Accelerating Virtual Network Embedding with Graph Neural Networks

Farzad Habibi*, Mahdi Dolati*, Ahmad Khonsari*†, and Majid Ghaderi‡
* University of Tehran, Tehran, Iran; Email: {farzad.habibi, mahdidolati, a_khonsari}@ut.ac.ir
† Institute for Research in Fundamental Sciences (IPM), Tehran, Iran; Email: ak@ipm.ir
‡ University of Calgary, Calgary, Canada; Email: mghaderi@ucalgary.ca

***Abstract*—Virtual Network Embedding (VNE) is an essential component of network virtualization technology. Prior works on VNE mainly focused on resource efficiency and did not address the scalability as a first-grade objective. Consequently, the ever-increasing demand and size render them less-practical. The few existing designs for mitigating this problem either do not extend to multi-resource settings or do not consider the physical servers and network simultaneously. In this work, we develop GraphViNE, a parallelizable VNE solution based on spatial Graph Neural Networks (GNN) that clusters the servers to guide the embedding process towards an improved runtime and performance. Our experiments using simulations show that the parallelism of GraphViNE reduces its runtime by a factor of 8. Also, GraphViNE improves the revenue-to-cost ratio by about 18%, compared to other simulated algorithms.**

I. Introduction

Motivation. Network virtualization is a prominent technology that provides the needed flexibility and isolation to deploy the next-generation heterogeneous network-applications in a shared infrastructure. An essential part of this technology is the problem of on-demand embedding of Virtual Networks (VNs) in a Physical Network (PN); which in the most basic form consists of mapping virtual nodes and links with specific resource demands to physical servers and paths with limited capacities. The embedding strategy substantially affects the utilization of resources, which, in turn, determines the revenue and cost of the production network. Given the NP-hardness of the problem [1], the design of efficient embedding algorithms has been the subject of extensive research [2].

However, in the face of ever-growing network sizes, most current studies suffer from scalability issues that restrict their practical usage in modern time-stringent environments. Furthermore, their solutions' quality drastically drops as the problem gets large. A promising method to alleviate these issues is using a pre-processing phase to help the embedding algorithm complete its task faster and with a higher success rate. As such, some approaches investigated different strategies, such as restricting the embedding options to a subset of physical nodes [3], shrinking the virtual network sizes [4], and rejecting the hard-to-embed virtual networks without running the actual embedding algorithm [5].

Although current proposals have demonstrated the effectiveness of the pre-processing technique, most of them fail to consider crucial aspects of the embedding problem (*e.g.*, network topology and multiple resource dimensions) to keep their computational overhead at a tolerable level. They are also typically designed based on the traditional model-driven paradigm that can not consider underlying patterns and hidden connections that generally are specific to the target system. Data-driven approaches can detect and exploit these patterns by applying the machine learning architectures (*e.g.*, convolutional deep neural networks), however, non-Euclidean nature of network structures (*i.e.*, capacitated graphs) [6] complicates the design process.

Graph Neural Network [7] is a new machine learning architecture that can aggregate multiple features across a graph's nodes while integrating its topology. Specifically, a GNN can automatically learn a condensed representation of each node in the network that incorporates the information about the node, its neighbors (up to the desired distance), and their inter-connecting topology. By applying this framework to a PN, it becomes possible to cluster the physical nodes (*i.e.*, servers) based on their resource capacities (*e.g.*, CPU and RAM) and topological positions. Then, the computed clusters can shrink the search space by eliminating the consideration of similar options and providing the opportunity of focusing on the examination of distinct options.

Our objective in this work is to exploit the spatial-based graph neural networks [8], [9], which provide parallelizable operations, to reduce the search space of the virtual network embedding problem in an efficient and meaningful way that appreciates the server resources and the network's topology.

Our Work. We present a VNE algorithm that employs GNNs for acceleration and efficiency. Specifically, we design an adversarially-regularized variational graph autoencoder to cluster the physical servers based on resource capacities and network status. The embedding procedure then uses the computed clusters to efficiently find and investigate the servers that collectively provide a diverse set of embedding options. Instead of using spectral-based GNNs (*e.g.*, [6]) that process the entire graph at once, we employ spatial-based GNNs that can operate on batches of nodes. Consequently, our model can take advantage of processing these batches of nodes in parallel, which significantly improves the scalability. Moreover, spectral-based approaches rely on a graph Fourier basis and assume a fixed graph, which drastically limits their

978-3-903176-31-7

generalization and transferability. Spatial-based GNNs mitigate this limitation by performing their convolution operator locally on each node. Thus, in dynamic environments where the network capacity changes after each virtual network arrival or departure, spatial-based GNNs provide better performance through their generalizability and better scalability through their local operations. Our main contributions can be summarized as follows:

- We model a physical network with multi-dimension resources (*e.g.*, CPU and GPU) as a graph with *node features*.
- We improve an existing graph autoencoder to employ spatial-based GNNs, which can perform the convolution by aggregating the neighbor's information. As the result, our method provides a higher level of parallelizability and generalizability.
- We define a suitable function that is used by the GNN to aggregate the information about each physical server, its neighbors, and their interconnecting network (*i.e.*, topology and bandwidth). We use the expected diameter of virtual networks to determine the depth of the information aggregation.
- We use the aggregated information to cluster the servers based on their embedding capability and find a diverse set of servers to start an embedding process from. We use the elbow method [10] to determine the number of clusters.
- We simulate our algorithm to demonstrate its speedup and performance and compare it with recent VNE algorithms.

A. Paper Organization

The paper is organized as follows. Section II reviews related works. The problem statement and technical background are presented, respectively, in Sections III and IV. The design of our algorithm and its evaluation are presented, respectively, in Sections V and VI. Section VII concludes the paper.

II. Related Work

We review related works in the literature by categorizing them into three groups. See [2] for a recent survey.

Generic VNE Algorithms. To handle the problem's size and complexity, authors in [11] used dynamic programming principles to break down virtual networks to a set of edge-disjoint path segments and used a multi-layer graph transformation to embed the obtained segments. Authors in [12] considered a multi-dimensional setting, where a security feature is associated with every physical node and link. They performed a greedy embedding to allocate the resources while minimizing the probability of malicious attacks aimed at virtual networks. Two resource dimensions (*e.g.*, CPU and RAM) are considered in [13], where the authors employ the ratios (*e.g.*, $\frac{\text{CPU}}{\text{RAM}}$) to obtain a metric for ranking the nodes for the embedding task. Authors in [14] considered guaranteeing the end-to-end delays for virtual links by modeling the latency in the physical links as random variables with known mean and variance. None of these works can use machine learning techniques to exploit the available operational data in the target system.

Learning-based VNE Algorithms. Authors in [6] employed an asynchronous advantage actor-critic algorithm to automate the embedding through the exploration-exploitation technique, where a spectral-based convolutional graph neural network is used to extract the physical network features that model the environment. To address the temporal dependency of physical network state, as it changes after serving requests, in [15], the node embedding task is formulated as a time-series problem. A recurrent neural network is then trained via the seq2seq model to learn the embedding location for virtual nodes. A method called DeepViNE is proposed in [16] that encodes the physical and virtual networks as images with multiple layers. It then feeds the images to a convolutional deep reinforcement learning agent to learn the embedding strategy that maximizes the profit. DeepViNE is restricted to networks with grid topologies. None of these works provide a mechanism to mitigate the large scale of the problem.

VNE Pre-processors. The method proposed in [3] uses the size of the virtual network to determine the suitable number of physical nodes for the embedding task. It then uses a Hopfield network (a form of recurrent artificial neural network) to select a subset of nodes with the determined size that maximizes the probability of a successful embedding. Authors in [5] used Recurrent Neural Networks to design an admission control mechanism that predicts whether the available physical resources are sufficient to embed new virtual network requests or not. The early rejection of infeasible requests eliminates the unsuccessful runs of the embedding algorithm and consequently reduces the response time. A network partitioning algorithm is employed in [4] to reduce the size of the virtual networks by dividing the virtual nodes into a specific number of groups (determined based on the physical node and link resources) that have lightweight connections. Authors in [17] applied field theory to extract the physical network features and then performed spectral clustering to aggregate the nodes with high resource and connectivity similarity into regions that provide sufficient capacity for the embedding process. None of these works uses a systematic mechanism, like GNN, to simultaneously address the servers with multi-dimensional resources and the network topology.

III. Problem Definition

This section presents the formal description of the virtual network embedding problem that is considered in this paper.

Physical Network. An un-directed graph $G^p = (\mathcal{N}^p, \mathcal{L}^p)$ is used to represent a physical network with S physical nodes (*i.e.*, servers) $\mathcal{N}^p = \{n_1^p, \ldots, n_S^p\}$ and T physical links $\mathcal{L}^p = \{\ell_1^p, \ldots, \ell_T^p\}$. Each node has a set of $\mathcal{R}$ resources (*e.g.*, CPU and RAM), where the amount of resource $r \in \mathcal{R}$ in node n_i^p is $f_r(n_i^p)$. The bandwidth of link ℓ_i^p is $b(\ell_i^p)$. Furthermore, we define $\mathcal{P}_{i,j}$ to be the set of all paths between physical nodes n_i^p and n_j^p, where each path $p_{i,j} \in \mathcal{P}_{i,j}$ is a list of physical links. Also, $\mathcal{Q}_i$ denotes the set of all paths that contain the physical link ℓ_i^p. We use real numbers to represent physical resources and bandwidth capacities with respect to a base unit.

Virtual Network. We assume that a virtual network $G^t = (\mathcal{N}^t, \mathcal{L}^t)$ arrives to the network at the time $t \in \mathcal{T}$, where $\mathcal{T}$ is the set of arrival moments. Here, $\mathcal{N}^t = \{n_1^t, \ldots, n_U^t\}$ and $\mathcal{L}^t = \{\ell_1^t, \ldots, \ell_V^t\}$, respectively, represent the set of U virtual nodes and the set of V virtual links. We define $\varepsilon(\ell_i^t) = \{a, b\}$ show the set of two virtual nodes n_a^t and n_b^t that form the endpints of the virtual link ℓ_i^t. Each virtual node n_i^t requires $g_r(n_i^t)$ units of resource r for its operation and each virtual link ℓ_i^t consumes $d(\ell_i^t)$ units of bandwidth to handle the communication of its endpoints. Furthermore, we assume that virtual network G^t finishes its operation after τ_t time units and releases its acquired physical resources upon leaving the network. We use real numbers to represent virtual resource and bandwidth demands with respect to the base unit.

VNE Problem. Each virtual network, upon arrival at time t, should explicitly be rejected or accepted. To accept a virtual network, the virtual nodes and links should be mapped, respectively, to physical nodes and paths with sufficient resource capacities. Let a_t denote the binary decision variable that represents the admittance status of G^t. An embedding algorithm should decide the value of a_t without the knowledge of arriving virtual networks after time t. Let $x_{i,t}^j$ be a binary decision variable which is equal to 1 if the virtual node n_i^t is mapped to physical node n_j^p. Similarly, the binary decision variable $y_{i,t}^{j,k}$ is defined to show whether the virual link ℓ_i^t is mapped to the physical path $p_{j,k}$ between the physical nodes n_j^p and n_k^p or not. A valid virtual network mapping satisfies the following constraints,

$$f_r(n_j^p) \geq \sum_{\substack{t' \in \mathcal{T} \\ t' \leq t}} \sum_{n_i^{t'} \in \mathcal{N}^{t'}} a_{t'} \delta_{t', \tau_{t'}}^t x_{i,t'}^j g_r(n_i^{t'}), \quad \forall n_j^p \in \mathcal{N}^p, r \in \mathcal{R} \quad (1)$$

$$y_{i,t}^{j,k} \leq \sum_{\substack{a,b \in \varepsilon(\ell_i^t) \\ a \neq b}} x_{a,t}^j x_{b,t}^k, \qquad \forall \ell_i^t \in \mathcal{L}^t, n_j^p, n_k^p \in \mathcal{N}^p, p_{j,k} \in \mathcal{P}_{i,j} \quad (2)$$

$$b(\ell_i^p) \geq \sum_{\substack{t' \in \mathcal{T} \\ t' \leq t}} \sum_{\ell_j^{t'} \in \mathcal{L}^{t'}} \sum_{\substack{p_{k,m} \\ \ell_i^p \in \mathcal{Q}_i}} a_{t'} \delta_{t', \tau_{t'}}^t y_{j,t'}^{k,m} d(\ell_j^{t'}), \qquad \forall \ell_i^p \in \mathcal{L}^p \quad (3)$$

where, $\delta_{t', \tau_{t'}}^t$ is an auxiliary function that determines whether the virtual network $G^{t'}$, with the duration of $\tau_{t'}$, is active at time t or not. The definition of $\tau_{t'}$ is for the sake of formulation and our solution does not need it. Constraint (1) ensures that the capacity of every type of resources in all the physical nodes is respected. Constraint (2) guarantees that each virtual link is mapped to a physical path that connects the physical nodes that embed the two endpoints of the virtual link. Constraint (3) enforces the bandwidth capacity of the physical links. If any of these constraints is violated, G^t is blocked. In this work, minimizing the virtual network blocking probability is considered objective, achieved by an efficient resource allocation strategy. This resource allocation wastes the minimum amount of physical resources during the mapping process. Formally, the acceptance ratio is defined by,

$$\lim_{|\mathcal{T}| \to \infty} \frac{\sum_{t \in \mathcal{T}} a_t}{|\mathcal{T}|}. \quad (4)$$

Furthermore, the *Revenue* and *Cost* of the algorithm can be computed as follows:

$$\text{Revenue} = \sum_{t \in \mathcal{T}} a_t \Big\{ \sum_{n_i^t \in \mathcal{N}^t} \sum_{r \in \mathcal{R}} \zeta_r g_r(n_i^t) + \sum_{\ell_i^t \in \mathcal{L}^t} d(\ell_i^t) \Big\}, \quad (5)$$

$$\text{Cost} = \sum_{t \in \mathcal{T}} a_t \Big\{ \sum_{n_i^t \in \mathcal{N}^t} \sum_{r \in \mathcal{R}} \xi_r g_r(n_i^t) + \sum_{n_i^p, n_j^p \in \mathcal{N}^p} \sum_{p_{i,j} \in \mathcal{P}_{i,j}} \sum_{\ell_k^t \in \mathcal{L}^t} y_{k,t}^{i,j} d(\ell_i^t) |p_{i,j}| \Big\}, \quad (6)$$

where, $|p_{i,j}|$ shows the length of the path $p_{i,j}$ and $\zeta_r \geq 0$ and $\xi_r \geq 0$ can be used to adjust the revenue and cost of type-r resource relative to the revenue and cost of the bandwidth.

IV. Graph Neural Networks

In this section, we briefly describe the background that is required in the rest of the paper. For more details and discussion please refer to [8] and [9].

A. Spatial Graph Neural Networks

Assume that a set of K aggregator functions $\Psi_k(.)$ are available that can, with the help of weight matrices W^k, aggregate the information of each node's depth-K neighborhood in the graph. The computation of $\Psi_k(.)$ and W^k is explained later. Let x_v denote the feature list of the node v. Also, let h_v^k denote the combined information about the node v and its depth-k neighborhood. Initially, $h_v^0 = x_v$ and the information aggregation happens incrementally and collectively for all nodes in the network, where, each node uses depth-k aggregated information of its neighbors to obtain its depth-$(k+1)$ aggregated information (similar to the backup operations that lie at the core of reinforcement learning approaches [18]). Specifically, each node v evaluates the following equations K times,

$$\psi \leftarrow \Psi_k(\{h_u^k : u \text{ is a neighbor of } v\}), \quad (7)$$

$$h_v^{k+1} \leftarrow \alpha\left(W^{k+1}.h_v^k \frown \psi\right), \quad (8)$$

where, ψ is an intermediate placeholder, $\alpha(.)$ is a non-linear activation function, and $\frown$ is the list concatenation operator. Notice that equation (8) is similar to a fully connected neural network layer, where W^{k+1} is the weight matrix. To control the runtime and memory consumption of this approach, typically, in each aggregation step, only a fixed-size random-sample of the neighbors are considered instead of computing ψ over all neighbors. There are multiple choices for the aggregator functions, which have to be invariant to their inputs' permutations. Mean, LSTM, and Pooling aggregators are common choices. For example, the Mean aggregator's operation is the element-wise mean of the vectors h_u^k in equation (7). Then, the stochastic gradient descent algorithm is applied to a graph-based loss function to learn the weight matrices W^k and parameters of $\Psi_k(.)$. Let $z_u = h_u^K$, the loss function for node u is defined as,

$$-\log\left(s(z_u^\intercal z_v)\right) - Q\mathbb{E}_w[\log\left(s(-z_u^\intercal z_w)\right)], \quad (9)$$

where, u is obtained from a fixed-length random walk from v, Q is the number of samples from the neighbors of u, w is a sampled neighbor, and $s(.)$ is the logistic sigmoid function.

B. Variational Graph Auto-encoding

In this approach, a node is mapped to distribution rather than a single representation. Then, a random representation is taken from the distribution and is used to reconstruct the original node. Then, by minimizing the reconstruction error, an optimized representation distribution of the graph is obtained. Let $\boldsymbol{Z}$ be the matrix of representation vectors z_u. A common approach for designing the encoder is to model the distribution of $\boldsymbol{Z}$ conditioned on the values of node features $\boldsymbol{X}$ and graph structure $\boldsymbol{A}$ (denoted by $q(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})$) by a Multivariate Gaussian distribution. Then, two graph neural networks can be used to learn the parameters of the distribution, *i.e.*, mean $\boldsymbol{\mu}$ and logarithm of variance $\log \boldsymbol{\sigma}^2$. A simple decoder, denoted by $p(\boldsymbol{A}|\boldsymbol{Z})$, is defined by $s(\boldsymbol{Z}\boldsymbol{Z}^\intercal)$. Then, the reconstruction error is computed by,

$$\mathbb{E}_{q(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}\big[\log p(\boldsymbol{A}|\boldsymbol{Z})\big] - \mathrm{KL}\big(q(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A}) \parallel p(\boldsymbol{Z})\big), \quad (10)$$

where, KL(.) is the Kullback-Leibler divergence and $p(\boldsymbol{Z})$ is a prior distribution, *i.e.*, the probability of observing $\boldsymbol{Z}$ if $\boldsymbol{\mu} = 0$ and $\boldsymbol{\sigma} = I$ (I is the identity matrix).

C. Adversarial Models

It is possible to improve the performance of an auto-encoder model (*i.e.*, reduce the reconstruction error) by using the distribution $q(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})$ as a generator model in an adversarial game. To this end, a a discriminator model (*e.g.*, a multi-layer perceptron) is used to discriminate the samples of the generator from the samples of the prior distribution $p(\boldsymbol{Z})$. The objective of the game is defined as,

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\boldsymbol{Z}\sim p(\boldsymbol{Z})}\big[\log \mathcal{D}(\boldsymbol{Z})\big] - \mathbb{E}_{\boldsymbol{X}\sim p(\boldsymbol{X})}\big[\log \mathcal{D}(1-\mathcal{G}(\boldsymbol{X},\boldsymbol{A}))\big], \quad (11)$$

where, $\mathcal{G}$ and $\mathcal{D}$ are, respectively, the generator and discriminator models. Equation (11) can be used in the training process of $q(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})$ to ensure that the latent values match the prior distribution.

V. VNE Algorithm

In this section, we present the design of a scalable algorithm based on the graph autoencoders to solve the network embedding problem (see Section III). We employ an adversarially-regularized variational graph autoencoder architecture to compute a representation for each physical server that contains information about: (1) the server's resource capacity, (2) the resource capacity of neighboring servers up to the desired distance, and (3) the inter-connecting network topology and its bandwidth capacity. Then, we cluster those physical nodes based on the obtained representations, where the servers in the same cluster would have a similar resource and bandwidth capacity. Consequently, the servers in the same cluster would provide similar qualifications for embedding a virtual network.

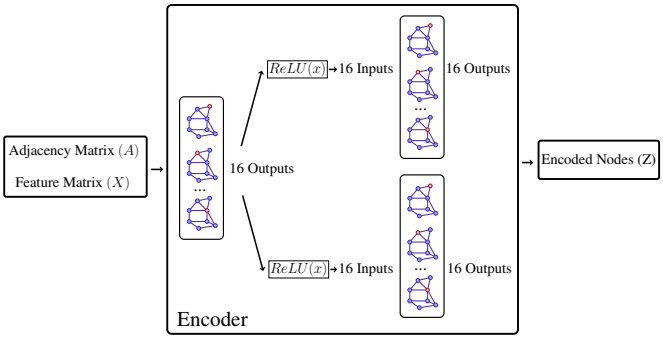


Fig. 1: Encoder model with spatial-based GNN layers

Then, we select a random server from each cluster upon which the virtual network embedding algorithm starts its process, and the result of the best starting point with the lowest cost is chosen as the solution. Then, the server representations and the clustering is updated efficiently.

To increase the scalability, we define a set of criteria for the clustering, which can control the quality-runtime trade-off. One criterion is the cluster's freshness, which allows the clustering to basically re-use the previous clustering and update the clustering only after a significant change occurs. For example, when the capacity of a server or a link is changed significantly. In what follows, we describe the encoder model, discriminator model, clustering method, and embedding algorithm.

Encoder. The topology of our proposed model for computing the representation of the physical servers (*i.e.*, the encoder model) is illustrated in Fig. 1. The encoder gets the server resource capacity matrix X (*e.g.*, CPU, memory, and GPU) and the weighted adjacency matrix A of the physical network, where the weights show the bandwidth, as the input, and generates a distribution that determines server representations. Similar to the existing architectures, we use the Multivariate Gaussian distribution to model the representation distribution. Remember that computing the representation distribution, instead of directly generating the representation, significantly improves the generalizability of the model. Thus, when the available link capacities change, the model still generates an appropriate representation. Specifically, the encoder is a spatial neural network with two layers. The second layer is divided into two parts for computing the mean and the variance of the representation distribution. The input channel's size in the first layer is equal to the number of server resources (*i.e.*, $\mathcal{R}$), and the size of the output channel is fixed to be 16. Also, the input and output channels' sizes in both parts of the second layer are equal to 16. The aggregator function $\Psi_k(.)$ is defined as,

$$W_1 \boldsymbol{x}^{k-1}(n) + W_2 . \frac{1}{|\mathcal{L}(n)|} \sum_{\ell \in \mathcal{L}(n)} b(\ell) \boldsymbol{x}^{k-1}(m) \quad (12)$$

where, W_1 and W_2 are learnable parameter matrices, $\boldsymbol{x}^{k-1}(n)$ is the feature vector of server n after $k-1$ round of aggregation, $\mathcal{L}(n)$ is the set of all links that are connected to server n, and m is the second endpoint of the physical link ℓ. Note that the initial value of the feature vector, *i.e.*,

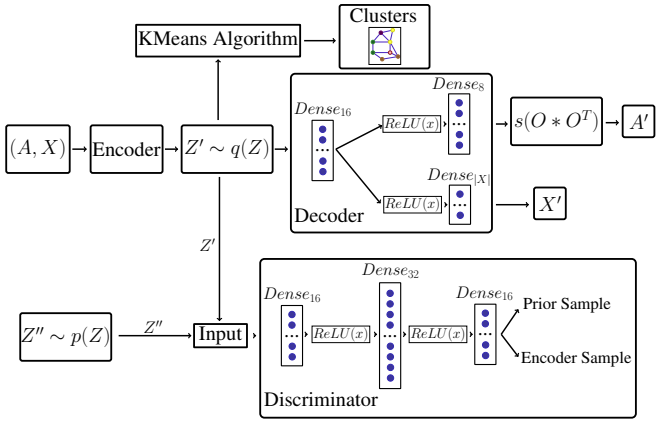


Fig. 2: Auto-encoder model with fully-connected feed-forward neural networks as decoder and discriminator

$\boldsymbol{x}^0(n)$, is set to be the list of resource capacities of server n, *i.e.*, $\boldsymbol{f}(n)$. Observe that the bandwidth of the link $b(\ell)$ is used to control the effect of the resource availability to the adjacent servers. The activation function of the first and second layers, respectively, are ReLU and linear. Finally, the number of aggregation steps K is chosen to be equal to the expected diameter of virtual networks. In our considered random networks in Section VI, this value was slightly more than 2, and hence the encoder in Fig. 1 has two layers. Note that the number of layers in the encoder does not limit the diameter of VNs and thus the method applies to VNs with arbitrary diameter. The number of layers only affects the depth from which neighbors' information is obtained.

Decoder and Discriminator. The topology of the employed auto-encoder is presented in Fig. 2. After encoder determines the representation distribution, a sample representation Z' is drawn and fed into the decoder, which is a multi-layer perceptron with two layers. Note that, here, we can not use a GNN, because the input Z is a graph *representation* and not a graph. The second layer of the decoder has two parts, where one of them reconstructs the adjacency matrix, and the other layer reconstructs the server resource matrix X. The outputs are then compared with the original inputs to compute the reconstruction error and train the encoder with equation (10). The first layer of the decoder has 16 neurons, and each part has 8 neurons. Also, Z' is fed to a discriminator model, which is illustrated in the lower tier of Fig. 2. This model is a 3-layer dense feed-forward neural network with 16, 32, and 16 neurons in each layer. The activation function in the discriminator layers is ReLU. The discriminator is trained to distinguish the representative samples of the prior distribution $p(Z)$ from the representative samples generated by the encoder. Simultaneously, the encoder is trained (see equation (11)) to generate samples that follow the prior distribution. This joint optimization of the encoder and the discriminator through the adversarial framework ensures that the encoder follows the selected prior distribution.

Clustering. The representation sample Z' generated by the encoder is used to cluster the physical servers into a set of groups, where the servers inside a group are similar to each other and different from the other groups' servers. Specifically, the servers inside a group provide similar resource and bandwidth capacities up to a specific distance for the embedding task. Thus, the embedding cost does not change when we choose two different physical servers in the same group as the embedding procedure's starting point. We employ the k-means algorithm to perform the clustering, where the number of clusters is determined by using the elbow method [10]. This method evaluates the sum of samples' squared distances to their closest cluster center (*a.k.a.*, inertia) for the different number of cluster centers. It selects the number that has the most improvement compared to the previous value. In our considered topologies, the number of clusters is typically between 4-6.

Algorithm 1: GraphViNE – Graph Neural Network Accelerated Virtual Network Embedding

```
   procedure GraphViNE(G^p, G^t, Z, C, M)
1      cost* ← ∞
2      sol* ← null
3      for C ∈ C do
4          n ← start_node(C)
5          cost, sol ← embed(n, G^p, G^t)        /* See Alg. 2 */
6          if cost < cost* then
7              cost* ← cost
8              sol* ← sol
9      allocate(G^p, G^t, sol*)
10     if Available resource of a server changed by more than κ1% or
         available bandwidth of a link changed by more than κ2%
         then
11         Z ← re_GNN(M, G^p, sol*)
12         C ← re_cluster(Z, sol*)
13     Return sol*, Z, C
```

Embedding. Proposed virtual network embedding method, called GraphViNE, is outlined in Alg. 1. When the virtual network G^t arrives at time t, it is passed to GraphViNE algorithm along with the physical network G^p, the server representation Z, the cluster of servers $\mathcal{C}$, and the auto-encoder model $\mathcal{M}$. The for loop in lines 3 to 8 finds and stores the embedding with the lowest cost. Specifically, GraphViNE selects a physical server from each cluster in line 4 and passes it to another subroutine called **embed**, which is outlined in Alg. 2. The embed subroutine uses the physical input server as the starting point of its embedding procedure based on a breadth-first search mechanism. We defined two thresholds, $\alpha \times |\mathcal{N}^t|$ and β, to limit the total number of inspected servers and the maximum search depth, respectively. We omit the detailed description of embed subroutine for the sake of space and brevity. After finding the best solution, GraphViNE applies it to the physical network in line 9. Then, GraphViNE checks the change in the available resources and bandwidth. When either of these items change by, respectively, more than $\kappa_1\%$ or $\kappa_2\%$, the representation matrix Z and the clustering $\mathcal{C}$ are updated to be used in the next embedding round in lines 11 and 12. These criteria allow GraphViNE to re-compute the clustering when it makes a practical difference and avoid unnecessary calculations that might limit the scalability. Note that, in spatial GNNs, the representations are computed locally for each node, which is a highly parallelizable procedure. For

Algorithm 2: Embedding Subroutine

procedure `embed` $(n,\ G^p(\mathcal{N}^p, \mathcal{L}^p),\ G^t(\mathcal{N}^t, \mathcal{L}^t), \alpha, \beta)$
- $q_p \leftarrow$ queue(n)
- $q_t \leftarrow$ prio_queue$(\mathcal{N}^t)$ /* Prioritizes virtual nodes with higher CPU demand */
- **while** *True* **do**
  - $n_p \leftarrow q_p$.pop()
  - Mark n_p visited
  - **while** *True* **do**
    - $n_t \leftarrow q_t$.pop()
    - **if** $\boldsymbol{g}(n_t) > \boldsymbol{f}(n_p)$ **then**
      - Break /* Not enough resource */
    - **if** $\nexists$ *path to connect* n_t *to its neighbors* **then**
      - Break /* Not enough bandwidth */
    - Embed n_t in n_p and store the decision in sol
  - **if** *dist*$(n_p,\ n) = \beta$ **then**
    - **continue**
  - counter $\leftarrow 0$
  - **for** $n' \in$ *neighbor*(n_p) ***and*** *counter* $\leq (\alpha \times |\mathcal{N}^t|)^{\frac{1}{\beta}}$ **do**
    - **if** n' *is not visited* **then**
      - q_p.push(n')
      - counter $\leftarrow$ counter $+1$
  - cost $\leftarrow$ Calculate the cost with Eq. (6)
  - **Return** cost, sol

example, libraries such as PyTorch Scatter [19] can be used to accelerate the update of the auto-encoder model $\mathcal{M}$ and the representation matrix Z.

VI. Evaluation

We conduct extensive simulations to demonstrate our proposed approach's performance in terms of VN acceptance ratio, revenue, cost, and utilization against other algorithms. We use the random graph model [20] to construct physical and virtual networks. All algorithms are implemented in Python 3.6.9. Computations are carried out on a computer with an Intel® Core™ $i5-9500T$ processor at $2.2-3.7$ GHz and 8 GB of RAM. For examining parallelizability of our algorithm, we used Google Colaboratory GPU notebook [21] which uses two Intel® Xeon® processors at 2.0 GHz, 13 GB of RAM, and an Nvidia® Tesla® P4 GPU.

A. Simulation Parameters

In this subsection, we define and explain the parameters that we used throughout the evaluation.

Physical Network. To model the physical network, we consider a typical real-world configuration that consists of 1.2 terabytes of RAM and one hundred processing and graphics cores similar to, respectively, Dell™ PowerEdge™ R910, two Intel® Xeon® Scalable processors after Hyperthreading, and one Nvidia® GeForce® GT 330. Then, we apply the normal distribution to introduce diversity and heterogeneity. Consequently, we consider a network of 100 servers with a 40% probability of a direct physical link between each server pair. Each physical link is characterized by its bandwidth (in Mbps), which is randomly selected from the normal distributions $\mathcal{N}(100, 400)$. Each physical node is characterized by its CPU power (number of cores), memory capacity (in GBytes), and GPU power (number of cores). These values are selected from the normal distributions $\mathcal{N}(100, 400)$, $\mathcal{N}(1200, 300)$, and $\mathcal{N}(100, 400)$, respectively.

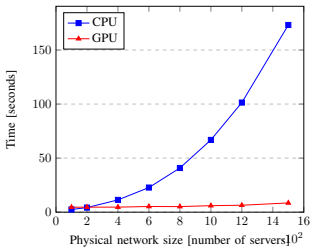


Fig. 3: Runtime comparison of GPU and CPU implementation

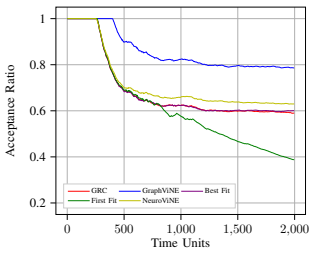


Fig. 4: Acceptance ratio comparison

Virtual Networks. The Number of virtual nodes in each VNR is randomly selected from the interval $[4, 10]$. The probability of having a virtual link between two virtual nodes is 0.7. Bandwidth demand of virtual links follows the normal distributions $\mathcal{N}(10, 4)$. The CPU, memory, and GPU demand of each virtual node come from the normal distributions $\mathcal{N}(10, 4)$, $\mathcal{N}(30, 9)$, and $\mathcal{N}(10, 4)$, respectively. Each VN has a lifetime that is also selected randomly from a normal distribution of $\mathcal{N}(100, 900)$. VNs arrive according to the arrival rate and remain in the physical network for the duration of their lifetime. The VN arrival rate is set to 2 per unit of time and the simulation is conducted for 2000 units of time.

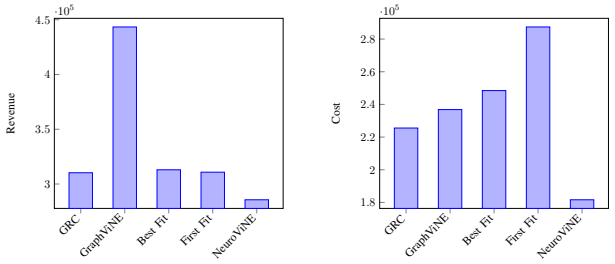
Comparing Algorithms. In addition to GraphViNE, we have implemented the following algorithms for comparison.

- **FirstFit**: an algorithm that embeds virtual nodes in the first physical node with sufficient capacity.
- **BestFit**: an algorithm that selects the physical node with maximum CPU capacity and fills it with the virtual node demands.
- **GRC** [22]: a node-ranking based algorithm.
- **NeuroViNE** [3]: an algorithm based on a search space reduction mechanism. This algorithm extracts relevant subgraphs by a Hopefield network. Then, it uses GRC to embed VN in candidate subgraphs.

Since GRC and NeuroViNE can only consider one server resource (*i.e.*, CPU), we employ them to evaluate the scalability and single-resource embedding performance of our method. Then, in a separate benchmark, we use another variations of Best Fit and First Fit algorithms to investigate the general setting. Parameters α, β, κ_1, and κ_2 are set to 30, 3, 10, and 10, respectively.

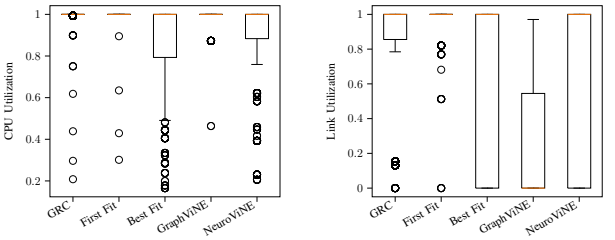
B. Benchmarks

Parallelizability. First, we examine the parallelizability of GraphViNE provided by the spatial GNN architecture, where the operations are local to the individual nodes. To this end, we increased the physical network size from 100 servers to 1500 servers and measured the runtime on both CPU and GPU. Figure 3 reveals a significant difference between the CPU and GPU results, where the former runtime grows exponentially while the latter runtime is relatively constant. Specifically, the parallelism has improved the runtime by about 8 times on average and 20 times in the best case. Furthermore, we



(a) Revenue (b) Cost

Fig. 5: Comparision of revenue and cost



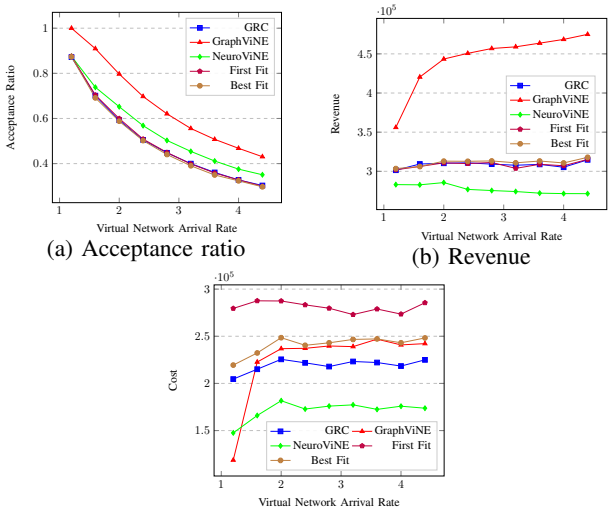
(a) CPU Utilization (b) Link Utilization

Fig. 6: Maximum utilization of servers and links throughout the simulation

investigated the sensitivity of GraphViNE to the virtual network size by increasing it up to 50% of the physical network (*i.e.*, 50 nodes) and observed only a 3 seconds increase in the embedding runtime. Thus, we conclude that the physical network size is the major factor that limits scalability. It is worth mentioning that neither GRC nor NeuroViNE provides a native mechanism to exploit available parallel processing capabilities.

Acceptance Ratio. A long-term acceptance ratio is an important metric that affects the system's profit. Figure 4 shows the acceptance ratio of different algorithms in a long simulation of about 2000 episodes. Note that during this period, the acceptance ratios reach a steady-state and remain constant due to a stationary virtual network arrival process. The First Fit algorithm fragments the resources too much and experiences a significant drop in its capacity for accepting new virtual networks. GraphViNE increases the acceptance ratio by about 20%, 25%, and 100% compared to the NeuroViNE, GRC, and First Fit algorithms, respectively. The performance of GRC and Best Fit are similar.

Revenue and Cost. Figures 5a and 5b, respectively, compare the cost and revenue of different algorithms. Computations are based on the equations (5) and (6), where $\zeta_r = 1$ and $\xi_r = 1$. Therefore, the effect of server resources is the same as the link bandwidth. Since GraphViNE embeds more VNs during the same period, its revenue is also higher. Despite a higher revenue, our proposed algorithm incurs a lower cost than First Fit and Best Fit methods. Since GRC optimizes the fraction of revenue to cost, it has a lower cost than First Fit and Best Fit algorithms while providing almost the same revenue. NeuroViNE achieves a low cost because it typically embeds smaller virtual networks. Consequently, it can not



(a) Acceptance ratio (b) Revenue

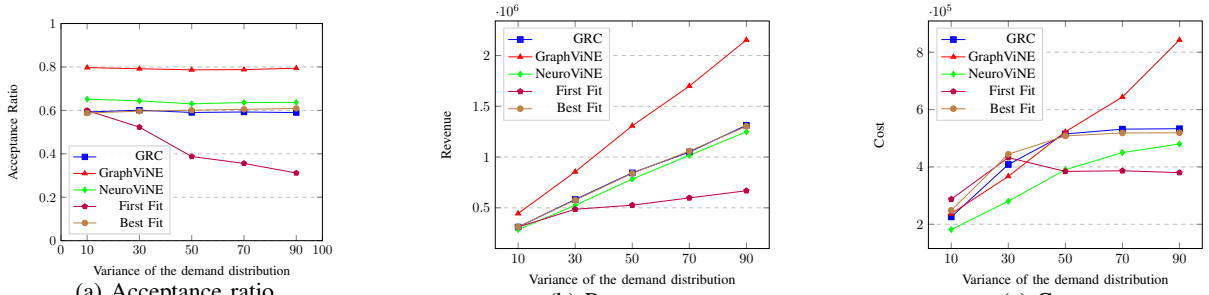
(c) Cost

Fig. 7: Comparison with different virtual network arrival rates

achieve a high revenue. Moreover, GraphViNE achieves the highest revenue-to-cost-ratio (*i.e.*, 1.87), while NeuroViNE, GRC, Best Fit, and First Fit, respectively, achieve 1.57, 1.37, 1.25, and 1.08 ratios. NeuroViNE strictly limits the embedding to a relatively small set of servers, which results to a poor revenue. This is an issue of all the pre-processing mechanisms that eliminate servers. GraphViNE, on the other hand, guides the embedding algorithm by introducing a diverse set of starting points, but allows it to use all the reachable servers.

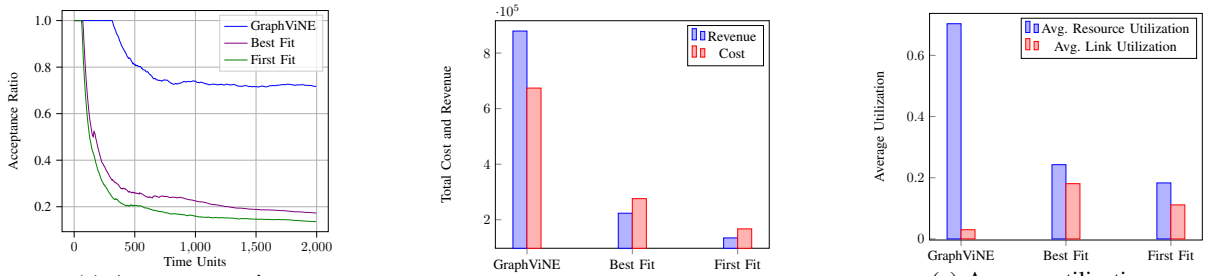
Utilization. Box plots in Fig. 6 show the distribution of the maximum server and link utilization throughout the simulation. Figure 6a shows that GraphViNE packs active servers with virtual nodes and only leaves few servers under-utilized. Moreover, GraphViNE keeps bandwidth utilization at a moderate level to maintain the connectivity (see Fig. 6b). Best Fit and NeuroViNE methods demonstrate a high level of CPU load-balancing to retain the ability of serving future demands. However, they exhibit a lower performance at bandwidth utilization, which can be attributed to not considering servers and the network simultaneously. Similarly, GRC and First Fit methods have a high server and link utilization, which limits their ability to embed virtual networks.

Arrival Rate. To further measure the scalability and performance, we compared GraphViNE with other algorithms under different virtual network arrival rates. Specifically, we examined different arrival rates between 1.2 and 4.4 and presented the results in Fig. 7. Evident from Fig. 7a, as the arrival rate increases, the acceptance rate decreases. Nevertheless, the acceptance rate achieved by GraphViNE remains consistently higher compared to other algorithms. We observe that NeuroViNE exhibits better scalability, but its performance is still about 18 percent lower than our method. NeuroViNE has a low revenue and cost that reveals its higher acceptance ratio stems from its ability to embed smaller VNs that provide less revenue. Although the acceptance ratio decreases, GraphViNE manages to increase the number of embedded



(a) Acceptance ratio (b) Revenue (c) Cost

Fig. 8: The effect of the virtual link demand



(a) Acceptance ratio (b) Cost and revenue (c) Average utilization

Fig. 9: The testing results of algorithms with GPU and memory as extra node resources

virtual networks, which results in higher revenue and cost (see Figs. 7b and 7c). However, other algorithms are already at their limits with 2 virtual networks per unit of time, and thus their revenue and cost do not change significantly.

Virtual Link Demand. In this experiment, we investigate the effect of the virtual link demand on different algorithms' performance. Thus, we change the variance of the virtual link demands, which is a normal distribution with mean 4, from 10 to 90 and present the results in Fig. 8. We observe that GraphViNE manages to embed the more extensive virtual links inside the physical servers, and thus its acceptance ratio does not change, which leads to an increase of the revenue (see 8b). This trend also happens for NeuroViNE, Best Fit, and GRC algorithms, which shows that they also acknowledge the importance of embedding large virtual links inside physical servers. However, the Firs Fit algorithm suffers from higher virtual link demands, which significantly reduces its acceptance ratio.

Multi-dimensional Resources. GraphViNE seamlessly supports virtual network embedding in multi-dimensional resource settings. In this experiment, we investigate the performance of GraphViNEwhile considering two extra resources (*i.e.*, GPU and memory). Since GRC and NeuroViNE do not support multi-dimensional resource allocation, we omit them here. We changed the First Fit to choose the first physical server with sufficient capacity for every resource type. Best Fit is also modified to choose the server that its summation of remaining resource capacities is maximum. Figure 9 shows that GraphViNE outperforms other algorithms in every metric. Specifically, GraphViNE accepts about 70% more virtual networks and improves the revenue by about 3 folds. From Fig. 9b, notice that GraphViNE embeds a significant number of large virtual links inside the physical servers and thus does not pay for their embedding in the physical network. Consequently, GraphViNE achieves a lower cost than its revenue, which is the ideal scenario. However, Best Fit and First Fit map many virtual links to multiple physical links, and thus their cost exceeds their revenue. This conclusion is further verified in Fig. 9c where GraphViNE produces a higher resource utilization and a lower network utilization compared to other methods.

VII. Conclusion

This work presented the design and evaluation of GraphViNE, a VNE algorithm based on an adversarially-regularized variational autoencoder with a spatial-based GNN. GraphViNE uses a server representation that is computed by the GNN to cluster the servers and efficiently detect distinct embedding options. We demonstrated that the employed spatial-based GNN model provides a level of parallelism, which significantly increases the scalability of our approach. We extensively evaluated the proposed algorithm by comparing it with several existing algorithms and demonstrated its performance and speedup. An attractive extension of this work is adding edge features to the GNN model for constraints such as link delay. Also, analyzing the trade-off between scalability and efficiency by investigating a finer-grained control over the number of clusters instead of computing it with the elbow method is interesting.

REFERENCES

[1] E. Amaldi *et al.*, "On the computational complexity of the virtual network embedding problem," *Electron. Notes Discret. Math.*, vol. 52, pp. 213–220, 2016.

[2] Y. Zong *et al.*, "Virtual network embedding for multi-domain heterogeneous converged optical networks: Issues and challenges," *Sensors*, vol. 20, no. 9, p. 2655, 2020.

[3] A. Blenk *et al.*, "NeuroViNE: A neural preprocessor for your virtual network embedding algorithm," in *Proc. IEEE INFOCOM*, 2018, pp. 405–413.

[4] D. Wang *et al.*, "RLS-VNE: Repeatable large-scale virtual network embedding over substrate nodes," in *Proc. IEEE GLOBECOM*, 2019, pp. 1–6.

[5] A. Blenk *et al.*, "Boost online virtual network embedding: Using neural networks for admission control," in *Porc. IEEE CNSM*, 2016, pp. 10–18.

[6] Z. Yan *et al.*, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, 2020.

[7] F. Scarselli *et al.*, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2008.

[8] W. L. Hamilton *et al.*, "Inductive representation learning on large graphs," in *Proc. NIPS*, 2017, p. 1025–1035.

[9] S. Pan *et al.*, "Learning graph embedding with adversarial training methods," *IEEE Trans. Cybern.*, vol. 50, no. 6, p. 2475–2487, 2020.

[10] R. L. Thorndike, "Who belongs in the family?" in *Psychometrika*, vol. 18, no. 4, Dec 1953, pp. 267–276. [Online]. Available: https://doi.org/10.1007/BF02289263

[11] G. Kibalya *et al.*, "A novel dynamic programming inspired algorithm for embedding of virtual networks in future networks," *Comput. Netw.*, vol. 179, p. 107349, 2020.

[12] H. Cao *et al.*, "NoViSec: Novel virtual network mapping framework for secure software-defined networking," in *Proc. IEEE WCNCW*, 2020, pp. 1–6.

[13] A. Pentelas *et al.*, "Network service embedding with multiple resource dimensions," in *Porc. IEEE/IFIP NOMS*, 2020, pp. 1–9.

[14] F. Hosseini *et al.*, "Probabilistic virtual link embedding under demand uncertainty," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1552–1566, 2019.

[15] H. Yao *et al.*, "A continuous-decision virtual network embedding scheme relying on reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 864–875, 2020.

[16] M. Dolati *et al.*, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM WKSHPS*, 2019, pp. 879–885.

[17] M. He *et al.*, "DROI: Energy-efficient virtual network embedding algorithm based on dynamic regions of interest," *Comput. Netw.*, vol. 166, p. 106952, 2020.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[19] "PyTorch Scatter," https://github.com/rusty1s/pytorch_scatter, accessed: 2020-07-07.

[20] P. Erdös and A. Rényi, "On the evolution of random graphs," in *Pub. Math. Inst. Hungarian Acad. Science*, vol. 5, 1960, pp. 17–61.

[21] "Colaboratory," https://colab.research.google.com/notebooks/intro.ipynb, accessed: 2020-07-07.

[22] L. Gong *et al.*, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. IEEE INFOCOM*, 2014, pp. 1–9.