International Conference on Network and Service Management*, 2019.

[7] Sabidur Rahman, Tanjila Ahmed, et al. Auto-scaling vnfs using machine learning to improve qos and reduce cost. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[8] Jianing Pei, Peilin Hong, and Defang Li. Virtual network function selection and chaining based on deep learning in sdn and nfv-enabled networks. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2018.

[9] R. Shi, J. Zhang, W. Chu, et al. Mdp and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization. In *IEEE International Conference on Services Computing*, 2015.

[10] Wolfgang Kellerer, Patrick Kalmbach, et al. Adaptable and data-driven softwarized networks: Review, opportunities, and challenges. *Proceedings of the IEEE*, 107(4):711–731, 2019.

[11] R. Mijumbi, S. Hasija, et al. A connectionist approach to dynamic resource management for virtualised network functions. In *International Conference on Network and Service Management (CNSM)*, 2016.

[12] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series, the handbook of brain theory and neural networks, 1998.

[13] Z. Liu and J. Zhou. *Introduction to Graph Neural Networks*. 2020.

[14] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[16] Franco Scarselli, Marco Gori, et al. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[17] Mohamed A Khamsi and William A Kirk. *An introduction to metric spaces and fixed point theory*, volume 53. John Wiley & Sons, 2011.

[18] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[19] Yin Zhang. Abilene traffic matrices, 2004.

[20] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.