Amirkabir University of Technology

A novel reinforcement learning algorithm for virtual network embedding

Ali Zamani

30 June 2018

Contents

1 Introduction

Network Function Virtualization (NFV) aims to simplify service deployment using Virtual Network Functions (VNFs). Service deployment involves placement of VNFs and in-sequence routing of traffic flows through VNFs comprising a Service Chain (SC). The joint VNF placement and traffic routing is called SC mapping. In a Wide-Area Network (WAN), where several traffic flows, generated by many distributed node pairs, require the same SC; a single instance (or occurrence) of that SC might not be enough. SC mapping with multiple SC instances for same SC is a very complex problem, since sequential traversal of VNFs has to be maintained while accounting for traffic flows in various directions [1].

1.1 Network Function Virtualization

Traditionally, communication networks have deployed network services through proprietary hardware appliances (e.g., network functions such as firewalls, NAT, etc.) which are statically configured. With rapid evolution of applications, networks require agile and scalable service deployment.

Network Function Virtualization (NFV) [2] offers a solution for an agile service deployment. NFV envisions traditional hardware functionality as software modules called Virtual Network Functions (VNFs). VNFs can be run on commercial-off-the-shelf hardware such as servers and switches in datacenters (DCs), making service deployment agile and scalable.

1.2 Service Chain

When several network functions are configured to provide a service, we have a "Service Chain". The term "service chain" is used "to describe the deployment of such functions, and the network operator's process of specifying an ordered list of service functions that should be applied to a deterministic set of traffic flows". So, a "Service Chain" (SC) specifies a set of network functions configured in a specific order. With NFV, we can form SCs where VNFs are configured in a specific sequence that minimizes the bandwidth usage in the network [3].

In Table 1 we show some well-known Service Chains (SCs).

Service Chains	Chained VNFs
Web Service	NAT-FW-TM-WOC-IDPS
VoIP	NAT-FW-TM-FW-NAT
Video Streaming	NAT-FW-TM-VOC-IDPS
Online Gaming	NAT-FW-VOC-WOC-IDPS

Table 1: Service Chain Requirements; Network Address Translator (NAT), Firewall (FW), Traffic Shaper (TM), WAN Optimization Controller (WOC), Intrusion Detection and Prevention System (IDPS), Video Optimization Controller (VOC).

1.3 Service Chain Mapping Issues

Unfortunately, since VNFs in a single SC may need to be traversed by several distinct traffic flows (i.e., flows requested by multiple geographically-distributed node pairs) in a specific sequence, it becomes difficult to improve network resource utilization.

For example, consider Figure 1(a) and 1(b), where three traffic requests r_1 (from node 4 to 13), r_2 (from node 6 to 3), and r_3 (from node 14 to 1) demand SC c_1 composed of VNF1, VNF2, and VNF3 (to be traversed in this order VNF1$\rightarrow$ VNF2 $\rightarrow$ VNF3).

In Figure 1(a), if we consider only one mapping occurrence (or instance) for SC c_1, then some traffic flows (in our example, r_3 and r_2) will be ineffectively routed over long paths. Instead, as shown in Figure 1(b), if we use two SC instances for the same SC, we can improve network resource utilization, at the expense of a larger number of VNFs to be deployed (or replicated) in the network to serve the same SC. This results in a more complex problem when, in a Wide-Area Network (WAN), a large number of distributed node pairs generate traffic flows, creating heavy traffic demands. Our objective in this work is to reduce the network resource consumption for a WAN with heavy traffic demands.

So the question is: how many SC instances for the same SC are required for optimal network resource utilization?

A possible (trivial) solution to the problem of SC mapping in case of multiple node pairs requiring the same SC is to use one single instance that would most likely lead to host SCs at a single node (e.g., a DC) which is centrally located in the network. However, traffic flows may have to take

long paths to reach the node hosting the SC, which will result in a high network resource consumption.

The other extreme case would be to use a distinct SC mapping per node pair (in other words, the number of SC instances is equal to the number of traffic node pairs). Now, we can achieve optimal network resource utilization as each node pair will use an SC effectively mapped along a shortest path in the network. However, this approach will increase the network orchestration overhead and increase capital expenditure, as there will be a large number of replicated VNF instances across nodes. To reduce excessive VNF replication, we bound the maximum number of nodes hosting VNFs.

Intuitively, the number of SC instances for a good solution will be a value between these two extremes. This solution will minimize the network resource utilization while not excessively increasing the number of nodes hosting VNFs [4].

1.4 Reinforcement algorithm

The purpose of virtual network embedding is to map virtual networks to a shared physical network while providing the requests with adequate computing and bandwidth resources. However, the virtual network embedding problem has been proved to be NP-hard. As a result, a large number of heuris- tic algorithms have been proposed, but most of them rely on artificial rules to rank nodes or make mapping decisions. parameters in these algorithms are always fixed and cannot be op- timized, making the embedding decisions sub-optimally. On the other hand, in prior works, the information about substrate net- work and the knowledge about virtual network embedding hidden in historical network request data have always been overlooked.

In recent years, big data, machine learning and artificial intel- ligence have exciting breakthroughs achieving state of the art re- sults such as natural language understanding and object detection. Machine learning algorithms process a large amount of data col- lected during a period and automatically learn the statistical in- formation from the data to give classification or prediction.Rein- forcement learning, as a widely-used technique in machine learn- ing, has shown a great potential in dealing with complex tasks, e.g., game of go, or complicated control tasks such as auto-driving and video games. The goal of a reinforcement learning system (or an agent) is to learn better policies for sequential decision mak- ing problems with an optimal cumulative future reward signal.

Notations	Descriptions
G^s	Substrate network
N^s	Nodes of substrate network
L^s	Links of substrate network
A_N^s	Node attribute of substrate network
A_L^s	Link attribute of substrate network
G^v	Virtual network of a certain virtual request
N^v	Nodes of a virtual network
L^v	Links of a virtual network
A_N^v	Constraints of substrate nodes
A_L^v	Constraints of substrate nodes

Table 2: Notations Descriptions

In this paper, we introduce reinforcement learning into the problem of virtual network embedding to optimize the node map- ping process. Similar to earlier works, our work is based on the assumption that all network requests follow an invariable dis- tribution. We divide our network request data into a training set and a testing set, to train our reinforcement learning agent (RLA) and evaluate its performance respectively.We devise an artificial neural network called policy network as the RLA, which observes the status of substrate network and outputs node mapping results. We train the policy network with historical network request data using policy gradient through back propagation. An exploration strategy is applied in the training stage to find better solutions, and a greedy strategy is applied in evaluation to fully evaluate the ef- fectiveness of the RLA. Extensive simulations show that the RLA is able to extract knowledge from historical data and generalize it to incoming requests.

2 Network modeling

In this section, we present a network model and formulate the virtual network embedding problem with description of its com- ponents. The notations used in this section are shown in table 2

Fig.2 shows the mapping process of two different virtual net- work requests. A substrate network is represented as an undi- rected graph $G^s = (N^s, L^s, A_N^s, A_L^s)$,where N^s denotes the set of all the substrate nodes, L^s

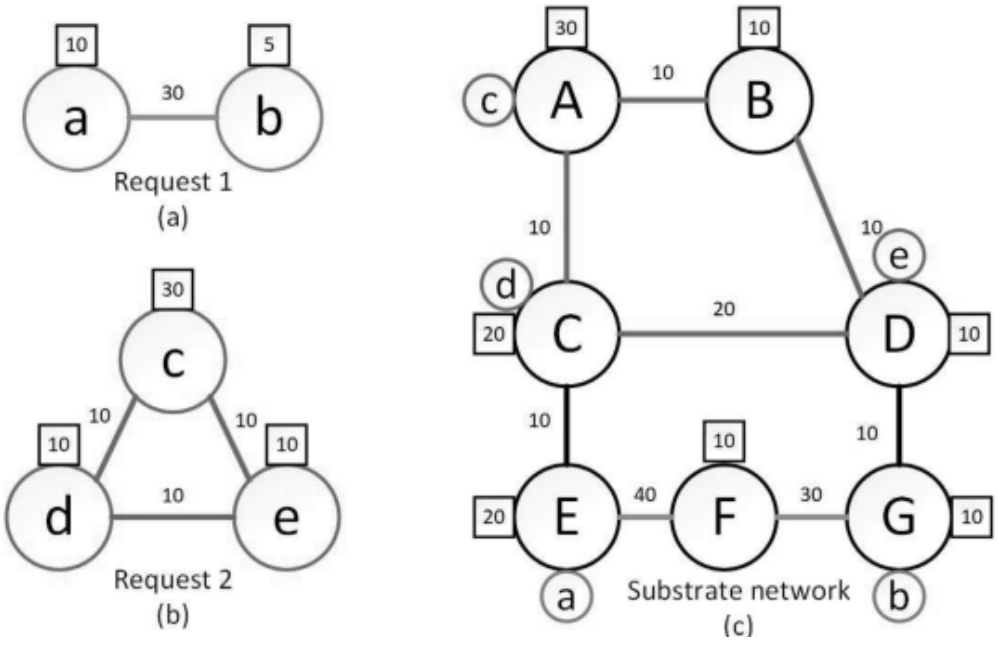

Figure 1: An example of virtual network embedding.

denotes the set of all the substrate links, A_N^s and A_L^s stand for the attributes of substrate nodes and links respectively.

Fig.2 (c)shows an example of a substrate network, where a circle denotes a substrate node, and a line connecting two circles denotes a substrate link. The number in a square box denotes the CPU (computing) capacity of that node, and the number next to a substrate link denotes the bandwidth of that link. Similarly, we also use an undirected graph $G^v = (N^v, L^v, A_N^v, A_L^v)$ to describe a virtual network request, where N^v denotes the set of all the virtual nodes in the request,L^v de- notes the set of all the virtual links in the request,C_N^v and C_L^v stand for the constrains of virtual nodes and links respectively.

Fig. 2(a) and (b) shows two different virtual re- quests. Additionally, we use t to denote the arrival time of a virtual request, and use t d to denote the duration of the virtual request.

When a virtual request arrives, the objective is to find a solu- tion to allocate different kinds of resources in the substrate net- work to the re- quest while satisfying the requirements of the re- quest. If such a solution

exists, then the mapping process will be executed, and the request will be accepted. Otherwise the request will be rejected or delayed. The virtual network embedding pro- cess can be formulated as a mapping M from G^v to $G^s : G^v(N^v, L^v) \Rightarrow G^s(N', L')$

The main goal of virtual network embedding is to accept as many requests as possible to achieve maximum revenue for an ISP, when the arrival of virtual network requests follows an unknown distribution of time and unknown resource requirements. Conse- quently, the embedding algorithm must produce efficient mapping decisions within an acceptable period.As shown in Fig.2, virtual nodes a and b in request 1 are mapped to substrate nodes E and G respectively, and virtual nodes c, d and e in request 2 are mapped to substrate nodes A, C and D respectively. Note that the embed- ding result of request 1 is not optimal. For example, the cost of bandwidth in the substrate network can be significantly reduced by moving a to F. To determine the performance of embedding algorithms, most works use certain metrics such as a long-term average revenue, a long-term acceptance ratio, and a long-term revenue to cost ra- tio. The revenue measures the profit of an ISP for accepting a cer- tain virtual request, and it depends on the amount of requested resources and the duration of it. Similar to the earlier works presented in, we define the revenue of accepting a virtual net- work request as follows:

$$R(G^v, t, t_d) = t_d \times [\sum_{n^v \in N^v} CPU(n^v) + \sum_{l^v \in L^v} BW(l^v)] \quad (1)$$

where CPU(n^v) and BW(l^v) denote the computing resource that a virtual node n^v requires and the bandwidth resource that a vir- tual link l^v requires respectively. Same as before, N^v denotes the set of all the virtual nodes in the request, and L^v denotes the set of all the virtual links in the request. As shown in the formula, the revenue for accepting a virtual request is proportional to the amount of requested resources and the duration of it.

When accepting a virtual request, the CPU consumption is al- ways fixed, but the bandwidth consumption may vary depending on the performance of embedding algorithm. Mapping a virtual link to a shorter path in the substrate network will result in less consumption of bandwidth resource. Therefore, a cost function is formulated to measure the total amount of bandwidth resources assigned to a virtual request, defined as follows:

$$C(G^v, t, t_d) = t_d \times [\sum_{l^v \in L^v} \sum_{l^s \in p'_{l^v}} BW(l^v)] \quad (2)$$

where p'_{l^v} denotes the set of substrate links where virtual link l^v is embedded. $C(G^v, t, t_d)$ computes the actual consumption of bandwidth resource for embedding request G^v. we use a long-term av- erage revenue to evaluate the overall performance of our embed- ding method defined as:

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T} R(G^v, t, t_d)}{T} \tag{3}$$

where T is the time elapsed. A higher long-term average rev- enue leads to a higher profit for the ISP. Another important metric to evaluate the mapping algorithm is a long-term acceptance ratio, which means the ratio of accepted requests to the total number of requests arrived. A higher long-term acceptance ratio means the proposed algorithm manages to serve more virtual requests. Finally, a better utilization of substrate network resources would lead to a high long-term average revenue with compara- tively low cost of substrate network. The long-term revenue to cost ratio, defined as follows, measures the utilization of substrate net- work resources:

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T} R(G^v, t, t_d)}{\sum_{t=0}^{T} C(G^v, t, t_d)} \tag{4}$$

A higher long-term revenue to cost ratio shows that the pro- posed algorithm is able to generate more profit with a compara- tively less cost to network resources. We will use these metrics mentioned above to evaluate the per- formance of our embedding method in the following sections.

3 Embedding algorithm

In this section, we present the details of the proposed policy network based reinforcement learning algorithm. Specifically, we apply the reinforcement learning agent in the node mapping stage to derive the probabilities of choosing nodes. The agent takes a feature matrix extracted from the substrate network as input, and makes decisions based on a policy network which is trained from historical data.

3.1 Feature extraction

Every substrate node has several attributes, such as CPU ca- pacity and the total amount of bandwidth of the adjacent links. A thorough knowledge of

substrate network is crucial for the re- inforcement learning agent to establish a basic understanding of its state and generate efficient mapping. To facilitate the agent to choose the substrate nodes, we need to extract features of each substrate node and use them as input to the policy network.

We extract four features for each substrate node listed as fol- lows:

- Computing capacity (CPU): The CPU capacity of a substrate node n S has a large impact on its availability. The substrate nodes with a higher computing capacity are likely to host more virtual nodes.

- Degree (DEG): The degree of a substrate node n S indicates the number of links connected to it. A substrate node with more adjacent links is more likely to find paths to other substrate nodes.

- Sum of bandwidth ($SUM^{(BW)}$): Every substrate node is con- nected to a set of links. A substrate node n S has a sum of band- width resources of its neighboring links:

$$SUM^{(BW)}(n^s) = \sum_{l^s \in L(n^s)} BW(l^s) \tag{5}$$

  where $L(n^s)$ is the neighboring links of n^s and $BW(l^s)$ is the bandwidth resource of a substrate link l^s . When a substrate node has access to more bandwidth, mapping a virtual node to it may lead to better link mapping options.

- Average distance to other host nodes $AVG^{(DST)}$: When mapping a virtual node, we also take into consideration the positions where other virtual nodes in the same request are mapped. By choosing a sub- strate node close to those already mapped, the cost of substrate link bandwidth can be reduced. We mea- sure the distance between two substrate nodes in terms of the number of links along the shortest path. The shortest path is computed following the FloydWarshall al- gorithm. We take an average of the distance from a substrate node n^s to another host nodes $N^s$for the same request:

$$AVG^{DST}(n^s) = \frac{\sum_{n^s \in N^s} DST(n^s, n^s)}{|N^s| + 1} \tag{6}$$

  where $DST(n^s, n^s)$is the distance from node n^s to node n^s.

In fact, the features that we can extract from the substrate nodes would be far more than listed above. More features would bring more information about the substrate network which leads to a better performance of the learning agent. It should be noted that extracting more features from the substrate network adds complexity in computation. After extracting the features of the kth substrate node n_k^s, we take their normalized values and concatenate them into a feature vector v_k :

$$v_k = (CPU(n_k^s), DEG(n_k^s), SUM^{BW}(n_k^s), AVG^{DST}(n_k^s))^T \tag{7}$$

The purpose of normalization is to accelerate the training pro- cess and enable the agent to converge quickly. We concatenate all feature vectors of substrate nodes to produce a feature matrix M_f where each row is a feature vector of a certain substrate node:

$$M_f = (v_1, v_2, ..., v_{|N^s|})^T \tag{8}$$

The feature matrix serves as an input to the learning agent. The feature matrix is updated along with the changing substrate net- work from time to time.

3.2 Policy network

Our work is based on the assumption that all network requests follow an invariable distribution. As a result, if an embedding al- gorithm works well for historical requests data, it is likely to have the same performance for on-line virtual requests. In this work, we implemented an artificial neural network called policy network as the learning agent. It takes the feature matrix as input and out- puts the probabilities of mapping virtual nodes to substrate nodes. Then we train the policy network and optimize its performance on historical virtual requests data in order to get comparative perfor- mance for on-line virtual requests.

For simplicity, we build a simple policy network with basic ele- ments of an artificial neural network as shown in Fig. 3.2 . The policy network contains an input layer, a convolutional layer, a softmax layer and finally a node filter. For each virtual node that requires a mapping, we use the policy network to choose a substrate node for it.

At the input layer, we compute the feature matrix and deliver it to the policy network. The policy network then passes the in- put feature matrix into a convolutional layer with one convolution kernel, where the policy

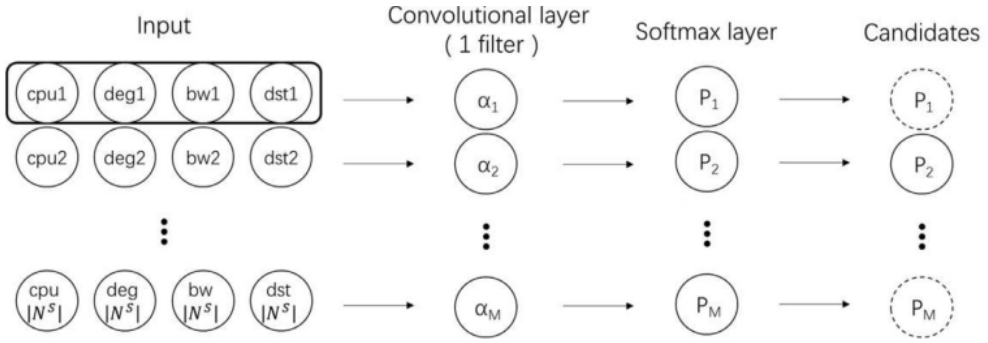

Figure 2: Policy network.

network evaluates the resources of each substrate node. The convolutional layer performs a convolution op- eration on the input to produce a vector representing the available resources of each node:

$$h_k^c = w.v_k + b \tag{9}$$

where h_k^c is the kth output of the convolutional layer, is the con- volution kernel weight vector, and b is bias.

Then the vector is transmitted to a softmax layer to produce a probability for each node which indicates the likelihood of yielding a better result if mapping a virtual node to it. For the k th node, the probability p k is computed as:

$$p_k = \frac{\exp(h_k^c)}{\sum_i \exp(h_k^c)} \tag{10}$$

The softmax function is a generalization of the logistic regres- sion. It turns the n-dimensional vector into real values between 0 and 1 that add up to 1. The output of the softmax function indi- cates a probability distribution over n different possible mappings. Some of the nodes are not able to host the virtual node in con- cern because they do not have enough computing resources. We add a filter to choose a set of candidate nodes with enough CPU capacities.

3.3 Training and testing

We first randomly initialize the parameters in the policy net- work, and train it for several epochs. For every virtual node in each iteration, a feature matrix

is extracted from the substrate network which serve as input to the policy network. The policy network outputs a set of available substrate nodes as well as a probability for each node. The probability of each node represents the likeli- hood that mapping a virtual node to it will yield a better result. In the training stage, we cannot simply select the node with a max- imal probability as the host because that the model is randomly initialized, which means the output could be biased and better so- lutions might exist. In other words, we need to reach a balance between the exploration of better solutions and the exploitation of current model. To this end, we generate a sample from the set of available substrate nodes according to their probability distribution that the policy network outputs, and select a node as the host. We repeat this process until all the virtual nodes in a virtual request are assigned and proceed to link mapping. If no substrate node is available, the mapping fails due to a lack of resources. For link mapping, we apply a breadth-first search to find the shortest paths between each pair of nodes.

In supervised learning, each piece of data in the training set corresponds to a label indicating the desired output of the model. With each output from model and the corresponding label, a loss value is computed which measures the deviation between them. The loss value for each piece of data in the training set sums up to an aggregated loss value, and the training stage aims to mini- mize the aggregated loss value. However, in reinforcement learn- ing tasks such as virtual network embedding, data in the training set does not have corresponding labels. The learning agent relies on reward signals to know if it is working properly. A big reward signal informs the learning agent that its current action is effective and should be continued. A small reward signal or even a nega- tive reward signal shows that the current action is erroneous and should be adjusted. The choice of reward is critical in reinforce- ment learning as it directly influences the training process and determines the final policy. Here, we use the revenue to cost ratio of a single virtual request as the reward for every virtual node in this request because this metric represents the utilization efficiency of the substrate resources. Then we apply policy gradient method to train the policy network. The actual implementation of the proposed algorithm is non- trivial since we cannot provide each output with a label. As a re- sult, we temporarily consider every decision that the agent makes to be correct by introducing a hand-crafted label into our policy network. Assume that we choose the i th node, then the hand- crafted label in policy network would be a vector y filled with zeros except the i th position which is one. Then we calculate the

cross- entropy loss:

$$L(y,p) = -\sum_{i} y_i log(p_i) \tag{11}$$

where y_i and p_i are the i th element of hand-crafted label and the output of policy network respectively. We use backpropagation to compute the gradients of parameters in the policy network. Since we use hand-crafted label , we stack the gradients g_f rather than applying them immediately. If our algorithm fails to embed a vir- tual request, the corresponding stacked gradients will be aborted since we cannot determine the reward signal. If a virtual request has been successfully mapped, we compute its revenue to cost ra- tio as a reward r . Then we multiply the stacked gradients by using the reward and an adjustable learning rate α to achieve the final gradients:

$$g = \alpha.r.g_f \tag{12}$$

The learning rate α is introduced to control the magnitude of gradients and the computation speed of training. If the gradients are too large, the model becomes unstable and may not improve through the training process. On the other hand, too small gra- dients make training extremely slow. Therefore the learning rate needs to be tuned carefully. It can be observed from Eq. 12 larger rewards make the corresponding gradients more significant than small ones. As a result, the choices that lead to larger rewards have larger impact on the learning agent, making it more prone to make similar decisions. When we stack a batch of gradients, we apply them to parameters and update the policy network. There are two reasons for batch updating—one is that parameter updat- ing normally takes a long time, but doing that in batches speeds up this process. Another reason is that batch updating averages over the gradients and is more stable. The training process is shown in Algorithm **??** . Lines 7–10 show node mapping stage where we com- pute the gradients in line 10, lines 11–13 show the link mapping stage. In the testing stage, we apply a greedy strategy where we di- rectly choose the node with the highest probability as the host. The testing algorithm is shown in Algorithm **??** .

Parameter	Value
G	It is shown in Figure 4.1
N^s	All nodes can be NFV node
$epoch_num$	$Numberofepochs : 100$
$batch_Size$	$Numberofbatches : 4$
$node_features$	$Numberofnodes'feature : 4$
$node_features$	$1e-7$
$chains_num$	$Numberofchains : 100$
fun_num	$Numberoffunctionsineachchain : 3$
$chain_ban$	$Maximumrequieredbandwithofeachchain : 90$
cpu_range	$Maximumrequieredcpucoreofeachvertiualnetwork : 3$
max_node_cap	$Maximumcpucoreofeachnode : 60$
min_node_cap	$Minimumcpucoreofeachnode : 30$
td_mean	$Meanrequieredtimeofeachchain : 100$
td_std	$Standarddeviationofrequieredtimeofeachchain : 10$

Table 3: Simulation Parameters

4 Simulation

4.1 Simulation Parameters Declaration

We tested our optimization process on a 14-node NSFNET WAN topology as shown in Figure 4.1. We generated 100 chains with up to 3 vnfs randomly. All of nodes can be NFV nodes. The link capacities are 100bps. Each traffic flow demand is up to 90 bps. Compute resource (CPU) at each NFV node is 100 cores per node. All other simulation parameters are shown in Table 3. All parameters can be change in Inputeconstants.py

4.1.1 Results

Results of the simulation are shown in Fig. 4.1.1. In Fig. 4.1.1, loss and cost are decreasing and the reward is increasing but revenue is so wiggly. For future works, we can consider the source and destination for each chain and also extract more features for each node and use more complex policy network.

5 Conclusion

I simulate this paper with TensorFlow and python is used. I also consider NFS-net network and 100 chains are generated randomly for training reinforcement learning algorithm. Eventually, I showed results.

Algorithm 1 Training process.

Input: Number of epochs, *numEpoch*; Learning rate, α; Training set;

Output: Trained parameters in policy network;

Initialize all the parameters in policy network;
while *iteration* $<$ *numEpoch* **do**
  for *req* $\in$ *trainingSet* **do**
    counter=0;
    for *node* $\in$ *req* **do**
      M_f=getFeatureMatrix();
      p=policyNet.getOutput(M_f) //Get the probability distribution from policy network;
      host=sample(p) //Sample from the probability distribution to choose a node as host;
      computeGradient(host);
    end for
    if isMapped($\forall$ node $\in$ req) **then**
      bfsLinkMap(req);
    end if
    if isMapped($\forall$ node $\in$ req, $\forall$ link $\in$ req) **then**
      reward=revToCost(req) //Compute revenue to cost ratio;
      multiplyGradient(reward, α) //Compute the final gradients;
    else
      clear the stacked gradients;
    end if
    ++counter;
    if counter reach the batch size **then**
      apply gradients to parameters;
      counter=0;
    end if
  end for
  ++iteration;
end while

Figure 3: Training process.

Algorithm 2 Testing process.

Input: testing set;
Output: long-term average revenue, acceptance ratio, long-term revenue to cost ratio;

```
Initialize all the parameters in policy network;
for req ∈ testSet do
    for node ∈ req do
        M_f=getFeatureMatrix();
        host=maxProbablity(p) //Greedy strategy;
    end for
    bfsLinkMap(req);
    if isMapped(∀ node ∈ req, ∀ link ∈ req) then
        signal(SUCCESS);
    end if
end for
```

Figure 4: Testing process.

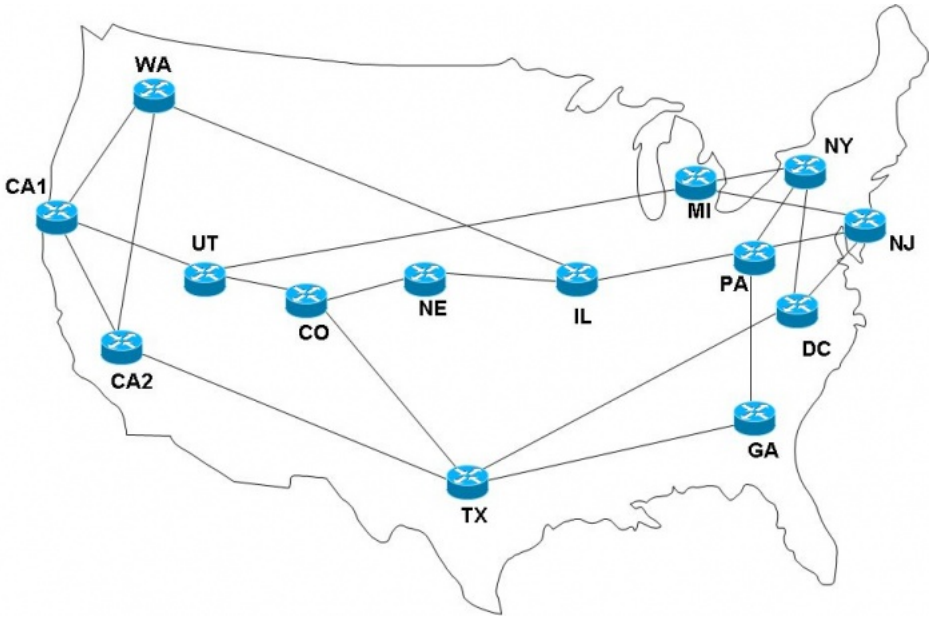

Figure 5: 14-node NSFNET WAN topology.

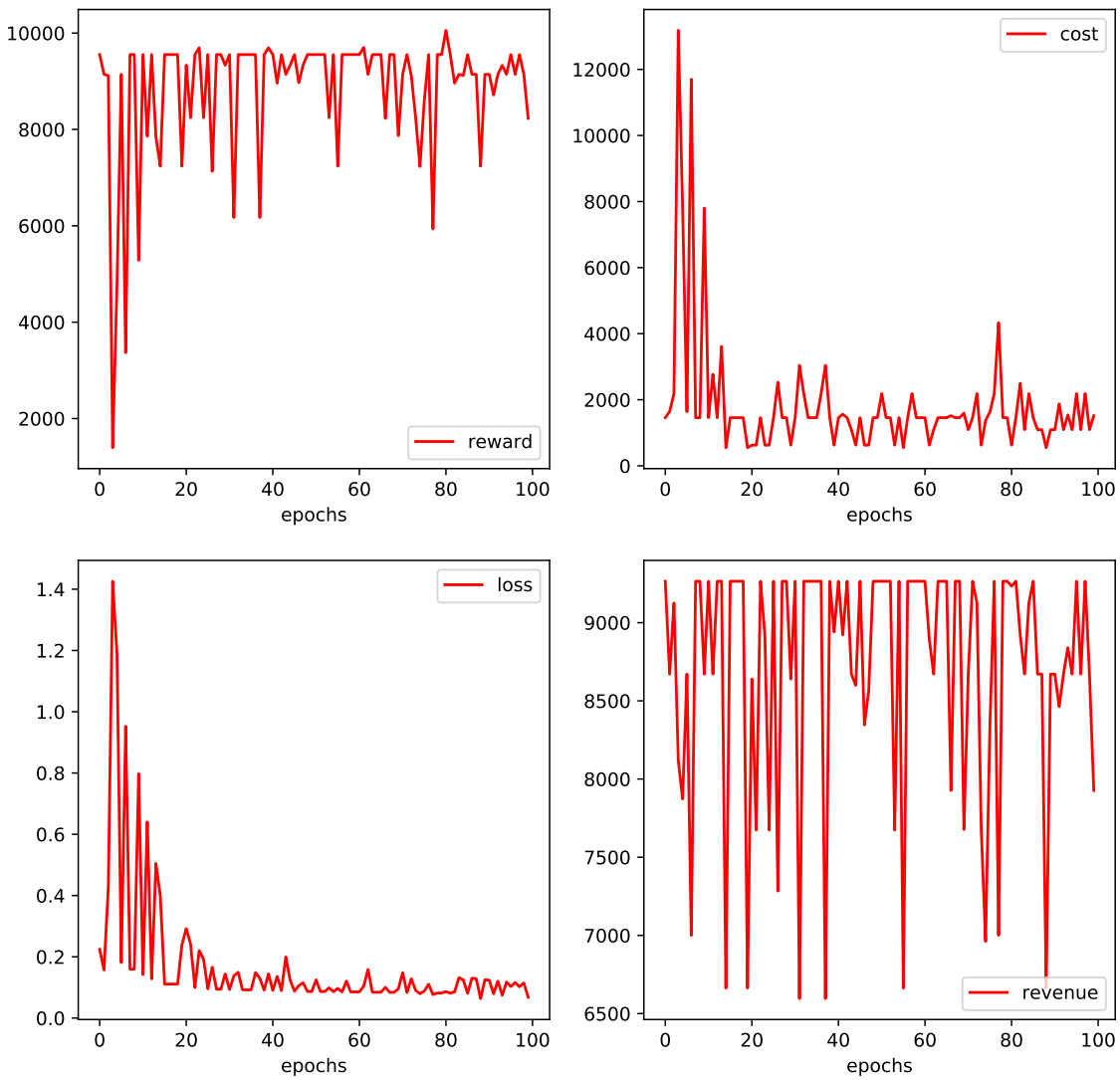

Figure 6: Results.

References

[1] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5g," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 84–91, April 2016.

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

[3] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research directions in network service chaining," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, Nov 2013, pp. 1–7.

[4] A. Gupta, B. Jaumard, M. Tornatore, and B. Mukherjee, "A scalable approach for service chain mapping with multiple sc instances in a wide-area network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 529–541, March 2018.