A Continuous-Decision Virtual Network Embedding Scheme Relying on Reinforcement Learning

Haipeng Yao, *Member, IEEE*, Sihan Ma, Jingjing Wang, *Member, IEEE*, Peiying Zhang, Chunxiao Jiang, *Senior Member, IEEE*, and Song Guo

***Abstract*—Network Virtualization (NV) techniques allow multiple virtual network requests to beneficially share resources on the same substrate network, such as node computational resources and link bandwidth. As the most famous family member of NV techniques, virtual network embedding is capable of efficiently allocating the limited network resources to the users on the same substrate network. However, traditional heuristic virtual network embedding algorithms generally follow a static operating mechanism, which cannot adapt well to the dynamic network structures and environments, resulting in inferior nodes ranking and embedding strategies. Some reinforcement learning aided embedding algorithms have been conceived to dynamically update the decision-making strategies, while the node embedding of the same request is discretized and its continuity is ignored. To address this problem, a Continuous-Decision virtual network embedding scheme relying on Reinforcement Learning (CDRL) is proposed in our paper, which regards the node embedding of the same request as a time-series problem formulated by the classic seq2seq model. Moreover, two traditional heuristic embedding algorithms as well as the classic reinforcement learning aided embedding algorithm are used for benchmarking our prpposed CDRL algorithm. Finally, simulation results show that our proposed algorithm is superior to the other three algorithms in terms of long-term average revenue, revenue to cost and acceptance ratio.**

***Index Terms*—Reinforcement learning, virtual network embedding, continuous decision, time-series, seq2seq.**

Manuscript received June 30, 2019; revised November 11, 2019 and January 29, 2020; accepted January 31, 2020. Date of publication February 4, 2020; date of current version June 10, 2020. This work was supported in part by the Director Foundation Project of National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data, in part by the National Key Research and Development Plan under Grant 018YFB1800805, in part by the Future Intelligent Networking and Intelligent Transportation Joint Laboratory (BUPT-CTTIC), in part by the "Fundamental Research Funds for the Central Universities" of China University of Petroleum (East China) under Grant 18CX02139A, and in part by the National Nature Science Foundation of China under Grant 61922050. The work of Dr. J. Wang was supported by the Shuimu Tsinghua Scholar Program. The associate editor coordinating the review of this article and approving it for publication was N. Zincir-Heywood. *(Corresponding author: Haipeng Yao.)*

Haipeng Yao and Sihan Ma are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: yaohaipeng@bupt.edu.cn; 1440432617@qq.com).

Jingjing Wang is with the Department of Electronic Engineering, Tsinghua University, Beijing 100084, China (e-mail: chinaeephd@gmail.com).

Peiying Zhang is with the College of Computer and Communication Engineering, China University of Petroleum (East China), Qingdao 266580, China.

Chunxiao Jiang is with the School of Information Science and Technology, Tsinghua University, Beijing 100084, China, and also with the Beijing National Research Center for Information Science and Technology, Beijing 100084, China (e-mail: jchx@tsinghua.edu.cn).

Song Guo is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: cssongguo@comp.polyu.edu.hk).

Digital Object Identifier 10.1109/TNSM.2020.2971543

I. Introduction

A. Motivations

NETWORK Virtualization (NV) has attracted more and more attention [1]–[3], which allows multiple virtual networks to share limited resources on the same substrate network. Specifically, a new business model Infrastructure as a Service (IaaS) is enabled by the Internet Service Providers (ISPs), which hosts multiple network services on its infrastructure. Therefore, embedding decision-making has become one of the important tasks for ISP, because an optimal embedding decision can provide services for more virtual networks and make resource utilization more effective. For the sake of simplification, a virtual network can be modeled as multiple virtual nodes, which are connected by virtual links. The purpose of virtual network embedding is to map virtual networks to shared physical networks while providing sufficient computation and bandwidth resources for requests. Virtual Network Embedding (VNE) has played a critical role in network resource allocation problems, which is considered as the main implementation method of NV [4], [5]. To elaborate a little further, SPs embed virtual network requests into the substrate network and offer the corresponding services for the sake of obtaining revenue. Additionally, the substrate network is comprised with substrate nodes with computational capability as well as substrate links with certain bandwidth. By contrast, the virtual network contains nodes having requirement of computational resources and links with requirement of transmission bandwidth.

Specifically, VNE process can be divided into two phases, i.e., node embedding and link embedding. Moreover, different optimization objectives and constraints in different contexts may formulate different VNE problems, which have been proved to be NP-Hard [5]. Liu and Wu [6] concluded that solving VNE problems mainly relied on direct solutions and heuristic solutions. However, direct solutions such as Integer Linear Programming (ILP), aim at finding the optimal embedding results yet with high computational complexity, which are not suitable for large-scale networks. Additionally, most of the heuristic algorithms [7]–[10] are based on static decision-making process, where network parameters cannot be adaptively optimized corresponding to dynamic network states.

With the development of artificial intelligence and machine learning [11], more and more intelligent algorithms have been taken into account for solving the VNE problems. In comparison to the heuristic algorithms, machine learning [12] algorithms can intelligently adjust network parameters relying on available data and make predictions for the future. Moreover, as one famous member of machine learning family, reinforcement learning [13], [14] methods are characterized with self-adaptivity in which the agent interacts with the environment to perform specific actions. Combining with the deep neural network [15], Deep Reinforcement Learning (DRL) algorithms are proposed for extracting and analyzing information of the substrate network relying on the perception of the neural network. Inspired by these, in this paper, the seq2seq model[1] is applied to formulate the node embedding process. Furthermore, a Continuous-Decision VNE scheme relying on Reinforcement Learning (CDRL) algorithm is proposed for solving the VNE problem and for optimizing their performance.

B. Related Work

In recent years, VNE algorithms have been widely studied in the literatures. As mentioned above, a VNE problem generally consists of two stages, i.e., node embedding and link embedding. In some works, these two processes were performed independently [7]–[9]. Specifically, the substrate nodes, first of all, were sorted according to their available resources, which determined their availability as well as their embedding priority. With the aid of some specific node embedding algorithms, virtual nodes were successfully embedded to the substrate network, followed by the link embedding process, where link embedding algorithms were implemented to find the best path with the optimal bandwidth. In [7], Yu *et al.* proposed a path splitting and migration strategy, where virtual links were embedded to multiple substrate links relying on the multi-commodity flow algorithm. Specifically, this method really saved bandwidth and improved network performances. However, this algorithm may result in a resource fragmentation problem and may be unpractical to apply to large-scale virtual networks. The closest node embedding algorithm was proposed by Razzaq and Rathore in [8], which aimed at embedding the virtual nodes to the substrate network in terms of their distance. In [9], Cheng *et al.* applied the markov random walk model to rank nodes in virtual networks in which both the resource availability of each substrate node and its adjacent links were considered. Moreover, Zhang *et al.* [18] proposed the VNE-DCC algorithm, which considered the topological attributes of its neighborhood nodes including the node degree and the clustering coefficient information. Then a five-node ranking method was raised by Zhang *et al.* to evaluate the importance of substrate nodes [19], which avoided unbalanced embedding results. In [20], Zhang *et al.* proposed a 3-D resource constraint model based on the computing, communication and storage. By contrast, in some works, these two processes were performed in an integrated stage [10], [21]–[23]. Chowdhury *et al.* [21] introduced the mixed integer program for solving the embeding problem and utilized the ILP relaxation as well as the rouding algorithm to calculate embedding result. Moreover, Lischka and Karl [22] proposed vnmFlib, an improved backtracking, which solved the subgraph isomorphism problem in a single stage. In [10], Hesselbach *et al.* used the strategy based on the path algebra to solve the VNE problem.

However, most of above-mentioned algorithms were based on artificial rules, which cannot automatically optimize network parameters and may lead to suboptimal embedding results. With the aid of self-learning and self-adaptivity capability of the machine learning algorithms, Haeri and Trajkovic [24] utilized a markov decision model to formulate the VNE problems, where virtual nodes were embeded relying on the Monte Carlo tree search algorithm. Furthermore, inspired by Mijumbi *et al.* [25] who introduced the neural network for solving the VNE problems, DRL algorithms were applied to NV in [26], [27]. According to these achievements, Yao *et al.* [28] introduced a reinforcement learning approach RLVNE to tackle with VNE problems and designed a policy network to make node-embedding decisions. Furthermore, a Reinforcement learning based Dynamic Attribute Matrix (RDAM) algorithm was presented by Yao *et al.* [29], which applied spectral analysis and perturbation theory to improve the VNE performance. However, these reinforcement learning methods all discretized the node embedding of same request and ignored continuity between nodes. It is obvious that embedding results of the previous node inevitably affect the current and even later embedding decisions. Therefore, in this paper, our proposed CDRL algorithm guarantees continuity of the node embedding process and beneficially improves the network performances.

C. Contributions and Organization

Our paper has the following three main contributions:

- To the best of our knowledge, it is the first time that the VNE is formulated as a time-series problem, where the continuity of node embedding process is modeled by a Recurrent Neural Network (RNN), which can substantially exploit the history states of the substrate network.
- We use the policy gradient algorithm to update the parameters of RNN relying on our innovatively defined long-term average revenue to cost ratio metric, namely CDRL, which is beneficial in terms of achieving the optimal network resource utilization efficiency.
- Finally, we define three evaluation metrics for benchmarking VNE algorithms. Simulation results show that relying on the seq2seq model aided node embedding process, our proposed CDRL algorithm outperforms traditional three VNE strategies in terms of computational and transmission resources of the substrate network.

The rest of paper is organized as follows. Section II introduces the VNE model and three evaluation metrics for benchmarking the VNE algorithms. Our proposed CDRL algorithm is proposed in Section III. The training and testing processes of the CDRL and its performance in comparison to other traditional algorithms are conducted in Section IV. Finally, our conclusion is given in Section V.

[1]The seq2seq is a common model in the field of Natural Language processing (NLP), which is a widely used architecture for machine translation and summarization relying on a recurrent neural network as one of its building blocks [16], [17].

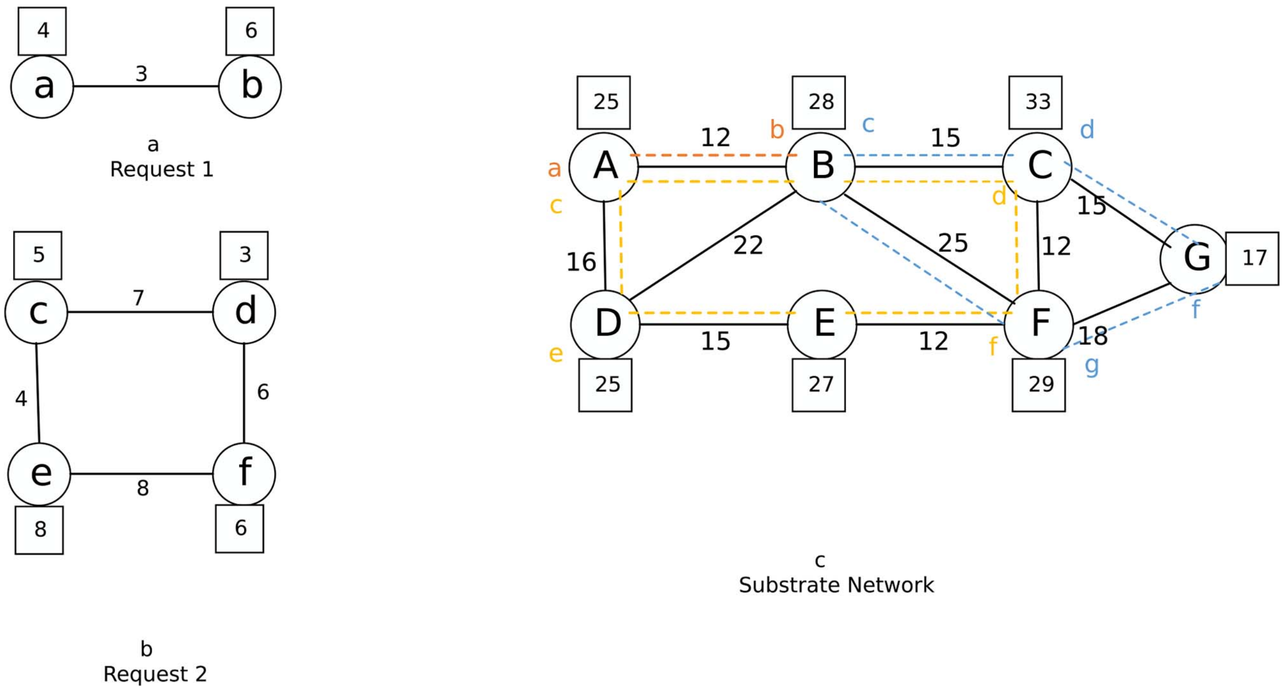


Fig. 1. Embedding results from the virtual network requests to the substrate network. Virtual nodes *a*, *b* are embedded to the substrate nodes *A*, *B* while the virtual link *ab* is embedded to the substrate link *AB*, and virtual nodes *c*, *d*, *e*, *f* are embedded to the substrate nodes *A*, *C*, *D*, *F* while the virtual links *cd*, *df*, *ef*, *ce* are embedded to the substrate paths *AC*, *CF*, *DF*, *AD*. With the aid of criterion of consuming less resource to achieve embedding result, it can be easily found that the bandwidth consumed by *AC* and *DF* are not optimal. Furthermore, After virtual nodes *c*, *d*, *e*, *f* are transferred to the substrate nodes *B*, *C*, *F*, *G* some bandwidth resources are saved for accepting other virtual network requests.

II. System Model and Evaluation Metrics

A. System Model

As mentioned above, VNE is the process of assigning virtual network requests to the substrate network according to their resource requirements. For the sake of analysis, the substrate network can be formulated by an undirected graph $G^S = (N^S, L^S, A_N^S, A_L^S)$, where N^S and L^S denote substrate nodes and links, while A_N^S and A_L^S represent the computational capability of nodes and the bandwidth of links, respectively. Moreover, let P^S represent a set of acyclic paths in the substrate network. Similarly, we also utilize an undirected graph $G^V = (N^V, L^V, C_N^V, C_L^V)$ to formulate a virtual network, where N^V and L^V are nodes and links in G^V, while C_N^V and C_L^V are computational resource requirements of nodes and bandwidth requirements of links, respectively. Relying on the symbols defined above, the VNE process can be easily expressed by $M : G^V(N^V, L^V) \rightarrow G^S(N', L')$, where $N' \subset N^S$ and $L' \subset L^S$. In our resource allocation problem, one objective is to find the node embedding relation $X = \{x_{ij} \mid n_i \in N^V, n_j \in N^S\}$, which can be formulated by:

$$\sum_{j=1}^{|N^S|} x_{ij} = 1, \tag{1}$$

where $x_{ij} = 1$ denotes that the virtual node n_i is embedded to the substrate node n_j, and we have $n_i \in N^V, n_j \in N^S$. Hence, another objective is to find the link embedding relation $Y = \{y_{ij} \mid l_i \in L^V, l_j \in L^S\}$, which can be given by:

$$\sum_{j=1}^{|L^S|} y_{ij} \geq 1, \tag{2}$$

where $y_{ij} \geq 1$ means that link l_i is embedded to one or more substrate links, and we have $l_i \in L^V,\ l_j \in L^S$. If a virtual network request G^V is eligible to be successfully embedded to the substrate network, it needs to satisfy the constrain of computational capability, which can be expressed by:

$$x_{ij} \cdot C_{n_i}^V \leq x_{ij} \cdot A_{n_j}^S, \tag{3}$$

where $A_{n_j}^S$ denotes the actual computational capability of n_j, while $C_{n_i}^V$ denotes the node resource requirement of n_i, and we have $n_i \in N^V, n_j \in N^S$. Furthermore, the bandwidth constrain also needs to be satisfied, which can be given by:

$$y_{ij} \cdot C_{l_i}^V \leq y_{ij} \cdot A_{l_j}^S, \tag{4}$$

where $A_{l_j}^S$ denotes the actual bandwidth of l_j, while $C_{l_i}^V$ denotes the bandwidth requirement of l_i, and we have $l_i \in L^V,\ l_j \in L^S$.

For the sake of simplification, figures in Fig. 1 are used to clarify our VNE process. Fig. 1 is consisting of request 1 and request 2 as well as a substrate network, where request 1 has 2 virtual nodes and 1 virtual links, while request 2 has 4 virtual nodes and 2 links. Moreover, the value on the line represents the bandwidth requirement of link, while the value next to the node denotes the computational resource requirement of node. Furthermore, for the substrate network in Fig. 1,

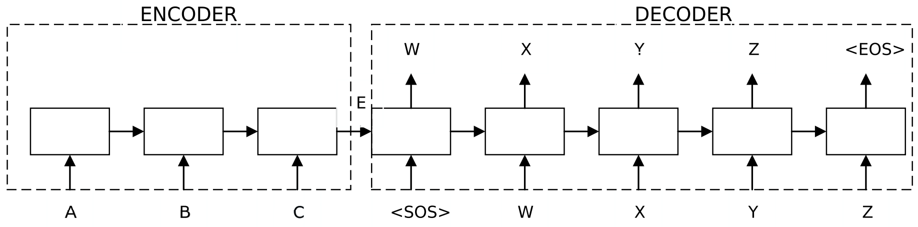


Fig. 2. The seq2seq model.

the value on the line represents the actual bandwidth of substrate link, while the value next to the node denotes the actual computational capability of substrate node. Whatever the case is, different embedding algorithms all aim at looking for an optimal embedding strategy, in which less resource is consumed in the case of successful embedding and ISPs gain much more revenue.

B. Evaluation Metrics

The revenue of ISP, namely $R(G^V, t, t_d)$, is computed in line with the duration time t_d of virtual network request, consumed node computational resources $CPU(n^V)$ as well as the link bandwidth $BW(l^V)$. Hence, the revenue can be formulated by:

$$R\Big(G^V, t, t_d\Big) = t_d\left[w_c \sum_{n^V \in N^V} CPU\Big(n^V\Big) + w_b \sum_{l^V \in L^V} BW\Big(l^V\Big)\right], \quad (5)$$

where $CPU(n^V)$ is the node computational resources consumed by the virtual node n^V, while $BW(l^V)$ is the bandwidth consumed by the virtual link l^V. Furthermore, w_c and w_b are revenue weights of nodes and links in the substrate network, which are only determined by the ISP. Specifically, some ISPs pay more attention to the benefits from consuming computational resources of nodes, then w_c is correspondingly a little bit higher. By contrast, if ISPs care much more about benefits from consuming bandwidth of links, w_b is correspondingly a little bit higher.

After a virtual network request is successfully embedded to the substrate network, the resource consumption can be calculated by:

$$C\Big(G^V, t, t_d\Big) = t_d\left[\sum_{n^V \in N^V} CPU\Big(n^V\Big) + \sum_{l^V \in L^V}\sum_{l^S \in L^S} BW\Big(f_{l^S}^{l^V}\Big)\right], \quad (6)$$

where $BW(f_{l^S}^{l^V})$ is the bandwidth consumed by l^V in the substrate link l^S, while $CPU(n^V)$ denotes the computational resources consumed by n^V. Moreover, during the link embedding stage, a virtual link l^V may be assigned to multiple substrate links, so the total bandwidth consumption of G^V needs to be calculated.

For the sake of easy comparison, there are generally three metrics to evaluate the performance of different VNE algorithms. The first metric is the $long-term\ average\ revenue$, which calculates the average revenue over an infinite period of time to evaluate the overall impact of a VNE algorithm. We have:

$$Rev = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} R\big(G^V, t, t_d\big)}{T}, \quad (7)$$

where *T* denotes the elapsed time. As mentioned above, VNE aims at getting as much revenue as possible with limited resource consumption, which gives us a second evaluation metric namely *long-term average revenue to cost ratio*. It can be calculated by:

$$Rev2Cos = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} R\big(G^V, t, t_d\big)}{\sum_{t=0}^{T} C\big(G^V, t, t_d\big)}, \quad (8)$$

where a higher *long-term average revenue to cost ratio* indicates a higher resource utilization efficiency to meet the resource requirements of more virtual network requests. The last evaluation metric is the *long-term acceptance ratio*, which calculates the percentage of all virtual network requests that are successfully embedded. We have:

$$Acp = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} Acp\ \big(G^V, t, t_d\big)}{\sum_{t=0}^{T} All\ \big(G^V, t, t_d\big)}, \quad (9)$$

where $Acp\ (G^V, t, t_d)$ denotes the number of accepted virtual network requests over time horizon *T*. Furthermore, we utilize these three metrics to evaluate performance of our proposed CDRL algorithm with other algorithms in following sections.

III. Embedding Algorithm

In this section, our CDRL algorithm and seq2seq model are described in detail. The reinforcement learning agent is applied to extract the information of nodes in the substrate network and form a feature matrix as the input of the seq2seq model. Then the model outputs the node embedding results of current virtual network request. The policy gradient algorithm is utilized to update the network parameters, and finally a model with a good embedding mechanism is produced.

A. Seq2Seq Model

The seq2seq model is commonly utilized to solve some sequence-to-sequence problems such as machine translation, QA system, etc. It is divided into two parts, as shown in Fig. 2, the encoder and decoder part, respectively. Each cell in two parts has a hidden vector h_t, which is related to the input vector x_t and the hidden vector of the previous cell h_{t-1}. The h_t can be calculated by:

$$h_t = f(x_t, h_{t-1}), \tag{10}$$

where *f* is a nonlinear activation function tanh or sigmoid. For the sake of simplification, we assume that a pair of source-target vectors is represented by (*A*, *B*, *C*) → (*W*, *X*, *Y*, *Z*). Specifically, in encoder part, the source vectors (*A*, *B*, *C*) are integrated into a encode vector *E*, which is actually the last hidden vector h_t in encoder part and represents the semantics of all input vectors. Then vector *E* and the target vectors (*SOS*, *W*, *X*, *Y*, *Z*) are used as the input of the decoder part, where *SOS* is the start signal of the decoder part. Furthermore, the training process of seq2seq model aims at making the output of the decoder part get close to the target vectors (*W*, *X*, *Y*, *Z*, *EOS*), where *EOS* is the end signal of the decoder part. In comparison to the VNE, we found the VNE problem can also be considered as a sequence-to-sequence problem in which information of the substrate network is encoded in the encoder part and be sent to the decoder part for decoding. As a result, we can get continuous node embedding results of current virtual network request.

In order to explain this process more clearly, here we define an input sequence $x_1, x_2, \ldots, x_t$ and an output sequence $y_1, y_2, \ldots, y_t$. Hence, for the seq2seq model, the purpose is to maximize the probability:

$$\prod_{t=1}^{T} P(y_t \mid v, y_1, \ldots, y_{t-1}), \tag{11}$$

where *v* denotes the input vector $x_1, x_2, \ldots, x_t$. Different from the encoder part, the hidden vector h_t in decoder part is calculated by:

$$h_t = g(v, y_{t-1}, h_{t-1}), \tag{12}$$

which is related to the output y_{t-1} of the last cell, where *g* denotes a nonlinear activation function. In order to prevent the numerical underflow problem of Eq. (11), the objective function of the seq2seq model can be rewritten as a log likelihood conditional probability function, which can be formulated by:

$$\max_{\theta} \frac{1}{T} \sum_{t=1}^{T} \log P_\theta(y_t \mid x_t), \tag{13}$$

where θ denotes the parameters of the seq2seq model.

B. Information Extraction

In order to vectorize the global state information of the substrate network, we need to extract the state information of each substrate node and link. Nodes in the substrate network have a range of characteristics such as remaining computational capability, bandwidth of adjacent links as well as the connection degree. Generally speaking, the more that features are extracted by the reinforcement learning agent, the more that the feature matrix can represent the entire substrate network. However, if the dimension of the feature matrix is large enough, it will lead to a high complexity, and hence will be likely to be over-fitting. As for the benchmark algorithm RLVNE [28], Yao *et al.* extracted four types of state information, including the computing capacity, degree, sum of bandwidth and average distance to other host nodes, which take into consideration the positions where other virtual nodes in the same request are embedded. However, as for our seq2seq model, it continuously outputs the embedding results of all virtual nodes in current virtual network each time, so we cannot take the average distance to other host nodes into account. Hence, three pieces of general information of the substrate network can be extracted in order to formulate the feature matrix, i.e.,

1) Computational capability (*CPU*): The remaining computational capability has a great impact on embedding results during the node embedding process, so this dimension needs to be taken into account in our feature matrix. The *CPU* can be given by:

$$CPU(n^s) = CPU(n^s)' - \sum_{n^v \to n^s} CPU(n^v), \tag{14}$$

where $CPU(n^s)$ denotes the initialization CPU value of n^s, while $\sum_{n^v \to n^s} CPU(n^v)$ represents the sum of CPU value of all virtual nodes embedded to n^s.

2) Degree (*DEG*): The degree is the number of links directly connected to the substrate node, which reflects the connectivity of the substrate network. Furthermore, the higher the degree, the easier it is to find the path between the other nodes. The *DEG* can be calculated as:

$$DEG(n^s) = \sum_{n \in N^s} L(n^s, n), \tag{15}$$

where $L(n^s, n)$ is equal to 1 when n^s and *n* are connected, 0 when not connected.

3) Sum of Bandwidth (*SUM*): Each substrate node may be connected to more than one link, we calculate the sum of bandwidth of all links connected to it, which can be formulated as:

$$SUM = \sum_{l^s \in L^{n^s}} BW(l^s), \tag{16}$$

where L^{n^s} denotes the substrate links connected to node n^s, while l^s denotes one of the L^{n^s}. Moreover, a higher *SUM* means more likely to complete the link embedding in this substrate node.

After these three features are extracted, their normalized values are connected into a feature matrix *A*, which can be given by:

$$A = [CPU, DEG, SUM]. \tag{17}$$

The matrix *A* is used as the input of the seq2seq model and is updated as the state of the substrate network changes.

C. Markov Decision Process

For our VNE problem, we have no way to obtain sufficient embedding data with labels, so we cannot solve this problem by using supervised learning algorithms. Hence, we introduce the reinforcement learning algorithm [30] to solve this problem. As is well known, the reinforcement learning algorithm is a method which is learned in practice and judges the performance of the current operation based on the value of a reward signal. For the sake of simplification, the process of reinforcement learning is usually assumed that there is Markov property between state probability transitions, and hence we use a Markov Decision Process (MDP) to model the reinforcement learning. In our model, the node embedding is considered as a continuous decision process, where the decision-making agent aims at collecting information of the substrate network to make node embedding decisions and the revenue can be obtained after the embedding. Therefore, the node embedding process of all virtual network requests can be simulated as (*S*, *A*, *P*, *R*, γ), where *S* and *A* represent two finite sets of states and actions, while *P* represents the state transition probability. It can be given by:

$$P_{ss'}^{a} = P\left[S_{t+1} = s' \mid S_t = s, A_t = a\right], \tag{18}$$

where S_t and A_t are state and action at time *t*. Moreover, *R* is the reward function, which can be expressed by:

$$R_s^a = E[R_{t+1} \mid S_t = s, A_t = a], \tag{19}$$

where R_s^a denotes the reward after action *a* is performed at state *s*, while $\gamma \in [0, 1]$ is a discount factor when the total discounted reward G_t is calculated from time *t*. The G_t can be formulated by:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{20}$$

where G_t reflects the impact on all subsequent states after action *a* is proformed, which is reduced over time. Furthermore, action policy π is a distribution of actions at state *s*, which can be expressed by:

$$\pi(a|s) = P[A_t = a \mid S_t = s]. \tag{21}$$

Additionally, the task of the decision-making agent is to find the best policy π which can be achieved by maximizing the reward. For MDP, there are two value functions including the state-value as well as the action-value. Moreover, the state-value function $V_\pi(s)$ is only related to the current state *s*, which can be formulated by:

$$V_\pi(s) = E_\pi[G_t \mid S_t = s]. \tag{22}$$

Similarly, the action-value function $q_\pi(s, a)$ is related to current action *a* as well as the state *s*, which can be expressed by:

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]. \tag{23}$$

According to the expression of the value function, we can derive the recursive relationship of the value function. For example, for the state-value function $v_\pi(s)$, we have:

$$v_\pi(s) = E_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \tag{24}$$

$$= E_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots \mid S_t = s] \tag{25}$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} | \; S_t = s] \tag{26}$$

$$= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | \; S_t = s]. \tag{27}$$

Obviously, there is a recursive relationship between the state S_t and S_{t+1}, which can be given by:

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | \; S_t = s]. \tag{28}$$

This recursive equation is generally called the Bellman equation, which indicates that the state-value of a state is composed of the reward of the state and the subsequent state-value combined in a certain ratio. In the same way, we can get the Bellman equation of the action-value function $q_\pi(s, a)$:

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | \; S_t = s, A_t = a]. \tag{29}$$

According to the above definition, we can easily conclude the translation relationship between $V_\pi(s)$ and $q_\pi(s, a)$, which can be given by:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s|a), \tag{30}$$

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in s} P_{ss'}^a V_\pi(s'). \tag{31}$$

Suppose now the decison-making agent receives a virtual request with *p* numbers of nodes, it has to make *p* numbers of embedding decisions at time *t* to embed these nodes to the substrate network. Obviously, S_t can be expressed by the feature matrix which is inputted to the model at time *t*. Moreover, the action A_t at time *t* is:

$$A_t = \left\{(n_t^v, \; n_t^s) \mid n_t^s \in N_t^S \cap N^S(n_t^s)\right\}, \tag{32}$$

where $N^S(n_t^s)$ denotes the substrate nodes which meet the CPU requirement of n_t^s. After action a_t is performed, the decision-making agent receives a reward signal R_t.

However, if we use the value-based reinforcement learning algorithms such as Q-learning algorithm [31] to make embedding decisions, we need to calculate the reward obtained by executing different actions under each state and choose the action with the largest reward. Obviously, the value-based algorithm does not fit our problem well. Because our state space is made up of continuous values and we cannot get the transition probability distribution between different states. Hence, we introduce a policy-based algorithm to address with the MDP, which directly optimizes the policy of actions. Furthermore, details of the policy gradient algorithm are introduced in the next section.

D. Policy Gradient

Policy gradient [32] algorithm is a reinforcement learning algorithm which optimizes the policy of actions directly. At the beginning, the policy π can be described as a function containing the parameter θ:

$$\pi_\theta(s, a) = P(a|s, \theta) \approx \pi(a|s). \tag{33}$$

After the policy function is represented as a continuous function, we can use continuous function optimization methods such as gradient descent algorithm to optimize the strategy.

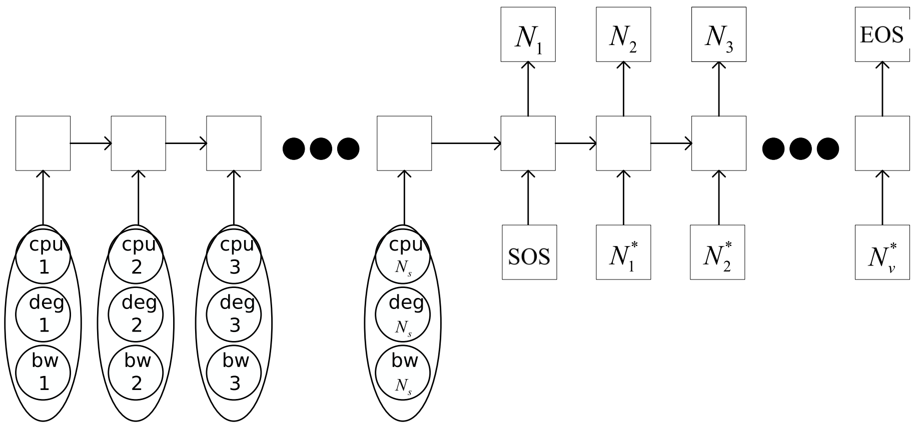


Fig. 3. The seq2seq training model. During the training process, the start decoding signal SOS and hand-crafted labels $N_1^*, N_2^*, \ldots, N_v^*$ are inputted to the decoder part. The output is the decoding result $N_1, N_2, \ldots, N_v$ and the end decoding signal *EOS*. Each output vector N corresponds to the location of the substrate network node to which the current virtual node is embedded.

The optimization function can be expressed by:

$$J(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R_s^a, \tag{34}$$

where θ denotes the parameters of policy gradient algorithm, while $d^{\pi_\theta}(s)$ is the probability distribution of states. Moreover, when gradient of the optimization function $J(\theta)$ is calculated, there is a little trick named likelihood ratio, which is:

$$\begin{aligned} \nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a). \end{aligned} \tag{35}$$

Relying on the likelihood ratio, gradient of the objective function $J(\theta)$ can be formulated by:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) R_s^a \\ &= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) r], \end{aligned} \tag{36}$$

where r is the total reward over the entire process. Furthermore, according to the policy gradient theorem [33], under the multi-step MDP, we have:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)], \tag{37}$$

where $Q^{\pi_\theta}(s, a)$ denotes the sum of multi-step reward. During the actual optimization process, unbiased sampling is performed on $Q^{\pi_\theta}(s_t, a_t)$. Then we can get v_t, and the parameters can be updated by the form of:

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t. \tag{38}$$

E. Training And Testing

The training process of our model is shown in Fig. 3, and network parameters of the seq2seq model need to be initialized at the beginning. In the training process, we input the feature matrix of the substrate network extracted by the reinforcement learning agent and get the encode vector *E*. Furthermore, the encode vector *E* and the start signal *SOS* are inputted into the decoder part. Let the output N_1 represent the location where the first virtual node is embedded to the substrate network. Then, it is calculated by a softmax function, which can be expressed by:

$$p_i = \frac{e^{h_i}}{\sum_j e^{h_j}}, \tag{39}$$

where *h* denotes the output vector of each cell in decoder part. Actually, parameters in our model are randomly initialized, and the node with the highest probability may not be the best result. Therefore, according to the probability of the output in the softmax layer, we select one node from the set of nodes with sufficient available resources as the host node. Moreover, after all the virtual nodes of the current virtual network are embedded, the Breadth-First-Search (BFS) algorithm is applied to find the path with the optimal bandwidth consumption between each pair of virtual nodes [34].

Furthermore, as for the supervised learning, each piece of data corresponds to a specific label. And the label is compared with the output of training process, where the cross-entropy or Mean Squared Error (MSE) formulation is used to calculate the error between them. Then some algorithms are used to minimize this error and update the parameters of the model. However, the reinforcement learning algorithm is a completely different algorithm from the supervised learning. Specifically, the reinforcement learning agent determines the effectiveness of each action based on the reward signal. If current action brings a large reward signal, it means the action is valid and will be encouraged to happen in next epoch. By contrast, if the current action brings a small or even negative reward signal, this action will be tried to avoid to happen in next epoch. Therefore, the reward signal is significant for agent to train the model and make embedding decisions. So in our experimental settings, we take the long-term average revenue to cost ratio as the reward signal, which is because this metric reflects the utilization efficiency of the substrate resources, and it also has a significant impact on the other two metrics.

To implement our algorithm, we need to assign a label to each output of the decoder part, and then update the parameters based on the error between the label and the specific output. Therefore, we use the hand-crafted label which is determined by an $|n_s|$ dimensions vector y_i to denote the label of virtual

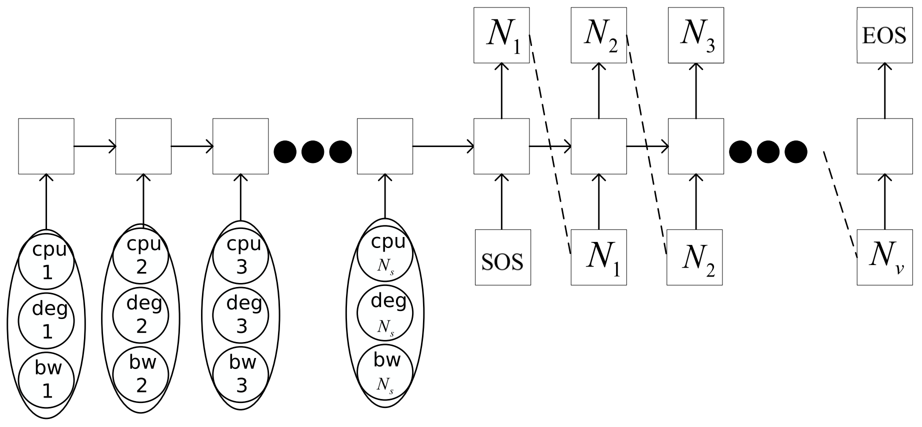


Fig. 4. The seq2seq testing model. During the testing process, we can find that when the start decoding signal *SOS* is inputted to the decoder part, the decoding process begins. Obviously, the output of the previous cell is used as the input to the next cell, and the decoding process ends when the output is *EOS*.

node, where is the number of substrate nodes. Specifically, if the label of a virtual node is set as the *i-th* substrate node, the *i-th* dimension of y_i is set to 1, while the other dimensions are set to 0. Moreover, the error between y_i and output vector p_i can be calculated by:

$$L(y,p) = -\sum_i y_i log(p_i), \tag{40}$$

and the cross-entropy of the decoder part can be given by:

$$loss = \sum_{i=1}^{|N^V|} L_i(y,p). \tag{41}$$

Then we use the gradient descent algorithm to calculate the gradient g_f of loss, then multiply the reward signal *r* and the learning rate α to get the parameter update formula of our model, which can be given by:

$$g = \alpha \cdot r \cdot g_f. \tag{42}$$

The learning rate α is a significant parameter in the process of updating the model parameters, which determines the speed of network updates. If α is set incorrectly, it may cause over-fitting or under-fitting of the network, affecting the final performance of the model. Hence, we need to find a moderate α to keep a stable update process and converge speed. The batch gradient descent algorithm is applied to update the network, which is beneficial in terms of improving the converge speed as well as making the network stable. Specifically, during the training process, we created a gradient stack to temporarily store these calculated gradients instead of using them immediately. When the number of storage reaches the batch amount, the model parameters are updated. However, if the link embedding process fails because of insufficient bandwidth, g_f will be deleted as we are unable to identify the reward signal. Our training algorithm is represented in Algorithm 1.

In testing process, the performance of our network is tested with the online virtual network requests which are slightly different from training process. The testing model is shown in Fig. 4, which shows that the output of each cell in decoder part is applied as the input of the next cell. The embedding result adopts a beam search strategy [35], where the nodes with the highest product of probabilities are chosen as host nodes. Algorithm 2 is our testing algorithm.

Algorithm 1 Training Process

```
Input: Number of epochs epochNum; Training data trainingSet; Batch size;
Output: Trained parameters in seq2seq model;
 1: Initialize all the parameters in seq2seq model;
 2: while epoch < epochNum do count=0
 3:     for request ∈ trainingSet do
 4:         input feature matrix of the substrate network to the seq2seq model;
 5:         get output of seq2seq model and select host nodes;
 6:         if embedded (∀ node ∈ request) then
 7:             compute gradient $g_f$ of request and add to stack;
 8:             link embedding process;
 9:         end if
10:         if embedded (∀ node ∈ request, ∀ link ∈ request) then
11:             compute the revenue to cost ratio as the reward signal;
12:             multiply reward signal and $g_f$ to compute the final gradient;
13:         else
14:             delete $g_f$;
15:         end if
16:         ++count
17:         if count equals to the batch size then
18:             update network parameters;
19:             count=0;
20:         end if
21:     end for
22:     ++epoch;
23: end while
```

F. Computational Complexity

For our CDRL algorithm, the computational complexity is *O*(*v*(*mr* + *p* + *q*)), where *v* and *m* are the number of virtual network requests and the substrate nodes, *p* and *q* are the maximum number of nodes and links in virtual network requests, respectively. Moreover, *r* is the dimension of the feature matrix.

Proof: for every virtual network request, the computational complexity of computing the feature matrix is *O*(*mr*). After

Algorithm 2 Testing Process

```
Input: Testing data testingSet; Trained parameters in seq2seq model;
Output: Long-term average revenue; Long-term average acceptance ratio; Long-term average revenue to cost ratio;
1:  for $request \in testingSet$ do
2:      input feature matrix of the substrate network to the trained seq2seq model;
3:      get output of the seq2seq model;
4:      select host nodes using beam search strategy;
5:      if embedded ($\forall$ node $\in$ request) then
6:          link embedding process;
7:      else
8:          return FALSE;
9:      end if
10:     if embedded ($\forall$ node $\in$ request, $\forall$ link $\in$ request) then
11:         return embedding result;
12:     else
13:         return FALSE;
14:     end if
15: end for
```

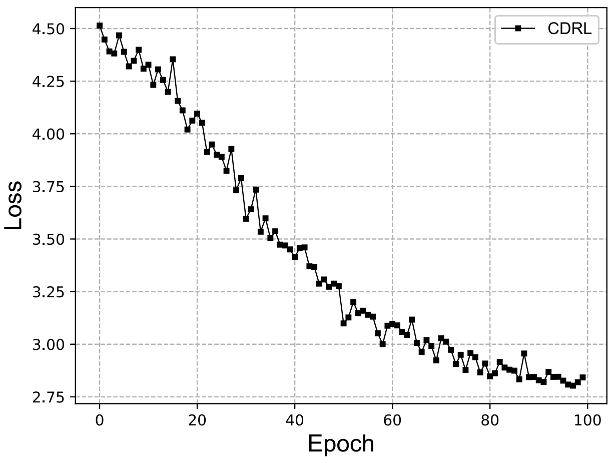


Fig. 5. Loss on the training set.

the feature matrix of the substrate network is inputted to our seq2seq model, the computational complexity is $O(p)$. Furthermore, when virtual nodes are successfully embedded, the computational complexity of the link embedding process is $O(q)$. Hence, for all v numbers of virtual network requests, the computational complexity is $O(v(mr + p + q))$. ∎

IV. Simulation Results

A. Experiment Setup

We used the GT-ITM tool [36] to generate a substrate network of 100 nodes and 550 links, which belongs to a medium ISP scale. The computational capability of nodes satisfies the uniform distribution between (50, 100), while the bandwidth of links satisfies the uniform distribution between (20, 50).

After the substrate network is generated, 2000 numbers of virtual networks will be produced. Furthermore, in these 2000 numbers of virtual networks, the training set consists of the first 1000 virtual networks, while the remaining 1000 virtual networks form the testing set. Specifically, every virtual network request has 2-10 nodes and the computational capability satisfies a uniform distribution between (0, 50), while the bandwidth of each link satisfies a uniform distribution between (0,50). The connection probability between virtual nodes is 0.5, which means the average number of links is $\frac{(n(n-1))}{4}$, where n represents the number of virtual nodes. The arrival speed of virtual network requests obeys a poisson distribution, where 4 requests will be reached within 100 time units, while the duration time of virtual network requests follows an exponential distribution with an average duration of 1000 time units. Hence, we have built a timeline with a length of 50,000 time units.

We apply the TensorFlow [37] frame to build the seq2seq model. Specifically, the encoder part has 100 LSTM cells which represent 100 substrate nodes, while the number of cells in the decoder part is determined by the number of virtual nodes in the current virtual network request. In general, the number of nodes in the encoder part is fixed, while the number of nodes in the decoder part varies with different virtual requests. Network parameters are initialized according to the normal distribution.

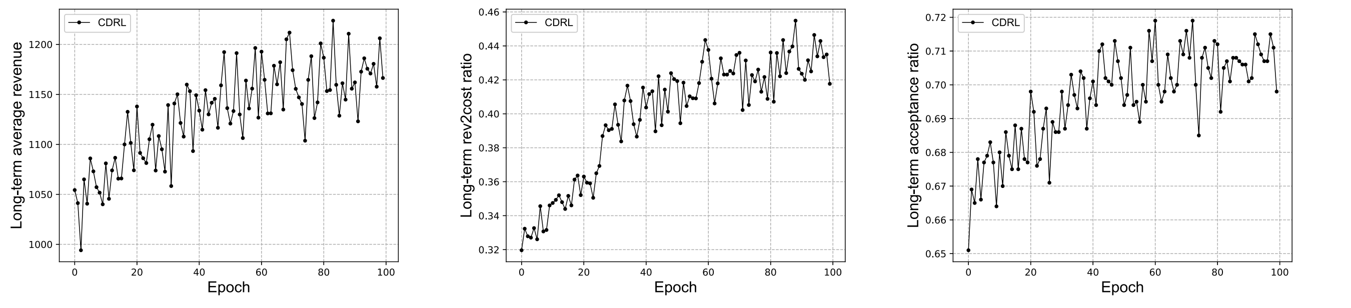
B. Training Performance

In comparison to the supervised learning, the converge speed of the reinforcement learning is slower, which is because it has no accurate label and network parameters need to be updated based on the hand-crafted labels and reward signal. During the training process, we utilized the training set with 1000 virtual requests to train the seq2seq model with random parameters initialization for 100 epochs. From the loss changing process in Fig. 5, we can conclude that the model converges well. Furthermore, Fig. 6 shows these three metrics changing processes within 100 epochs. Furthermore, we can find at the beginning of the training process, every metric is relatively low because the parameters are randomly initialized. As the training process continues, the reinforcement learning agent applies the policy gradient algorithm to update the network according to the reward signal, where the embedding policies are adjusted according to the magnitude of the reward signal. With the constant update of network parameters, the model performance is getting better. Furthermore, after 100 epochs, because of the limitation of the resources of the substrate network, these three metrics have reached a certain point.

C. Simulation Result

From figures of the training performance, we can conclude that our CDRL algorithm can be applied to the training set well. However, our optimization goal is not only perform well on the training set, but also on the online virtual network requests. This requires us to test our algorithm on the testing set which is different from the training set. Moreover, if our CDRL algorithm performs well in both training and testing sets, it proves that our proposed algorithm is robust and our research topic is meaningful.

In testing process, the node with the highest probability is selected. In order to test our algorithm performance convincingly, the contrast experiments with other three algorithms are



(a) The changing process of the long-term average revenue during training process (Eq. (7)). (b) The changing process of the long-term average Rev2Cost ratio during training process (Eq. (8)). (c) The changing process of the long-term average acceptance ratio during training process (Eq. (9)).

Fig. 6. The algorithm performance on the training set.

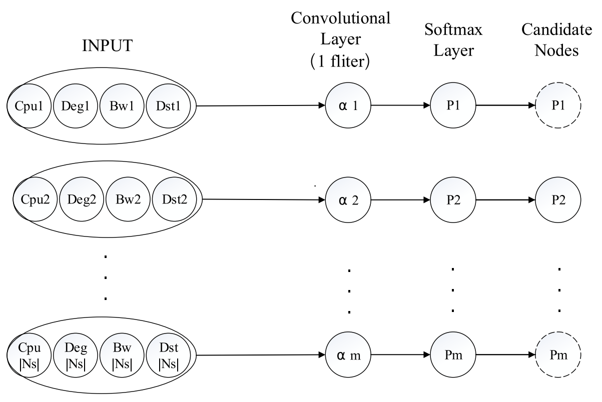


Fig. 7. The RLVNE model.

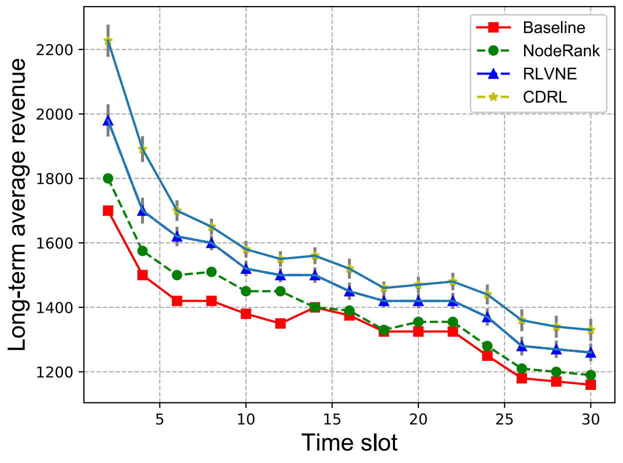


Fig. 8. The long-term average revenue of four algorithms during testing process (Eq. (7)).

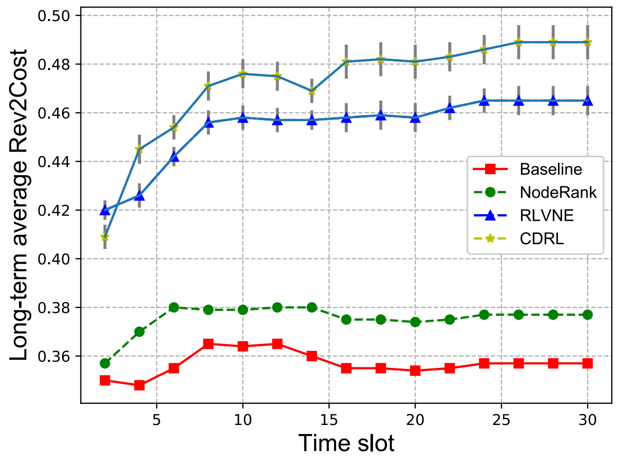


Fig. 9. The long-term average Rev2Cost ratio of four algorithms during testing process (Eq. (8)).

conducted, which includes two heuristic algorithms as well as a reinforcement learning aided algorithm. The first is the baseline algorithm proposed in [7], which introduced the path splitting and migration to the link embedding part. The second algorithm is the NodeRank algorithm proposed in [9], which calculated $H(n^s)$:

$$H(n^s) = CPU(n^s) \sum_{l^s \in L(n^s)} BW(l^s), \tag{43}$$

to rank substrate nodes, where the $H(n^s)$ represents the resource availability of node n^s. The last algorithm is RLVNE algorithm proposed in [28], which introduced a policy network based on the reinforcement learning to make node embedding decisions. The policy network architecture is shown in Fig. 7.

In order to avoid interference caused by random initialization of neural network parameters, the CDRL and RLVNE algorithms need to run for 30 different initializations and to apply the testing set for the sake of evaluating the algorithm performance. The evaluation metrics of four different algorithms are shown in Fig. 8, 9, 10. Furthermore, we added the error bar for each time slot. As we can see from Fig. 8, 9, 10, the long-term average revenue and acceptance ratio are relatively high in previous epochs, because the substrate network has more available resources at the beginning. As the testing process continues, available substrate resources are gradually consumed, which results in the decrease of the long-term average revenue and acceptance ratio. By contrast, the long-term revenue to cost ratio has no particularly substantial fluctuation from the beginning to the end, because it has no relationship with the available resources. Moreover, as we can see from the average metric figures and error bars, we can conclude that our algorithm has significantly improved these three evaluation metrics.

For the sake of determining whether our algorithm is better than RLVNE, a statistical hypothesis test is performed based on 30 testing results. The Wilcoxon [38] testing method is

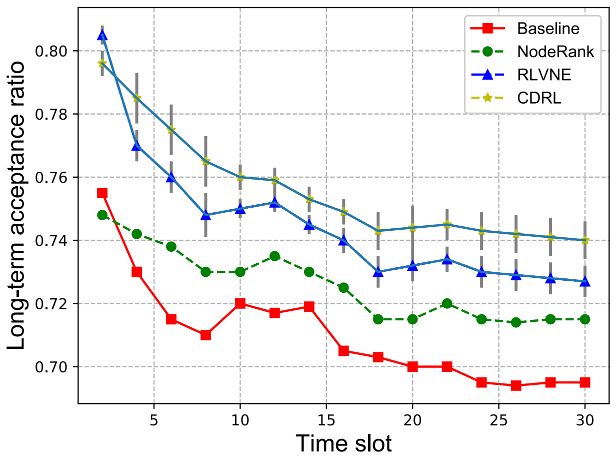


Fig. 10. The long-term average acceptance ratio of four algorithms during testing process (Eq. (9)).

chosen and the average value over all time slots is regarded as the testing data. Furthermore, the original hypothesis H_0 is set as: RLVNE and CDRL algorithms have no significant difference on performance, while the alternative hypothesis H_1 is set as: CDRL algorithm performs better than RLVNE. According to the calculation results, the one-sided significance value P < 0.05, so H_0 is rejected and H_1 is accepted. Therefore, we can conclude that our algorithm is better than RLVNE.

V. Conclusion

In this paper, we propose a Continuous-Decision VNE scheme relying on Reinforcement Learning (CDRL), which regards node embedding of the same virtual network request as a time-series problem formulated by the classic seq2seq model. The simulation results indicate that our CDRL algorithm achieves good progress in three evaluation metrics and improves the overall embedding effect.

In the future work, we still have three details to improve. Firstly, we can introduce the attention [39] mechanism to our seq2seq model to avoid information loss due to excessively long encoder part. Secondly, we can extract much more representative information about the substrate network. At last, RNN cannot be parallelized which causes a slow training speed. Moreover, we can consider introducing the transformer [39] mechanism to solve this problem.

References

[1] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.

[2] N. M. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Oct. 2010.

[3] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Comput.*, vol. 17, no. 2, pp. 20–27, Mar. 2013.

[4] Y. Zeng and R. Zhang, *Efficient Mapping of Virtual Networks Onto a Shared Substrate*, Washington Univ., St. Louis, MO, USA, Jan. 2006.

[5] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proc. 25th IEEE Int. Conf. Comput. Commun.*, Barcelona, Spain, Apr. 2006, pp. 23–29.

[6] Z. Liu and M. Wu, "Exact solutions of VNE: A survey," *China Commun.*, vol. 13, no. 6, pp. 48–62, Jul. 2016.

[7] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Apr. 2008.

[8] A. Razzaq and M. S. Rathore, "An approach towards resource efficient virtual network embedding," in *Proc. Int. Conf. Evol. Internet*, Valcencia, Spain, Sep. 2010, pp. 68–73.

[9] X. Cheng *et al.*, "Virtual network embedding through topology-aware node ranking," *Comput. Commun. Rev.*, vol. 41, no. 2, pp. 38–47, Apr. 2011.

[10] X. Hesselbach, J. R. Amazonas, S. Villanueva, and J. F. Botero, "Coordinated node and link mapping VNE using a new paths algebra strategy," *J. Netw. Comput. Appl.*, vol. 69, pp. 14–26, Apr. 2016.

[11] J. Wang, C. Jiang, H. Zhang, Y. Ren, K.-C. Chen, and L. Hanzo, "Thirty years of machine learning: The road to Pareto-optimal wireless networks," *IEEE Commun. Surveys Tuts.*, early access, doi: 10.1109/COMST.2020.2965856.

[12] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," in *Proc. Ethnographic Praxis Ind. Conf.*, Jan. 1988, pp. 3104–3112.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Apr. 1996.

[14] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 1054–1054, Sep. 1998.

[15] Y. Lecun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, Jan. 2015.

[16] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, Doha, Qatar, Jun. 2014, pp. 1724–1734.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Montreal, QC, Canada, Dec. 2014, pp. 3104–3112.

[18] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on the degree and clustering coefficient information," *IEEE Access*, vol. 4, pp. 8572–8580, 2016.

[19] P. Zhang, H. Yao, C. Qiu, and Y. Liu, "Virtual network embedding using node multiple metrics based on simplified electre method," *IEEE Access*, vol. 6, pp. 37314–37327, 2018.

[20] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on computing, network, and storage resource constraints," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3298–3304, Oct. 2018.

[21] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. 28th IEEE Int. Conf. Comput. Commun.*, Rio de Janeiro, Brazil, May 2009, pp. 783–791.

[22] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. 1st ACM SIGCOMM Workshop Virtualized Infrastruct. Syst. Archit.*, Barcelona, Spain, Oct. 2009, pp. 81–88.

[23] S. Shanbhag, A. R. Kandoor, W. Cong, R. Mettu, and T. Wolf, "VHub: Single-stage virtual network mapping through hub location," *Comput. Netw.*, vol. 77, pp. 169–180, Dec. 2015.

[24] S. Haeri and L. Trajkovic, "Virtual network embedding via Monte Carlo tree search," *IEEE Trans. Cybern.*, vol. 48, no. 2, pp. 510–521, Feb. 2018.

[25] R. Mijumbi, J. L. Gorricho, J. Serrat, M. Claeys, J. Famaey, and F. D. Turck, "Neural network-based autonomous allocation of resources in virtual networks," in *Proc. Eur. Conf. Netw. Commun.*, Bologna, Italy, Jun. 2014, pp. 1–6.

[26] Y. Kawamoto, H. Takagi, H. Nishiyama, and N. Kato, "Efficient resource allocation utilizing Q-learning in multiple UA communications," *IEEE Trans. Netw. Sci. Eng.*, vol. 6, no. 3, pp. 293–302, Jul.–Sep. 2018.

[27] F. Tang, Z. M. Fadlullah, B. Mao, and N. Kato, "An intelligent traffic load prediction-based adaptive channel assignment algorithm in SDN-IoT: A deep learning approach," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 5141–5154, May 2018.

[28] H. Yao, C. Xu, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.

[29] H. Yao, B. Zhang, L. Maozhen, P. Zhang, and L. Wang, "RDAM: A reinforcement learning based dynamic attribute matrix representation for virtual network embedding," *IEEE Trans. Emerg. Topics Comput.*, early access, doi: 10.1109/TETC.2018.2871549.

[30] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[31] C. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, May 1992.

[32] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[33] D. Silver *et al.*, "Deterministic policy gradient algorithms," in *Proc. 31st Int. Conf. Mach. Learn. (ICML)*, vol. 1, Jun. 2014, pp. 387–395.

[34] S. Hougardy, "The Floyd–Warshall algorithm on graphs with negative cycles," *Inf. Process. Lett.*, vol. 110, no. 8, pp. 279–281, Apr. 2010.

[35] P. Koehn, *Pharaoh: A Beam Search Decoder for Phrase-Based Statistical Machine Translation Models*, Massachusetts Inst. Technol., Cambridge, MA, USA, Sep. 2004, pp. 115–124.

[36] E. Z. M. Thomas, *Generation and Analysis of Random Graphs to Model Internetworks*, College Comput., Georgia Inst. Technol., Jul. 1994.

[37] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 25th IEEE Int. Conf. Comput. Commun.*, May 2016, pp. 265–283.

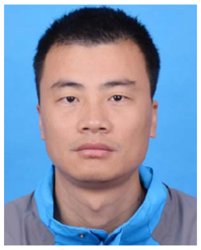
[38] F. Wilcoxon, "Individual comparisons of grouped data by ranking methods," *J. Econ. Entomol.*, vol. 39, no. 2, pp. 269–270.

[39] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. New York, NY, USA: Curran Assoc., Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

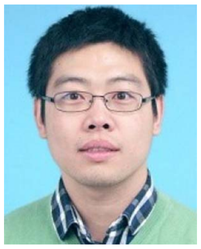
Haipeng Yao (Member, IEEE) received the Ph.D. degree from the Department of Telecommunication Engineering, University of Beijing University of Posts and Telecommunications in 2011, where he is an Associate Professor. He has published more than 90 papers in prestigious peer-reviewed journals and conferences. He has been engaged in research on future Internet architecture, network AI, big data, cognitive radio networks, and optimization of protocols and architectures for broadband wireless networks.

Sihan Ma received the bachelor's degree from the Beijing University of Posts and Telecommunications in 2018, where he is currently pursuing the master's degree with the School of Information and Communication Engineering. His research interests include semantic computing and big data for networking.

Jingjing Wang (Member, IEEE) received the B.S. degree (Highest Hons.) in electronic information engineering from the Dalian University of Technology, Dalian, China, in 2014, and the Ph.D. degree (Highest Hons.) in information and communication engineering from Tsinghua University, Beijing, China, in 2019. From 2017 to 2018, he visited the Next Generation Wireless Group chaired by Prof. L. Hanzo, University of Southampton, U.K. He is currently a Postdoctoral Researcher with the Department of Electronic Engineering, Tsinghua University. His research interests include resource allocation and network association, learning theory-aided modeling, analysis and signal processing, as well as information diffusion theory for mobile wireless networks. He was a recipient of the Best Journal Paper Award of IEEE ComSoc Technical Committee on Green Communications and Computing in 2018 and the Best Paper Award from IEEE ICC and IWCMC in 2019.



Peiying Zhang received the Ph.D. degree in information and communication engineering from the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He is currently an Associate Professor with the College of Computer and Communication Engineering, China University of Petroleum (East China). His research interests include semantic computing, artificial intelligence for networking, network virtualization, and future network architecture.



Chunxiao Jiang (Senior Member, IEEE) received the B.S. degree (Highest Hons.) in information engineering from Beihang University, Beijing in 2008, and the Ph.D. degree (Highest Hons.) in electronic engineering from Tsinghua University, Beijing, in 2013. He is an Associate Professor with the School of Information Science and Technology, Tsinghua University. His research interests include application of game theory, optimization, and statistical theories to communication, networking, and resource allocation problems, in particular space networks and heterogeneous networks. He is the recipient of the Best Paper Award from IEEE GLOBECOM in 2013, the Best Student Paper Award from IEEE GlobalSIP in 2015, the IEEE Communications Society Young Author Best Paper Award in 2017, the Best Paper Award IWCMC in 2017, the IEEE ComSoc TC Best Journal Paper Award of the IEEE ComSoc TC on Green Communications & Computing 2018, the IEEE ComSoc TC Best Journal Paper Award of the IEEE ComSoc TC on Communications Systems Integration and Modeling 2018, and the Best Paper Award ICC 2019. He has served as an Editor for the IEEE INTERNET OF THINGS JOURNAL, IEEE NETWORK, IEEE COMMUNICATIONS LETTERS, and a Guest Editor for *IEEE Communications Magazine,* the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, and the IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING. He has also served as a member of the Technical Program Committee as well as the Symposium Chair for a number of international conferences, including the IEEE ICC 2018 Symposium Co-Chair, the IWCMC 2018/2019 Symposium Chair, the WiMob 2018 Publicity Chair, the ICCC 2018 Workshop Co-Chair, and the ICC 2017 Workshop Co-Chair.

Song Guo (Fellow, IEEE) is a Full Professor and an Associate Head of the Department of Computing, Hong Kong Polytechnic University. His research interests are mainly in the areas of big data, cloud computing, mobile computing, and distributed systems. He is the recipient of the 2019 IEEE TCBD Best Conference Paper Award, the 2018 IEEE TCGCC Best Magazine Paper Award, the 2017 IEEE Systems Journal Annual Best Paper Award, and other six Best Paper Awards from IEEE/ACM conferences. His work was also recognized by the 2016 Annual Best of Computing: Notable Books and Articles in Computing in ACM Computing Reviews. He was an IEEE ComSoc Distinguished Lecturer from 2017 to 2018 and served in IEEE ComSoc Board of Governors from 2018 to 2019. He has also served as the general and program chair for numerous IEEE conferences. He is the Editor-in-Chief of the IEEE OPEN JOURNAL OF THE COMPUTER SOCIETY, and an Associate Editor of the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING, and the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING.