NeuroViNE: A Neural Preprocessor for Your Virtual Network Embedding Algorithm

Andreas Blenk* Patrick Kalmbach* Johannes Zerwas* Michael Jarschel† Stefan Schmid‡ Wolfgang Kellerer*
*Technical University of Munich, Germany †Nokia Bell Labs, Munich, Germany ‡University of Vienna, Austria

***Abstract*—Network virtualization enables increasingly diverse network services to cohabit and share a given physical infrastructure and its resources, with the possibility to rely on different network architectures and protocols optimized towards specific requirements. In order to ensure a predictable performance despite shared resources, network virtualization requires a strict performance isolation and hence, resource reservations. Moreover, the creation of virtual networks should be fast and efficient. The underlying NP-hard algorithmic problem is known as the Virtual Network Embedding (VNE) problem and has been studied intensively over the last years. This paper presents NeuroViNE, a novel approach to speed up and improve a wide range of existing VNE algorithms: NeuroViNE is based on a search space reduction mechanism and preprocesses a problem instance by extracting relevant subgraphs, i.e., good combinations of substrate nodes and links. These subgraphs can then be fed to an existing algorithm for faster and more resource-efficient embeddings. NeuroViNE relies on a Hopfield network, and its performance benefits are investigated in simulations for random networks, real substrate networks, and data center networks.**

I. Introduction

Context: Virtual Networks Providing Predictable Performance. Today's communication networks are challenged with increasing diversity of applications (e.g., live streaming, IoT applications, 5G, etc.) as well as frequently changing demands, e.g., due to user mobility. Network virtualization is an attractive paradigm that allows accommodation of different applications on a shared infrastructure, while supporting application-specific network architectures and optimizations. Whereas resource sharing enables high network utilization, efficient performance isolation mechanisms need to be in place, to ensure a predictable application performance. This in turn introduces the need for mechanisms to quickly and efficiently provision virtual networks and their resources.

The Problem: Fast and Efficient Embeddings. The problem of computing a minimal viable resource footprint for a virtual network is known as the Virtual Network Embedding (VNE) problem. The VNE problem is NP-hard in general [1] and there exist many exact and heuristic algorithms [2]–[8]. While exact solutions are attractive for their resource efficiency (in terms of resource footprints of the virtual networks), they are expensive to compute [9]. In contrast, heuristic algorithms solve the VNE problem in acceptable time, but their embedding footprints can be far from optimal. Especially problematic are heuristic algorithms that split the embedding problem into a node and a link mapping step [10]: substrate nodes that may be ranked highly in the node step (e.g., due to available node resources) might be located far away from each other, thus requiring a lot of bandwidth resources.

The Idea: Subgraph Extraction. Our paper is motivated by the observation that efficient solutions to the VNE problem place frequently communicating nodes close to each other. Moreover, we observe that many time-intensive VNE algorithms may benefit when executed on subgraphs selected intelligently from the substrate network. Consequently, a preprocessing mechanism that extracts subgraphs providing (1) high probabilities for being able to accommodate virtual networks and (2) ensuring low-cost embeddings can potentially lead to shorter runtimes while preserving high embedding qualities.

Our Contributions. We propose a generic preprocessing mechanism called *NeuroViNE*, to both, speed up and improve existing rigorous VNE algorithms, by performing an effective search space reduction and subgraph extraction. Concretely, for a given virtual network request, *NeuroViNE* leverages a Hopfield network [11], [12], which is a special artificial neural network, to preselect a subset of "good" substrate nodes — the Hopfield network extracts whole valuable subgraphs. Hopfield networks do not require any kind of learning; the Hopfield network designed and used in this paper computes a probability for each node to be part of the subgraph. An existing VNE algorithm is then used to find the final embedding solution. Using extensive simulations, for which we reimplemented five well-known embedding algorithms (namely GRC [10], GREEDY [7], SDP [2], and the two VINEYARD [5] algorithms D-VINE and R-VINE), we evaluate the performance of *NeuroViNE* and find that it provides an attractive extension to many algorithms. We also find that our approach can either improve runtime or embedding quality, and in some cases even both metrics at the same time.

Paper Organization. The remainder of this paper is organized as follows. Section II introduces our model. Our solution, *NeuroViNE*, is presented in Section III and evaluated in Section IV. After reviewing related work in Section V, we conclude and discuss future work in Section VI.

II. The Model

The Virtual Network Embedding (VNE) problem [2] is defined as follows.

Substrate Network. We consider an undirected graph $\mathcal{G}^p := (\mathcal{N}^p, \mathcal{L}^p, \mathcal{C}^p, \mathcal{B}^p)$ to describe the substrate network. $\mathcal{N}^p$ is the set of physical nodes $\mathcal{N}^p := \{N_i^p\}_{i=1}^s$, where s is the number of nodes of the substrate. $\mathcal{L}^p$ is the set of physical edges with $\mathcal{L}^p \subseteq \mathcal{N}^p \times \mathcal{N}^p$ and $L_{ij}^p = (N_i^p, N_j^p)$ denoting a physical

link. Each node $N_i^p \in \mathcal{N}^p$ has CPU capacity C_i^p and residual capacity $C_i^p(t)$ at time t. Every link $L_{ij}^p \in \mathcal{L}^p$ has bandwidth B_{ij}^p and residual bandwidth $B_{ij}^p(t)$ at time t.

Virtual Network Requests (VNRs). A Virtual Network Request (VNR) is an undirected graph $\mathcal{G}^v := (\mathcal{N}^v, \mathcal{L}^v, \mathcal{C}^v, \mathcal{B}^v)$. $\mathcal{N}^v$ is the set of all virtual nodes $\mathcal{N}^v := \{N_m^v\}_{m=1}^r$ of a VNR, where r is the number of virtual nodes of the VNR. $\mathcal{L}^v$ contains all virtual links with $\mathcal{L}^v \subseteq \mathcal{N}^v \times \mathcal{N}^v$ and $L_{mn}^v = (N_m^v, N_n^v)$. Vice versa, every virtual node $N_m^v \in \mathcal{N}^v$ has a CPU requirement C_m^v and every virtual link $L_{mn}^v \in \mathcal{L}^v$ has a bandwidth requirement B_{mn}^v.

Virtual Network Embedding. VNE algorithms try to map an arriving VNR $\mathcal{G}^v$ to the substrate network $\mathcal{G}^p$. A successful embedding is then defined by a node mapping f_N and a link mapping f_L function [7]:

$$f_N : \mathcal{N}^v \to \mathcal{N}^p \tag{1}$$

$$f_L : \mathcal{L}^v \to 2^{\mathcal{L}^p} \setminus \emptyset \tag{2}$$

such that

$$\forall N_m^v \in \mathcal{N}^v : C_m^v \leq C^p_{f_N(N_m^v)}(t) \tag{3}$$

$$\forall L_{mn}^v \in \mathcal{L}^v : \forall L_{ij}^p \in f_L(L_{mn}^v) : B_{mn}^v \leq B_{ij}^p(t) \tag{4}$$

For a valid mapping of VNR $\mathcal{G}^v$ to $\mathcal{G}^p$, it is necessary to map all virtual nodes $\mathcal{N}^v$ to distinct substrate nodes $\mathcal{N}^p$ (Eq. 1) and to map all virtual links $\mathcal{L}^v$ to paths in the substrate network (Eq. 2), i.e., a subset of links $\mathcal{L}^{p\prime} \in 2^{\mathcal{L}^p} \setminus \emptyset$. Note that we assume unsplittable flows in this work. The CPU requirements of all virtual nodes (Eq. 3) and the bandwidth requirements of all virtual links must be fulfilled (Eq. 4); virtual nodes and links can only be mapped on substrate nodes and links providing enough residual CPU or bandwidth (Eq. 3, Eq. 4).

III. The NeuroViNE Preprocessor

In a nutshell, *NeuroViNE* is a preprocessor extracting subgraphs. Using subgraphs composed of good candidate nodes can provide various benefits in terms of algorithm efficiency or solution quality: e.g., such subgraphs can help improve solution qualities of embedding algorithms, as we demonstrate for heuristics [6], [10], or can provide time savings when combined with runtime-expensive VNE algorithms, such as algorithms based on mathematical programming [2]. Before presenting our Hopfield network adaptation for subgraph extraction, we will first provide background on Hopfield networks in the next section.

A. Background: Hopfield Network

Hopfield networks are a form of recurrent artificial neural networks [12]. A Hopfield network has one layer of neurons that are all interconnected, as shown in Fig. 1. The following parameters specify a Hopfield network at time t:

- Number of neurons m.
- Vector $\mathbf{V}(t) \in [0,1]^m$ representing each neuron's external state.
- Vector $\mathbf{U}(t) \in \mathbb{R}^m$ representing each neuron's internal state.

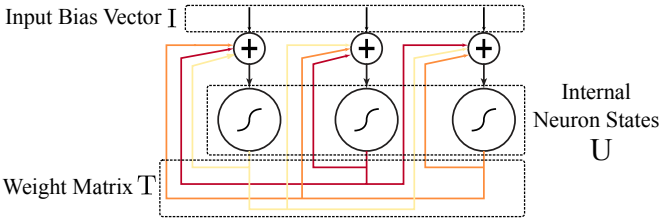


Fig. 1. A Hopfield Network with 3 neurons. The output of one neuron is fed back as input to the other neurons. Input and output have the same size. The final output is given by the values of the neurons.

- Bias vector $\mathbf{I} \in \mathbb{R}^m$, serving as independent input to each neuron.
- Symmetric weight matrix $\mathbf{T} \in \mathbb{R}^{m \times m}$, with $\mathbf{T}_{ij} = \mathbf{T}_{ji}$ being the weight of the connection between neuron i and j and $\mathbf{T}_{ii} = 0$.
- Activation function $g : \mathbb{R}^m \to [0,1]^m$, calculating the *activation*, i.e., the external state of each neuron.

One way to fully describe the behavior of neurons of a Hopfield network is the differential equation [11]:

$$\frac{d\mathbf{U}(t)}{dt} = -\frac{\mathbf{U}(t)}{\tau_{HF}} + \mathbf{T}\mathbf{V}(t) + \mathbf{I}. \tag{5}$$

The input of one neuron $\mathbf{U}_i(t)$ is the sum of the output of all other neurons plus the bias value $\mathbf{I}_i$. The external neuron state $\mathbf{V}(t)$ at point in time t is determined by using a smooth approximation to the step function as activation function $g(\mathbf{V}(t))$, as originally proposed in [11]:

$$\mathbf{V}(t) = g(\mathbf{U}(t)) = \frac{1}{2} \cdot \left(1 + \tanh\left(\frac{\mathbf{U}(t)}{u_0}\right)\right). \tag{6}$$

The free parameter u_0 controls the steepness of the curve. We fix u_0 to 1 in our experiments since its variation showed no significant impact on our results. When implementing a Hopfield network in software, parameter τ_{HF} of Eq. 5 can be set to 1 [11]. To avoid notational clutter, we drop the time index t if the context is clear: e.g., $\mathbf{U}$ instead of $\mathbf{U}(t)$.

When executing a Hopfield network, its neurons potentially converge towards stable states. Given the stable states, it is shown that the neuron values imply a local minimum of the so-called *energy* equation of the Hopfield network:

$$E = -\frac{1}{2}\mathbf{V}^T\mathbf{T}\mathbf{V} - \mathbf{V}^T\mathbf{I}. \tag{7}$$

When used for optimization, parameters $\mathbf{T}$ and $\mathbf{I}$ have to be chosen in such a way that they relate to the objective of the optimization problem to be minimized. By solving Eq. 5, we minimize Eq. 7 and, thus, the original objective represented by $\mathbf{T}$ and $\mathbf{I}$ as well.

B. System Overview

NeuroViNE's preprocessor uses a Hopfield network. Fig. 2 shows an overview of *NeuroViNE*. The main components of *NeuroViNE* are the ratings for nodes and links of the substrate, a selection function $\zeta(\mathcal{G}^v)$ determining the number of physical nodes in the subgraph, and the parameters of the

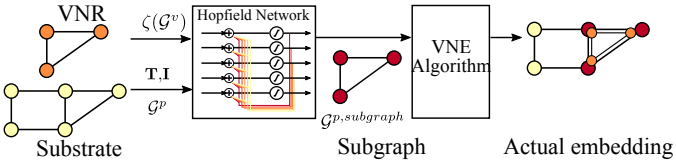


Fig. 2. *NeuroViNE* uses a Hopfield network to extract a subgraph from the substrate and then applies a VNE algorithm. Dependent on the VNR size, a selection function $\zeta(\mathcal{G}^v)$ determines the number of selected nodes. The size of $\mathcal{G}^p$ determines the number of neurons of the Hopfield network, here five. The current state of $\mathcal{G}^p$ determines the weight matrix $\mathbf{T}$ and the bias vector $\mathbf{I}$. The VNE algorithm embeds the VNR on the subgraph $\mathcal{G}^{p,\text{ subgraph}}$.

Algorithm 1 Preselection and Virtual Network Embedding.

Require: $\mathcal{G}^p, \mathcal{G}^v$
1: $\boldsymbol{\Xi}(t) \leftarrow \beta \cdot$ calculate_noderanks($\mathcal{N}^p$)
2: $\boldsymbol{\Psi}(t) \leftarrow$ calculate_edgeranks($\mathcal{L}^p$)
3: $\zeta \leftarrow$ set_number_of_preselected_nodes ($\mid \mathcal{N}^v \mid$)
4: $(\mathbf{T}, \mathbf{I}) \leftarrow$ create_hopfield_network($\boldsymbol{\Xi}$, $\boldsymbol{\Psi}$", ζ)
5: $\mathbf{V} \leftarrow$ execute_hopfield_network($\mathbf{T}$, $\mathbf{I}$)
6: $\mathcal{G}^{p,\text{ subgraph}} \leftarrow \mathcal{G}^p$
7: **for** $V_i \in \mathbf{V}$ **do**
8: **if** $V_i < 0.5$ **then**
9: remove_node($\mathcal{G}^{p,\text{ subgraph}}$, $N_i^{p,\text{ subgraph}}$)
10: **end if**
11: **end for**
12: $f_N \leftarrow$ map_nodes($\mathcal{G}^v$, $\mathcal{G}^{p,\text{ subgraph}}$)
13: $f_L \leftarrow$ map_edges($\mathcal{G}^v$, $\mathcal{G}^p$, f_N)
14: **return** (f_N, f_L)

Hopfield network (weight matrix $\mathbf{T}$, bias vector $\mathbf{I}$ and energy function E). Our goal is to accept as many virtual networks as possible while reducing cost, i.e., increasing revenue-cost-ratios. Algorithm 1 shows how all components interplay. The relevant parts are explained throughout the next sections.

C. Substrate Node and Edge Ranking

In *NeuroViNE*, neurons in the Hopfield network represent substrate nodes. In order to select the potentially best nodes from the substrate, the neurons representing those nodes must be associated with low values of Eq. 7. We introduce a node and edge (node connectivity) rating for this purpose, which will be used for the setting $\mathbf{T}$ and $\mathbf{I}$ in Sec. III-E.

The node calculation step (line 1) can consider any node attribute, e.g., memory or CPU, or any attribute of a node's connected edges, e.g., latency or bandwidth. In this work, the node ranking vector $\boldsymbol{\Xi}(t) \in \mathbb{R}^{|\mathcal{N}^p|}$ considers the residual CPU capacities at time t as follows:

$$\boldsymbol{\Xi}_i(t) = \beta \cdot \frac{\max_{N_j^p \in \mathcal{N}^p} C_j^p(t) - C_i^p(t)}{\max_{N_j^p \in \mathcal{N}^p} C_j^p(t)} \qquad \forall N_i^p \in \mathcal{N}^p, \quad (8)$$

where β is a parameter that weighs the importance of the node ranking. By taking the residual CPU, the Hopfield network tries to identify subgraphs with a high remaining CPU capacity, thus providing a high chance for acceptance. Dividing by the highest available CPU capacity normalizes the rankings to the interval from 0 and 1. Setting $\beta = 7$ showed the best performance in the conducted simulations.

The edge ratings (line 2) are represented in the matrix $\boldsymbol{\Psi}(t) \in [0,1]^{|\mathcal{N}^p| \times |\mathcal{N}^p|}$. Determining the values of $\boldsymbol{\Psi}(t)$ involves two steps; (1) setting the weights of all links and (2) calculating the shortest paths between all nodes based on these weights. The weights of all links are set as

$$w_{\text{HF}}(ij) = B_{max}^p(t) - \frac{B_{ij}^p(t)}{B_{max}^p(t)}, \quad (9)$$

where $B_{max}^p(t) := \max_{L_{ij}^p \in \mathcal{L}^p} B_{ij}^p(t)$ is the maximum residual bandwidth at time t among all links. The idea behind the weight setting is to integrate the distance between nodes while simultaneously considering the remaining capacities of links. The link weights w_{HF} are then used to calculate the distance matrix $\mathbf{D}(t)$ containing the costs of shortest paths between all nodes at time t. Finally, the values of the matrix $\boldsymbol{\Psi}$ are

$$\boldsymbol{\Psi}_{ij}(t) = \gamma \cdot \frac{\mathbf{D}_{ij}(t)}{\max(\mathbf{D}(t))}, \quad (10)$$

where every value is normalized by the maximum value of the matrix $\mathbf{D}(t)$, and the parameter γ weights the edge ratings. $\gamma = 3$ provided the best results in our simulations. Note that we assume that at least one path with remaining capacity exists, otherwise the algorithm would not be executed at all. The obtained ranking results in the selection of a subset of substrate nodes with small distance in terms of hop count to each other, but high residual capacity on connecting links/paths. Combining distance and residual capacity into the path calculation ensures that the virtual nodes can be connected successfully with low cost.

D. Node Number Selection Functions

The node number selection functions determine the amount of nodes that should be preselected, i.e., that determine the size of the subgraph. As this function is interchangeable, it allows to tailor the embeddings to specific goals. For instance, the selection function can weight smaller networks higher than larger networks. To realize such strategy, an operator would simply have to select more nodes for smaller networks than larger networks. Thereby, the acceptance ratio for smaller networks might be larger than for larger networks. We propose three selection functions: $\zeta_{\text{const}}(\mathcal{G}^v)$, $\zeta_{\text{factor}}(\mathcal{G}^v)$, and $\zeta_{\text{inverse}}(\mathcal{G}^v)$.

Line 3 calculates the number of nodes based on one specific function. Function $\zeta_{\text{const}}(\mathcal{G}^v)$ simply sets ζ to any pre-defined constant value κ for all VNRs; it is independent of the requested graph $\mathcal{G}^v$:

$$\zeta = \zeta_{\text{const}}(\mathcal{G}^v) = \kappa \quad (11)$$

Function $\zeta_{\text{factor}}(\mathcal{G}^v)$ uses the size of a VNR $\mathcal{G}^v$, i.e, $\mid \mathcal{N}^v \mid$:

$$\zeta = \zeta_{\text{factor}}(\mathcal{G}^v) = \kappa \cdot \mid \mathcal{N}^v \mid . \quad (12)$$

κ allows to linearly scale the number of selected nodes: e.g., $\kappa = 1$ forces the Hopfield network to select exactly the number of requested VNR nodes. Larger κ should increase the probability of accepting a VNR. Function ζ_{inverse} selects

ζ inversely proportional to the number of requested virtual nodes:

$$\zeta = \zeta_{\text{inverse}}(\mathcal{G}^v) = \frac{\kappa}{\mid \mathcal{N}^v \mid} \tag{13}$$

ζ_{inverse} is tailored towards the demands of optimal algorithms. When embedding small networks, optimal algorithms can use the entire network, while for VNRs, the substrate search space should be reduced for runtime reasons. For correct operation of *NeuroViNE*, it is important that the Hopfield network preselects a number of nodes equal or greater than the number of requested nodes of the VNR (i.e., $\zeta \geq \mid \mathcal{N}^v \mid$). Note that the Hopfield network may not always select exactly the requested number of nodes. Instead, the number of selected nodes might slightly vary around ζ. Generally, increasing the amount of preselected nodes increases the likelihood that a VNR can be accepted, but also decreases the efficiency of *NeuroViNE*. Accordingly, Sec. IV analyzes the effect of the choice and the parameter settings of the selection functions.

E. Hopfield Network Creation

With the ranking and number of nodes to be selected, we can create a Hopfield network with $\mid \mathcal{N}^p \mid$ neurons. The creation of the network is done in line 4 of Algorithm 1. This step involves the calculation of the neuron weight matrix $\mathbf{T}$ and the bias vector $\mathbf{I}$. First, we calculate the parts of $\mathbf{T}$ and $\mathbf{I}$ that are caused by the constraints $\mathbf{T}^{\text{constraint}}$ and $\mathbf{I}^{\text{constraint}}$. This is done for all elements of the weight matrix and bias vector using the k-out-of-n rule proposed by Page and Tagliarini [13] as follows

$$\begin{aligned} \mathbf{T}_{ij}^{\text{constraint}} &= \begin{cases} 1 & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \\ \mathbf{I}_{k}^{\text{constraint}} &= -(2 \cdot \zeta - 1), \end{aligned} \tag{14}$$

where i and j are the indexes of the weight matrix $\mathbf{T}^{\text{constraint}}$ and k is the index of the bias vector $\mathbf{I}^{\text{constraint}}$. The k-out-of-n rule ensures that for a decision problem with n variables, k variables will be chosen, i.e., be set to 1. Accordingly, the Hopfield network should choose ζ substrate nodes out of all $\mathcal{N}^p$. Finally, the actual weight matrix $\mathbf{T}$ and the actual bias vector $\mathbf{I}$ can be calculated:

$$\mathbf{T} = -2(\mathbf{\Psi}(t) + \alpha \cdot \mathbf{T}^{\text{constraint}}) \tag{15}$$

$$\mathbf{I} = -(\mathbf{\Xi}(t) + \alpha \cdot \mathbf{I}^{\text{constraint}}) \tag{16}$$

The parameter α weighs the constraint terms against the optimization terms. A high α value ensures that the number of selected nodes in the subgraph is close to the number of nodes of a VNR. Setting $\alpha = 10$ showed the best performance in the presented simulations.

The energy function of the Hopfield network is given as:

$$E = \mathbf{V}^T(\mathbf{\Psi}(t) + \alpha \cdot \mathbf{T}^{\text{constraint}})\mathbf{V} + \mathbf{V}^T(\mathbf{\Xi}(t) + \alpha \cdot \mathbf{I}^{\text{constraint}}), \tag{17}$$

where $\mathbf{V}$ is the vector of the neuron states. The inclusion of the terms $\mathbf{T}^{\text{constraint}}$ and $\mathbf{I}^{\text{constraint}}$ ensures that the constraint is satisfied, i.e., the correct number of nodes are selected. The minimum of Eq. 17 implies the best possible combination of nodes according to the edge and node ranking.

F. Finding the Stable Neuron State

To solve Eq. 5 we use the Runge-Kutta-Method as described in Algorithm 2. First, the internal neuron states $\mathbf{U}$, the iteration count i, and the state change variable Δ are initialized (line 1 to line 3). The while loop (line 4 to line 19) repeats until either the state change variable Δ is smaller than a threshold δ or the maximum number of iterations $i_{\max}$ is reached. Inside the while loop, the next iteration of $\mathbf{U}$ and the difference to the last iteration are calculated (line 6 to line 11). Finally, the activation function calculates the output of the neurons (line 11). The parameter τ controls the step size of the updates. Results presented in Sec. IV are obtained with τ fixed to 0.1. The value of 0.1 yielded the best performance.

G. Modifications for Data Center Use Case

So far, we made the usual assumption that virtual nodes can be embedded on any substrate node. In order to apply our Hopfield network-based algorithm in data centers, we introduce additional concepts.

Data center topologies come in many flavors, including FatTree (FT) [14] or BCube (BC) [15]. In the context of VNE, we need to distinguish two types of nodes: servers and switches. Servers host the virtual machines carrying out the customers' operations, whereas switches are used to connect servers. As such, only servers can host virtual machines.

To model such physical networks, we leverage that Hopfield networks allow to fix some variables in advance, i.e., before executing them. This method is called *clamping* [16]: certain neurons are fixed to certain values. Clamping can be useful when variables should be excluded from the search space or if some variable solutions are already given.

Concretely, the modified Hopfield network system requires additional information given by the set $\mathcal{C}_0$ containing all switches, and the following constraint:

$$V_i = 0 \qquad \forall N_i^p \in \mathcal{C}_0. \tag{18}$$

The constraint is implemented using clamping. Generally, the algorithm now **requires** additional information about whether we face a data center topology and also the set $\mathcal{C}_0$.

Line 11 to line 14 provide the data center modification; they are marked in blue. If the topology is a data center, the elements of the internal state vector $\mathbf{U}$ representing switches ($\mathcal{C}_0$) are set to a large negative value while the deviations $\mathbf{dU}$ are set to 0. When the Hopfield network execution converges, this will result in a solution where the nodes $\mathcal{C}_0$ are all set to 0 — excluding switches from the subgraph.

To be a compatible alternative, the data center algorithm uses a fallback algorithm (any VNE algorithm) in case the preprocessor-based subgraph leads to an early rejection. This might increase the computational resources and runtime; however, as the results will demonstrate, the benefits in terms of cost savings and revenues favor this design. Furthermore, using a heuristic algorithm as alternative still guarantees a compatible algorithm runtime.

Algorithm 2 Execution of the Hopfield Network.

Require: $\mathbf{I}, \mathbf{T}, \Delta, i_{\max}$, dc_flag, $\mathcal{C}_0$

1: $\mathbf{U} \leftarrow$ Random
2: $i \leftarrow 0$
3: $\Delta \leftarrow \infty$
4: **while** $(\Delta > \delta) \wedge (i < i_{max})$ **do**
5: $\quad i \leftarrow i + 1$
6: $\quad \mathbf{k_1} \leftarrow \mathbf{TV} + \mathbf{I} - \mathbf{U}$
7: $\quad \mathbf{k_2} \leftarrow \mathbf{T}(\frac{1}{2}(1 + \tanh(\frac{\mathbf{U} + \frac{1}{2}\tau\mathbf{k_1}}{u_0}))) + \mathbf{I} - (\mathbf{U} + \frac{1}{2}\tau\mathbf{k_1})$
8: $\quad \mathbf{k_3} \leftarrow \mathbf{T}(\frac{1}{2}(1 + \tanh(\frac{\mathbf{U} - \tau k_1 + 2\tau k_2}{u_0}))) + \mathbf{I} - (\mathbf{U} - \tau\mathbf{k_1} + 2\tau\mathbf{k_2})$
9: $\quad \mathbf{dU} \leftarrow \frac{\mathbf{k_1} + 4\mathbf{k_2} + \mathbf{k_3}}{6}$
10: $\quad$ **if** $dc\_flag == True$ **then**
11: $\quad\quad$ **for** $i \in \mathcal{C}_0$ **do**
12: $\quad\quad\quad U_i \leftarrow -\infty$
13: $\quad\quad\quad dU_i \leftarrow 0$
14: $\quad\quad$ **end for**
15: $\quad$ **end if**
16: $\quad \mathbf{U} \leftarrow \mathbf{U} + \tau \cdot \mathbf{dU}$
17: $\quad \Delta \leftarrow |\mathbf{dU}|$
18: $\quad \mathbf{V} \leftarrow \frac{1}{2}(1 + \tanh(\frac{\mathbf{U}}{u_0}))$
19: **end while**
20: **return** $\mathbf{V}$

H. Embedding of the Links and Nodes

After the preselection is complete, the subgraph $\mathcal{G}^{p,\text{ subgraph}}$ is created (line 6 to line 11 of Algorithm 1.). $\mathcal{G}^{p,\text{ subgraph}}$ contains only the substrate nodes whose neuron states are active, i.e., $\mathcal{N}^{p,\text{ subgraph}} = \{N_i^p \mid N_i^p \in \mathcal{N}^p \wedge V_i > 0.5\}$. We call the node mapping with the subgraph $\mathcal{G}^{p,\text{ subgraph}}$ in line 12 of Algorithm 1. After all virtual nodes are mapped, we map the edges by calling the edge mapping function in line 13. The edge mapping is performed using the complete substrate network $\mathcal{G}^p$. If node and link mappings are successful, the network is embedded to the substrate.

IV. Evaluation

NeuroViNE can be employed together with many existing algorithms, and we have evaluated different combinations in different settings using extensive simulations, both on artificial and real network topologies.

A. Methodology

The following sections summarize the simulation setups involving VNE algorithms, substrate network graphs, virtual network requests and VNE performance metrics. For all settings, we performed at least 10 runs for every substrate setting: e.g., we used 10 substrate graphs for both random network models. For every setup, we run the online embedding until 2500 VNRs were processed.

1) Virtual Network Embedding Algorithms: In our evaluation, we compare five embedding algorithms with and without *NeuroViNE*: the Global Resource Capacity algorithm (GRC) [10], the Greedy algorithm (GREEDY) [7], the optimal Shortest Distance Path algorithm (SDP) [2], the two VINEYARD [5] algorithms D-VINE and R-VINE. We use *NeuroViNE* in combination with GRC, SDP, D-VINE and R-VINE as VNE algorithms; we refer to the algorithm variants with Hopfield preprocessing HF-GRC, HF-SDP, HF-D-VINE and HF-R-VINE. The Hopfield data center variant is called HF-GRC-DC.

GRC introduces the Global Resource Capacity metric for every node of the substrate network. GRC embeds the virtual nodes to the substrate nodes with the highest metric values. The parameters of GRC are set as proposed in [10]. GREEDY rates the nodes based on their residual CPU capacity only. In the edge embedding stage, GRC and GREEDY embed all virtual edges consecutively depending on their rank. The virtual edges are mapped using the shortest path in terms of number of hops. D-VINE and R-VINE use a relaxed integer programming formulation of the VNE problem. Either deterministic (D-VINE) or randomized (R-VINE) rounding is applied to provide a solution for a VNR. We use the shortest path versions of D-VINE and R-VINE. SDP is an (exact) mixed integer programming-based algorithm that tries to find a cost optimal embedding. As SDP does not scale to large topologies (> 50), the execution is prematurely interrupted after 30 seconds.

2) Substrate Network Graphs: We evaluate the performance of the algorithms with five substrate graph types: the two random network graph models Erdős-Rényi (ER) [17] and Barabási-Albert (BA) [18], realistic network graphs from the Topology Zoo (TPZ) [19], and the two data center topologies FatTree (FT) and BCube (BC).

Random Network Graphs. All ER graphs are generated with 100 nodes and a connection probability of 0.11. The CPU capacities of the nodes and the bandwidths of the links are equally distributed in the range from 50 to 100. This substrate setup is the same as used by Gong et al. [10], allowing a close comparison of the performance of GRC and HF-GRC. The same resource generation process is used for BA graphs; however, BA has two more parameters m_0 and m. Both parameters can be used to analyze different network densities. The parameter m_0 is set to 20. The value m of the BA model, which controls the number of links in the network, ranges from 1 to 10 in order to analyze the impact of varying densities: a higher m leads to a higher density.

Realistic Network Graphs. To evaluate the performance on more realistic network graphs, we use all substrates with $\mid \mathcal{N}^p \mid \geq 50$ nodes [19]. Overall, we analyze 28 topologies from the Topology Zoo. If the data set provides unconnected topologies, we use the largest components of the graphs. Using TPZ graphs does not only provide insights on more realistic setups, it also provides insights on the algorithms' performances on larger topologies: e.g., the KDL network graph has 709 nodes and 815 links. Compared to existing VNE evaluations, this network is quite large. We randomly assign CPU values between 50 and 100 and bandwidth values between 250 and 500. The increased bandwidth in comparison to the random network graphs is necessary because the topologies are much more sparsely connected than the random network graphs.

Using the same bandwidth values as for the random graphs would result in rejection of nearly all VNRs.

Data Center Networks. Two data center types are investigated: BCube (BC) and FatTree (FT). The host and link capacities are set to 100. The bandwidth value is chosen to resemble realistic setups with 1 Gbps; the VNRs resemble requests between 10 Mbps and 100 Mbps (see VNR generation description). Since switches cannot host virtual machines, their CPU capacity is set to 0; this also excludes the investigation of needed processing for network traffic.

3) Virtual Network Requests: For ER, BA and TPZ substrates, the VNRs are created using the ER method [17]. The number of virtual nodes is equally distributed in the range between 2 and 20. The connection probability is set to 0.5. The required CPU capacities and bandwidths of virtual nodes and links are equally distributed in the range from 0 to 50. The arrival rate λ of VNRs is set to 5 arrivals per 100 time units. Every VNR has a negative exponentially distributed lifetime with an average of 500 time units. This VNR generation is identical to the one used by Gong et al. [10].

For data center topologies, the Waxman graph model (WAX) is used for VNRs. The model parameter are set as follows: $\alpha_{WAX} = 0.2$ and $\beta_{WAX} = 0.4$. The number of nodes is equally distributed between 3 and 10; the CPU capacities of the nodes between 2 and 20; the virtual link demands between 1 and 10. This is in compliance with a recently published study of VNE in data center networks [20].

4) Metrics: The following metrics are used to quantify the quality of the solution for the online VNE problem; the metrics are standard in literature to quantify VNE algorithms [7], [9].

Acceptance Ratio (**AR**) provides one general dimension to compare VNE algorithms. It is the ratio of accepted VNRs among all arrived VNRs. A high AR is generally one indicator for an efficient embedding. Formally, the AR for a given time interval $\mathcal{T} := [t^{\text{start}}, t^{\text{end}}]$ is defined as

$$\text{AR}(\mathcal{T}) := \frac{\mid \mathcal{R}^{\text{acc}}(\mathcal{T}) \mid}{\mid \mathcal{R}^{\text{rej}}(\mathcal{T}) \cup \mathcal{R}^{\text{acc}}(\mathcal{T}) \mid}, \tag{19}$$

where $\mathcal{R}^{\text{acc}}(\mathcal{T})$ is the set of accepted VNRs and $\mathcal{R}^{\text{rej}}(\mathcal{T})$ is the set of rejected VNRs during $\mathcal{T}$.

Cost (**COS**) comes from the invested node and link resources. To realize a virtual network on a substrate, node resources and link resources need to be assigned to a virtual network, i.e., a substrate operator needs to invest. While there is a 1-to-1 mapping between requested and assigned node resources, the costs for realizing a virtual path depend on the physical path length. The cost $\text{COS}(\mathcal{G}^v)$ of a VNR is the sum of all substrate resources that have been used to embed the VNR and is defined as:

$$\text{COS}(\mathcal{G}^v) := \sum_{N_m^v \in \mathcal{N}^v} C_m^v + \sum_{L_{mn}^v \in \mathcal{L}^v} \mid f_L(L_{mn}^v) \mid \cdot B_{mn}^v, \tag{20}$$

where $\mid f_L(L_{mn}^v) \mid$ provides the length of the physical path on which the virtual edge L_{mn}^v has been mapped.

Revenue (**REV**) of a VNR is determined by the requested virtual resources. The more resources a virtual network requests, the higher the revenue. The revenue $\text{REV}(\mathcal{G}^v)$ is the sum of all virtual resources and is given as:

$$\text{REV}(\mathcal{G}^v) := \sum_{N_m^v \in \mathcal{N}^v} C_m^v + \sum_{L_{mn}^v \in \mathcal{L}^v} B_{mn}^v. \tag{21}$$

Simply building the sum of, e.g., CPU and data rate resources, is a common approach in VNE [7].

Total Revenue (**TR**) in a given time interval $\mathcal{T}$ is the sum of the revenues of accepted VNRs $\mathcal{R}^{\text{acc}}(\mathcal{T})$ of this time interval

$$\text{TR}(\mathcal{T}) := \sum_{\mathcal{G}^v \in \mathcal{R}^{\text{acc}}(\mathcal{T})} \text{REV}(\mathcal{G}^v). \tag{22}$$

Revenue-Cost-Ratio (**RCR**) of a VNR is defined as the fraction of $\mathcal{G}^v$'s revenue $\text{REV}(\mathcal{G}^v)$ over $\mathcal{G}^v$'s cost $\text{COS}(\mathcal{G}^v)$

$$\text{RCR}(\mathcal{G}^v) := \frac{\text{REV}(\mathcal{G}^v)}{\text{COS}(\mathcal{G}^v)}. \tag{23}$$

The best possible RCR can be achieved by a 1-to-1 mapping between requested and allocated demands, i.e., all virtual nodes and links are implemented on one physical node and link respectively. A high RCR indicates low embedding cost: an inevitable target for network operators.

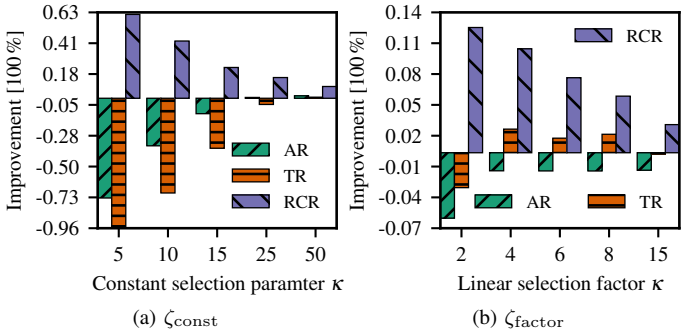


Fig. 3. Illustration on the impact of κ for ζ_{const} and ζ_{factor}. Figures show the improvement in percent for each metric: AR, TR, and RCR. Positive values indicate a performance improvement achieved by *NeuroViNE*. GRC vs. HF-GRC. Larger κ values let *NeuroViNE* improve metrics.

B. Optimization Opportunities

We first exemplarily analyze the impact of the two node selection functions ζ_{const} and ζ_{factor} on the three key performance metrics AR, TR, and RCR. This evaluation provides initial insights on the tuning possibilities of *NeuroViNE*. Fig. 3a shows the performance improvement for the constant function ζ_{const} and Fig. 3b shows the improvement for the linear function ζ_{factor}. For each selection size/factor κ, the figures show the bar plots of the improvement defined as $\text{Improvement} = (\text{HF-GRC} - \text{GRC})/\text{GRC} \cdot 100\,\%$, where HF-GRC and GRC are placeholders for any performance metric of the respective algorithm. A positive value indicates a better performance with *NeuroViNE*.

Constant Selection Size. Fig. 3a illustrates that HF-GRC blocks 73 % VNRs when $\zeta = \kappa\ (= \zeta_{\text{const}})$ is set to a small value like 5 nodes. Consequently, the TR decreases as less virtual networks are accepted. The RCR, however, improves:

the nodes of the small virtual networks are always embedded very close to each other. Increasing the number of nodes to 25 increases the AR to a level that is minimally higher than the one of GRC; whereas TR is still minimally lower. Yet, *NeuroViNE* improves the RCR by 18 %. When *NeuroViNE* selects subgraphs with $\kappa = 50$ nodes, it shows a similar or slightly higher AR and TR, while it still provides a 9 % higher RCR: with *NeuroViNE*, all virtual networks with a size up to 20 nodes can be embedded much more efficiently. However, the problem of ζ_{const} is the insensitivity to the number of substrate network- and VNR nodes. The number of nodes needs to be predetermined based on the VNR request size to function well.

Selection Factor - Dependent on the Number of Virtual Nodes. Fig. 3b shows that *NeuroViNE* using the factor-based function ζ_{factor} has almost the same AR as GRC, except for $\kappa = 2$. The TR is statistically insignificantly better than with GRC, again besides for $\kappa = 2$ where it is slightly lower. For $\kappa = 2$, the Hopfield network does not always select enough nodes for bigger virtual networks. Thus, bigger networks are rejected even before the VNE algorithm can be run. Fig. 3b indicates that HF-GRC improves the RCR ratio up to 8 % for factors ranging from 2 to 8. We believe that trading off the increased RCR for the slightly worse AR and TR is justifiable: decreasing the cost by 10 % for 1 % lower TR. As we aim at a significantly higher RCR for real topologies in the following studies, we use ζ_{factor} with $\kappa = 2.5$, which shows a still acceptable tradeoff between RCR and AR or TR.

C. How does NeuroViNE perform on different topology types?

To illustrate the impact of different substrate topology models, Fig. 4 shows the performance of all algorithms for BA, ER and substrates from the TPZ.

Random graphs vs. realistic ones. Fig. 4a demonstrates that GRC accepts slightly more VNRs on random network graphs, whereas HF-GRC accepts the most VNRs on realistic network graphs (TPZ) - HF-GRC is highly efficient when faced with sparsely connected real topologies. While D-VINE and R-VINE are generally not able to compete with GRC and HF-GRC, *NeuroViNE* still increases the AR for both algorithms, as indicated by HF-D-VINE and HF-R-VINE. Fig. 4b confirms the observations from Fig. 4a: GRC has slightly higher revenues than HF-GRC on random graphs, whereas *NeuroViNE* accepts similar VNRs like the VINEYARD algorithms on random graphs. For TPZ, HF-GRC shows again the overall best performance: *NeuroViNE* accepts more and larger networks. As Fig. 4c illustrates for RCR, *NeuroViNE* outperforms all other VNE algorithms, independent of the topology type — *NeuroViNE* always achieves the highest RCR values.

An illustrative example. Fig. 5 illustrates an exemplary embedding for one VNR for GRC (Fig. 5a) and HF-GRC (Fig. 5b) on the KDL graph (709 nodes and 815 links) from the Topology Zoo: GRC embeds virtual nodes to central nodes on this topology; unfortunately, these central nodes do not need to be near each other, as the spatially distributed green squares illustrate in Fig. 5a; long paths are required to connect the virtual nodes. In contrast to GRC, HF-GRC selects nodes that are close to each other and have high capacities in their vicinities, as indicated by the orange-filled nodes centrally located in the network in Fig. 5b. Since all nodes are close, the paths between them are also shorter. This improves not only the embedding quality (RCR), but also the AR. We conclude that *NeuroViNE* is very useful on realistic network topologies. In particular, it finds valuable nodes in sparsely connected topologies. Furthermore, independently of the VNE algorithm, it can help improve the RCR and reduce cost.

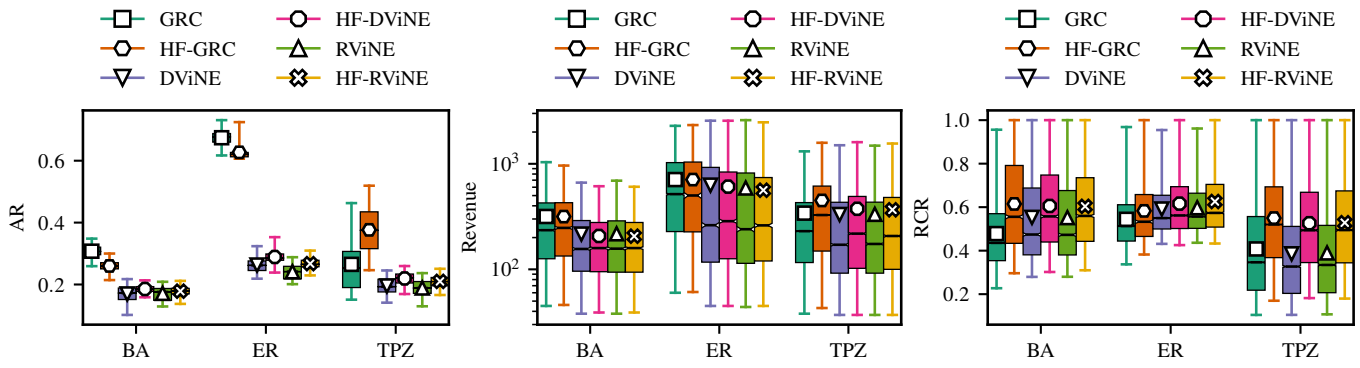
D. Do the benefits extend to data centers?

Fig. 6 reports on the results for data center topologies when using GRC, HF-GRC-DC, and GREEDY. The Hopfield variant HF-GRC-DC accepts slightly more VNRs (AR) for the FatTree (FT) topology, whereas almost no change is observable for the BCube (BC) topology. Note the high AR values that are already obtained without applying *NeuroViNE*; the values actually range from 0.95 to 0.98 for the FT topology. We omit the presentation of the TR because it is hardly affected. However, the improvements achieved by *NeuroViNE* in RCR are significant; HF-GRC-DC improves the performance by 10 % for FT and by 7 % for BC. The average cost of all embeddings is reduced by placing clusters within racks where nodes are close to each other. This brings numerous advantages: the clusters might yield lower inter-communication latency or the provider can operate the network more energy-efficient.

E. Can we speed up exact (optimal) algorithms?

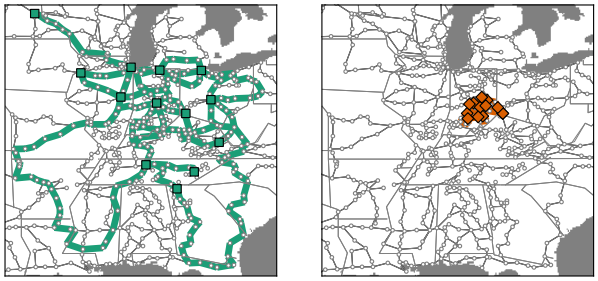
To answer this question, we compare the performance of SDP and the HF variants HF-SDP in combination with the selection functions ζ_{factor} and ζ_{inverse}. For ζ_{factor}, κ is set to 2.5; for ζ_{inverse} it is set to 300. The results for AR and TR are quickly summarized without illustrating them: *NeuroViNE* does neither improve nor worsen the results with statistical significance. For the RCR over the number of nodes, Fig. 7a illustrates that both Hopfield variants are better than SDP, demonstrating again that *NeuroViNE* selects subgraphs with nodes that are physically close to each other. Interestingly, the improved RCR does not improve the AR: the embeddings are still suffering from the blocking due to bottleneck links.

However, *NeuroViNE* highlights another interesting angle for improvement: the preprocessing efficiently decreases the model creation time and model solving time of the exact algorithm while preserving its solution quality. The model creation time encompasses the time to create the model for the solver used by the optimal algorithm, e.g., to acquire memory and to set all variables and constraints. For the HF variants, it also involves the creation and execution of the Hopfield networks. The linear selection function decreases the model creation and solving time the most, as shown in Fig. 7c: the solver spends the least time to find a feasible solution; the solver even achieves a ten times lower model creation time. In contrast, SDP and HF-SDP (inverse) are consuming the whole given processing time ($30\,s$) for VNR requests having more than 8 nodes when solving the model. For HF-SDP (inverse), this can be explained by the behavior of ζ_{inverse}: for smaller



(a) Acceptance Ratio. (b) Revenue. (c) Revenue-cost-ratio.

Fig. 4. Figures compare the performance between baseline algorithms and *NeuroViNE*-enhanced versions among BA, ER and TPZ topologies. Subfigures show results for AR, REV (log-scaled), and RCR. For each measure, one group of boxplots is shown per topology type. The boxplots show mean, median, and the upper 97.5 % and lower 2.5 % of values through the whiskers. REV is shown to illustrate which VNRs (smaller or larger ones) are accepted.



(a) GRC (b) HF-GRC

Fig. 5. Comparison of node locations for a single VNR between GRC (green-filled squares) and HF-GRC (orange-filled diamonds).

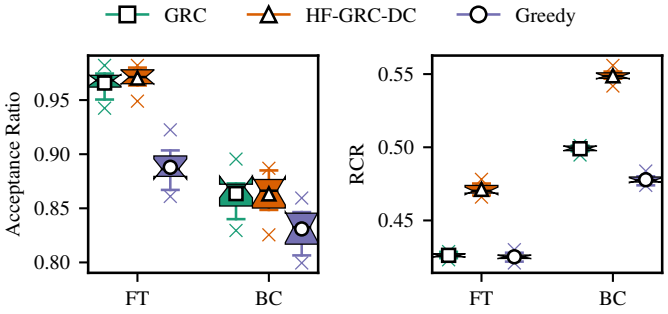


Fig. 6. Comparison between GRC, GREEDY and HF-GRC for data center topologies. Two metrics are shown: AR and RCR. *NeuroViNE* significantly improves the RCR of the GRC algorithm.

networks it selects more substrate nodes whereas for larger networks it reduces the number of nodes. Using *NeuroViNE*, exact algorithms based on mathematical programming can become credible alternatives to heuristics on larger topologies.

V. Related Work

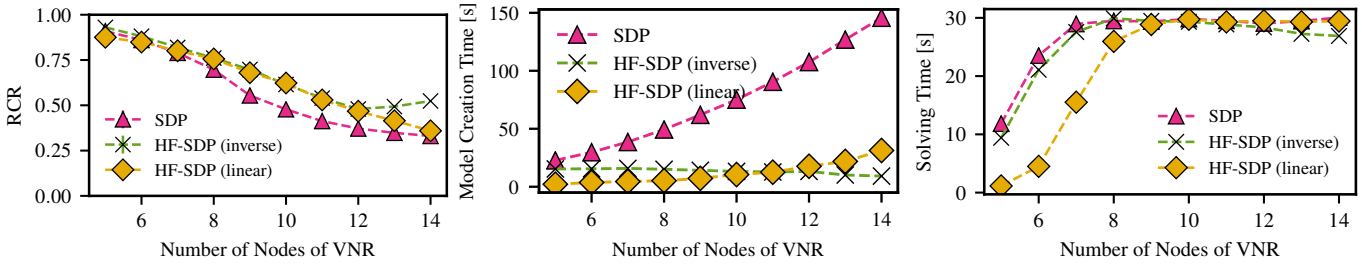
Artificial neural networks have already been successfully applied on a variety of combinatorial optimization problems, including the traveling salesman problem [11], [12] (which introduced the Hopfield network), the shortest path problem [21], the hub-placement problem [22], or the Knapsack problem [23]–[25]. In the context of communication networks, recent work used Hopfield networks e.g., for channel selection in radio networks [26]. Recent work has successfully applied deep neural networks [27] for cloud resource management. Beside, in our recent work we have successfully used neural networks for admission control for VNE [28].

To the best of our knowledge, Hopfield networks have not yet been applied to VNE. A general survey of existing VNE algorithms is given in [9]. Algorithms for solving the VNE problem broadly fall in two categories: those which solve the VNE problem in one step, during which both nodes and links are mapped, and those which divide the problem into separate node and link mapping steps. An example for the former are exact algorithms based on mathematical programming [2]. An example for a non-optimal one step algorithm is [6].

Most heuristic algorithms are 2-step algorithms: in the first step, every virtual node is assigned to the best substrate node that fulfills the capacity constraints. In order to determine the best substrate node for every virtual node, the substrate nodes are usually ranked according to substrate node attributes, e.g., based on their remaining CPU [7] or graph measure [8], [29]. To integrate the physical connections between nodes, i.e., physical links and paths, into the ranking, heuristic algorithms apply a global ranking procedure. For instance, the MCRank [4] or the GRC rating [10] make a global rating of the nodes. For each node, their global ranking metric integrates also the distance to all other nodes in the network, e.g., by using random walks through the substrate network. Generally, after all nodes are rated and embedded, shortest path or multi commodity flow approaches are used to interconnect them.

VI. Future Work

We believe that our work opens interesting directions for future research. For example, it will be interesting to explore alternative approaches such as Boltzmann machines and Pointer Networks, introducing additional challenges in modeling and adapting those networks to accommodate resource constraints. Systems that can adapt, e.g., to changing



(a) Revenue-cost-ratio. (b) Model Creation Time. (c) Solver Time.

Fig. 7. The mean values with 95 % confidence intervals of RCR, modeling time and solving time for SDP and HF-SDP in combination with linear and inverse selection function. Performance metrics are depicted over number of nodes of VNRs. *NeuroViNE* improves RCR and reduces model creation and solving time.

networking environments or dynamically changing virtual network demands, open further intriguing research directions. In particular in changing environments, the parameters of neural networks, e.g., Hopfield networks, might be needed to adapt and be set automatically at runtime. As finding the correct parameter setting is a time demanding and exhaustive task, machine learning techniques for automatically finding parameter settings will be particularly interesting to study.

Acknowledgment

This work has been performed in part in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1) funded by the German BMBF (Project ID 16KIS0473), and in part in the framework of the EU project FlexNets funded by the European Research Council under the European Unions Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets). This work reflects only the authors' view and the funding agency is not responsible for any use that may be made of the information it contains. Stefan Schmid was supported by Aalborg University's talent project PreLytics. The authors would like to thank Michael Manhart for his invaluable implementation work.

References

[1] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, Jun. 2016.

[2] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha, "Optimal Virtual Network Embedding: Node-Link Formulation," *IEEE Trans. Netw. Serv. Manage.*, vol. 10, no. 4, pp. 356–368, Dec. 2013.

[3] M. Feng, J. Liao, J. Wang, S. Qing, and Q. Qi, "Topology-aware Virtual Network Embedding based on multiple characteristics," in *Proc. IEEE ICC*, Jun. 2014, pp. 2956–2962.

[4] S. Zhang, Z. Qian, J. Wu, and S. Lu, "An Opportunistic Resource Sharing and Topology-Aware mapping framework for virtual networks," in *Proc. IEEE INFOCOM*. IEEE, Mar. 2012, pp. 2408–2416.

[5] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 783–791.

[6] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[7] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 17–29, Mar. 2008.

[8] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM CCR*, vol. 41, no. 2, p. 38, Apr. 2011.

[9] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, Feb. 2013.

[10] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1–9.

[11] J. J. Hopfield and D. W. Tank, ""Neural" computations of decisions in optimization problems," *Biological Cybernetics*, vol. 52, pp. 141–152, 1985.

[12] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons." *Proc. Natl. Acad. Sci. USA*, vol. 81, no. 10, pp. 3088–3092, May 1984.

[13] G. Tagliarini, J. Christ, and E. Page, "Optimization using neural networks," *IEEE Trans. Comp.*, vol. 40, no. 12, pp. 1347–1358, Dec. 1991.

[14] M. Al-Fares and A. Loukissas, Alexanderand Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.

[15] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," *ACM SIGCOMM CCR*, vol. 39, no. 4, pp. 63–74, October 2009.

[16] L. Gislén, C. , and B. Söderberg, ""Teachers and Classes" with Neural Networks," *International Journal of Neural Systems*, vol. 01, no. 02, pp. 167–176, January 1989.

[17] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[18] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[19] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[20] S. Haeri and L. Trajković, "Virtual Network Embeddings in Data Center Networks," in *Proc. IEEE ISCAS*, 2016.

[21] H. Rauch and T. Winarske, "Neural networks for routing communication traffic," *IEEE Ctrl. Sys. Mag.*, vol. 8, no. 2, pp. 26–31, Apr. 1988.

[22] K. Smith, M. Krishnamoorthy, and M. Palaniswami, "Neural versus traditional approaches to the location of interacting hub facilities," *Location Science*, vol. 4, no. 3, pp. 155–171, Oct. 1996.

[23] S. Abe, J. Kawakami, and K. Hirasawa, "Solving inequality constrained combinatorial optimization problems by the hopfield neural networks," *Neural Networks*, vol. 5, no. 4, pp. 663–670, 1992.

[24] B. J. Hellstrom and L. N. Kanal, "Knapsack Packing Networks," *IEEE Trans. Neural Netw.*, vol. 3, no. 2, pp. 302–307, Mar. 1992.

[25] M. Ohlsson, C. Peterson, and B. Söderberg, "Neural Networks for Optimization Problems with Inequality Constraints: The Knapsack Problem," *Neural Computation*, vol. 5, no. 2, pp. 331–339, Mar. 1993.

[26] M. Alsenwi, H. Elsayed, and M. Elghoneimy, "Optimization of channel selection in cognitive heterogeneous wireless networks," in *11th ICENCO*, Dec. 2015, pp. 38–43.

[27] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Hotnets*, ser. HotNets '16. New York, NY, USA: ACM, 2016, pp. 50–56.

[28] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in *Proc. CNSM*. IFIP/IEEE, Oct. 2016.

[29] Y. Zhu and M. H. Ammar, "Algorithms for Assigning Substrate Network Resources to Virtual Network Components." in *Proc. IEEE INFOCOM*, vol. 1200, 2006, pp. 1–12.