Abstract—In management of computer network systems, a service function chaining (SFC) module plays a vital role in generating an efficient network traffic path that connects virtualized network functions (VNF) on network topology to serve a user request. The SFC module needs to generate a complete path quickly even in various network situations, including dynamic VNF resources, various types of requests, and various network topologies to provide the best quality of service. The previous supervised learning method demonstrated that graph neural networks (GNN) could represent network features for the SFC task. However, the supervised learning method works properly only in the network situation in which the model was trained with labels. Due to the limitation, it showed poor performance on new unseen network situations. In this paper, we apply a reinforcement learning algorithm to train GNN based models in various network situations even without label information. In the experiments, compared to the previous supervised learning method, the proposed methods demonstrate remarkable generalization effects, showing that the proposed methods could work successfully on unseen network situations without re-designing and re-training.

Index Terms—Network Management, Service Function Chaining, Graph Neural Network, Reinforcement Learning

I. INTRODUCTION

Development of computer network systems based on software such as software-defined networking (SDN) and network function virtualization (NFV) opens the softwarization of networking service systems [1], [2]. Based on the advanced technologies, the scale of computer networks and the types of services have become large and diverse. As a side effect of these developments, managing networks in an optimal way has become a complex problem, and the automatic network management system has become a promising technology [3].

An automatic management system includes several modules such as virtualized network function (VNF) deployment, resource demand prediction, auto-scaling, anomaly detection, and service function chaining (SFC). Among those modules, we focus on the SFC module which is about the first step to serve user requests. To conduct the SFC task, the SFC module should consider three components of the network situation: network topology, VNF deployment, and user request. The network topology can be considered a graph whose nodes and
edges are physical servers and physical connections between them. The VNF is a virtualized software that conducts specific network services. The instance of VNF is deployed on a physical server at a cost of computing resources (e.g., CPU and memory). Usually, the deployment strategy is determined by the VNF deployment module, which is based on an integer linear programming (ILP) algorithm or a model trained for the VNF deployment task [4]. Finally, the user request consists of a sequence of requested VNF types, source and destination nodes, arrival time, duration, and maximum latency.

Given this network situation, as in Fig. 1, the SFC module carries out the essential task of generating a path for network traffic’s traverse. Through the path, the network traffic should pass all of the requested VNF instances from the source node to the destination node to satisfy the user request [5]. However, among the many possible cases of paths, the SFC module should find an efficient path with the shortest delay. It is challenging to generate an optimal path if the scale of network topology is large or the types of requests are diverse and complicated. The problem even gets worse if the network situation changes over time.

Recently, deep learning techniques have been applied to this challenging SFC task with high-capacity neural network models. To the best of our knowledge, the first deep learning approach to the SFC task was based on simple deep belief networks (DBNs) as proposed in [6], whose performance was not sufficient enough to be a practical SFC module. Another deep learning approach improved the performance based on the gated graph neural network (GG-NN) [7] as proposed in [8]. They called their encoder-decoder architecture model GG-RNN and demonstrated its advantages in terms of performance. As an additional advantage, it can be applied to new network topologies without re-designing the model architecture since the GG-NN based encoder can encode any network topology into representation vectors.

The previous work trained the GG-RNN models in the supervised learning (SL) approach. Even though the model can be applied to a new network topology, the performance is not guaranteed because the SL approach can train the model only on a labeled topology. Therefore, whenever the model receives an unseen network situation (unseen network topology, VNF deployment, and user requests), its performance is unreliable. There is a possible solution to train the model in an online fashion with the label given by the ILP algorithm. However, the ILP algorithm requires a lot of computational costs especially when the network topology is large. It may cause unstable interactions with other modules when it is modularized in the automatic management system. Therefore, we need the training algorithm that trains the SFC modules to be flexible about changes in network situations while preserving good performance.

To make the SFC modules flexible to various network situations, we propose to use reinforcement learning (RL) algorithms to train the models. Compared to the SL algorithm, RL algorithms have the advantage that it is able to train the model even in unlabeled network situations since they can easily receive a reward from the network. In this paper, we design the components of the RL algorithm for the network domain and train GG-RNN models with the standard policy gradient algorithm, REINFORCE [9]. Additionally, to make the models even more flexible, we train the models with randomly changed network topologies and randomly located VNF deployments during the training stage where the labels (optimal paths) are unavailable.

Our experiments were conducted on the simulated network topology dataset, which reflects the realistic networking system. In the results, our proposed RL approaches show similar performances to the SL approach in the original Internet2 topology. Furthermore, our proposed RL approaches significantly outperform the SL approach in the randomly changed network situations. These results demonstrate that the proposed RL approaches are remarkably more flexible than the SL approach. Additionally, we visualize the generated paths from several approaches to demonstrate the flexibility of our approaches qualitatively.

II. BACKGROUND

A. Service Function Chaining

We briefly describe the problem definition for the SFC task in this section. For more details, see [8].

As we mentioned above, the SFC task needs network topology, VNF deployment, and user requests. The network topology is a graph with the set of nodes \( N \) and the set of edges \( E \). Before conducting the SFC task, several VNF instances \( M \) are deployed on the network topology based on the decision of the VNF deployment module. Then, the SFC module should generate network traffic paths with high-quality services (QoS) according to the received user requests. The QoS is generally measured by packet loss, bit rate, throughput, transmission delay, and availability. However, because our experiments are conducted by a model on a simulated dataset, the QoS is measured by the failure ratio and the total transmission delay of the complete generated path. The generated path is classified as a complete case if it processes all of the requested VNFs and arrives at the destination node. Otherwise, it is classified as a failed case. Delays are caused by traversing the network traffic and processing VNF instances. Given a complete path \( P \), the delay of the path is calculated by

\[
\text{Delay}(P) = \sum_{uv \in E} d_{uv} P_{E,uv}^P + \sum_{m \in M} d_m P_{M,m}^P, \tag{1}
\]

where \( d_{uv} \) and \( d_m \) are delays of that edge and VNF instance, respectively. \( P_{E,uv}^P \) and \( P_{M,m}^P \) are logical variables indicating whether or not the path \( P \) includes edge \( uv \) and VNF instance \( m \), respectively.

In this paper, the basic network topology is Internet2 [10]. At each time step, the optimal VNF deployment is determined by the ILP algorithm. It guarantees a sufficient amount of VNF instances corresponding to total user requests at each time step while minimizing the extra instances. It follows the realistic scenario of user requests. We generated the simulated dataset
The encoder part is based on the GG-NN is a different GRU model from the one in A is the function of the gated equal to \( \theta \)

\[ h^{(l)}_u, h^{(l)}_{u,in} = GRU(h^{(l-1)}_u, h^{(l-1)}_{u,in}) \]  

where \( h^{(0)}_u \) is the initial hidden state of node \( u \), which is the annotation of that node. It consists of \( f_u \), a feature vector of node \( u \) and then zero-padding vector. In Eq. (3), \( |N| \) is the number of nodes in the network topology. \( A_u \) is the adjacency vector of the node \( u \). It indicates the connections between node \( u \) and other nodes. With the adjacency vector \( \mathbf{A}_u \), \( GRU(\cdot, \cdot) \) is the function of the GRU layer in the decoder part. By repeating this step, the decoder generates a sequence of \((\hat{n}_t, \hat{p}_t)\), which is a resulting path \( P \).

3) Loss: The loss of the GG-RNN model at the generation step \( t \) is composed of two cross-entropy (CE) loss terms. The first term is CE on the node distribution and its label, and the second one is CE on the VNF process distribution and its label. The total loss is the summation of all CE values along with the generation steps, which are computed as follows:

\[ L(\theta) = \sum_{t=0}^{T} \{ CE(Pr_u(\hat{n}_t; \theta), n^*_t) + CE(Pr_p(\hat{p}_t; \theta), p^*_t) \} \]  

where \( T \) is the last generation step, \( n^*_t \) and \( p^*_t \) are the label of the node and whether to process the VNF instance, respectively. \( \theta \) is the parameter set of the entire GG-RNN model.

Fig. 2. GG-RNN model architecture with encoder (a) and decoder (b). These illustrations are slightly modified from [8] with permission.
III. REINFORCEMENT LEARNING METHOD FOR SFC

A. Reinforcement Learning Components

To apply RL methods to the SFC task, we need to define the components of Markov decision process: state, action, reward, and policy. In our setting, the policy network is defined by the GG-RNN model as follows:

$$\pi_\theta = \text{GG-RNN}_\theta(s_t, a_t).$$

The state consists of inputs to the GG-RNN model, such as annotation matrix, adjacency matrix, and three additional inputs to the decoder as mentioned in section II-B2. The previous hidden state of the $GRU_{dec}$ is also part of the state. To sum it up, the state at the generation step $t$ is composed as follows:

$$s_t = [h_t^{(0)}, A, V_{all}, V_{now}, n_t^{-1}, c^{(t-1)}].$$

The action consists of selecting the next node and the decision to process the top priority type of VNF instance on the selected node. The action at the generation step $t$ is composed as follows:

$$a_t = [\hat{n}_t, \hat{p}_t].$$

We assign a positive reward only when the agent successfully generates a complete path at the end of the episode. To make the model generate a path with a small delay, we add a negative reward computed by the total delay of the generated path as a penalty. The negative delay reward is multiplied by $\lambda$, which is the hyperparameter to control the magnitude of the penalty. The reward is defined by

$$r_t = \begin{cases} 10000 - \lambda \times \text{Delay}(P), & \text{if } t \text{ is terminal}, \\ 0, & \text{otherwise}, \end{cases}$$

(12)

where $P$ is a sequence of actions $\{a_1, a_2, \ldots, a_{T-1}, a_T\}$ in an episode. The total delay, $\text{Delay}(P)$ is computed by Eq. (1).

B. REINFORCE

The REINFORCE algorithm [9] is one of the policy gradient algorithms in RL. According to its policy, the agent makes an action given a state and receives the reward and a new state from the environment. The agent repeats this transition until the new state reaches a terminal state, then eventually completes an episode. After an episode is finished, the updating algorithm of REINFORCE calculates the returns for all the actions from the last to the first. After these calculations, the policy network’s parameters are updated by the derivatives of the log probabilities of those actions with respect to the parameters, then each derivative is weighted by the return of its action. The parameter updating and computing return processes are as follows:

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t)G_t,$$

(13)

$$G_t = r_t + \gamma G_{t+1},$$

(14)

where $G_t$ is the return value of the generation step $t$. $\alpha$ and $\gamma$ are a learning rate and a discount factor, respectively. As a result, the policy network is trained to maximize the returns for each episode.

C. Training on Random Network Situations

In the SL approach [8], the performance deteriorated in the changed topology because the model could not be trained on various network situations without labels. In our proposed RL methods, we can randomly change the network situation at every episode during the training stage. We propose two strategies to change the network situation as illustrated in Fig. 3.

In the first strategy, we add a random amount of nodes and edges at random locations. The probability and the number of trials to add a node and an edge are set to $(0.1$ and $12)$ and $(0.3$ and $15)$, respectively. We add two edges with randomly selected neighbor nodes whenever we add a new node. After this adding process, we remove randomly selected edges. The probability and the number of trials for removing edges are set to $(0.3$ and $30)$. However, if a trial of the removing process disconnects a connection from any node to any other nodes, we dismiss the trial.

In the second strategy, we conduct the same processes as the first strategy and we additionally change the locations of the already deployed VNF instances randomly. Compared to the first strategy where only network topology is changed, the second strategy changes network topology and VNF locations.

For the experiments with the first strategy, we generate 100 random network topologies before training. Then, we select one topology randomly at the beginning of every episode. For the experiments with the second strategy, we also select one from the generated 100 random topologies and we additionally change the locations of the already deployed VNF instances at the beginning of every episode. We also generate 100 different random topologies for the testing and do the same processes.
as above before starting each test episode.

IV. EXPERIMENTS AND RESULTS

A. Experimental Settings

After pre-training the GG-RNN model with the SL approach, we trained the pre-trained model with RL approaches. The hyperparameters are the same as [8] except for the learning rate of the proposed RL approaches, which is 0.00001. We used the ϵ-greedy sampling for exploration where ϵ was 0.01. The discount factor γ in Eq. (14) was set to 0.999. In the experiments, the models were trained and tested on the simulated dataset generated by the methodology in [10] to get a highly realistic and dynamic simulation. The numbers of simulated network situations for training, validation, and testing were 13134, 100, and 1500, respectively.

We trained models with seven different configurations. The first model is the baseline GG-RNN which is trained with the SL approach as mentioned above. To the best of our knowledge, the GG-RNN model architecture is the only SFC model that can handle changeable network topology without re-designing. That is why our RL approaches are compared with only the SL approach applied to GG-RNN model architecture. For the RL approaches, we set λ in Eq. (12) to 0 or 1 to check the effects of the delay penalty in reward. We also trained the models with the two strategies of random network situations. The first and second strategies are symbolized with ‘CS1’ and ‘CS2’, respectively. In training process for each configuration setting, we selected the best checkpoint based on validation.

B. Tests and Evaluations

We conducted three types of tests, (1) ‘Tests on Original Topology’, (2) ‘Tests on Random Topologies’, and (3) ‘Tests on Random Topo.+VNFs’. That is, the model generates paths (1) on the original topology ‘Internet2’, (2) on random topologies using the first strategy, and (3) on the random topologies and VNF deployment using the second strategy. The test types (2) and (3) follow the two strategies as mentioned in section III-C.

In our experiments, we used two evaluation metrics, ‘Failure Ratio’ and ‘Delay Ratio’. ‘Failure Ratio’ is the ratio of the failed cases out of all the trials to generate network traffic paths. For example, if the model could not complete a path corresponding to its request by its own error, it is a failed case. ‘Delay Ratio’ is the generated complete path’s total delay over the label path’s total delay. Note that this metric is only measurable for the test type (1), because labels are not available for the other test types (2) and (3). Additionally, we added the deterioration rate ‘Deter. Rate’, which is the ratio of ‘Failure Ratio’ of the test (2) or (3) over the ‘Failure Ratio’ of the test (1). It measures how much the performance deteriorates due to the changes in the network topology and VNF deployment.

TABLE I

| Approach + Test Types | Tests on Original Topology | Failure Ratio | Delay Ratio |
|-----------------------|----------------------------|---------------|-------------|
| SL (Baseline)         | 1.1515                     | 0.0080        |             |
| RL(λ = 0)             | 1.4615                     | 0.0064        |             |
| RL(λ = 1)             | 1.4447                     | 0.0072        |             |
| RL(λ = 0) + CS1       | 1.6150                     | 0.0092        |             |
| RL(λ = 1) + CS1       | 1.4979                     | 0.0105        |             |
| RL(λ = 0) + CS2       | 1.5823                     | 0.0103        |             |
| RL(λ = 1) + CS2       | 1.3424                     | 0.0135        |             |

TABLE II

| Approach + Test Type | Test on Random Topologies | Failure Ratio | Deter. Rate |
|----------------------|---------------------------|---------------|-------------|
| SL (Baseline)        | 0.5133 (64.1)             | 0.7399 (92.5) |
| RL(λ = 0)            | 0.2627 (41.0)             | 0.4410 (68.9) |
| RL(λ = 1)            | 0.5088 (41.7)             | 0.4753 (66.0) |
| RL(λ = 0) + CS1      | 0.0432 (4.7)              | 0.0850 (9.2)  |
| RL(λ = 1) + CS1      | 0.0420 (4.0)              | 0.0941 (8.9)  |
| RL(λ = 0) + CS2      | 0.0403 (3.9)              | 0.0663 (6.4)  |
| RL(λ = 1) + CS2      | 0.0496 (3.7)              | 0.0817 (6.1)  |

C. Results and Analysis

1) Results on the Original Topology: Table I summarizes the experiment results on the original network topology. In the ‘Tests on Original Topology’ test, our proposed methods have similar or better performances in terms of ‘Failure Ratio’, though they are slightly worse than baseline in terms of ‘Delay Ratio’. It is because the RL method’s reward is mainly based on whether the model generates a complete path or not, while the SL method trains the model with labels that embed the direct information of the optimal path. Nevertheless, we found that λ reduces ‘Delay Ratio’ at a cost of worse ‘Failure Ratio’. Even though the ‘Delay Ratio’ values of RL approaches are slightly worse than the baseline, ‘Failure Ratio’ is a more critical evaluation for QoS. Therefore, RL approaches on the ‘Tests on Original Topology’ preserve the performances well.

2) Results on Random Topology: Table II summarizes the experiment results on random topologies and random VNF deployments. As we expected, the baseline model performs poorly on the ‘Tests on Random Topologies’ and ‘Tests on Random Topo.+VNFs’ tests with high ‘Failure Ratio’ and ‘Deter. Rate’. However, our RL approaches show significantly better performances on these tests. Even though ‘RL(λ=0)’ and ‘RL(λ=1)’ approaches were not trained on random topologies and random VNF deployment like the baseline, they show better generalization effects. With this, we found that RL approaches are more flexible than the SL approach.

Also, as we expected, when the RL approaches were trained with ‘CS1’ or ‘CS2’, they significantly improved the performances even without re-designing and re-training on those unseen testing topologies. Furthermore, we found that
V. CONCLUSION

In this paper, to make the SFC module applicable to a complicated computer networking system that has various situations, we proposed the RL methods to train the GNN based SFC model after formulating the SFC task as an RL problem. The proposed methods could train the model on various network situations without labels, while the SL method could not. The experiment results demonstrated that the proposed approaches make the SFC module work well in various random network situations, which shows the potential for modularization to the practical computer networking system.

In the future, we can apply the methods to more practical testbed with actual computer networks to confirm that our methods can be applied to real computer networking environments where a lot of different network situations exist without labels.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimino, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, Jan 2015.

[2] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turk, and R. Boutaba, “Network function virtualization: State-of-the-art and research challenges,” IEEE Communications Surveys Tutorials, vol. 18, no. 1, pp. 236–262, 2016.

[3] A. Datta, A. Rastogi, O. R. Barman, R. D’Mello, and O. Abuzaghleh, “An approach for implementation of artificial intelligence in automatic network management and analysis,” Lecture Notes in Networks and Systems, vol. 22, pp. 901–909, Jan 2018.

[4] H.-G. Kim, S. Park, S. Lange, D. Lee, D. Heo, H. Choi, J.-H. Yoo, and J. W.-K. Hong, “Graph neural network-based virtual network function management,” in 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2020, pp. 13–18.

[5] D. Bhame, R. Jain, M. Samaka, and A. Erbad, “A survey on service function chaining,” J. Netw. Comput. Appl., vol. 75, no. C, p. 138–155, Nov. 2016.

[6] J. Pei, P. Hong, and D. 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), 2018, pp. 1–6.

[7] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” 2015.

[8] D. Heo, S. Lange, H.-G. Kim, and H. Choi, “Graph neural network based service function chaining for automatic network control,” in 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2020, pp. 7–12.

[9] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. Learn., vol. 8, no. 3–4, p. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[10] S. Lange, H.-G. Kim, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, “Predicting vnf deployment decisions under dynamically changing network conditions,” in 15th International Conference on Network and Service Management (CNSM), 2019, pp. 1–9.

[11] K. Cho, B. van Merriensboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” 2014.

[12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[13] H. Choi, “Persistent hidden states and nonlinear transformation for long short-term memory,” Neurocomputing, vol. 398, pp. 458–464, 2019.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in ICLR 2013, Jan 2013, pp. 1–12.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.