Reinforcement Learning-based Multi-domain Network Slice Provisioning

Zhouxiang Wu[1], Genya Ishigaki[2], Riti Gour[3], Congzhou Li[1], Feng Mi[1], Subhash Talluri[4], and Jason P. Jue[1]
1. Department of Computer Science, The University of Texas at Dallas, Richardson, Texas 75080, USA
2. Department of Computer Science, San Jose State University, San Jose, CA 95192, USA
3. Department of Computer Electronics and Graphic Technology,
Central Connecticut State University, New Britain, Connecticut 06050, USA
4. Amazon Web Services, Seattle, WA, 98108, USA

***Abstract*—We address the problem of establishing an end-to-end network slice across multiple domains and propose a Reinforcement Learning-based framework that enables multiple domains to collaborate on end-to-end network slicing admission and allocation. The objective is to maximize the long-term revenue of the network operator. We employ a Graph Neural Network (GNN) to capture the topology features as the encoder. The simulation results show that our framework improves the profit of the network operator by up to 15% compared to a greedy algorithm.**

***Index Terms*—Network Slice, Resource Allocation, Machine Learning, Reinforcement Learning, Graph Neural Network**

I. Introduction

With the diversification of service requirements, users' expectations for the network are not limited to homogeneous services. Network slicing, which enables the virtual partitioning of network and computing resources, is a promising approach to fulfill the tailored service requirements of different applications. Network Function Virtualization (NFV) and Software Defined Network (SDN) are two enablers for network slicing. NFV allows virtual network functions (VNFs) to be implemented in commodity hardware, which increases deployment flexibility, while SDN provides flexible resource management and collects network data. Network slices share the same network infrastructure while meeting different service requirements. Each network slice contains required VNFs, computation resources, RAM, storage resources, and network links between VNFs.

There are many types of network slice [1], including multiple core networks sharing a common radio access network (RAN), common spectrum sharing, multiple RANs sharing a common core network, etc. We consider the end-to-end network slice crossing over multiple domains. When a slice needs support from multiple domains, the growing time complexity will make related problems intractable if we treat multiple domains as a single network. The network operator must collect information from each node to obtain the optimal global solution. The overhead of transmitting network device state information is also non-negligible. Furthermore, each domain may belong to different entities, and the willingness to share network information is uncertain. We propose a distributed encoder to resolve scalability and privacy problems. Each domain maintains a Graph Neural Network (GNN) based encoder, and the encoder outputs a representation vector of the domain's network state. The global network operator makes admission and allocation decisions based on the domains' representation vectors and gives feedback to the encoders. The size of the representation vector is much smaller than the size of the actual topology, which alleviates the pressure of network state collection. Furthermore, it is difficult to infer the network state based on the representation vector. Thus, privacy is guaranteed to a certain degree.

In this paper, we focus on resource management for network slice requests (NSRs), and we assume that there is an intelligent agent to take the responsibility of the management. Resource management for network slicing consists of four stages: NSR admission control, resource allocation, resource orchestration, and resource scheduling. In the admission control stage, the intelligent agent must decide whether to accept the current NSR. Once the NSR is accepted, the agent needs to locate the resources required by the NSR in the network infrastructure. Resource orchestration coordinates multi-domains to generate an end-to-end network slice. Resources required by the network slice are reserved for the duration of the network slice holding time. In this paper, we employ a Reinforcement Learning (RL) agent that organically combines the admission control stage and the resource allocation stage to maximize the long-term revenue of the network provider.

The remainder of this paper is organized as follows. Section II introduces related papers of applying RL in network slicing admission and allocation. We formulate the problem in Section III, including network slice design, feature selection, GNN encoder design, and RL training algorithm. In Section V, we design experiments to prove the effectiveness of the proposed algorithm. Finally, we conclude the paper in Section VI.

II. Related Work

This paper extends our previous work [2], [3], in which we consider the admission control problem and resource allocation strategy for elastic slice requests within a single domain.

In [4], the authors assume that network slices exist over a long period of time and that the network operator needs to rearrange resources among slices at a specific frequency.

For the radio access network, RL is applied to allocate radio resources among slices to achieve a higher spectrum efficiency and quality of experience ratio. For the core network, the authors apply RL to schedule VNFs based on their priority in order to achieve a lower average waiting time.

In [5], the authors propose a framework for a network broker that includes slice traffic prediction, slice admission control, and slice scheduling. The slice admission control strategy is based on the computer network constraints and prediction results. The slice scheduling algorithm employs the RL methods to provide feedback to the prediction algorithm.

In this work, we consider a topology representation of the network slice, which is more flexible compared with previous works. We assume that the network slice tenants generate the network slice requests based on their prediction of their users. Once the network slice request is generated and accepted by the network operator, we assume that the QoS is fulfilled. Our network operator has a single objective to maximize the long-term profit based on the network condition, the price, and the cost of the slice request.

III. Problem Formulation

A. Multi-domain Network Infrastructure

We assume that multiple domains exist in the system, which work together to provide network slice services. Each domain is represented as a graph G with a node set V and an link set E. Each node $n \in V$ accommodates a set of VNFs and the computational resource capacity CMP, RAM capacity RAM, and disk storage capacity $DISK$, which supports VNFs located in the node. Each link has a given bandwidth capacity, which supports data transmission between computational nodes. The nodes and links do not have to be physical nodes and links, but could be virtual nodes and virtual links from a hierarchical virtualization network structure. We assume there is a link with unlimited bandwidth between each domain and its neighbors. With this assumption, we can focus on resource allocation and constraints within a domain.

B. Network Slice

The network service request is translated to a Network Slice Instance (NSI) by the Service Management Function. Translating the service request to a tightly matched network slice request is an open problem beyond the scope of this paper. Rather, we focus on the strategy of the network slice allocation to maximize the network operator's profit. The NSI includes the end-to-end requirement of VNFs and related resources. The End-to-end (E2E) network slice instance is split into multiple network slice subnet instances (NSSIs) [6] by the global coordinator, which are distributed to every domain's orchestrator. How to split the NSI into multiple NSSIs is another open question. In this paper, we assume there is a pre-defined strategy to create NSSIs. According to the NSSIs, the domain orchestrators propose several allocation plans or the decision of rejection to the global coordinator. Each allocation plan indicates a set of potential resources that would be allocated to the NSSI. The allocation plans are encoded by considering the network state in the domain after the allocation and passing this information through a GNN encoder to avoid leaking the network state information. The global coordinator evaluates every combination, which consists of one plan from each domain, and selects the most promising combination. If the combination contains one rejection plan from any domain, then the E2E NSR is rejected by the global coordinator. If the combination is valid, then the selected plans are handed back to each domain's orchestrator, and the orchestrator implements the plan accordingly. We summarize the procedure described above in Fig. 1.

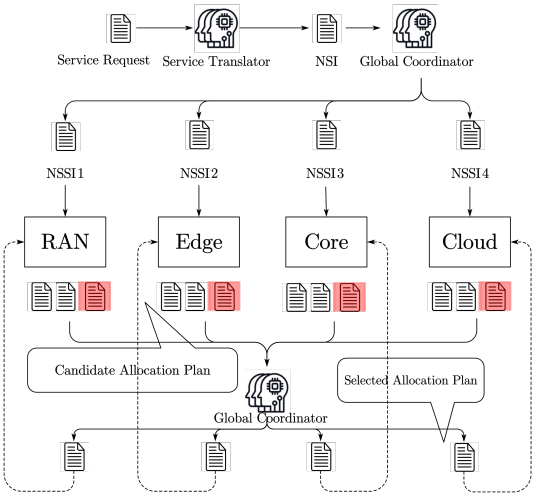

Fig. 1. Procedure of obtaining NSI

C. Network Slice Resource Allocation

Ideally, the intelligent agent in each domain could choose a subset of nodes and a subset of links to support its NSSI. However, the number of combinations of nodes and links could make the selection space almost infinite. Usually, there is a sequencing requirement among VNFs to form a Service Function Chain (SFC). Instead of choosing a random node subset and a link subset, the intelligent agent chooses an allocation strategy from certain number of shortest possible paths which meet the SFC requirements. We use the following data structure to represent a SFC:

$$\begin{aligned} VNF &= [VNF_1, VNF_2, ..., VNF_n] \\ CMP &= [CMP_1, CMP_2, ..., CMP_n] \\ RAM &= [RAM_1, RAM_2, ..., RAM_n] \\ DISK &= [DISK_1, DISK_2, ..., DISK_n] \\ BW &= [BW_{1,2}, BW_{2,3}, ..., BW_{n-1,n}] \end{aligned} ,$$

where the VNF list denotes VNFs and their sequence in SFC; CMP_i, RAM_i, and $DISK_i$ denotes the computer resource requirement, RAM requirement, and disk storage requirement for VNF_i respectively; and $BW_{i,j}$ denotes the bandwidth requirement between VNF_i and VNF_j.

We formalize the path calculation problem as follows. Given (1) a graph G with node set N and edge set E, each node

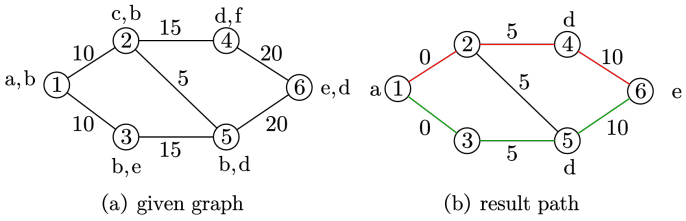

Fig. 2. Example of shortest candidate path

maintains a certain number of VNFs, and each edge has a certain level of bandwidth, (2) SFC, and (3) number K, the problem is to find K shortest disjoint paths in G to implement the SFC. For example, given (1) the graph shown in Fig. 2(a), (2) the SFC ($VNF = [a, d, e], BW = [10, 10]$), and (3) $K = 2$, the outputs are $1 \rightarrow 3 \rightarrow 5 \rightarrow 6$ and $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$. Path $1 \rightarrow 2 \rightarrow 5 \rightarrow 6$ is not valid since the bandwidth between Node 2 and Node 5 is 5 units which is below the required 10 units. We show the candidate paths and residual network in Fig. 2 (b).

We design the algorithm to calculate the shortest path based on Breadth First Search. Domain orchestrators call Algorithm 1 K times to obtain K shortest paths.

D. Reward

The reward function gives feedback on the RL agent's reaction based on the current environment. The RL agent attempts to maximize the accumulated reward over a long period of time instead of maximizing the instant reward. We assume that the network operator gains reward through accepting network slice requests. We assume the profit equals the resource occupied multiplied by the network slice holding time. To formalize the reward function definition, we assume that an end-to-end network slice goes through m domains. Each domain i needs to provide support for SFC_i by allocating SFC_i to $PATH_i$. We use CMP_i, $DISK_i$, RAM_i, and BW_i to represent the total units of computational resource, disk storage resource, ram resource, and bandwidth resource, which are occupied by $PATH_i$, respectively. Each type of resource has its price per unit. We use p_c^i, p_d^i, p_r^i, p_b^i to denote the price per unit of computational resource, disk storage resource, ram resource, and bandwidth resource in domain i,respectively. The holding time of the network slice is denoted by t.

$$Reward([SFC_1, ..., SFC_m]) = \sum_{i=1}^{m} (p_c^i CMP_i + p_d^i DISK_i + p_r^i RAM_i + p_b^i BW_i) \cdot t \quad (1)$$

However, different resource allocation plans lead to additional infrastructure cost. For simplicity, we use the following equation to represent the reward after considering the infrastructure cost:

$$Reward_{cost}([SFC_1, ..., SFC_m]) = Reward([SFC_1, ..., SFC_m]) \cdot p^{l_i}, \quad (2)$$

where $p \in [0, 1]$, and l_i is the number of hops in $PATH_i$.

Algorithm 1 Search the Shortest SFC Path

Input:
Graph G in adjacent list format, SFC
Output: The shortest path to support SFC

Initialize heap h which sorts according to the length of the third item in each entry in decreasing order.
If length of third items in multiple entries have same length, then these entries is sorted according to the length of second item in increasing order.
for node n in V **do**
  if n contains VNF_1 **then**
    h.push((n, [n,], [VNF_1,]))
  end if
end for
while h is not empty **do**
  current node n, current path $[node_1, node_2, ..node_n]$, current VNF list: $[VNF_1, .., VNF_i] \leftarrow h$.pop()
  if length of current VNF list is equal to length of SFC[VNF] **then**
    return current path
  end if
  for node m in neighbors of n **do**
    required bandwidth = SFC[$BW_{i,i+1}$]
    if bandwidth on $edge_{n,m}$ > required bandwidth **then**
      bandwidth on $edge_{n,m}$ = bandwidth on $edge_{n,m}$ - required bandwidth
      if m contains VNF_{i+1} and CMP in m >= CMP_{i+1} and RAM in $m >= RAM_{i+1}$ and DISK in $m >= DISK_{i+1}$ **then**
        h.push((m,[$node_1, .., node_m$], $[VNF_1, ..., VNF_{i+1}]$))
        CMP in $m-$ = CMP_{i+1}; RAM in $m-$ = RAM_{i+1}; DISK in $m- = DISK_{i+1}$
      else
        h.push((m, [$node_1, .., node_m$], $[VNF_1, ..., VNF_i]$))
      end if
    end if
  end for
end while
return None

The network operator will obtain the above reward if the network slice request is accepted. If it is rejected, then the reward is 0. In our assumption, the network slice requests arrive randomly. Long-term revenue is the sum of rewards for an extended period.

E. Problem Description

We formulate this problem in the infinite horizon discrete-time deterministic dynamic system [7]. We first introduce finite-horizon discrete-time deterministic dynamic programming problems.

1) Finite Horizon Problem Formulation: Assume there are N of time stages, and the system makes a control decision for

each stage. Since we only consider the deterministic problems, the state of the next step is determined only by the state and the determination of the current stage. This process can be formalized in the following equation:

$$x_{k+1} = f_k(x_k, u_k), k = 0, 1, ..., N-1, \quad (3)$$

where k is the index of time, x_k is the state of the system at time step k, u_k indicates the decision at time step k, and f_k is a function describe the state transition from x_k to x_{k+1} based on decision u_k. At each step k, the system obtains a reward based on the state x_k and decision u_k, which is denoted by $g_k(x_k, u_k)$. The total reward of a control sequence $\{u_0, ..., u_{N-1}\}$ is

$$J(x_0; u_0, ..., x_{N-1}; u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k), \quad (4)$$

where $g_N(x_N)$ is the final reward. We want to maximize J.

2) Infinite Horizon Problem Formulation: We replace N with ∞ and introduce the discount factor α to avoid unlimited rewards for states. The total reward for x_k is

$$J(x_k) = g_k(x_k, u_k) + \sum_{i=k+1}^{\infty} \alpha^{(i-k)} g(x_i, u_i). \quad (5)$$

The goal is maximize $J(x_k)$ for each state x_k.

3) Network Slice Admission and Allocation Problem: We treat this problem as an infinite horizon problem. The slice request arrives at time step k. The request itself with network state is the state x_k. The reinforcement learning agent makes decision u_k based on x_k. $x_{k+1} = f_k(x_k, u_k)$ and reward $g_k(x_k, u_k)$ will be obtained after the action is applied.

At time step k, the state x_k contains m domain topologies: $[G_1, G_2, ..., G_m]$ with their usage condition and a network slice request which is represented by m SFCs for m domains: $[SFC_1, SFC_2, ..., SFC_m]$, where domain i needs to support SFC_i.

Assuming each domain i provides K_i paths to accommodate SFC_i, the action u_k is to choose one path from each domain and stitch them together to provide end-to-end network slice service or to reject the request.

For each (x_k, u_k) pair, we can calculate $g(x_k, u_k)$ as follows:

$$g(x_k, u_k) = \begin{cases} 0 & \text{if } u_k = \text{reject} \\ Reward_{cost}(u_k) & \text{else} \end{cases}. \quad (6)$$

For state x_{k+1}, the request part always updates to the $k+1^{th}$ request. If the u_k is to reject, then the $x_k + 1$'s domain topologies and usage condition remain the same as x_k. Otherwise, the domain topologies with the usage condition changes to the residual graph after the resources of u_k are occupied. We show the $x_{k+1} = f_k(x_k, u_k)$ as follows:

$$f_k(x_k, u_k) = \begin{cases} \text{graph from } x_k \text{ and request } k+1 & \text{if reject} \\ \text{residual graph and request } k+1 & \text{else} \end{cases} \quad (7)$$

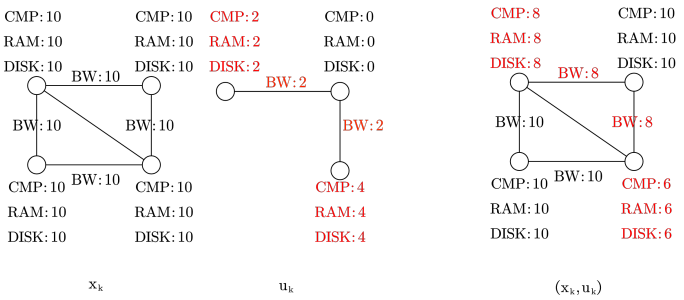

Fig. 3. Residual graph example

IV. Reinforcement Learning Agent Design

We use J^* to denote the optimal total reward function.

$$J^*(x_k) = \max_{u_k \in U_k} \{g_k(x_k, u_k) + \sum_{i=k+1}^{\infty} \alpha^{(i-k)} g(x_i, u_i)\}, \quad (8)$$

where U_k is the action space at time step k. We use u_k^* to denote the optimal action take at time step k to achieve $J^*(x_k)$.

$$u_k^* = \text{argmax}_{u_k \in U_k} \{g_k(x_k, u_k) + \sum_{i=k+1}^{\infty} \alpha^{(i-k)} g(x_i, u_i)\}. \quad (9)$$

Then, we can rewrite the J^* function as follow:

$$J^*(x_k) = \max_{u_k \in U_k} \{g_k(x_k, u_k) + \alpha J^*(x_{k+1})\}, \quad (10)$$

We use $Q^*(x_k, u_k)$ to represent the optimal total reward after the agent chooses action u_k at time step k, which is calculated as follows:

$$Q^*(x_k, u_k) = g_k(x_k, u_k) + \alpha J^*(x_{k+1}). \quad (11)$$

Thus, we can rewrite the J^* function as follows:

$$J^*(x_k) = \max_{u_k \in U_k} Q^*(x_k, u_k). \quad (12)$$

Then we get the new Q^* function as follows:

$$Q^*(x_k, u_k) = g_k(x_k, u_k) + \alpha \max_{u_{k+1} \in U_{k+1}} Q^*(x_k + 1, u_k + 1). \quad (13)$$

Since the state space size for x_k is enormous, it's impossible to maintain a lookup table to log all the optimal Q values for each (x_k, u_k) pair. Thus, we use a Deep Neural Network (DNN) with a Graph Neural Network (GNN) encoder to calculate the approximate Q value, since our input is a graph.

A. Multi-domain Encoder

We use the residual graph which is obtained after the action u_k is applied to represent the input (x_k, u_k). We give an example in Fig. 3.

The agent takes the residual network state and outputs the Q value. We can naively concatenate all the elements' numerical values in the topology to a vector, which is treated as the encoding vector. However, the result vector of this method can be affected by the permutation of nodes and edges. To

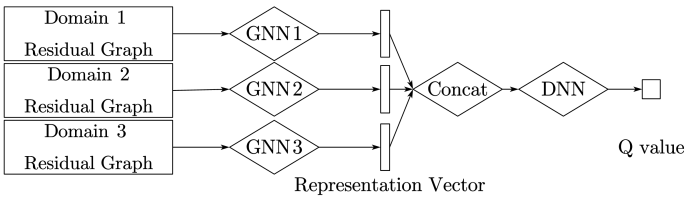

Fig. 4. Encoding Procedure

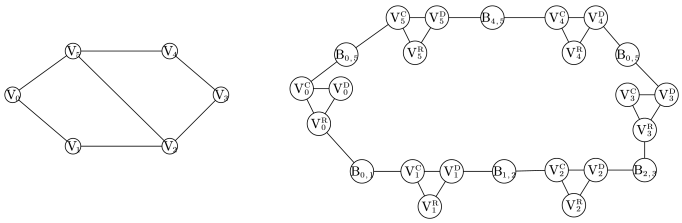

Fig. 5. Topology Transformation

make the result vector more robust to different permutations, we choose a Graph Neural Network (GNN) to process the residual graph information. The encoding procedure is shown in Fig. 4.

We employ NN4G [8] as the online GNN encoder. The output of NN4G is the input of the DNN. The gradient of loss of the Q value's difference back propagates to the NN4G, and the parameter is updated based on the gradient.

For each domain $i \in (1, ..., m)$, there is a $NN4G_i$ model to encode the residual network in domain i. There are four types of resources: computational resources, disk storage, RAM, and bandwidth. Computational resources, disk storage, and RAM are located in the node, and bandwidth is located in the link. To make the representation more uniform, we split each node into three virtual nodes, where each virtual node holds only one type of resource. We also create a virtual node for the edge to hold the bandwidth resource. We give an example of the topology transformation in Fig. 5.

The scale of value on each node should be similar. We use the ratio of the remaining resource to the total resource as the value on each node. For example, if the total resource is 10 units, the current remaining resource is 8 units, and the required resource for this strategy is 5 units, then the value of the node should be $(8-5)/10 = 0.3$. Thus, all values on nodes are from 0 to 1.

B. Action Space

Each domain can choose a path from K candidate paths, or reject the NSR. Thus, each domain has $K+1$ choices, and the action space for m domains is $(K+1)^m$. If any domain rejects the network slice request, then the network slice request will be rejected by the global coordinator.

C. Agent Optimization

Our RL agent concatenates the GNN encoder with a DNN and output a scalar which indicates the $Q(x_k, u_k)$. We need an algorithm to train the agent to predict the Q value more accurately. We choose the Double Deep Q Network (DDQN) algorithm to optimize the RL agent, shown in Algorithm 2.

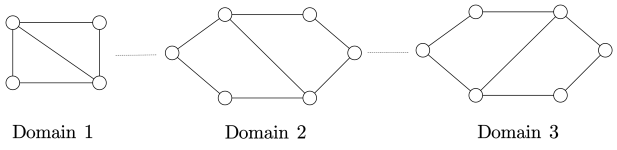

Fig. 6. Experiment Domains

D. Time Complexity

The overall time complexity for inference includes time to find K shortest paths and time for calculating Q values. The time complexity of the first part is $O(KN \log N)$, where K is the number of candidate paths, and N is the number of nodes. The time complexity of the second part is $O(WNL)$, where W is the time complexity for parameter matrix multiplication with the node features, and L is the number of representation layers. The total time complexity is $O(KN \log N + WNL)$.

Algorithm 2 DDQN

Initialize Q model Q, and target Q model $\hat{Q}$
for each episode **do**
  for each time step k **do**
    Given state x_k and action space U_k, choose $u_k \in U_k$ based on Q value (epsilon greedy)
    Obtain reward $r_k = g(x_k, u_k)$ and new state $x_{k+1} = f_k(x_k, u_k)$
    Store (x_k, u_k, r_k, x_{k+1}) in to buffer
    Sample (x_i, u_i, r_i, x_{i+1}) from buffer
    Target $y = r_i + \alpha\hat{Q}(x_{i+1}, u_{i+1})$, where $u_{i+1} =$ argmax $_{u_{i+1} \in U_{i+1}} Q(x_{i+1}, u_{i+1})$
    Update parameters of Q to minimize $(Q(s_i, a_i) - y)^2$
    Every C step reset $\hat{Q} = Q$
  end for
end for

V. Experiments

A. Environment Setup

We select the network topology as shown in Fig. 6, which contains three domains and links with unlimited bandwidth between domains 1 and 2, and domains 2 and 3. We assign four service functions to each domain, and each node in the domain are randomly assigned two of the four functions to maintain. The links within a domain are randomly assigned a bandwidth level from (10 GB/s, 15 GB/s, 20GB/s). The computation resource level of each node is randomly selected from (2 cores, 4 cores, 8 cores), the RAM resource level is selected randomly from (2 GB, 4 GB, 8 GB), and the disk resource level is selected randomly from (64 GB, 128 GB, 256 GB).

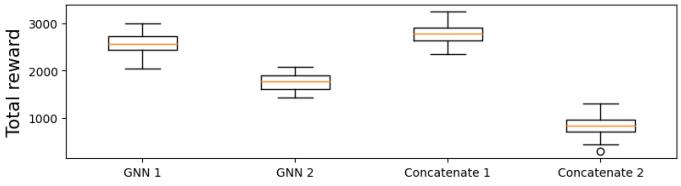

Fig. 7. Comparison between GNN and Concatenation

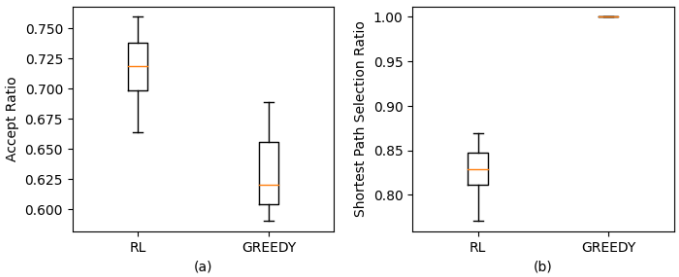

Fig. 8. Comparison of RL vs. Greedy algorithm with respect to (a) slice acceptance ratio, and (b) the probability of selecting the shortest path.

B. Network Slice Request Generation

A network slice request contains three parts for three domains, respectively. We generate random network resource requests for each domain and stitch them together as the network slice request. Specifically, for domain 1, the network resource request randomly contains two of four service functions supported by domain 1. The request contains three service functions for domains 2 and 3. The support resource for the function is selected from (0.5 core, 1 core, 2 cores) for computational resource, (1 GB, 2 GB, 4 GB) for RAM resource, and (8 GB, 16 GB, 32 GB) for disk resource. The request arrival event follows Poisson distribution with $\lambda = 6$. The holding time of request follows an exponential distribution with $\mu = 8$.

C. Performance evaluation

We compared the RL algorithm with the greedy algorithm. In the greedy algorithm, the agent always chooses the shortest path in each domain. If there is no such path, then the agent rejects the request. We evaluated different discount factors and found that a discount factor of 0.4 achieved the best performance.

We compared the performance of the GNN encoder with a method which naively concatenates the numbers in the residual graph into a vector of a fixed size. We show the results in Fig. 7[1]. GNN1 and Concate1 are tested on the topology that is the same as the topology used for training. The topology for GNN2 and Concate2 is the topology obtained by randomly removing one edge from the topology used for training. When the topology is the same for training and testing, the performance of concatenation is better than the GNN. However, even if the topology changes slightly, the performance of the concatenation drops heavily. The GNN encoder drives the RL agent to be more robust when the topology varies.

We compare the accept ratio between the RL agent and the greedy approach in Fig. 8(a). The RL's accpet ratio is higher than the greedy approach. We also calculated the probability of the RL agent choosing the shortest path, shown in Fig. 8(b), and this probability is around 82%.

[1] All of the boxplots in this paper are drawn based on 50 experiments. The circles denote the outlier. The upper bound of the vertical line, the upper bound of the box, the middle horizontal line within the box, the lower bound of the box, and the lower bound of the vertical line represent the minimum, the first quartile, the median, the third quartile and maximum, respectively.

VI. Conclusion

In this paper, we provide an RL algorithm to automatically allocate a multi-domain slice request to achieve a higher long-term reward compared with a greedy algorithm. We also include a GNN encoder to transform the topology information into a representation vector. Compared with vanilla concatenation representation, the GNN encoder makes the agent more robust when the topology changes. Furthermore, each domain has its own encoder. Thus, there is no leak of the domain's structure information. If the topology is large, then the overhead associated with the transfer of topology information is expensive. By only sharing the representation vector, the bandwidth requirement can be reduced. Furthermore, we provide a topology transformation to unify the representation of nodes and edges and reduce the numerical influence in inference and training.

Acknowledgment

This work was supported in part by the National Science Foundation under Grant No. CNS-2008856.

References

[1] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5g network slice broker," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, 2016.

[2] Z. Wu and J. P. Jue, "A reinforcement learning-based routing strategy for elastic network slices," in *ICC 2022 - IEEE International Conference on Communications*, pp. 5505–5510, 2022.

[3] Z. Wu, G. Ishigaki, R. Gour, and J. P. Jue, "A reinforcement learning-based admission control strategy for elastic network slices," in *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 01–06, 2021.

[4] R. Li, Z. Zhao, Q. Sun, C.-L. I, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[5] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "Rl-nsb: Reinforcement learning-based 5g network slice broker," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1543–1557, 2019.

[6] T. Taleb, I. Afolabi, K. Samdanis, and F. Z. Yousaf, "On multi-domain network slicing orchestration architecture and federated resource control," *IEEE Network*, vol. 33, no. 5, pp. 242–252, 2019.

[7] S. Keerthi and E. Gilbert, "An existence theorem for discrete-time infinite-horizon optimal control problems," *IEEE Transactions on Automatic Control*, vol. 30, no. 9, pp. 907–909, 1985.

[8] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.