Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach with Graph Convolutional Networks

Zhongxia Yan, Jingguo Ge, Yulei Wu, *Senior Member, IEEE,* Liangxiong Li, Tong Li

***Abstract*—Virtual network embedding arranges virtual network services onto substrate network components. The performance of embedding algorithms determines the effectiveness and efficiency of a virtualized network, making it a critical part of the network virtualization technology. To achieve better performance, the algorithm needs to automatically detect the network status which is complicated and changes in a time-varying manner, and to dynamically provide solutions that can best fit the current network status. However, most existing algorithms fail to provide automatic embedding solutions in an acceptable running time. In this paper, we combine deep reinforcement learning with a novel neural network structure based on graph convolutional networks, and propose a new and efficient algorithm for automatic virtual network embedding. In addition, a parallel reinforcement learning framework is used in training along with a newly-designed multi-objective reward function, which has proven beneficial to the proposed algorithm for automatic embedding of virtual networks. Extensive simulation results under different scenarios show that our algorithm achieves best performance on most metrics compared with the existing state-of-the-art solutions, with upto 39.6% and 70.6% improvement on acceptance ratio and average revenue, respectively. Moreover, the results also demonstrate that the proposed solution possesses good robustness.**

***Index Terms*—Network Virtualization, Virtual Network Embedding, Reinforcement Learning, Graph Convolutional Network**

I. Introduction

THE existing Internet architecture and management schemes become increasingly powerless when facing the growing pressure given by the emerging network services. Many promising techniques have been proposed to tackle this issue. Network virtualization is able to decouple the network services from its underlying network hardware and allows network users to program their own network services. With the useful technologies such as software defined network (SDN) [24] [25] and network functions virtualization (NFV) [13] [32] [5], network virtualization has become the main enabling technology for the management of modern Internet.

By adopting network virtualization, traditional network providers can act as two main roles. One is the service provider (SP) who creates customized virtual networks (VNs) to form end-to-end network services under various circumstances and provides these services to users. The other one is the infrastructure provider (InP) who keeps substrate network (SN) devices so that SPs can use them to host their services with acceptable commercial cost. A good coordination of these two roles can enhance the utility of physical network devices and the commercial incoming of both SPs and InPs. Virtual network embedding (VNE), which forms the embedding from a VN, representing a nework service, to its corresponding SN resources under given resource and service constraints, is therefore an important and challenging task in network virtualization for optimal network performance and resource utilization.

However, the VNE problem is known to be NP-hard [45], which means it is impossible to get exact solutions in a larger network environment. To reduce the computational complexity and running time, many heuristic methods have been applied to VNE algorithms for suboptimal solutions. Fischer et al. [14] investigated most existing algorithms and characterized them in detailed categories. These algorithms have three common drawbacks. First, the features and constraints for modelling are manually designed, which are usually sparse and trivial, making feature engineering inefficient and inflexible in problem solving. Second, the optimization objectives explicitly focus on a single metric or a simple combination of multiple metrics, which only suit for highly specialized situations. Third, the running time of these algorithms in large-scale networks is still excessive despite of the introduction of heuristic methods.

In this paper, we propose a new VNE algorithm based on deep reinforcement learning (DRL) technique to counter existing shortcomings. DRL uses reinforcement learning (RL) as learning architecture and deep learning techniques (e.g., multi-layer neural networks) as automatic feature extractor; it has performed well on a set of complicated sequential decision-making problems, such as the game of Go [34] and adaptive video streaming [22]. In RL, an agent starts from knowing nothing and gradually learns desirable behaviour (e.g., optimal VNE strategy, in this paper) purely by probing the external environment. When the agent gets the network

Manuscript received April 15, 2019; revised December 2, 2019; accepted January 28, 2020. (*Corresponding authors: Jingguo Ge and Yulei Wu.*)

This work is supported by the National Key R&D Program of China (Grant No.:2017YFB0801801), and the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDC02070200 and XDC02060400).

Z. Yan and J. Ge are with the Institute of Information Engineering, Chinese Academy of Science, Beijing, 100093, China, and the School of Cyber Security, University of Chinese Academy of Science, Beijing, 100049, China. E-mail: {yanzhongxia, gejingguo}@iie.ac.cn

Y. Wu is with the College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, U.K. E-mail: y.l.wu@exeter.ac.uk

L. Li and T. Li are with the Institute of Information Engineering, Chinese Academy of Science, Beijing, 100093, China. E-mail: {liliangxiong, litong}@iie.ac.cn

status, it transforms the status to advanced features and generates an action (e.g., an embedding decision from the virtual network component to a certain substrate network resource). The environment executes this action and returns a reward signal to the agent. The reward signal, which differs from the traditional optimizing objective, is not necessarily to make performance maximization for current network status, but can aim for better performance in the future. The agent then uses the reward signal to dynamically improve its next action generation. As the improvement iteratively goes on, the agent finally converges to a strategy that maximizes the cumulative reward in a long term without any explicit objective functions and optimization targets. Once the agent completes the training, it only needs a single step (i.e., a forward computation of neural networks) to generate valid embeddings through raw state inputs, which can reduce the computational complexity while solving VNE problems. After this approach, the procedure of (high-level) feature extraction and the generation of embedding strategy can be handled automatically by the trained learning model without much human effort.

DL-based algorithms have been successfully used in many data-driven problems such as image recognition [15] and text classification [19] due to its powerful capability of automatic feature extraction and the ability of utilizing hidden advanced features. However, applying DRL techniques directly into VNE problem solving is not yet trivial, but may face the following challenges. First, the sampled experiences from the external environment come slow as the environment status usually takes some time to be probed by the agent. Second, despite being widely used in processing regular data like images and voice signals, traditional neural networks are not suitable for irregular data such as network topologies. To overcome these challenges, we adopt a well-known RL framework which can apply parallel training, and use a novel neural network structure to handle feature extraction in a random graph topology. The main contributions of this paper can be summarized as follows:

- This paper proposes a new and efficient algorithm for automatic virtual network embedding based on deep reinforcement learning. In particular, a novel neural network based on graph convolutional networks is adopted in learning agent to automatically extract spatial features in an irregular graph topology (i.e., the substrate network). To the best of the authors' knowledge, this approach of feature extraction is the first time used in VNE problems.
- We use a popular parallel policy gradient training method to guarantee the efficiency of sampling training experiences and the robustness while optimizing the learning agent. Validation tests show that when other settings are identical, the parallel version of our algorithm can converge more quickly in training episode and has an advantage up to 5.8% over the single-processed version.
- A new reward function is precisely designed to match multiple optimizing objectives in the VNE problem which considers multiple factors including request acceptance, long-term revenue, load balance and policy exploration, leading to a better performance up to 6.3% comparing with the VNE models with traditional reward functions.
- Extensive simulation experiments have been conducted to validate the performance of the proposed solution. The results show that the proposed VNE algorithms with the above innovative designs can achieve up to 39.6% and 70.6% improvement on acceptance ratio and average revenue, respectively, in comparison with the existing state-of-the-art solutions. Moreover, the results also demonstrate that the proposed solution possesses good robustness, owing to different data distributions used in training and testing phases.

The remaining of this paper is constructed as follows: Section II describes the related work. Section III depicts the VNE problem along with the essential network components. The design and implementation of the VNE learning agent are presented in Section IV. Section V compares the performance of the proposed algorithm with the state-of-the-art ones, and finally, Section VI concludes this study.

II. Related Work

The VNE problem has been well discussed in the last decade. The first academic document focusing on this problem and its virtual network resource allocation counterpart is presented in [1]. Since then, many VNE algorithms have been proposed. Typically, the VNE problem can be formulated as a mixed-integer programming (MIP) problem along with predefined resource constraints. Chowdhury et al. [7] added some location constraints to limit the maximum distance between any two embedded virtual nodes and applied a round-based relaxation method to transform the MIP-formulated VNE problem into a linear programming (LP) problem for gaining exact solutions. However, this approach suffered from the requirement of excessive computational resources for a linear programming solution. Shahriar et al. [33] proposed an algorithm based on integer linear programming, which jointly focused on the full resource utility and request survivability. Although with a heuristic version that accelerates execution, its running time is still unacceptable and needs to be further reduced. Benkacem et al. [2] introduced a content delivery network as a service (CDNaaS) platform and addressed virtual slicing problem on the platform by formulating two MIP problems and applying bargaining game theory to achieve optimal tradeoff between cost efficiency and service quality. This work is mainly to ensure the quality of content-based service.

The majority of VNE algorithms use heuristic methods for approximate solutions with acceptable time cost. Cheng et al. [6] focused on topology-aware node ranking methods which sort substrate nodes and virtual nodes using the rules inspired by the Google's Pagerank algorithm [29] and then use a "big-to-big, small-to-small" matching strategy to embed virtual nodes onto substrate nodes that have similar ranking positions. However, the node ranking is fixed per network topology, which means the embedding decisions are hard to be optimized unless the ranking rules are changed. Soualah et al. [35] turned the VNE problem into a coordination game based on the identical interest game in game theory. Each substrate node is considered as a player that shares the same utility function and tries to achieve a Nash Equilibrium for optimal embedding solutions. However, this approach separates the node and link embeddings as individual games and then nests them together; it not only lacks the coordination between node and link embedding decisions, but also harms time efficiency. Gong et al. [16] addressed the VNE algorithm with a new cost metric to perform a greedy-based load-balancing embedding strategy among substrate nodes. However, the metric was defined with a certain scenario and the performance will probably drop as the environment changes. Lischka et al. [21] detected the isomorphic subgraph inside the substrate network and restricted the maximum path length of an embedded virtual link to accomplish the representing VNE topology embedding. This algorithm is based on the subgraph isomorphism detection which is also NP-hard, thus the time complexity still needs to be optimized. Dehury et al. [10] proposed an algorithm based on global and local fitness value functions to maximize the resource utilization given that requests are dynamically changing. However, to reduce the computational complexity, this approach considered the candidate embeddings in a subset rather than the full set of substrate nodes, resulting in the limitation of performance.

There exist several VNE algorithms using metaheuristic methods, as the VNE problem can be treated as a combinatorial optimization problem with a large search space. Zhang et al. [47] adopted particle swarm optimization as a stochastic global optimizer where each particle (i.e., possible embedding solution) iteratively made improvements following the embedding cost function, and a global evolution procedure finally took place to get a complete solution. Metaheuristic solutions usually follow static improvement steps which are designed manually by specialized human experiences; they are therefore not flexible for optimizing and being applied to other VNE scenarios. Yu et al. [44] adopted an evolutionary membrane computing mechanism to perform embeddings in a parallel manner. However, this approach was based on a sophisticated encoding mechanism for virtual network components with a manually-designed set of evolutionary rules, which causes the model inflexible and lack of automation.

In conclusion, the exisiting algorithms have several main drawbacks:

- The constraints, features and optimizing procedures in problem solving are manually determined, which are unchangeable throughout the running of algorithms. This indicates a heavy workload and a narrow improvement gap for optimization.
- Explicit objective functions and optimization targets may harm the flexibility and robustness of VNE algorithms. For example, embedding algorithms focusing on reducing cost will prohibit the solutions that use long substrate paths as virtual link embeddings and may cause performance drop on acceptance ratio.
- The execution time to generate VNE solutions is yet to be reduced. VNE is time-sensitive; if an algorithm cannot finish under the computational constraints, the revenue of service providers would be reduced, and the negative effects would be placed on the quality-of-experience of end users.

The RL framework has been proposed to solve decision-making problems by automatically interacting between a *learning agent* and an *environment*. At each time step t, the learning agent gets the *state* s_t by observing the environment status and generates an *action* a_t following its internal algorithms. The environment executes a_t, moves onto next state s_{t+1}, and sends *reward* r_t, which judges how good a_t is doing back to the agent. The primary aim of an RL algorithm is to maximize the estimation of discounted accumulative reward: $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$ instead of optimizing a manually defined objective function, where $\gamma \in (0, 1)$ is the discount factor that balances instant return and long term reward. With proper state, action and reward proposals, the VNE problem can fit the RL framework well, which is shown in Fig. 1.

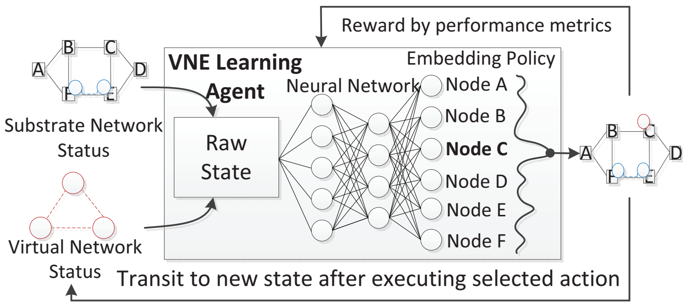


Fig. 1. The RL framework of a VNE problem.

Several papers in recent years explored the utilization of RL techniques in VNE algorithms. Haeri and Trajkovic [18] modelled the VNE problem as a Markov decision procedure and picked the embedding actions using Monte-Carlo tree search (MCTS) driven by a reward function based on the difference between embedding cost and revenue. However, MCTS needs to run a complete procedure for every single embedding decision, thus it makes the algorithm time consuming. In addition, the simple reward function is not enough for representing a complicated VNE problem. Mijumbi et al. [26], [27] managed the virtual network resource allocation using a tabular Q-learning algorithm [36] and a typical fully-connected neural network, using the percentage of free resources in each substrate network node as input features. However, the state and action of this algorithm are coarsely defined, which limits the search space and may cause suboptimal solutions. In addition, this work focused on dynamically adjusting the resources that have already been used by embedded VNs, which differs from the VNE problem that allocates substrate network resources to VNs from scratch. Yao et al. [43] used a 1-dimensional convolution neural network to extract substrate node features and trained the neural network by minimizing the cross-entropy loss which used the current selected action as a "correct label". This approach only leveraged the spatial information of network topology in a brief way and only allocated one single filter for a substrate node. Yuan et al. [46] used a Q-learning algorithm, designed a reward function related to the effect of virtual link embedding and applied unsupervised learning to update it. However, the features in the algorithm are manually obtained. Xiao et al. [41] used a deep-RL based model to formulate the problem of automatic deployment of service function chains and adopted a policy gradient based method to improve the training efficiency and convergence to optimality. Unfortunately, they did not take spatial features into account. Sciancalepore et al. [31] optimized on the joint of VNF orchestration and monitoring process based on a Q-learning algorithm, while the solution requires a huge complexity to solve the problem in an affordable time. Wang et al. [38] adopted a deep RL algorithm to dynamically allocate resources in 5G networks and achieve resource efficiency and end-to-end reliability. However, this approach did not take spatial features into account. Pham et al. [30] managed VNF forwarding graph embedding problem using deep deterministic policy gradient (DDPG) to fulfill resource demands from virtual nodes and service quality of virtual links (e.g., latency, loss rate, etc.). However, this approach used many auxiliary variables to denote actions which burdened the training process. Dolati et al. [11] converted an arbitrary VNR to a specific image representation and adopted general CNN for a DRL approach in solving VNE problems. The authors made an assumption that networks are having grid-like topologies to make the image representation easier to get, but it is not true in many other situations (e.g., tree-like and star-like topologies). Wang et al. [39] made a temporal-difference learning approach towards VNE problem solving. However, this approach used all possible states as the input of a neuron network, which increases the computational complexity.

We summarize the related works mentioned above in Table I. In addition, this journal paper is an extension of the previously published conference version [42]. The main extension of this journal version is summarized below:

- We provide a more comprehensive investigation on recent related VNE algorithms (published in 2019 and 2020), especially DRL approaches (e.g., the VNE algorithms based on DDPG, Temporal-Differential method, CNN-based DRL algorithm, etc.).
- We provide more detailed derivation and illustration on system modelling and add the pseudocode for algorithm implementation of critical steps, facilitating the understanding of the model and make it easier to follow.
- We introduce new (and realistic) network topology and new performance metrics (e.g., latency, node utilization and link utilization) for performance evaluation, strengthening the robustness of the proposed algorithm.
- The comparison experiments and analysis are massively expanded, such as using different neuron network structures, different reward functions, etc. The comparisons of model convergence are added. All these new results demonstrate the effectiveness of the proposed model under different network configurations and working conditions.

III. The Description of A VNE Problem

In this section, we describe the VNE problem in mathematical language for a formal definition. Meanwhile, we discuss some critical performance metrics and optimizing objectives that are widely used for evaluating VNE algorithms. To make the following descriptions easier to read, we denote some major notations in Table II.

A. Substrate Networks

A substrate network can be treated as a weighted undirected graph $G_s = (N_s, L_s, A^n, A^l)$, where N_s represents a collection of substrate nodes and L_s refers to a set of substrate links. A^n and A^l are node attributes (e.g., CPU processing capability, memory space and node reliability) and link attributes (e.g., bandwidth, latency and packet loss rate), respectively. For the sake of illustration, in this paper we consider the CPU processing capability of each substrate node as the node attribute and the bandwidth of each substrate link as the link attribute.

B. Virtual Network Requests

A virtual network can similarly be described as a weighted undirected graph $G_v = (N_v, L_v, R^n, R^l)$. Here, N_v and L_v are the set of virtual nodes and links, while R^n and R^l denote the virtual node and link requests for substrate resources, respectively. A virtual network request (VNR) can therefore be denoted as $VNR = (G_v, t_a, t_d)$, where t_a and t_d are the arrival and departure time of a VNR, respectively. When a VNR is generated with a certain network topology and arrives the substrate network at t_a, it demands the substrate resources characterized by R^n and R^l. If this demand can be satisfied by the available resources in the substrate network, the VNR will hold these substrate resources until t_d. If there is not enough resource available or the embedding algorithm cannot work out a solution, the VNR will be rejected or postponed.

C. The VNE Problem

The VNE problem can be defined as a mapping: $M : G_v(N_v, L_v) \rightarrow G_s'(N_s', L_s')$ from G_v to a subgraph of G_s, where $N_s' \subset N_s$ and $L_s' \subset L_s$. The mapping procedure can be decomposed into two stages: 1) the node mapping procedure for hosting virtual nodes on substrate nodes with sufficient resources, and 2) the link mapping procedure for assigning virtual links onto loop-free paths of the substrate network which satisfy virtual link resource requests. It is worth noting that virtual nodes from different VNRs may share the same substrate node, and a virtual link can not only share substrate links with other virtual links, but may also cross over multiple substrate links that form a substrate path between source and target nodes. In the crossing case, the substrate bandwidth a virtual link actually takes is more than the resources it initially demands, which is decided by the length of the crossed substrate path. Fig. 2 shows an illustration of virtual node sharing (blue node c and red node d) and link crossing (the link between blue nodes a and b).

D. Optimization Objectives

The purpose of a VNE algorithm is to make efficient utility of substrate network resources by performing smart VNR embedding decisions over time. Several key metrics have

TABLE I
THE SUMMARY OF RELATED WORKS

Author	**Methodology**	**Features**	**Spatial Features**	**Feature Extraction**	**DRL Usage**	**Problem Solving Time**
Chowdhury et al. [7]	MIP modelling and LP relaxation	CPU and bandwidth	Yes	No	No	Massive with larger networks.
Shahriar et al. [33]	Integer LP formulation with heuristic solver	mean substrate and virtual path lengths	Yes	No	No	Affordable
Dehury et al. [10]	MIP formulation	CPU, memory and bandwidth	Yes	No	No	Affordable
Lischka et al. [21]	Isomorphic subgraph detection	CPU and bandwidth	No	No	No	Affordable
Cheng et al. [6]	Node ranking	CPU and bandwidth	No	Yes	No	Affordable
Zhang et al. [47]	A variant of discrete particle swarm optimization algorithm	CPU, bandwidth and location constraint	Yes	No	No	Affordable
Soualah et al. [35]	Game theory	CPU and bandwidth	No	No	No	Extra time cost due to nested games.
Benkacem et al. [2]	Two integer LPs and bargaining game theory	Paid cost and streaming quality	Yes	No	No	Affordable
Yu et al. [44]	Dual parallel computation	CPU and bandwidth	Yes	No	No	Affordable
Mijumbi et al. [26]	Tabular Q-learning	Resource allocation, link delay and packet loss	Yes	No	No	Not mentioned
Mijumbi et al. [27]	3-layer neuron network	Resource allocation status	Yes	No	No	Costly due to a slow convergence to optimal.
Haeri and Trajkovic [18]	MCTS	CPU and bandwidth	No	No	No	Costly due to search procedure.
Yao et al. [43]	Deep RL approach	CPU and bandwidth	Yes	No	Yes	Affordable.
Yuan et al. [46]	Q-learning	CPU and bandwidth	Yes	No	No	Affordable.
Sciancalepore et al. [31]	Q-learning	CPU and bandwidth	Yes	Yes	No	Costly due to huge solution complexity.
Xiao et al. [41]	Deep RL and policy gradient training	Throughput and operation cost	No	Yes	Yes	Not mentioned.
Wang et al. [38]	Deep RL	CPU, memory and bandwidth	No	Yes	Yes	Computational overhead increases in larger scale networks.
Pham et al. [30]	DDPG	CPU and quality of service	No	No	Yes	Not mentioned.
Dolati et al. [11]	Deep RL	CPU and bandwidth	Yes	Yes	Yes	Not mentioned.
Wang et al. [39]	temporal difference	CPU and quality of service	No	No	Yes	Not mentioned.

TABLE II
MAJOR NOTATIONS TO DESCRIBE A VNE PROBLEM

Notation	**Description**
N_s	The full collection of network nodes in substrate network.
N_s'	The subset of N_s.
L_s	The full collection of network links in a substrate network.
L_s'	The subset of L_s
A^n	The node attributes (e.g., CPU processing capability, memory space and node reliability) of substrate nodes.
A^l	The link attributes (e.g., bandwidth, latency and packet loss rate) of substrate links.
N_v	The full collection of network nodes in a virtual network request.
L_v	The full collection of network links in a virtual network request.
R^n	The resource request of each virtual node corresponding to A^n.
R^l	The resource request of each virtual link corresponding to A^l.

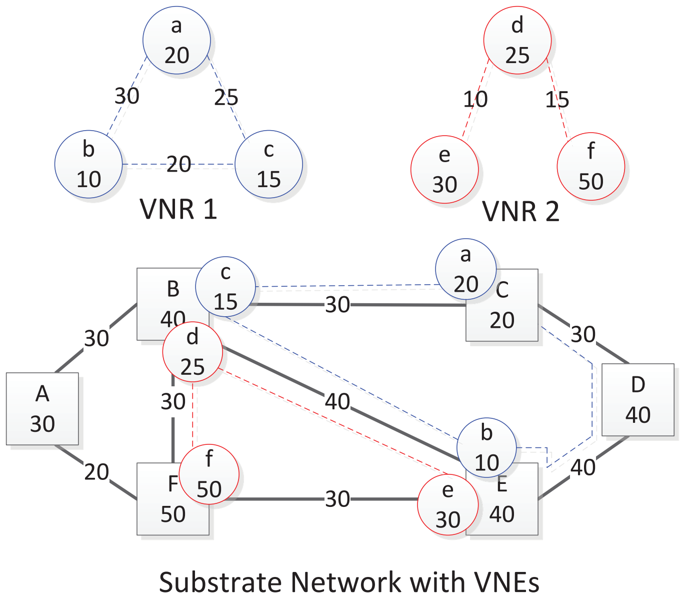


Fig. 2. A typical scene of virtual network embedding problem

been widely used for measuring the performance of a VNE algorithm, and thus they are the main objectives, in this study, a VNE algorithm will try to optimize.

1) Acceptance Ratio: The VNR acceptance ratio of the substrate network at time T can be defined as:

$$AC\_Ratio(T) = \frac{\sum_{t=0}^{T} NUM\_VNR\_S}{\sum_{t=0}^{T} NUM\_VNR} \tag{1}$$

where NUM_VNR_S and NUM_VNR are the number of successful VNR embeddings and the number of total VNR embeddings, respectively. A successful VNR means it can be embedded immediately without any violation of resource constraints. If a VNR cannot be embedded when it arrives, it will be discarded and will be considered as a failure. The embedding algorithm manages to have a high acceptance ratio to fulfill network requests as many as possible.

2) Long-term Average Revenue: In the perspective of InPs, the substrate network devices gain revenue in proportion to the scale of requests from SPs. The more resources SPs demand, the more infrastructure resources can be loaned out, and the more profit will be made. Hence, the resource requests of upcoming VNRs are related to the revenue, and therefore we denote the revenue of a successful VNR embedding by its resource demands, which is shown as:

$$Rev(G_v) = \sum_{n_v \in N_v} CPU(n_v) + \sum_{l_v \in L_v} BW(l_v) \tag{2}$$

where $CPU(n_v)$ and $BW(l_v)$ are the CPU and bandwidth requests from each virtual node and link in the virtual network G_v, respectively. The two attributes are normalized with the maximum to make the addition valid. The long-time average revenue at time T can be decided by:

$$Rev(T) = \frac{\sum_{t=0}^{T} Rev(G_v^t)}{T} \tag{3}$$

where G_v^t denotes the revenue of successful VNR embedding arrived at time t. A high long-term average revenue is desirable since it represents a high resource utility, leading to a better commercial profit.

3) Running Time: There is a common trade-off between performance and running time for every algorithm, and the approximate VNE solutions are usually preferable over exact ones. In practice, a VNE algorithm needs to stay online and processes VNRs in a time-critical manner since the VNRs may represent real-time network services. Thus, the time efficiency of VNE algorithms need to be guaranteed.

IV. THE SYSTEM MODELLING

In this section, we discuss the whole modelling procedure from model input to output. We firstly describe how to define critical RL elements in a VNE problem. Then, we present the procedure that the learning agent generates actions from scratch, including feature extraction, policy generation and training method, where novel technologies are applied. Finally, we discuss the implementation details.

A. Definition of RL Environment

There are three main components in an RL framework: state, action and reward. In what follows, we will illustrate each one in detail.

1) State Representation: The state representation in an RL framework defines the information an agent can acquire from the environment and serves as the raw input for the upcoming feature extraction phase. In a VNE problem, a state is the real-time representation of network status, which contains the features shown in Table III.

The first five features subtracted from the substrate network are the vectors with the length that is consistent with the number of substrate nodes, and the following three features come from the VNR which are scalars. The eight attributes are the most relevant and well-accepted ones for a VNE problem while keeping the model as concise as possible. Note that link features are not included in the state representation. Link features are possibly useful in solving VNE problems, but the number of links is usually far more than that of nodes in an arbitrary network topology. For the simplicity of state representation and the efficiency of computing, instead of explicitly using link features in the state representation, we introduce an implicit way to exploit link information in the following stages in Section IV-B1.

2) Action Definition: An action is a valid embedding process that allocates virtual network requests onto a subset of substrate network components. However, the number of subgraphs of a network topology grows exponentially as the node and link sizes increase. If we consider every possible subgraph as embedding actions, the action space will be computationally large and inconsistent per VNR, which is not desirable for the RL framework. Thus, we decompose a VNR embedding process into a sequence of virtual node embeddings.

At every single step, the learning agent focuses on exactly one virtual node from the current VNR, and it generates a certain substrate node to host the virtual node. If the substrate node keeps enough spare resource for this virtual

TABLE III
THE NETWORK FEATURES USED FOR REPRESENTING A STATE

State Representations	Description
S_CPU_Max	The maximum of the CPU resources over all SN nodes.
S_BW_Max	The max bandwidth of each substrate node. We define the bandwidth of a node as the sum of all links' bandwidth that directly link to this node.
S_CPU_Free	The amount of the CPU resources that are currently free on every substrate node.
S_BW_Free	The bandwidth resources that are yet to be allocated on all substrate nodes.
Current_Embedding	The (partial) embedding result of the current VNR. Each substrate node is set to 1 if it hosts a virtual node in the current VNR and 0 otherwise. This feature works as a mask to prevent virtual nodes in the same VNR from sharing one substrate node, as most previous works did.
V_CPU_Request	The number of virtual CPUs the current virtual node needs to fulfill its requirement.
V_BW_Request	The total bandwidth the current virtual node demands according to the current VNR.
Pending_V_Nodes	The number of unallocated virtual nodes in the current VNR.

node, the adjacent virtual links between this virtual node and other already-embedded virtual nodes will be automatically processed following a hybrid search procedure: for the two substrate nodes that host one end of a virtual link respectively, the agent will firstly try the shortest substrate path; if failed, the agent will then search in a set of edge disjoint paths that connect the two substrate nodes. This link embedding procedure is a typical trade-off mechanism between performance and efficiency. Fig. 3 shows how the hybrid search works when embedding virtual links between node A and node D with the shortest path $(A-B-C-D)$ and edge-disjoint path set $((A-B-C-D),(A-F-E-D))$.

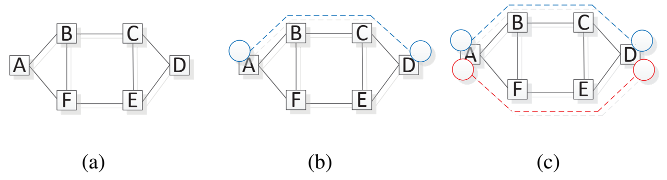


Fig. 3. An example illustration for the hybrid search. (a) is a substrate network, (b) is a link embedding between A and D using the shortest path, and (c) is another link embedding using a path which is edge-disjoint with the shortest path if the shortest path is full.

Comparing with the shortest path finding method, this search procedure explores more candidate paths and thus increases the possibility of successful embeddings. In addition, this procedure reduces computational requirements than the full search (e.g., breadth-first search), leading to better time efficiency. If the current virtual node is the first node of the corresponding VNR, there is no link to be processed after this node is embedded and the hybrid search will not be executed.

The action space of a VNE problem is then represented as the set of substrate nodes. Once the substrate network topology is set, the shape of state representation and action space is also fixed. With this definition of action space, the complete two-stage embedding approach is shown in Algorithm 1.

3) Reward Description: Instead of following an explicit objective function (e.g., linear programming) or a predefined label (e.g., supervised machine learning) to optimize the behaviour of an algorithm, an RL learning agent improves its performance by constantly receiving reward function from the external environment. Unlike other methods mentioned above, the reward function is not a definitive indicator of "correct"

Algorithm 1 Virtual Network Embedding Procedure

```
Input:
    The VNR topology $G_v$;
    The substrate network topology $G_s$;
    The already embedded virtual node list $l$;
    The substrate node list $s$ that hosts nodes in $l$;
    The currently processing virtual node $n$;
    The selected action (i.e. substrate node) $a$.
Output:
    The embedding result of action $a$.
1: if $S\_CPU\_Free[a] < V\_CPU\_Request[n]$ then
2:     return $ACTION\_FAILED$;
3: end if
4: Embed virtual node $n$ onto substrate node $a$;
5: for $i = 0$; $i < length(l)$; $i++$ do
6:     if $(l[i], n) \in EdgeSet(G_v)$ then
7:         Find a substrate path $p$ in $G_s$ that links $s[i]$ and
           $a$ following given search type: shortest-path, full or
           hybrid;
8:         for all substrate links $e$ in path $p$ do
9:             if $BW\_Free(e) < BW\_Request((l[i], n))$ then
10:                Undo all previous embedding actions and re-
                   lease all resource secured by the current VNR;
11:                return $ACTION\_FAILED$;
12:            end if
13:        end for
14:        Embed virtual link $(l[i], n)$ onto substrate path $p$;
15:    end if
16: end for
17: return $ACTION\_SUCCESSFUL$;
```

actions, but only tells the agent how good the current action is doing relatively. To maximize the estimation of discounted accumulative rewards, the agent may give up the actions with the best current reward to get a better long-term performance. Typically, a successful action (e.g., all virtual network components are embedded without any resource constraint violation) is treated to be good and returns a positive reward to enforce the probability that the current action is selected. Otherwise, a failed action (e.g., the remaining resource cannot fulfill the VNR requirement or the execution is running out of time) should be prevented, thus it will receive a negative reward to let the agent search alternative decisions. The reward function steers the optimizing direction of an VNE algorithm, which is carefully designed in section IV-C1.

B. Learning Algorithm

The learning agent is responsible for generating appropriate policies which are the probability distribution over actions with the raw inputs defined in Section IV-A1. To achieve this, the agent needs to use input features efficiently, go through various states and actions sufficiently, and optimize the policy iteratively. Since the number of states are infinite (e.g., the resource-related features are continuous real numbers) which are impossible to record, an approximator based on neural network with a set of trainable parameters is applied to manage the feature extraction and the policy generation using raw state inputs. To train these parameters of the approximator, a policy gradient training algorithm [37] is adopted to improve the embedding policy. In what follows, we introduce the most critical techniques used in our learning agent. The architecture of our learning algorithm is shown in Fig. 4.

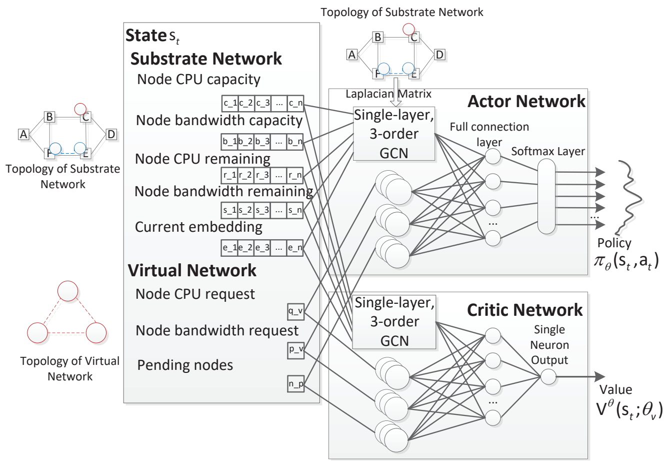


Fig. 4. The learning algorithm used in the proposed RL agent.

1) Feature Extraction: Convolutional neural network (CNN) extracts advanced features automatically and has been widely used in the fields including image and text processing. A critical contribution made by CNN is that it uses the kernels with different sizes to manage the spatial features in different input scales. However, the typical object a CNN can deal with is Euclidean (i.e., all input elements are ordered neatly in rows and columns, as shown in Fig. 5a), which is not applicable for an arbitrary network topology depicted in Fig. 5b.

In VNE solutions, the spatial features of an (irregular) substrate network topology are critical. To manage spatial features more effectively while preventing our model from accepting excessive input features such as explicit link features, we need an alternative approach. For auto feature extraction in non-Euclidean domains, based on spectral graph theory [8], which mainly uses the Laplacian matrix and orthogonal factorization to characterize the spatial features of a certain graph topology, the convolution on a graph topology is proposed by [9] and has been used efficiently on semi-supervised learning as Graph

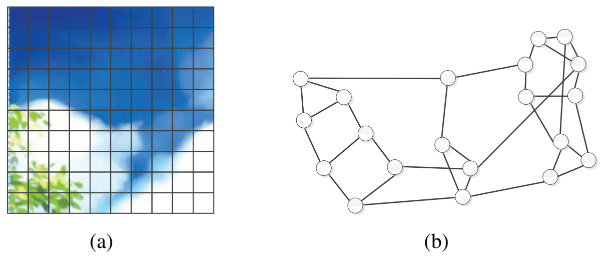


Fig. 5. Examples of Euclidean and non-Euclidean graph. (a) is an Euclidean graph (an image in pixel squares) and (b) is non-Euclidean (the ARPANET topology).

Convolutional Network (GCN) [20]. The graph convolution defines the Fourier transformation in an n-dimensional space, which shares the idea with the traditional (function) Fourier transform: a real-value function can be decomposed into a set of functions that are orthogonal to each other. Similarly, a vector in an n-dimensional space can also be represented as a set of orthogonal vectors (which is called the orthogonal basis of this space). Fig. 6 shows this intuition.

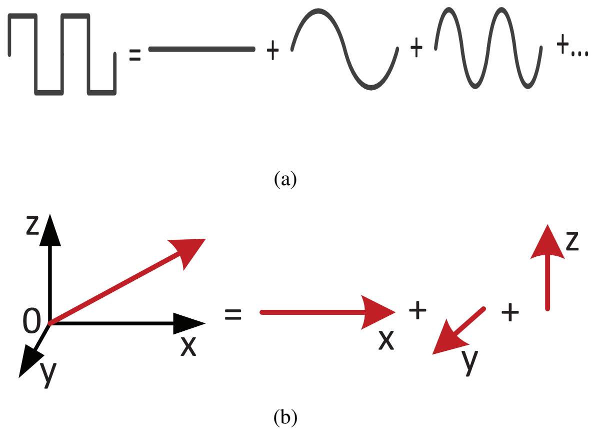


Fig. 6. The similar intuition of Fourier Transform of normal functions and spatial vectors. (a) shows the decomposition of a square wave into a set of orthogonal functions, while (b) depicts the decomposition of a 3-dimensional vector into orthogonal basis vectors.

Assume a graph G has n nodes, and the features from each node are gathered together as a feature vector $x \in \mathbb{R}^n$. To acquire the graph-specific orthogonal basis of this n-dimensional space, one approach is to factorize the (normalized) Laplacian matrix L of graph G since L is semi-definite. The eigenvectors of L can therefore form an orthogonal basis U of the n-dimensional space, and the Fourier transformation of the vector x, represented by $\hat{x}$, on graph G can be computed as:

$$\hat{x} = U^T x \tag{4}$$

The inverse Fourier transform is subsequently defined as:

$$x = U\hat{x} \tag{5}$$

With the convolution theorem [4] where the Fourier transform of a convolution of two signals is the pointwise product of their

Fourier transforms, the convolution of x and a convolution kernel y on graph G can be stated as:

$$(x * y)_G = U((U^T y) \odot (U^T x)) \tag{6}$$

where $\odot$ is the element-wise Hadamard product. By adjusting parameters in y, the convolution is able to output various results. For a better locality and spatial feature extraction, in our implementation we set the kernel filter $y = \sum_{k=0}^{K} \alpha_k \Lambda^k$, where α_k is trainable parameters of kernel filters, and $\Lambda \in \mathbb{R}^{n \times n}$ is a diagonal matrix with the eigenvalues of L . With this kernel, the final output of GCN is:

$$y_{output} = \sigma(\sum_{k=0}^{K} \alpha_k L^k x) \tag{7}$$

where σ is the activation function of the neural network. The order index K indicates the locality, which means the recognization field of a network node can reach the neighbouring nodes up to K hops under this kernel. For a detailed interpretation, the reader can refer to Section 2 in [9].

We therefore use a single layer GCN with an order index of three to manage the automatic feature extraction on the SN topology. If the order index is too small, the center node will be short of adjacent feature information since it can only recognize few neighbors. If it is set too large, all nodes in a graph will have an unwillingly similar representation since nearly every node is recognized and shared as neighbors no matter how far the distance is. Meanwhile, as Equation 7 describes, a GCN with a higher order will increase the computational overhead greatly. The filter size (i.e., the number of extracted features from raw state input through GCN) is set to 60, and each feature generated by a filter is a vector with the length of $Size(N_s)$. The original features from a VNR are passed through a full connection layer separated from GCN. The extracted features from a substrate network and a VNR are merged together and form a matrix with a shape of (60, $Size(N_s)$) as the final extracted features.

2) Policy Generation: A policy is the probability distribution over candidate actions under a certain state $\pi : \pi(s_t, a_t)$, which shows the probability to perform an action a_t in s_t. We firstly transform the extracted features into a single column vector and pass it through a full connection layer to make the output size consistent with the size of substrate network nodes. The parameter size of the full connection layer is (60* $Size(N_s)$, $Size(N_s)$). To interpret the output as a probability distribution, we adopt the well-known softmax layer [3] after the output of the full connection layer. The softmax function turns an arbitrary real vector into a vector with a range of (0,1) on each index, which also has a sum of 1 without changing the relative order in the previous vector. The learning agent now is capable of selecting actions using this probability distribution.

It is worth noting that all resource-related features, defined in Section IV-A1 and used in the state representation, are normalized into [0, 1] interval, with the $Max(S\_CPU\_Max)$ and $Max(S\_BW\_Max)$ as two benchmarks. If the feature representations are not normalized, the output probability distribution will be prone to extreme inputs and become hard for optimization.

3) Parallel Policy Gradient Training: After the environment executes the action sampled from the policies generated by the learning agent, the corresponding reward signal will be sent back to the agent, and a single process of experience sampling is completed. To optimize the policy and train the network parameters using the sampled experience, we select Asynchronous Advantage Actor-Critic (A3C) algorithm [28], an improved version of actor-critic based policy gradient method [37] which makes two major improvements. First, it uses *advantage function* instead of the mere state-action value function which can reduce the variance of training experience. Second, it adopts a "master-worker" parallel training scheme to improve the sampling efficiency. Two networks are constructed in this algorithm: the *actor network* (with a set of parameters θ), which generates embedding policy π_θ, and the *critic network* (with a set of parameters θ_v), which generates the estimation of values in different states $V^{\pi_\theta}(s_t; \theta_v)$ and helps compute the advantage function. Typically, these two networks share a similar structure except for the output layer.

The traditional policy gradient method uses the following gradient of accumulative discounted expected rewards:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)] \tag{8}$$

where $Q^{\pi_\theta}(s, a)$ is the state-action value function that estimates the expected long-term return of action a derived from policy π_θ under state s. However, the variance of $Q^{\pi_\theta}(s, a)$ is usually high, making the training process unstable. In addition, if a certain state is in a good position, the Q value will stay high regardless of actions picked under this state, which means the difference between actions is overlooked. Therefore, the advantage function, which represents the difference between the expected reward if we deterministically choose action a and the average state value, can be computed as:

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \tag{9}$$

where $V^{\pi_\theta}(s)$ is the state value function that estimates the accumulative return under state s. The advantage function indicates how better is the current action a from the "average action" derived from the corresponding policy under a certain state s, and the variance while training can be reduced without changing the bias ($V^{\pi_\theta}(s)$ has nothing to do with the selected actions). Meanwhile, the difference among actions under the same state is more significant. In practice with a given experience (s_t, a_t, r_t, s_{t+1}), the estimation of $Q^{\pi_\theta}(s, a)$ is computed as $r_t + \gamma V_\theta^\pi(s_{t+1})$ based on Bellman Equation [36]. Now $A^{\pi_\theta}(s, a)$ is subsequently estimated as $r_t + \gamma V_\theta^\pi(s_{t+1}) - V^{\pi_\theta}(s)$ and replaces $Q^{\pi_\theta}(s, a)$ in Equation 8.

With the help of advantage function, the update of actor network can follow the policy gradient training method below:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log \pi_\theta(s_t, a_t) A^{\pi_\theta}(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)) \tag{10}$$

where α is the learning rate, and $\pi_\theta(\cdot|s_t)$ is the policy over all the actions under state s_t. The gradient of the parameter set θ indicates the direction to adjust the network parameters that increase the probability $\pi_\theta(s_t, a_t)$, and the advantage function value (along with the learning rate) decides how far this step should be taken by the learning agent through the gradient direction. As the learning agent gradually experiences more training episodes, it reinforces the action with better empirical reward under certain states. Meanwhile, $H(\cdot)$ is the entropy of the policy at each time step. This entropy term is used as the regularization to encourage exploration and prevent the algorithm from trapping into local optima by pushing the policy to a more uniform distribution. The decaying parameter β is set to be large at the beginning, and is gradually shrinking during the training phase. In our implementation, β is set to 0.5 at the start of training and gradually shrinks by 50% in every 10000 training episodes.

The critic network can also learn a better evaluation by following the standard Temporal-Difference (TD) method [36]:

$$\theta_v \leftarrow \theta_v + \alpha' \sum_t \nabla_{\theta_v} (r_t + \gamma V_\theta^\pi(s_{t+1};\theta_v) - V_\theta^\pi(s_t;\theta_v))^2 \quad (11)$$

where $V_\theta^\pi(\cdot;\theta_v)$ is the estimation of a value function under a certain state, and α' is the learning rate of the critic network. The intuition of standard TD method is to use the estimated value function of next step instead of the true value function (since the agent takes one more step in the environment, it generally moves one step near the true value function) and minimizes the square loss between the two steps.

The training from episodic experiences faces two main challenges. First, the experience taken from the environment usually comes slow, and the learning agent has to wait for the environment to perform actions before training steps. Second, the $\{s_t, a_t\}$ pairs in one trajectory (i.e., starts from scratch to a terminal state — succeed or fail) are highly correlated, degrading the robustness and efficiency of the training process. To overcome these shortcomings, A3C uses the parallel training to speed up the training process while enhancing its robustness. We use 24 worker agents to individually collect $\{s_t, a_t, r_t, s_{t+1}\}$ experiences from their own environments and send them to one central agent which is responsible for training and updating network parameters when a sample trajectory is completed. The 24 network environments are independent when performing the training of embedding actions (i.e., the actions in one environment will not affect the resource utilizations in other different environments). In other words, it is similar with creating 24 copies of the network to get more embedding experiences while training the policy. In case of the realistic network applications where multiple substrate network deployments are costly, we can investigate the behaviors of VNRs (i.e., the frequency of request arrivals, the distribution of resource requests, etc.) of realistic network users and "mock" them in multiple simulation networks. For this case, the simulation experiences help train the policy together with realistic experiences. The simulation experiences only affect in training episode or when the policy needs to be updated. The trained policy will then be used online. These 24 worker agents also share one same copy of actor-critic network parameters acquired from the central agent. The central agent collects the various experiences as a minibatch from the worker agents and train the central network. Once this minibatch finishes training, the network parameters are updated and passed to all the worker agents for the next collection step. The parallel training procedure is shown in Algorithm 2 and Algorithm 3.

Algorithm 2 Parallel Training Algorithm - Master

```
1:  Initialize the $actor\ network$ and $critic\ network$;
2:  Initialize the number of workers $NUM\_WORKERS$;
3:  for $i$ in $NUM\_WORKERS$ do
4:      Create a $worker$ agent $w[i]$ with a same copy of
        $actor\ network$ and $critic\ network$;
5:  end for
6:  while TRUE do
7:      for $i$ in $NUM\_WORKERS$ do
8:          Collect the experiences generated by $worker[i]$;
9:      end for
10:     Adjust the parameters in both $actor\ network$ and
        $critic\ network$ using previously collected experiences
        under policy gradient training method;
11:     for $i$ in $NUM\_WORKERS$ do
12:         Push the newest version of $actor\ network$ and
            $critic\ network$ to $w[i]$;
13:     end for
14: end while
```

Algorithm 3 Parallel Training Algorithm - Worker

```
1:  Initialize the $actor\ network$ and $critic\ network$;
2:  Initialize the independent $environment$ for VNE;
3:  while TRUE do
4:      Receive the parameters of the $actor\ network$ and
        $critic\ network$ from $master$;
5:      Sample a trajectory from the $environment$ using the
        copied network;
6:      Send the trajectory as $\{s_t, a_t, r_t, s_{t+1}\}$ experience tuples
        to $master$;
7:  end while
```

By simultaneously activating multiple experience collectors, the waiting delay of learning agent for experience sampling has largely shortened. Besides, the cental agent uses the experiences from independent sources, and therefore breaks the correlation among state,action pairs in one trajectory.

C. Implementation

1) Reward Shaping: Generally, a return positive reward is returned for a successful embedding and a negative one returns vice versa. However, successful actions themselves may also vary as they may lead to many possible state representations, and thus make the long term accumulative rewards different. To make the difference between "slightly good" and "really good" actions, we need to design the reward function more precisely. This procedure is called reward shaping.

To achieve high acceptance ratio, we should encourage successful embedding actions. For an action that satisfies the virtual resource requirement (i.e., the chosen substrate node has enough resources to host the current processing virtual node), we set an initial reward of 100. Alternatively, we set a reward of -100 for an action that cannot fulfill the resource requirement. Recall from Section IV-A2 that we have decomposed a VNR embedding into a sequence of virtual node embedding actions, hence we add a discount factor to this part of reward function. The reward due to the acceptance result is therefore:

$$r_a = \begin{cases} 100\gamma_a & a_t \text{ is successful} \\ -100\gamma_a & \text{otherwise} \end{cases} \tag{12}$$

where γ_a is the discount factor that starts from $(1/Size(N_v))$ and gradually increases to 1 when the last node of a VNR is in processing, and $Size(N_v)$ means the node size of a VN topology. For instance, if a VNR has 5 virtual nodes, then $\gamma_a = 0.2$ for the first node, $\gamma_a = 0.4$ for the second, and so on. This is because latter nodes have less embedding options and available resources than former nodes, thus they are more important and deserve a larger weight.

In addition, the learning agent needs not only to make successful, but also cost-efficient actions. A better embedding policy will consume less substrate resources (especially substrate link resources, by making virtual links lay on shorter substrate paths) when processing the same VNR. We therefore add another factor into the reward function:

$$r_c = \frac{\delta(revenue)}{\delta(cost)} \tag{13}$$

where $\delta(revenue)$ and $\delta(cost)$ are the newly-added revenue and cost due to the current action compared with the last step, respectively. Obviously the cost is always no less than the revenue, so this factor has a range of (0,1]. Even an action only brings minor revenue (e.g. the resource demands of the given VNR are little), there is also a possibility to generate a large reward if it effectively reduces the consuming substrate network resources.

Load balancing is also a critical feature in virtualized networks. To balance workload among substrate network nodes, we encourage the learning agent to pick substrate nodes with more spare resources. Thus, another factor reflecting this point is added to the reward function:

$$r_s = \frac{S\_CPU\_Remaining[a]}{S\_CPU\_Max[a]} \tag{14}$$

where a is the selected action representing the corresponding substrate node.

If the trained policy is trapped into a sub-optimal and becomes more likely to keep its current decision, it will lose the chance to further improve itself due to lack of exploration. To address this issue while training, we should avoid the embedding policy that repeatedly generates the same actions and encourage the policy to move onto the actions that have not been picked up in a while. Therefore, we use *eligibility trace* for every action i at time step t:

$$egb\_trace_t[i] = \begin{cases} \gamma_e(egb\_trace_{t-1}[i] + 1) & i == a_t \\ \gamma_e egb\_trace_{t-1}[i] & \text{otherwise} \end{cases} \tag{15}$$

where γ_e is the decay factor that shrinks the eligibility trace a little in each time step, which is set to 0.99 in our implementation. This approach guarantees that frequently-picked actions will receive constant inspiration and keep a high value in a while, and unpicked actions will gradually decay to 0. We use the eligibility trace as a divider in the reward function, making the infrequently-picked actions more preferable.

Finally, the reward function of the action a_t is given as follows:

$$Reward[a_t] = \frac{r_a r_c r_s}{egb\_trace[a_t] + \epsilon} \tag{16}$$

where ϵ is a small positive number to avoid the denominator being zero.

2) Global Training Settings: A summary of the training settings is presented in Table IV. We use a fixed setting of parameters to randomly generate VNRs for training the algorithm. We notice that the features S_CPU_Max and S_BW_Max remain the same once the substrate network is fixed, and we only use them to determine the action our learning agent made during the training phase, so we can subtract these two features to reduce the model complexity. The whole network architecture is implemented using Tensorflow [12]. The server used for training has a 32-core CPU and a memory capacity of 128GB.

TABLE IV
THE PARAMETER SETTINGS IN THE TRAINING PHASE

Parameter Name	Parameter Value
Shape of the raw inputs from substrate network	(3,$Size(N_s)$)
Shape of the raw inputs from VNR	(3,1)
The learning rate of actor network	0.00025
The learning rate of critic network	0.0025
The resource requests of virtual nodes*	[15,30]
The resource requests of virtual links*	[15,30]
The node size of each VNR*	[2,10]
The lifetime of each VNR*	[2000,3000]

* All uniformly distributed.

V. Performance Evaluation and Analysis

In this section, we first describe our evaluation environment, and then compare the proposed algorithm with the state-of-the-art VNE algorithms under various circumstances to show the effectiveness and robustness of our algorithm. For the sake of clarity of illustration, we name the proposed algorithm as "$A3C + GCN$" in the following performance evaluation.

A. Evaluation Settings

To facilitate the comparison and evaluation of multiple algorithms under the same platform, a VNE simulator is implemented. We generate a random substrate network topology following the Waxman random graph [40] using the parameters $\alpha = 0.5$ and $\beta = 0.2$; this approach of topology generation has been commonly used in previous works [45] [6]. Following this approach, a substrate network with 100 nodes and about 500 links is generated, which can represent a medium-sized Internet service provider (ISP). The CPU and bandwidth of the substrate network and links are uniformly distributed between 50 and 100 units. Following the previous research [23], the VNRs are generated following a Poisson process. As many previous works did, the expected arriving rate is 4 VNRs per 100 time units. In addition, each VNR has an exponentially distributed lifetime with an average of 500, and each VNR's node size is uniformly distributed from 2 to 10 virtual nodes. The initial virtual node and link resource requests in a VNR are uniformly distributed from 0 to 30. Each pair of virtual nodes has 50% chance to form a link. The testing phase for each group of evaluation lasts 50,000 time units, so there are about 2,000 VNRs per evaluation. These basic evaluation settings are fixed except for three particular ones: the arrival rate of VNRs, the distribution of virtual nodes and link resource requests in a VNR, and the distribution of each VNR's node size. In the following subsections, we separately change these three settings and form three different test groups to evaluate various VNE scenarios.

In the testing phase, the learning agent only uses the actor network to generate embedding policy and selects the action with the highest probability from the set of substrate nodes possessing enough node resources to host the current virtual node. The agent we use for testing has been trained for nearly 72 hours and been experienced 70,000 episodes for training, with approximately 1,680,000 different VNRs.

B. Comparable Algorithms and Evaluation Metrics

We select five algorithms including R-ViNE [7], D-ViNE [7], GRC [16], MCVNE [18] and NodeRank [6], which can cover most perspectives of the existing algorithms. We use the public VNE simulator VNE-Sim [17] to collect the performance of these algorithms. The description of each algorithm is shown in Table V.

We use the three optimization objectives introduced in Section III-D to compare among embedding algorithms under different testing situations. Note that the resource metrics for computing average revenues are normalized into (0,1] interval.

C. Arrival Rate Tests

For the first group of tests, we are interested in the effect of arrival rates of VNRs. The virtualized network faces frequently-arriving VNRs at busy time and slowly-arriving VNRs at non-busy time. We therefore change the arrival rate of VNRs to mimic this changing situation, from an average of 4 arrivals per 100 time units to 20 arrivals with an increasing step size by 2.

Fig. 7 shows the acceptance ratio and the long-term average revenue of the total six algorithms, where our proposed algorithm leads NodeRank, MCVNE, GRC, D-ViNE and R-ViNE with a gap of 10.0%, 12.5%, 22.7%, 26.6% and 23.2% on acceptance ratio, and 15.6%, 15.7%, 27.9%, 30.7% and 28.1% on average revenue. The results also show that when the arrivals of VNRs are low, all the algorithms perform quite well. However, as the arrival rate keeps increasing, our algorithm (i.e., A3C+GCN) and the NodeRank algorithm keep a relatively better performance, where A3C+GCN outperforms NodeRank, and the performance of the other algorithms drops quickly.

The average revenue rises at the first several tests and then stays steady. This is because at the first several tests, most VNRs are successfully satisfied with all algorithms, and the increase in VNR arrival rate brings the expansion on average revenue. After those tests, the drop of acceptance ratio achieves a balance with the increase of revenue due to the more densely-arrived VNRs. Different from the other four algorithms, the drop trend of A3C+GCN and NodeRank algorithms on acceptance ratio is not that severe, and thus the balance constantly stays on the side of revenue increment.

Though our algorithm is slightly worse than MCVNE algorithm at the beginning in acceptance ratio, it outperforms the other algorithms in long-term average revenue. This implies that our algorithm can perform better embedding quality and make more efficient use of substrate resources.

D. Resource Request Tests

For the second group of tests, we observe the performance of these six algorithms under various resource request modes. Different network services have different modes of resource requirements (e.g., computation-intensive and communication-intensive tasks demand more node resource and link resources, respectively). To simulate this situation, we change the node and link resource request distribution from a [0,30] uniform distribution to [0,100]. In each step, we rise the upper bound by 10.

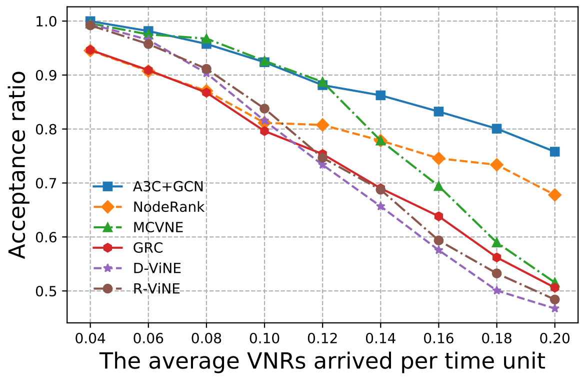
From the results shown in Fig. 8, we can compute the advantage of our algorithm over NodeRank, MCVNE, GRC, D-ViNE and R-ViNE, which are 29.4%, 1.8%, 34.1%, 23.6% and 9.8% improvement on acceptanve ratio, and bring 29.1%, 2.8%, 36.5%, 22.5% and 12.7% more average revenue. We also observe that all the algorithms suffer a huge loss on acceptance ratio when the virtual resource requests keep growing. As the resource request increases, the substrate network components gradually become much harder to host virtual network components since they are short of capacity for allocation, and the former embedding policies which fits VNRs with lower resource requests may not perform well. Thus, the algorithms need to find alternative solutions where candidate actions are limited due to the resource constraints, following by inevitably more embedding failures.

In addition, the average revenue appears an "ascending-descending" trend with the turning point at an upperbound of 50 or 60 for most algorithms. Although the acceptance ratio is decreasing, all the algorithms perform a relatively high acceptance level on the first 4 or 5 tests. As the resource requests increase test by test, the average revenue keeps rising. But after then, the algorithms accelerate to fall down on acceptance ratio, and the potential revenue increase brought by the growth of resource requests fails to offset this effect, bringing the total average revenue falls down.

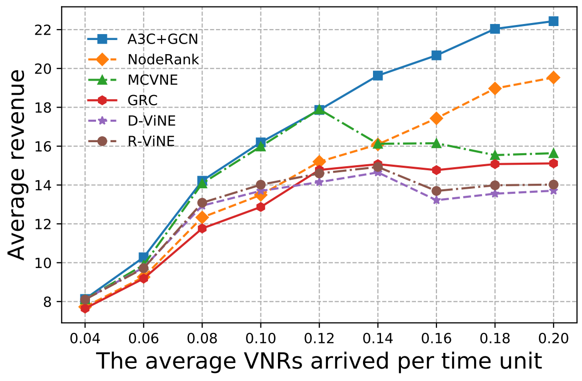
Back to the comparison, we discover that our algorithm again outperforms the other algorithms in a wide range of different resource request distributions. It is worth noting that when training the learning agent, we only use one resource request distribution which is shown in Table IV. This means our algorithm shows good robustness over different request

TABLE V
THE EXISTING ALGORITHMS FOR COMPARISON

VNE Algorithm	Description
D-ViNE [7]	Use a deterministic rounding-based approach to attain a linear programming relaxation of the MIP that corresponds to the VNE problem, aiming to minimize the cost of VNRs.
R-ViNE [7]	Same as D-ViNE, except for its rounding approach which is randomly decided.
NodeRank [6]	A node-ranking based algorithm that inspires from the intuition of Google's PageRank algorithm.
GRC [16]	A node-Ranking based algorithm that manages global resource capacity.
MCVNE [18]	A reinforcement learning-based algorithm that uses Monte-Carlo MCTS to search the action space.

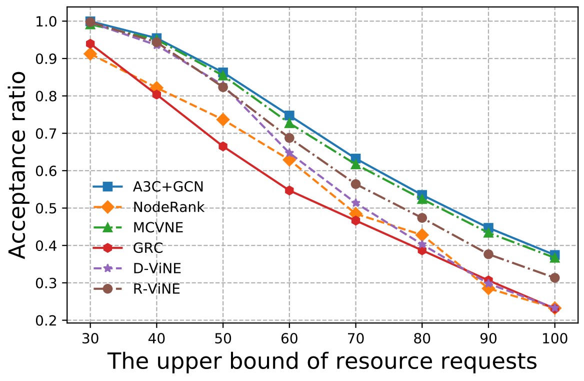


(a)

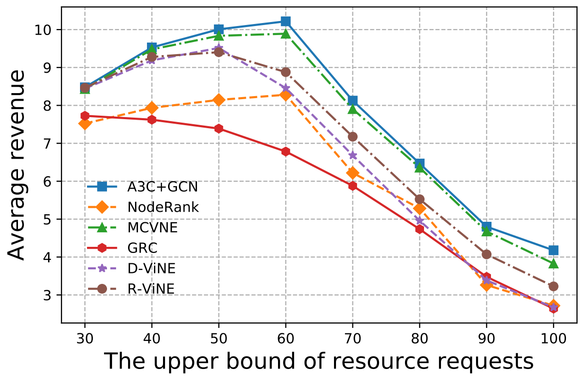


(b)

Fig. 7. The testing results of different VNR arrival rates: (a) represents the acceptance ratio and (b) depicts the average revenue.



(a)



(b)

Fig. 8. The testing results of various virtual resource requests: (a) represents the acceptance ratio and (b) depicts the average revenue.

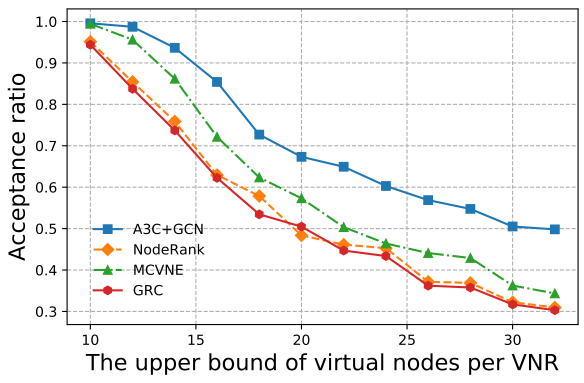
situations. The performance of MCVNE is pretty close with our algorithm in this test group.

E. Node Size Expansion Tests

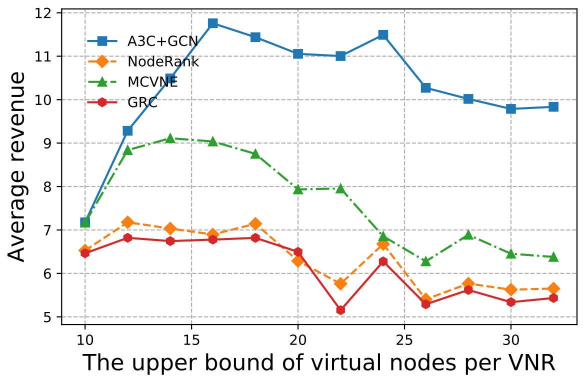
For the third group of tests, we focus on the effect of VNR sizes. Typically, group or enterprise users adopt larger network services, and individual users utilize services with less virtual nodes. Hence, to simulate various VNR sizes, we increase the number of virtual nodes in a VNR from a uniform distribution of [2,10] to [2,32], rising the upper bound by 2 in each step.

From the results shown in Fig. 9, our algorithm outperforms NodeRank, MCVNE and GRC with an average of 36.4%, 22.0% and 39.6% on acceptance ratio, and 64.2%, 36.8% and 70.6% on average revenue, respectively. It is obvious that as the size of each VNR increases, the acceptance ratio of each algorithm drops in some degree. This is because each VNR must be successfully embedded as a whole; a bigger VNR means more chances to fail in intermediate steps of embedding. In addition, more nodes in a single VNR limit the candidate action space since two virtual nodes in the same VNR cannot share one particular substrate node, which is mentioned in Section IV-A1.

The average revenue of NodeRank and GRC stays at a relatively low level, comparing with the other two algorithms.



(a)



(b)

Fig. 9. The results by changing the distribution of virtual node number in a VNR: (a) represents the acceptance ratio and (b) depicts the average revenue.

TABLE VI
THE AVERAGE RUNNING TIME OF A SUCCESSFUL EMBEDDING IN SECONDS

	Arrival Rate Tests	**Resource Request Tests**	**Node Size Expansion Tests**
A3C+GCN	0.219	0.227	0.476
NodeRank	0.103	0.125	0.388
GRC	0.086	0.079	0.135
MCVNE	1.815	1.893	9.649
D-ViNE	23.778	21.488	-*
R-ViNE	22.181	21.925	-*

* Runs in excess of computing resource and unable to get a result.

This is potentially because that the acceptance ratio of NodeRank and GRC drops quickly, reflecting their shortage when processing VNRs with complex topologies. Our algorithm reaches the highest performance again. Recall from Table I that we only use a single distribution of VNR node size in the training of our model, which reinforces the conclusion that our algorithm is robust against different changes of VNR topologies.

An interesting fact is that, unlike previous test results where the advantages on acceptance ratio and average revenue are quite similar, the advantage of our algorithm on average revenue in this test group is much more obvious than that of acceptance ratio. Though the computation of average revenue over different works may vary, it follows the same procedure in one particular paper, which makes the fact of comparison more hard to explain. This is potentially because that VNRs with fewer virtual nodes are easier to be embedded, and all algorithms can perform well on them. But when VNRs are with more nodes (and thus more links) which can provide much more potential revenues, our algorithm shows a better policy to accept them compared with the other algorithms, leading to a larger revenue. Therefore, the boost of average revenue becomes more significant than that of acceptance ratio.

Note that this group of tests does not show the results of D-ViNE and R-ViNE algorithms. We will discuss this in Section V-F.

F. Average Running Time Statistics

The average running time describes the average time cost that a VNE algorithm processes a complete VNR. The statistics are shown in Table VI. This table indicates that our algorithm, NodeRank and GRC run relatively faster, while MCVNE, D-ViNE and R-ViNE take much longer time on solving a VNE problem. The short running time of NodeRank and GRC may possibly be because that these two algorithms do not contain any space for modification following a static greedy mechanism, which indicates that the embedding solution is fixed once the VNR and the SN topology are revealed and no additional computation is required. However, this is not desirable for achieving an efficiency-performance balance.

Changing the resource request of a VNR does not necessarily burden the solving process, but adding more nodes into a VNR definitely increases the computation complexity. When the node size of a VNR increases (e.g., in the node size expansion tests), the overall processing time of a single VNR also rises since the steps for node embedding increases (for the two-stage algorithms) or the constraints become more complicated (for the MIP-based algorithms). Table VI does not show the performances of D-ViNE and R-ViNE in node size expansion tests since they cannot provide any results when processing larger VNRs. Meanwhile, MCVNE receives a huge slow down compared to the other test groups in the node size expansion test.

Despite of not having the optimal running time, our algorithm still have an acceptable time efficiency while achieving the best embedding performance.

G. Validation Tests

In this section, we evaluate how the improvement of our model from existing RL-based works can enhance the performance of VNE algorithms. We implement comparative algorithms to train their own VNE models individually. The comparative algorithms are shown in Table VII. These algorithms along with our proposed algorithm use the same copy of substrate networks and receive the identical VNRs for training.

1) Training Efficiency and Convergence: We first observe their differences during the training phase. We construct a set of VNRs for validation, with basic settings introduced in Section V-A, and show the algorithm performance on this set in every training episode (i.e., every 1000 VNRs in training) to evaluate their training efficiencies. The results are shown in Fig. 10(a).

The results show that all the algorithms improve their performance during training, although multiple thrashings take place. The thrashings on the validation set are mainly caused by the change of model policies, which seek for exploration during the training process. TR and NOEGB ascend faster at the beginning, but our algorithm comes from the behind and shows the best training performance in the latter stage. A3C+GCN, TR and TAC finally converges to a relatively similar good policy under the judgement of the validation set, while CNN is poorly performed. All the three "good" algorithms use GCN as feature extractor, which enforces the assertion that GCN is more applicable than traditional CNN in irregular graph topologies. In addition, though with a slow start, our algorithm quickest converges to a good policy, and the parallel training mechanism which allows our agent adequately meet various training samples is most likely to behave. In this group of test, our algorithm receives the training samples nearly four times more than the single-processed version in the same time.

2) Validation on Resource Request Tests: In the second phase, we evaluate the acceptance ratio of the comparative algorithms by the testing cases in Section V-D. The results are depicted in Fig. 10(a). It has shown that our original algorithm outperforms any other comparative versions in acceptance ratio when facing various amount of resource requests with an average advantage of 4.5%, 3.4%, 16.5% and 8.4% to TR, TAC, CNN and NOEGB respectively. The testing results have proved that the utilization of A3C training algorithm, the feature extractor based on GCN, and the new design of reward functions all play a role in boosting the VNE performance. In particular, the performance of CNN remains the last position, which shows that the use of GCN instead of traditional CNN for feature extraction is a critical part of our algorithm.

H. Feasibility Study for the Model Under Additional Parameters

In this section, we conduct the experiments considering additional substrate network topology, parameter and evaluation metrics. The experimental results show the potential of the proposed model to be applied in more complicated and realistic network environments.

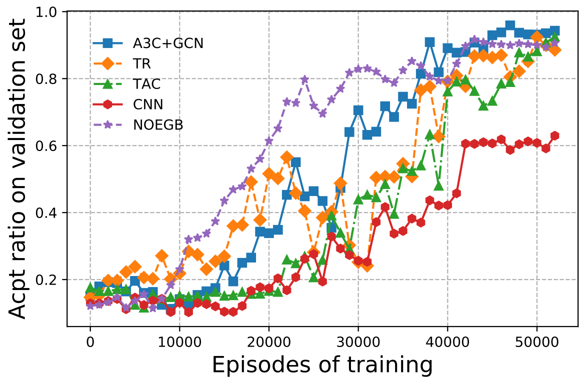
1) Additional Network Topology and Parameter: CSTNET is a real network operator in China, with its network topology connecting the institutes of Chinese Academy of Sciences. The topology is depicted in Fig. 11(a). The red edges are 100Gb links, green ones have 10Gb bandwidth, orange links are 2.5Gb and the black edges are 1Gb. The weights shown on some edges are the average transmission latency in milliseconds, while the edges without a weight value have the default latency of 1 millisecond. We mock some random point-to-point data transfer tasks; the source and target pairs are randomly selected, the arrival rate and transfer time are the same as the ones mentioned in Section V-A, and the data transfer rate is uniformly distributed from 500Mbps to 3Gbps. In addition, we add the average latency of each node (with a similar manner as the definition of node bandwidth in the second row of Table III) to our model and observe the average transform latency between A3C+GCN and Noderank. The results depicted in Fig. 11(b) show that our model obtains less latency than the Noderank algorithm, which demonstrates the feasibility of expanding our model with more relevant network attributes.

2) Additional Metrics: In this section, we evaluate two additional parameters, node resource utilization and link resource utilization. Their definitions are straightforward, i.e., the amount of node/link substrate resources which are used by VNRs divided by the total amount of resources. We use the test settings in Section V-C and investigate the node and link utilizations among our approach and the five algorithms introduced in Section V. The results shown in Fig. 12 demonstrate that our algorithm reaches the best in these two parameters. In addition, the trend is analogous with Fig. 9(b), which means the resource utilization is highly related with revenue.

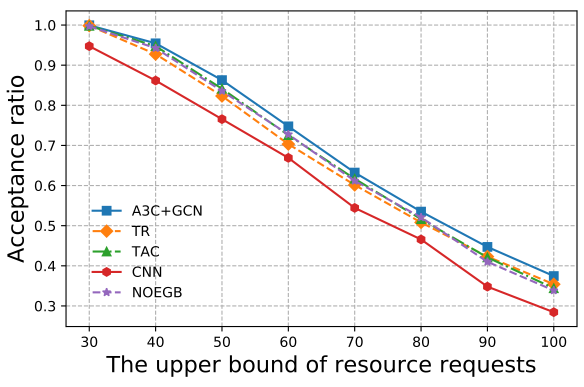
VI. CONCLUSION AND FUTURE WORK

In this paper, to solve the automic virtual network embedding problem, we have adopted the state-of-the-art deep reinforcement learning techniques. To speed up the training procedure and generate the training experience more efficiently, we have used A3C algorithm to train the policy generation algorithm. Inside our learning agent, to extract spatial features from raw state inputs more efficiently, we have designed a single-layer, 3-order GCN instead of traditional CNN. Meanwhile, we have defined the new reward function using multiple objectives to guide the learning agent into desirable behaviours. Extensive simulation results have shown that our algorithm outperforms the previous typical and state-of-the-art VNE algorithms by 1.8% to 39.6% (with an average of 22.4%) on acceptance ratio and by 2.8% to 70.6% (with an average of 30.2%) on average revenue with a fairly acceptable running time. Moreover, the results have also demonstrated that the proposed solution possesses good robustness, owing to different data distributions used in training and testing phases.

In the future, we are planning to train our agent on a larger SN topology (containing thousands of substrate nodes) and larger VNRs (with hundreds of virtual nodes). We also intend



(a)

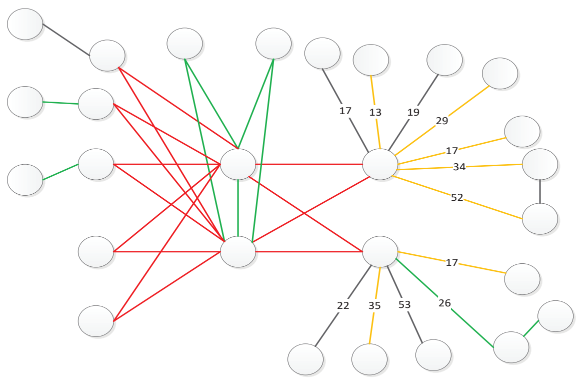


(b)

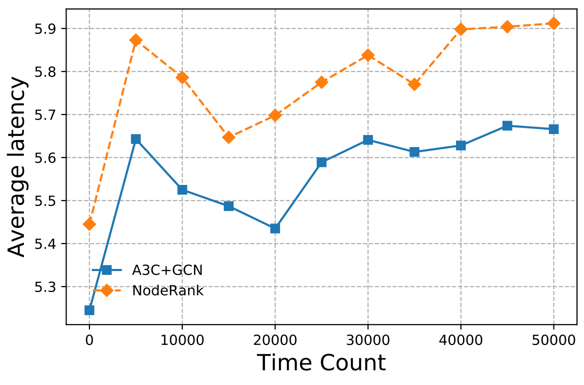
Fig. 10. The results of validation tests: (a) represents the convergence trend while training and (b) depicts the acceptance ratio with varying resource requests during the testing phase.

TABLE VII
THE COMPARATIVE ALGORITHMS FOR VALIDATION

VNE Algorithm	Description
TR	Uses a traditional reward function (the subtraction of VNE cost and reward which is used in [18]) to manage the reward instead of the reward function discussed in Section IV-C1.
TAC	The single-process version of A3C algorithm, which reduces the number of work agents to 1.
CNN	Uses traditional node features and CNN architecture proposed by [43] to extract features and generate policies.
NOEGB	Uses reward function in Section IV-C1, except for the eligibility trace values, they are always set to 1.



(a)



(b)

Fig. 11. The study of new topology and parameters, where (a) is the CSTNET topology and (b) is the latency test results under this topology.

to extend our model definition and training algorithms to support the following features: 1) the VNRs change dynamically even they have already been embedded; 2) substrate nodes and links may fail during embedding; 3) one virtual node can be embedded over multiple substrate nodes. In addition, we are preparing to perform tests on a real testbed based on Openstack. Additionally, the link requests can be replaced by a traffic matrix over pairwise virtual nodes, which also need a new feature extraction mechanism.

REFERENCES

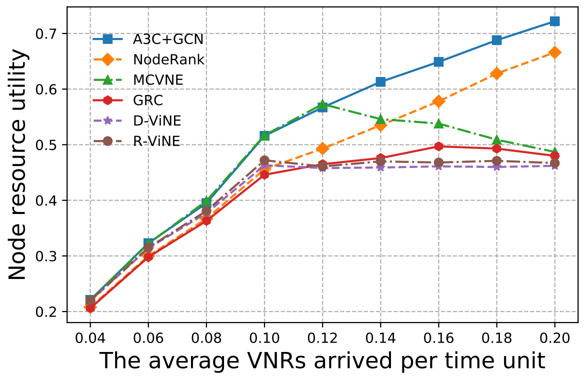
[1] David G Andersen. Theoretical approaches to node assignment. *Computer Science Department*, page 86, 2002.

[2] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck. Optimal vnfs placement in cdn slicing over multi-cloud environment. *IEEE Journal on Selected Areas in Communications*, 36(3):616–627, March 2018.

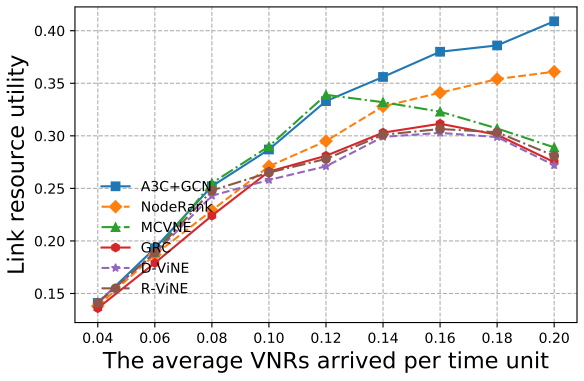
[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2007.

[4] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.

[5] X. Cheng, Y. Wu, G. Min, and A. Y. Zomaya. Network function virtualization in dynamic networks: A stochastic perspective. *IEEE*



(a)



(b)

Fig. 12. The effects of (a) node resource utilization and (b) link resource utilization.

Journal on Selected Areas in Communications, 36(10):2218–2232, Oct 2018.

[6] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *SIGCOMM Comput. Commun. Rev.*, 41(2):38–47, April 2011.

[7] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Trans. Netw.*, 20(1):206–219, February 2012.

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

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[10] Chinmaya Kumar Dehury and Prasan Kumar Sahoo. Dyvine: Fitness based dynamic virtual network embedding in cloud computing. *IEEE Journal on Selected Areas in Communications*, 2019.

[11] Mahdi Dolati, Seyedeh Bahereh Hassanpour, Majid Ghaderi, and Ahmad Khonsari. Deepvine: Virtual network embedding with deep reinforcement learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 879–885. IEEE, 2019.

[12] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[13] ETSI. Nfv official site. https://www.etsi.org/technologies/nfv.

[14] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[16] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *INFOCOM, 2014 Proceedings IEEE*, pages 1–9. IEEE, 2014.

[17] Soroush Haeri and Ljiljana Trajković. Vne-sim: a virtual network embedding simulator. In *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, pages 112–117. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2016.

[18] Soroush Haeri and Ljiljana Trajković. Virtual network embedding via monte carlo tree search. *IEEE transactions on cybernetics*, 48(2):510–521, 2018.

[19] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[20] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[21] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88. ACM, 2009.

[22] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[23] MARA82 Marathe and W Hawe. Predicted capacity of ethernet in a university environment. In *Proceedings of Southcon*, pages 1–10, 1982.

[24] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[25] Wang Miao, Geyong Min, Yulei Wu, Haozhe Wang, and Jia Hu. Performance modelling and analysis of software-defined networking under bursty multimedia traffic. *ACM Trans. Multimedia Comput. Commun. Appl.*, 12(5s), September 2016.

[26] Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, Maxim Claeys, Filip De Turck, and Steven Latré. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[27] Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, Maxim Claeys, Jeroen Famaey, and Filip De Turck. Neural network-based autonomous allocation of resources in virtual networks. In *Networks and Communications (EuCNC), 2014 European Conference on*, pages 1–6. IEEE, 2014.

[28] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[30] Quang Tran Anh Pham, Yassine Hadjadj-Aoul, and Abdelkader Outtagarts. Evolutionary actor-multi-critic model for vnf-fg embedding. In *IEEE Consumer Communications & Networking Conference (CCNC 2020)*, 2020.

[31] Vincenzo Sciancalepore, Faqir Zarrar Yousaf, and Xavier Costa-Pérez. z-torch: An automated NFV orchestration and monitoring solution. *CoRR*, abs/1807.02307, 2018.

[32] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, 2012.

[33] Nashid Shahriar, Shihabur Rahman Chowdhury, Reaz Ahmed, Aimal Khan, Siavash Fathi, Raouf Boutaba, Jeebak Mitra, and Liu Liu. Virtual network survivability through joint spare capacity allocation and embedding. *IEEE Journal on Selected Areas in Communications*, 36(3):502–518, 2018.

[34] David Silver and Demis Hassabis. Alphago: Mastering the ancient game of go with machine learning. *Research Blog*, 9, 2016.

[35] Oussama Soualah, Ilhem Fajjari, Nadjib Aitsaadi, and Abdelhamid Mellouk. A reliable virtual network embedding algorithm based on game theory within cloud's backbone. In *Communications (ICC), 2014 IEEE International Conference on*, pages 2975–2981. IEEE, 2014.

[36] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[37] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[38] Haozhe Wang, Yulei Wu, Geyong Min, Jie Xu, and Pengcheng Tang. Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach. *Information Sciences*, 498:106 – 116, 2019.

[39] Sen Wang, Jun Bi, Jianping Wu, Athanasios V Vasilakos, and Qilin Fan. Vne-td: A virtual network embedding algorithm based on temporal-difference learning. *Computer Networks*, 161:251–263, 2019.

[40] Bernard M Waxman. Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9):1617–1622, 1988.

[41] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, pages 21:1–21:10, New York, NY, USA, 2019. ACM.

[42] Z. Yan, J. Ge, Y. Wu, H. Zheng, L. Li, and T. Li. Automatic virtual network embedding based on deep reinforcement learning. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 625–631, Aug 2019.

[43] Haipeng Yao, Xu Chen, Maozhen Li, Peiying Zhang, and Luyao Wang. A novel reinforcement learning algorithm for virtual network embedding. *Neurocomputing*, 284:1–9, 2018.

[44] Chunyan Yu, Qi Lian, Dong Zhang, and Chunming Wu. Pame: Evolutionary membrane computing for virtual network embedding. *Journal of Parallel and Distributed Computing*, 111:136–151, 2018.

[45] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[46] Ying Yuan, Zejie Tian, Cong Wang, Fanghui Zheng, and Yanxia Lv. A q-learning-based approach for virtual network embedding in data center. *Neural Computing and Applications*, Jul 2019.

[47] Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *International Journal of Communication Systems*, 26(8):1054–1073, 2013.