Boost Online Virtual Network Embedding: Using Neural Networks for Admission Control

Andreas Blenk*, Patrick Kalmbach*, Patrick van der Smagt†‡, Wolfgang Kellerer*
*Chair of Communication Networks, Department of Electrical and Computer Engineering
Technical University of Munich, Germany
†fortiss, Associate Institute of Technical University of Munich, Germany
‡BRML, Department of Informatics, Technical University of Munich, Germany

***Abstract*—The allocation of physical resources to virtual networks, i.e., the virtual network embedding (VNE), is still an on-going research field due to its problem complexity. While many solutions for the online VNE problem exist, only few have focused on methods that can be generally applied for optimization of online embeddings. In this paper, we propose an admission control based on a Recurrent Neural Network (RNN) to improve the overall system performance for the online VNE problem. Before running a VNE algorithm to embed a virtual network request, the RNN predicts whether the request will be accepted by the VNE algorithm based on the current state of the substrate and the virtual network request (VNR). The RNN prevents VNE algorithms from spending time on VNRs that are either infeasible or that cannot be embedded in acceptable time. In order to train and operate the RNN efficiently, we additionally propose new representations for substrate networks and virtual network requests. The representations are based on topological and network resource features to represent the substrate network and the VNRs with low computational complexity. Via simulations, we show that our admission control reduces the overall computational time for the online VNE problem by up to $91\,\%$ while preserving VNE performance on average. Using our new substrate and request representations, the RNN achieves an accuracy ranging between $89\,\%$ and $98\,\%$ for different VNE algorithms, substrate sizes, and VNR arrival rates.**

***Index Terms*—Virtual network embedding, mathematical optimization, admission control, machine learning, neural networks**

I. Introduction

Network virtualization plays an important role for future communication networks such as 5G [1], [2]. With network virtualization, end-service providers can request, use and control virtual networks (VNs) according to their service specific demands. For instance, VNs can be used to dynamically and flexibly interconnect virtual network functions [3]. VNs are sliced out of the infrastructure and provide the needed networking resources, i.e., network link and node capacity for virtual network tenants. In order to operate virtualized infrastructures efficiently, infrastructure operators need to deploy sophisticated resource allocation mechanisms. The allocation of physical to virtual resources, also called the virtual network embedding (VNE), is still an ongoing research area due to its computational complexity [4].

Due to runtime and computational efficiency, infrastructure operators mostly depend on heuristic embedding algorithms, providing only sub-optimal solutions. In contrast, VNE algorithms based on mathematical programming, e.g., mixed integer programing (MIP), guarantee optimality [5], [6], [7]. However, optimal algorithms suffer the problem of exponentially growing runtime for increasing problem sizes. Hence, mechanisms are needed that are able to improve the runtime efficiency in particular for the online VNE problem.

In the mathematical programming research area, on-going studies investigate the potential of applying concepts from machine learning (ML) in order to improve the performance of algorithms and solvers for mathematical programming formulations [8], [9], [10]. For instance, ML is used to learn branching during optimization [9] or to detect the best solver configurations for varying problem instances [10]. In this paper, we use ML techniques to predict whether a problem instance, i.e., virtual network request (VNR), will be accepted by a VNE algorithm. In detail, we propose to exploit the behavior of VNE algorithms. If faced with similar problem instances, e.g., similar VNRs, the outputs of VNE algorithms, i.e., accepted or rejected, are likely to be equal. We therefore provide an admission control based on ML to predict the acceptance decisions of VNE algorithms. In detail, the admission control uses a Recurrent Neural Network (RNN) to predict the probability whether VNRs will be accepted by VNE algorithms. The VNE algorithms only process those requests that are accepted by the RNN. Hence, such admission control improves the runtime of systems that face the online VNE problem as VNE algorithms do not spend time on infeasible VNR requests. This improves the computational efficiency as it decreases the amount of needed computational resources.

It has already been shown that graph features and topological features can be used for heuristic-based VNE algorithms [11], [12], [13], [14], [15], [16]. Those algorithms use the knowledge that topological graph features of VNRs show a high correlation with topological features of substrate networks for efficient embeddings. In those embeddings, virtual nodes are placed on substrate nodes with similar graph features. In this work, we use graph features to provide a compact representation of the substrate state and the VNRs.

Beside VNE, the application of graph features for classification has already been successfully applied to data mining tasks in chem-informatics and bioinformatics [17], [18]. For substrate networks and virtual networks, we propose two representations based on feature-vectors constructed from graph as well as resource features (attributes) of the networks. One

978-3-901882-85-2

representation for the substrate network and one representation for the VNRs. The RNN-based admission control uses the new representations as input in order to predict whether VNRs will be accepted. More specifically, the RNN filters out VNRs that are likely to be infeasible or not solvable in acceptable time, given experience from previous runs. Thus, the admission control reduces the computational time for the online VNE problem while not increasing the cost of embedding networks.

We make the following main contributions:

- We initiate the study of using Artificial Neural Networks (ANNs) in admission control for VNE algorithms.
- We present a novel procedure to improve the runtime performance for the online VNE problem.
- We propose network representation models depending on graph and resource features.
- We present and publish an RNN-based admission control and a VNE simulator together with this paper [19].

Our simulations demonstrate that our admission control based on an RNN works for two different VNE algorithms. The admission control reduces the runtime of the online VNE problem between $20\,\%$ and $91\,\%$ while even improving embedding metrics, such as acceptance ratio, revenue, cost, and revenue-cost-ratio in some cases on average.

II. Related Work

We categorize the related work into two areas. The first area classifies embedding algorithms according to the solution techniques they rely on, i.e., mathematical programming or heuristics. An overview and classification of VNE algorithms is given in [4]. The second area classifies existing admission control mechanisms that are based on Artificial Neural Networks (ANNs) and are deployed for resource management in communication networks.

Exact solutions for VNE algorithms guarantee optimality, i.e., the best possible solution for a given objective function. However, as the VNE problem is NP-hard [4], exact solutions suffer from exponentially growing runtime. Still, problem formulations are proposed that can serve as baselines for heuristic algorithms for small problem instances [5], [6], [7], [20]. Further VNE algorithms exist that solve only subproblems via mathematical programming, for instance the path embedding [21]. Other algorithms exist that use rounding techniques to derive feasible solutions from Linear Programming (LP) solutions [22].

Many heuristic solutions have proposed to take topological attributes into account while solving the embedding. Those solutions try to identify the importance of substrate nodes via their graph or resource attributes [14]. Further, the importance of nodes can be quantified via their hop distance [12]. Besides, instead of taking only one or two node metrics into account, node ranks are also done on multiple node characteristics [13], [11]. Others apply techniques such as Markov Random Walks or Markov Chains to rank nodes [15], [16]. Those techniques try to order nodes with respect to the overall network state.

Admission control systems in VNE have been used to postpone the embedding of VNRs [21] or for selecting VNRs

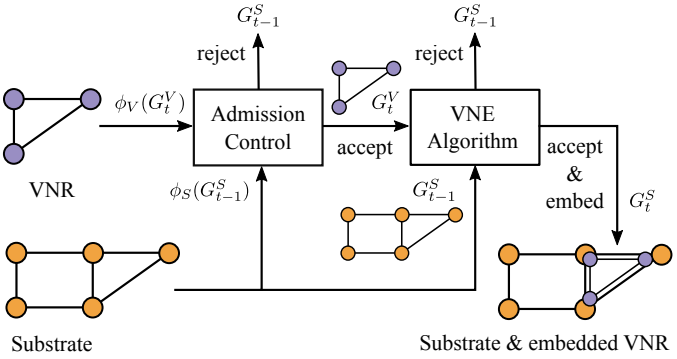


Fig. 1. Admission Control for a VNE Algorithm for online VNE.

that provide a high benefit, e.g., in terms of revenue [23], [24], [25]. In communication networks, the use of ANNs for admission control is not new. For instance, it has been proposed in the context of ATM networks, cellular networks, or multimedia wireless networks [26], [27], [28], [29], [30]. As an example, in case of ATM, an ANN was used to rapidly decide whether a flow request might lead to network over-utilization [30], [27], [26].

In this paper, we propose an admission control using an RNN to improve the system runtime for the online VNE problem. In contrast to the state-of-the-art, our admission control has to classify graphs, i.e., VNRs, based on substrate states, i.e., a changing graph. For this, we propose a network representation based on topological and resource features.

III. An Admission Control for Online Virtual Network Embedding

A schematic view of our proposed system is shown in Figure 1. The system consists of two main blocks, the admission control part and the VNE algorithm. In general, it is the goal of the admission control to assess the chances for the successful generation of a solution by the VNE algorithm. The admission control classifies VNRs that arrive over time t. A VNR G^V_t can be either *infeasible*, unsolvable in acceptable time (*no solution*), or *accepted*. A VNR is classified as *no solution* if a solver cannot find a feasible solution in time T. Note that this does not imply that the VNR is generally infeasible. The time constraint T allows our system to trade off embedding quality and needed computational runtime.

The admission control uses an RNN to realize the classification. The input of the RNN needs to be compact and independent of the size of substrates and VNRs. For this, the functions ϕ_S and ϕ_V create feature vector representations of substrate networks and VNRs, explained in Sec. V. Using the VNR representations $\phi_V(G^V_t)$ and $\phi_S(G^S_{t-1})$ as input, the RNN filters VNRs. In case a VNR is classified as not solvable in acceptable time (*no solution*) or *infeasible*, it will be rejected. Otherwise, the VNR is forwarded to the VNE algorithm for processing. We chose an RNN to capture changing VNRs, which we want to analyze in future work.

In our online VNE system, the (computational) runtime needed for one VNR is compound of the mathematical

model creation time (including the creation of $\phi_V(G_t^V)$ and $\phi_S(G_{t-1}^S)$ and filtering for the RNN case) and the model solving time. Note that the solving time is bounded by T.

The VNE algorithm tries to embed the VNR G_t^V on the current substrate network G_t^S under a time constraint T. If the VNE algorithm can actually embed the VNR in time T, the VNR is placed on the substrate. This transfers the substrate network state G_{t-1}^S into G_t^S. If the VNR cannot be embedded as it is *infeasible* or *no solution* is found in T, the substrate network state is not affected.

IV. Virtual Network Embedding Formulation

In this section, we mathematically describe the online VNE problem. For integrity, we also briefly describe four VNE performance metrics, which are used in the results (Sec. VII).

A. Substrate Network

We denote an undirected graph $G^S := (\mathcal{N}^S, \mathcal{E}^S, \mathcal{A}^S)$ as a substrate network. $\mathcal{N}^S$ describes the set of physical nodes $\mathcal{N}^S := \{n_i\}_{i=1}^n$ where n is the total number of network nodes. $\mathcal{E}^S$ is the set of physical edges with $\mathcal{E}^S \subseteq \mathcal{N}^S \times \mathcal{N}^S$ and $e_{ij}^S = (n_i^S, n_j^S)$ denoting a physical edge. Similar to [21], we denote by $\mathcal{P}^S$ the set that contains all loop free paths in the substrate network. A path $p_{in} \in \mathcal{P}^S$ consists of a set of edges that connect n_i^S to n_n^S, i.e., $\forall p_{in} \in \mathcal{P}^S, p_{in} := \{e_{ij}^S, e_{jk}^S, \cdots, e_{lm}^S\} \subseteq \mathcal{E}^S$. Each element $o \in \mathcal{N}^S \cup \mathcal{E}^S$ is associated with a set of attributes $\mathcal{A}^S(o) := \{a_1^S, \ldots, a_l^S\}$, where l specifies the type, e.g., CPU or bandwidth (bw). In this paper, we consider CPU capacity for nodes and link capacity (bandwidth) for edges as resource attributes. This means $\forall n \in \mathcal{N}^S : a_{\text{CPU}}^S \in \mathcal{A}^S(n)$ with $\text{CPU}(n) \in \mathbb{R}_0^+$ specifying the CPU of a substrate node and $\forall e \in \mathcal{E}^S : a_{\text{bw}}^S$ with $\text{bw}(e) \in \mathbb{R}_0^+$ giving the bandwidth of a substrate link.

B. Virtual Network Requests (VNRs)

We denote an undirected graph $G^V := (\mathcal{N}^V, \mathcal{E}^V, \mathcal{A}^V)$ as a VNR. $\mathcal{N}^V$ contains all virtual nodes n_i^V of a VNR. $\mathcal{E}^V$ contains all edges with $\mathcal{E}^V \subseteq \mathcal{N}^V \times \mathcal{N}^V$ and $e_{ij}^V = (n_i^V, n_j^V)$. For each virtual node or link $o \in \mathcal{N}^V \cup \mathcal{E}^V$, the set $\mathcal{A}^V := \{a_1^V, \ldots, a_l^V\}$ specifies its attributes. Again, $\forall n \in \mathcal{N}^V : a_{\text{CPU}}^V \in \mathcal{A}(n)$, $\text{CPU}(n)$ gives the demanded CPU of a virtual node. The demanded link bandwidth is given by $\text{bw}(e)$, i.e., $\forall e \in \mathcal{E}^V : a_{\text{bw}}^V \in \mathcal{A}(e)$. Each VNR has a given lifetime following a pre-defined distribution. Further node and link attributes being necessary for the network state model proposed in this paper will be introduced in Sec. V.

C. Virtual Network Embedding and Performance Metrics

Online VNE algorithms map a VNR G^V to a substrate network G^S. As VNRs arrive and leave over time, we will use the subscript t to refer to a VNR and the substrate graph at a certain time.

A mapping of a VNR G_t^V to the substrate G_t^S is only valid, if all virtual nodes $\mathcal{N}^V$ are allocated to a substrate node $\mathcal{N}^S$, and all virtual nodes are connected via their demanded edges $\mathcal{E}^V$. All virtual node and link demands in G_t^V need to be satisfied. For this, a node mapping $f_\mathcal{N}$ and a link mapping $f_\mathcal{E}$ is defined [4]:

$$f_\mathcal{N} : (\mathcal{N}^V, a_{CPU}^V) \rightarrow (\mathcal{N}^S, a_{CPU}^{alloc}) \tag{1}$$

$$f_\mathcal{E} : (\mathcal{E}^V, a_{bw}^V) \rightarrow (\mathcal{P}^S, a_{bw}^{alloc}) \tag{2}$$

where a_{CPU}^{alloc} and a_{bw}^{alloc} are the allocated CPU and bw.

In order to get all virtual nodes that are mapped to a substrate node and all virtual edges that are mapped to a substrate link, we define the inverse mapping functions providing the set of mapped virtual nodes or virtual edges

$$f_\mathcal{N}^{-1}(n_i^S) := \{n_j^V \mid f_\mathcal{N} : n_j^V \rightarrow n_i^S\} \tag{3}$$

$$f_\mathcal{E}^{-1}(e_{ij}^S) := \{e_{km}^V \mid f_\mathcal{E} : e_{km}^V \rightarrow e_{ij}^S\} \tag{4}$$

We define the residual CPU capacity $\triangle CPU(n), \forall n \in \mathcal{N}^S$ and the residual bandwidth capacity $\triangle bw(e), \forall e \in \mathcal{E}^S$ as:

$$\triangle CPU(n_i^S) := \text{CPU}(n_i^S) - \sum_{n_j^V \in f_\mathcal{N}^{-1}(n_i^S)} \text{CPU}(n_j^V) \tag{5}$$

$$\triangle bw(e_{ij}^S) := \text{bw}(e_{ij}^S) - \sum_{e_{km}^V \in f_\mathcal{E}^{-1}(e_{ij}^S)} \text{bw}(e_{km}^V) \tag{6}$$

The following four metrics are used to quantify the quality of the solution for the online VNE problem.

1) Acceptance Ratio: The acceptance ratio AR for a given time interval $\mathcal{T} := [t^{\text{start}}, t^{\text{end}}]$ is defined as

$$AR(\mathcal{T}) := \frac{\mid \mathcal{R}^{\text{acc}}(\mathcal{T}) \mid}{\mid \mathcal{R}^{\text{rej}}(\mathcal{T}) \cup \mathcal{R}^{\text{acc}}(\mathcal{T}) \mid} \tag{7}$$

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 $\mathcal{T}$

2) Revenue: The revenue $R(G^V)$ of a VNR is given by

$$R(\text{G}^V) := \sum_{n_i^V \in \mathcal{N}^V} \text{CPU}(n_i^V) + \sum_{e_{km}^V \in \mathcal{E}^V} \text{bw}(e_{km}^V). \tag{8}$$

3) Cost: The cost $C(G^V)$ of a VNR is defined as:

$$C(\text{G}^V) := \sum_{n_i^V \in \mathcal{N}^V} \text{CPU}(n_i^V) + \sum_{e_{km}^V \in \mathcal{E}^V} \mid f_\mathcal{E}(e_{km}^V) \mid \cdot bw(e_{km}^V), \tag{9}$$

where $\mid f_\mathcal{E} \mid$ provides the length of the physical path on which a virtual edge has been mapped.

4) Revenue-Cost-Ratio: The revenue-cost-ratio $RCR(\text{G}^V)$ of a VNR is defined as the fraction of its revenue $R(\text{G}^V)$ over its cost $C(\text{G}^V)$

$$RCR(\text{G}^V) := \frac{R(\text{G}^V)}{C(\text{G}^V)}. \tag{10}$$

V. Network Representations and RNN-based Admission Control

In order to achieve an efficient admission control, compact representations of substrate networks and virtual network requests (VNRs) are needed. The representations sizes should be independent of the number of nodes and edges. Thus,

TABLE I
GRAPH FEATURES

Graph Feature	Computational Complexity
Average degree	$O(n+m)$
Standard Deviation degree	$O(n+m)$
Average clustering coefficient	$O(\frac{m^2}{n})$
Standard deviation clustering coefficient	$O(\frac{m^2}{n})$
Average effective eccentricity	$O(2n^2+nm)$
Standard deviation effective eccentricity	$O(2n^2+nm)$
Maximum effective eccentricity	$O(2n^2+nm)$
Minimum effective eccentricity	$O(2n^2+nm)$
Average Path length	$O(2n^2+nm)$
Standard Deviation Path Length	$O(2n^2+nm)$
Percentage of central points	$O(2n^2+nm)$
Percentage of endpoints	$O(n+m)$
Number of nodes	$O(n+m)$
Number of edges	$O(n+m)$
Spectral radius	$O(n^3)$
Second largest eigenvalue	$O(n^3)$
Energy	$O(n^3)$
Number of eigenvalues	$O(n^3)$
Label Entropy	$O(n)$
Neighborhood Impurity	$O(nd_{max})$
Link Impurity	$O(n+m)$

representations that simply consider a feature per node and edge cannot be generalized and are not efficient.

As graphs are used in many disciplines, different methods have been developed to use graphs together with ML concepts. Examples of such applications are graph classification or community detection. Graph kernels are one method for graph classification. For graph kernels, a function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is defined, where $\mathcal{G}$ is the set of all possible graphs for which a mapping $\varrho : \mathcal{G} \to \mathcal{H}$ into a Hilbert Space $\mathcal{H}$ exist, s.t. $k(G, G') = \langle \varrho(G), \varrho(G') \rangle, \quad \forall G, G' \in \mathcal{G}$ [31]. Depending on the kernel, the complexity is at best $O(n^3)$ [32], with n being the number of nodes. Besides, the computation of a complete kernel is even at least as complex as the graph isomorphism problem, which is an NP-hard problem [31].

Another approach is the calculation of node, edge, and graph features. The goal is to obtain a fixed length real valued feature vector $\phi : \mathcal{G} \to \mathbb{R}^n$. When used for graph classification, the approach potentially outperforms graph kernels [32]. Table I summarizes all graph features and their complexity used in this work. Although spectral properties have also a complexity of $O(n^3)$ like graph kernels, efficient algorithms such as power iteration exist that exploit special properties of the adjacency matrix to calculate eigenvalues. Generally, algorithms exist that efficiently solve eigenvalue problems in less than $O(n^3)$ [33]. As our goals are the reduction of computational overhead and a speedup in the embedding process, we use graph features (see Table I) as presented in [32], [34] for our network representations. In this way, our feature vectors are independent of the number of nodes and edges of substrate networks and VNRs.

A. Substrate Network Representation

Features for the substrate graph G^S must allow the distinction of states in which a request can be embedded. Accordingly, this demands in particular a representation that captures the changing meta-distribution $\mathcal{A}^S$, i.e., the changing amount of network resources of nodes and edges. To obtain a representation that captures the changes of the substrate network over time, we consider the graph $G^{S\prime} := (\mathcal{N}^{S\prime}, \mathcal{E}^{S\prime})$ induced by the embedded requests. Formally, $\forall o \in \mathcal{N}^S \cup \mathcal{E}^S : a_{cm} \in \mathcal{A}^S(o)$ with $a_{cm} \in \mathbb{N}$ indicating the number of mapped virtual nodes/edges respectively. Then $\mathcal{N}^{S\prime} := \{n^S \in \mathcal{N}^S \mid 0 <| f_{\mathcal{N}}^{-1}(n^S) |\}$ and $\mathcal{E}^{S\prime} := \{e^S \in \mathcal{E}^S \mid 0 <| f_{\mathcal{E}}^{-1}(e^S) |\}$. As $f_{\mathcal{E}}^{-1}$ and $f_{\mathcal{N}}^{-1}$ change over time, we obtain a network with a changing topology, for which we always recalculate all graph features (see Table I) whenever the substrate network changes.

We additionally use eight resource features, which we refer to as *VNE-Features (VF)* in the remainder. For resource features, we additionally calculate three CPU features (*free CPU*, *occupied CPU*, and *total CPU*), three bandwidth features (*free bw*, *occupied bw*, and *total bw*) as well as two embedding features (*total number of currently embedded edges* and *total number of currently embedded nodes*).

We define the mapping $\phi_S(G) : \mathcal{G} \to \mathbb{R}^n$ with $n = 29$ and the resulting feature vector $\mathbf{x}^S := (x_1^{GF}, \ldots, x_i^{GF}, x_{i+1}^{VF}, \ldots, x_n^{VF})$. For proof of concept of our admission control, we use all 21 graph features $x_1^{GF}, \ldots, x_{21}^{GF}$ and the eight VF features $x_{22}^{VF}, \ldots, x_{29}^{VF}$. In this way, the representation allows to distinguish different states of the substrate and to capture the current resource distribution.

B. Virtual Network Request (VNR) Representation

Different models exist specifying the distribution of edges in graphs [35]. The distribution of edges has an impact on the embedding. For example [36] decomposes requests into stars leveraging their topology to simplify the embedding problem. Therefore the features extracted for graphs of requests must allow a distinction between VNRs based on different models, i.e., representations of similar graphs should be similar. For graphs G^{V_1}, G^{V_2} and G^{V_3} where $\mathcal{E}^{V_1}, \mathcal{E}^{V_2} \sim B(n, p_1)$ and $\mathcal{E}^{V_3} \sim B(n, p_2)$ with p_1 sufficiently different from p_2 we thus require that $d\left(\phi\left(G^{V_1}\right), \phi\left(G^{V_2}\right)\right) \leq d\left(\phi\left(G^{V_1}\right), \phi\left(G^{V_3}\right)\right) \simeq d\left(\phi\left(G^{V_2}\right), \phi\left(G^{V_3}\right)\right)$ for some distance function d defined on $\mathbb{R}^n$.

Consider two VNRs G^{V_1} and G^{V_2} with the same total number of nodes and the same amount of requested resources, but different connectivity probabilities p, e.g., $p_1 < p_2$. The substrate G_t^S is highly utilized but has enough spare resources to accommodate one of the requests. We assume that the used VNE algorithm has load balancing as objective. The spare resources can then be expected to be distributed evenly on the substrate graph. G^{V_2} is then more likely to be accepted, as each edge is expected to have less bandwidth demand as G^{V_1}. I.e., it is harder to find physical edges that are able to accommodate the edges of G^{V_1} with higher bandwidth requests.

To obtain a compact representation, we performed a Principal Component Analysis (PCA) on a set of feature vectors $\{\theta(G^{V_1}), \ldots, \theta(G^{V_m})\}$ consisting of all graph features. The features are extracted from a set of graphs $\{G^{V_1}, \ldots, G^{V_m}\}$ generated according to different models. We select features with high load factors, i.e., high coefficients in the linear

TABLE II
GRID SEARCH PARAMETERS. CHOSEN VALUES ARE PRINTED IN BOLD.

Parameter	Values
Optimizer	adam [37], RMSprop
Size of hidden layer	100, **200**, 300
RMSprop step-rate	0.1, 0.01, **0.001**
RMSprop momentum	0.3, 0.6, **0.7**, 0.8, 0.9
RMSprop decay	0.3, 0.6, 0.7, 0.8, **0.9**

transformation of the input data obtained via the PCA. The selected features are *number of nodes*, *spectral radius*, *maximum effective eccentricity*, *average neighbor degree*, *number of eigenvalues*, *average path length* and *number of edges*. They allow the recovery of the models using Multinomial Linear Regression with an accuracy of $99.0\,\%$ and thus satisfy the requirements on the representation stated above. The VNR representation is then given by the mapping $\phi_V : \mathcal{G} \to \mathbb{R}^n$ with $n = 7$ and $\phi_V(G^V) = \mathbf{x}^V = (x_1, \ldots, x_7)$ where x_i corresponds to one of the features listed above. A deeper analysis of the feature selection is planned for future work.

C. Recurrent Neural Network (RNN)-based Admission Control

The input of the RNN is the vector $\mathbf{x} := (\mathbf{x}^S, \mathbf{x}^V) = (\phi_S(G^S(t)), \phi_V(G^V))$, which is a combination of $\mathbf{x}^S$ and $\mathbf{x}^V$ for the substrate and the request as defined in the previous section. The RNN uses this vector to decide whether the VNR will be accepted by the VNE algorithm. Formally, the RNN learns the following mapping: $c : \mathbb{R}^n \to z$ with $z \in \{0, 1\}$ where $n = 36$ (as the dimension of $\phi_S(G)$ is 29 and 7 for $\phi_V(G)$). For z, 0 represents *reject* and 1 *accept*.

On each VNR arrival, the RNN predicts $y \in (0, 1)$ allowing a direct probabilistic interpretation: $P(z = 1) = y$ and $P(z = 0) = 1 - y$. This reflects how confident the RNN is for each class. We take the class with the maximum probability, which amounts to label a request as accepted if $y > 0.5$ and else as rejected. The respective request is then either passed on to the VNE algorithm or discarded.

The RNN has one hidden layer with 200 hidden units and is trained using the RMSprop [38] optimizer implemented in the publicly available *climin* [39] library. The parameters of RMSprop are step-rate of 0.001, momentum of 0.7 and decay of 0.9. The output transfer function is the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$, the hidden transfer function is the hyperbolic tangent function $\tanh(x) = (e^{2x} - 1)(e^{2x} + 1)^{-1}$, the loss optimized on is the Bernoulli cross-entropy loss $L(c \mid y) = -z \log(y) - (1 - z) \log(1 - y)$, where $z \in \{0, 1\}$ is the class label and y the prediction. The RNN is implemented in the publicly available *breze* library [40]. We used the class *SupervisedRNN* from this library. We selected the values for all parameters using grid search over the different sizes for the hidden layer, different optimizers and different optimizer settings as given in Table II. The chosen parameters, printed in bold letters in Table II, achieved the highest accuracy on the validation set.

VI. SIMULATION SETUP AND RNN SETUP

A. Simulation Setup

TABLE III
SIMULATION PARAMETERS

Parameter	Values
Embedding Algorithm	SDP, LB
CPU weight α and link weight β	0.5
Algorithm time constraint T	30 s,60 s,90 s
Substrate graph type	Erdos-Renyi
Number of substrate nodes $\lvert\mathcal{N}^S\rvert$	50, 75, 100
Substrate node capacity (CPU)	Uniformly distr. $\sim U(50, 100)$
Substrate edge connectivity prob.	0.1
Substrate edge capacity (bw)	Uniformly distr. $\sim U(50, 100)$
Virtual Network (VN) graph types	Erdos-Renyi
Number of VN nodes $\mathcal{N}^V$	Uniformly distributed $\sim U(5, 14)$
VN node capacity demand (CPU)	$\sim U(0, 50)$
VN edge connectivity prob.	0.5
VN edge capacity (bw)	Uniformly distributed $\sim U(0, 50)$
VN lifetime	Exponentially distr. with mean 1 000
VN arrival rate λ	Exp. distr. with $\lambda = \frac{1}{100}, \frac{3}{100}, \frac{5}{100}$
Gurobi CPU Cores	1
Gurobi Version	Version 6.5.1

1) Virtual Network Embedding Algorithm Objectives: In this paper, we evaluate two existing VNE algorithms [5], namely *Shortest Distance Path (SDP)* and *Load Balancing (LB)*, briefly described in the following. Due to lack of space, we omit a detailed listing of constraints and binary variables and refer the reader to the original work [5].

a) Load Balancing (LB): The target of the load balancing (LB) algorithm is to minimize the maximum load utilization per physical resource. The cost function is

$$\min_{f_\mathcal{N}, f_\mathcal{E}} \{\alpha \cdot \mathrm{CPU}^{\max}(n_i^S) + \beta \cdot \mathrm{bw}^{\max}(e_{ij}^S)\} \qquad (11)$$

where $\mathrm{CPU}^{\max}(\mathcal{N}^S)$ is the maximum CPU utilization over all substrate nodes and $\mathrm{bw}^{\max}(\mathcal{E}^S)$ is the maximum utilization over all substrate edges. For instance, for a node n_1^S with $\mathrm{CPU}(n_1^S) = 100$ and a virtual node n_1^V with $\mathrm{CPU}(n_1^V) = 50$, the resulting utilization would be $\frac{\mathrm{CPU}(n_1^V)}{\mathrm{CPU}(n_1^S)} = 0.5$ or $50\,\%$. Two weighting parameters α and β are used to trade off the load cost per substrate node and substrate link.

b) Shortest Distance Path (SDP): *SDP* uses as few physical edges for a valid VNE embedding as possible. The algorithm always chooses substrate nodes and substrate links with the highest available capacity. According to [5], this algorithm targets scenarios with scarce substrate resources. The cost function is

$$\min_{f_\mathcal{N}, f_\mathcal{E}} \{\alpha \cdot \frac{1}{\triangle CPU(f_\mathcal{N}(n_i^V))} + \beta \cdot \sum_{e_{ij}^S \in f_\mathcal{E}(e_{km}^V)} \frac{1}{\triangle bw(e_{ij}^S)}\} \qquad (12)$$

where α and β are used to trade off the CPU and bandwidth cost. Over all possible assignments, the cost function choses the node and link mapping that provides the minimal value.

2) Environment: For our online VNE setup, virtual network requests (VNRs) arrive over time and need to be embedded on a capacity-constrained substrate network. All parameters are additionally shown in Table III. The topologies of substrate networks and of VNRs are generated based on an Erdos-Renyi model (Random Graph). We generate substrate topologies with $50, 75$ and 100 number of nodes and an edge connection

probability of 0.1. Both substrate node capacities and edge capacities, which are unitless, are uniformly distributed (uni. distr.) between 50 and 100, i.e., $U(50, 100)$. For VNRs, the edge probability is 0.5, the number of nodes is uniformly distributed ($U(5, 14)$), and node and edge capacity are also uniformly distributed ($U(50, 100)$). The VNR requests arrive with an exponentially distributed (exp. dstr.) arrival rate $\lambda = \frac{1}{100}, \frac{3}{100}, \frac{5}{100}$ and stay for an exp. dstr. lifetime with mean 1 000. All chosen parameter settings in this paper are in accordance with existing VNE research [41], [5], [42].

For each setup, i.e., for each combination of embedding algorithm, algorithm time constraint T, number of substrate nodes $|\mathcal{N}^S|$, and arrival rate λ, we perform simulation runs with and without RNN until we have processed 5 000 requests. We apply the batch-means approach to calculate the mean values of all metrics used in the results per setup. We form batches of 100 VNRs and calculate the confidence intervals of the mean values of all batches per setup. For runtime, all VNRs are considered per batch, while for revenue, cost, and revenue-cost-ratio, only the accepted VNRs are considered. The solver for the MIP problems is Gurobi version 6.51 [43]. A gurobi instance is restricted to one core to produce reliable runtime results.

B. Recurrent Neural Network (RNN) Setup

The RNN used in this paper is trained offline, i.e., we apply supervised learning. Accordingly, we describe the generation of samples that are used for learning and the training procedure in the following.

1) Sample Generation: To acquire the necessary training data, we perform 10 simulations for the setups as described in Table III and outlined in the previous section without the admission control, i.e., the RNN. Each run contains 2 500 requests. For each arriving VNR G_t^V, we store the feature vector $\mathbf{x}_t$ together with the MIP solution (accepted, no solution, infeasible) and the time needed to create the model and the time needed to solve the MIP problem.

2) Training of RNN: We train one RNN for each combination of number of substrate nodes, algorithm type, and T. Note that one RNN is always trained for all λ values to prove it can be used for different system utilizations. Thus, we train 18 RNNs in total. Note, we plan to analyze RNNs for larger parameter combinations for future work. We use a corpus of three setups combining all λ values to train one RNN model. As a result, the corpus consists of $3 \cdot 10 \cdot 2\,500 = 75\,000$ samples. In matrix notation we have $\mathbf{X} \in \mathbb{R}^{75\,000 \times 36}$ where each row in $\mathbf{X}$ is one feature vector x_t.

60 % of the samples are used as training set, 20 % as validation set and 20 % as test set. Every feature vector in training, validation and test set is standardized using $\mathbf{x}\prime = \frac{\mathbf{x} - \mu_{\text{train}}}{\sigma_{\text{train}}}$ with the mean μ_{train} and standard deviation σ_{train} calculated from the training set. Each RNN is then trained for at least 30 000 iterations with patience of 5 000 iterations. Every 50 iterations, the performance of the RNN is evaluated on the validation set. If a new best loss has been achieved, the number of iterations is extended about 5 000 iterations. Training terminates if no new best validation loss is found when the maximum number of iterations is reached. Using this technique, we ensure that the RNN will find a good optimum but also keep the training time low. We take the parameters for which the best validation loss has been achieved and evaluate the model on the test set.

TABLE IV
MODEL PERFORMANCE. ER50-SDP30 MEANS AN RNN TRAINED FOR ERDOS-RENYI-BASED SUBSTRATE NETWORK WITH 50 NODES, SDP ALGORITHM, AND $T = 30\,s$.

RNN Model	Accuracy [%]	TPR [%]	TNR [%]	MV [%]
ER50-SDP30	92.70	86.57	95.35	69.86
ER50-SDP60	92.60	87.88	94.98	66.53
ER50-SDP90	92.91	89.71	94.59	65.61
ER75-SDP30	90.57	88.68	92.39	50.89
ER75-SDP60	90.52	91.88	89.16	50.03
ER75-SDP90	89.64	89.85	89.44	50.23
ER100-SDP30	90.61	92.36	87.85	61.26
ER100-SDP60	90.30	92.97	85.87	62.45
ER100-SDP90	90.15	91.82	87.31	62.91
ER50-LB30	97.77	92.30	98.78	84.38
ER50-LB60	98.02	92.88	99.01	83.93
ER50-LB90	97.68	92.33	98.68	84.17
ER75-LB30	96.24	88.98	98.29	77.97
ER75-LB60	96.34	90.10	98.03	78.62
ER75-LB90	96.22	89.10	98.17	78.46
ER100-LB30	94.21	87.52	96.70	72.87
ER100-LB60	93.48	86.39	96.26	71.79
ER100-LB90	92.86	87.61	94.89	72.01

VII. RESULTS

We first inform on the performance of our RNN. Second, we elaborate on the VNE results with respect to the metrics as introduced in Sec. IV. Further, we show the results of the runtime of the online VNE process with and without RNN.

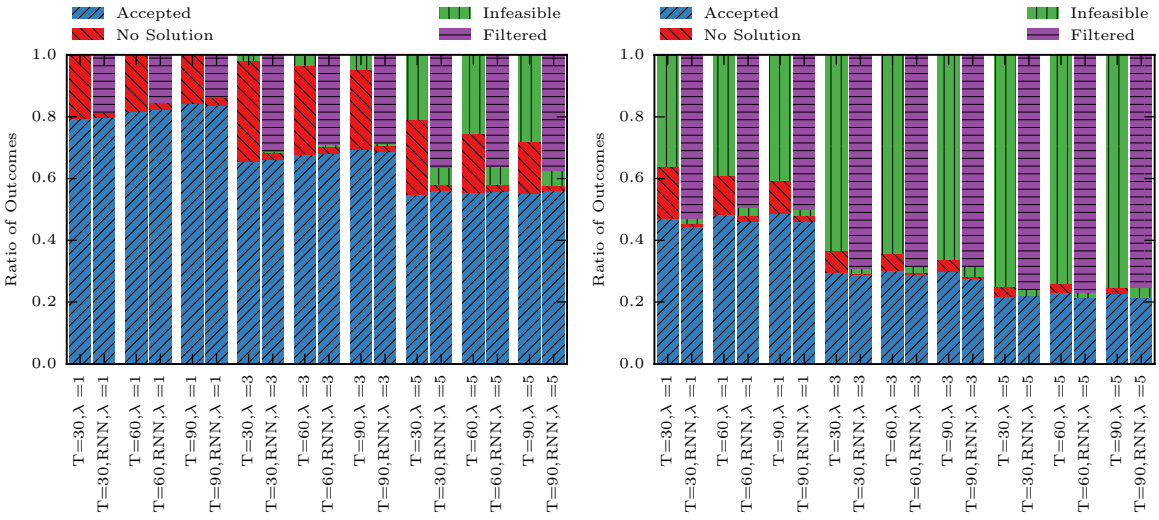
A. RNN Performance Results

Table IV gives an overview of the classification performance of the 18 RNNs. The accuracy gives ratio of correctly classified samples. The True Positive Rate (TPR) gives proportion of correctly identified positive samples (accepted) and the True Negative Rate (TNR) the proportion of correctly identified negative outcomes (infeasible, no solution). All models achieve accuracies between 89 % and 98 % and are better than the respective majority vote. The accuracies for LB are in general higher than SDP. As can be seen from the majority vote column, the datasets are often very skewed. The high values for both TPR and TNR show how well the models learn the mapping c. As the results are similar for different network sizes $(50, 75, 100)$, we will focus on substrate networks with sizes of 100 in the next section.

Besides, note that we used $y > 0.5$ as threshold for our RNN, however, other thresholds than 0.5 can also be chosen to reflect different application scenarios. For example, in case of abundant computational resources, one might want to use a smaller value than 0.5, as this will lead to a higher TPR on the cost of a lower TNR.

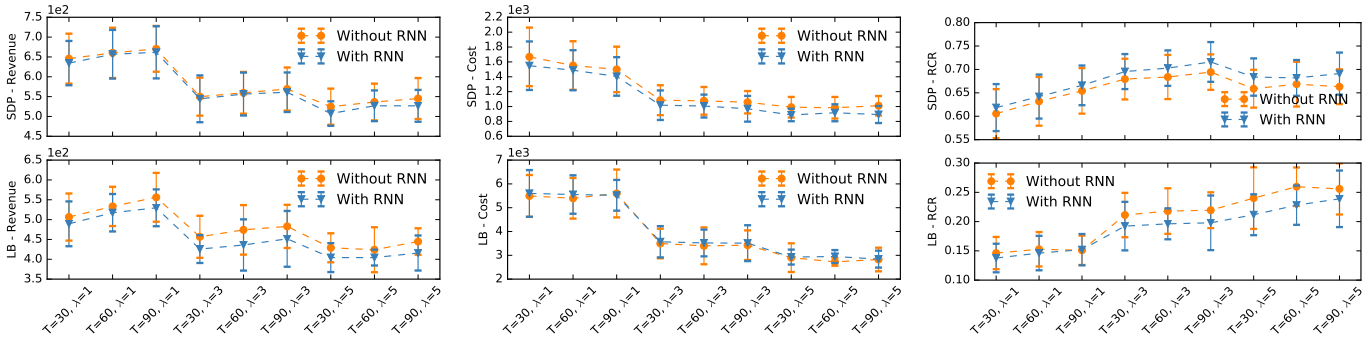
B. VNE Performance Results

Figs. 2a-b show the filtering behavior of SDP and LB with and without RNN. The four colors represent the ratios



(a) SDP VNE Algorithm (b) LB VNE Algorithm

Fig. 2. Comparison between both VNE algorithms with and without RNN. The four colors represent the ratios of accepted (blue), no solution (red), infeasible (green), and filtered (purple) ratios of VNRs by the system, i.e., by the admission control and the VNE algorithm. Substrate network size: 100.



(a) Revenue R (b) Cost C (c) Revenue-cost-ratio RCR

Fig. 3. VNE metrics for substrate network size 100 and both VNE algorithms (SDP,LB). The graphs show mean values and their 95 % confidence intervals over at least 10 batches. The run without RNN is indicated via orange circles and orange dashed lines. The run with RNN is indicated via blue triangles and blue dashed lines. The dashed lines illustrate trend over different simulation settings. Note the scientific scaling of y-axis.

of accepted (blue), no solution (red), infeasible (green), and filtered (purple) ratios of VNRs by the system, i.e., by the admission control and the VNE algorithm. The results are shown for all $T = 30, 60, 90$ and $\lambda = 1, 3, 5$ with and without RNN. In general, the acceptance ratio (AR) of SDP is higher than LB. The reason is that LB allocates more VNR resources to bottleneck links, which leads to a higher blocking. In contrast to LB, SDP always tries to prevent allocating resources to bottleneck links. For increasing λ, the RNN still achieves a high filtering ratio, i.e., it filters efficiently infeasible or no solution VNRs. This is independent of the ratio of infeasible to no solution VNRs, as the RNN recognizes and filters them reliably for SDP and LB. In all cases, the RNN slightly affects the overall acceptance ratio compared to the case without RNN, i.e., it accepts 7.01 % less (LB, $T = 60, \lambda = 5$) or 2.37 % more VNRs (SDP, $T = 60, \lambda = 5$), which is due to TNR and TPR.

Fig. 3a-c show mean values and confidence intervals for Revenue (R), Cost (C), and Revenue-cost-ratio (RCR). Similar to AR, it can be observed that R and C are decreasing for increasing λ. The reason is that with increasing λ, more smaller VNRs are embedded, as the available resources are generally scarce. For smaller networks, C decreases faster than the revenue, which leads to an increasing RCR. In case of SDP, the RNN rejects more larger networks (in terms of number of nodes), which provide a lower RCR, thus the mean values are slightly better. The mean values for LB, however, are slightly worse with RNN. The reason is that generally less networks are accepted, which are embedded with higher costs due to LB's objective to balance node and link utilizations. In general, no statistical significant difference between all metrics with and without RNN can be observed in our setups.

Fig. 4 shows the results of the mean runtime of batches of 100 VNRs. For both algorithms, the RNN can lead to a statistical significant runtime gain. However, for SDP, the relative runtime gain is smaller than for LB. The reason is that

TABLE V
COMPARISON OF EMBEDDING METRICS. POSITIVE VALUES MEAN INCREASE, NEGATIVE VALUES MEAN DECREASE DUE TO USING RNN.

Algorithm	T [s]	λ	Δ Acceptance Ratio [%]	Δ Runtime [%]	Δ Revenue [%]	Δ Cost [%]	Δ Revenue-Cost-Ratio [%]
SDP	30	1	-1.28	-30.40	-1.75	-7.12	2.13
SDP	30	3	-0.38	-54.29	-0.98	-6.18	2.37
SDP	30	5	0.65	-53.39	-3.30	-10.46	3.78
SDP	60	1	-0.40	-29.33	-0.52	-4.10	1.62
SDP	60	3	1.06	-56.19	-0.56	-6.48	2.78
SDP	60	5	2.37	-47.49	-1.77	-6.89	2.00
SDP	90	1	-1.32	-20.34	-1.31	-6.31	1.80
SDP	90	3	0.67	-38.88	-1.42	-8.32	3.12
SDP	90	5	1.07	-50.14	-3.37	-11.70	4.22
LB	30	1	-5.11	-78.27	-3.37	1.95	-5.80
LB	30	3	-6.34	-84.00	-6.78	2.08	-8.97
LB	30	5	0.00	-85.94	-5.77	0.86	-11.83
LB	60	1	-5.12	-69.04	-2.99	2.94	-4.30
LB	60	3	-5.87	-85.09	-8.03	3.57	-10.00
LB	60	5	-7.01	-91.50	-4.66	8.26	-12.14
LB	90	1	-5.71	-70.18	-4.76	-1.36	0.71
LB	90	3	-6.52	-81.38	-6.49	2.51	-9.82
LB	90	5	-6.32	-85.82	-6.51	0.57	-6.63

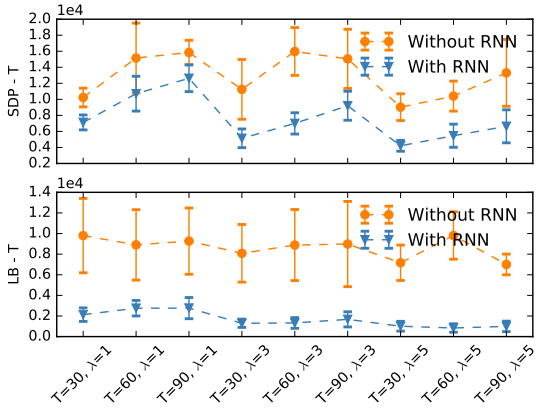


Fig. 4. Mean runtime comparison. Batches contain 100 VNRs.

SDP generally accepts more VNE. Thus, more VNRs need to be processed by the VNE algorithm. For LB, the RNN always reduces the runtime by more than 69 %. As the amount of infeasible solutions is generally high for LB, the RNN saves efficiently the processing time of the VNE algorithm.

Table V provides an overview over the relative performance improvements. The performance values are computed by dividing the performance over all runs with RNN through the performance without RNN. While parameters such as λ affect the absolute metrics, Table V shows that the relative performance differences for the VNE metrics are generally small. Note again, however, that the RNN has a more positive effect on SDP (RCR improvement up to 4.22 %) than on LB (RCR decreases by 12.14 %). We conclude that our admission control saves significantly computational time while it does not significantly diminish the VNE performance in particular for a VNE algorithm that shows a high acceptance ratio.

VIII. CONCLUSIONS

Rapidly and efficiently solving the online virtual network embedding (VNE) problem is inevitable in particular for next generation networks. In this paper, we proposed an admission control for the online VNE problem. The admission control uses a Recurrent Neural Network (RNN) that classifies virtual network requests based on newly developed network representations. Via simulations, we demonstrate that the RNN predicts whether a request will be accepted by the VNE algorithm or rejected, as the request is either infeasible or needs too long for being efficiently processed. Our admission control reduces the overall system runtime, thus improves calculation efficiency for the online VNE problem. We demonstrate the ability to learn from the history of algorithms, i.e., how they performed over time, and how to use this knowledge for admission control of future problem instances.

For future work, we would like to investigate the feature selection in more detail. Further, we want to extend our analysis to a wider range of VNE algorithms and different embedding objectives, e.g., to accept only networks providing a high revenue-cost-ratio. An online learning procedure is also in our future research scope. Further, we want to focus on systems that need to adapt, e.g., to changing networking environments or dynamically changing virtual network demands. Besides, we would like to investigate how our network representations can be beneficial in other research areas, e.g., for Software-defined Networking (SDN) network hypervisor placements [44], [45] or NFV function placements [3], where virtual networks are an inevitable component.

ACKNOWLEDGMENT

This work has been performed in part in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1) and is partly 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). The authors alone are responsible for the content of the paper. Finally, we would like to thank all reviewers for their fruitful comments.

REFERENCES

[1] N. Nikaein, E. Schiller, R. Favraud, K. Katsalis, D. Stavropoulos, I. Alyafawi, Z. Zhao, T. Braun, and T. Korakis, "Network Store," in *Proc. ACM MobiArch*, Paris, France, 2015, pp. 8–13.

[2] N. Omnes, M. Bouillon, G. Fromentoux, and O. Grand, "A programmable and virtualized network & IT infrastructure for the internet of things: How can NFV & SDN help for facing the upcoming challenges," in *Proc. International Conference on Intelligence in Next Generation Networks*. IEEE, 2015, pp. 64–69.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. c, pp. 1–1, jan 2015.

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

[5] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha, "Optimal Virtual Network Embedding: Node-Link Formulation," *IEEE Transactions on Network and Service Management*, vol. 10, no. 4, pp. 356–368, dec 2013.

[6] A. Blenk and W. Kellerer, "Traffic pattern based virtual network embedding," in *Proc. of ACM CoNEXT Student Workshop*, 2013, pp. 23–26.

[7] M. Rost, S. Schmid, and A. Feldmann, "It ' s About Time : On Optimal Virtual Network Embeddings under Temporal Flexibilities," in *Proc. IEEE IPDPS*, 2014, pp. 1–10.

[8] F. Hutter, H. H. Hoos, and K. Leyton-brown, "Automated Configuration of Mixed Integer Programming Solvers," Tech. Rep.

[9] E. B. Khalil, P. L. Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to Branch in Mixed Integer Programming," in *Proc. AAAI*, 2016.

[10] F. Hutter, H. H. Hoos, and K. Leyton-Brown, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. Automated Configuration of Mixed Integer Programming Solvers, pp. 186–202.

[11] M. Feng, J. Liao, J. Wang, S. Qing, and Q. Qi, "Topology-aware Virtual Network Embedding based on multiple characteristics," in *Proc. IEEE ICC*, no. 61372120, jun 2014, pp. 2956–2962.

[12] J. Wang and J. Liao, "Topology-aware virtual network embedding through bayesian network analysis," in *Proc. IEEE Globecom*. IEEE, dec 2012, pp. 2621–2627.

[13] M. Feng, L. Zhang, X. Zhu, J. Wang, Q. Qi, and J. Liao, "Topology-aware virtual network embedding through the degree," in *National Doctoral Academic Forum on Information and Communications Technology*. Institution of Engineering and Technology, 2013, pp. 16–16.

[14] N. F. Butt, N. M. M. K. Chowdhury, and R. Boutaba, "Topology-Awareness and Reoptimization Mechanism for Virtual Network Embedding," in *Proc. IFIP Networking*, 2010, pp. 27–39.

[15] 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.

[16] 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*, no. i. IEEE, mar 2012, pp. 2408–2416.

[17] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine Learning*, vol. 75, no. 1, pp. 3–35, apr 2009.

[18] C. Bilgin, C. Demir, C. Nagi, and B. Yener, "Cell-graph mining for breast tissue modeling and classification," in *Proc. Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2007, pp. 5311–5314.

[19] "cnsm-vne-simulator," 2016. [Online]. Available: https://github.com/tum-lkn/cnsm-vne-simulator

[20] I. Houidi, W. Louati, and D. Zeghlache, "Exact Multi-Objective Virtual Network Embedding in Cloud Environments," *The Computer Journal*, 2015.

[21] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding," *ACM SIGCOMM CCR*, vol. 38, no. 2, p. 17, mar 2008.

[22] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, feb 2012.

[23] G. Even, M. Medina, G. Schaffrath, and S. Schmid, "Competitive and deterministic embeddings of virtual networks," *Theoretical Computer Science*, pp. 1–22, oct 2012.

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

[25] D. Dietrich and P. Papadimitriou, "Policy-compliant virtual network embedding," in *Proc. IFIP Networking Conference*. IEEE, jun 2014, pp. 1–9.

[26] D. Liu, Y. Zhang, and H. Zhang, "A self-learning call admission control scheme for cdma cellular networks," *IEEE Transactions on Neural Networks*, vol. 16, no. 5, pp. 1219–1228, Sept 2005.

[27] H. Perros and K. Elsayed, "Call admission control schemes: a review," *IEEE Communications Magazine*, vol. 34, no. 11, pp. 82–91, 1996.

[28] C. W. Ahn and R. S. Ramakrishna, "Qos provisioning dynamic connection-admission control for multimedia wireless networks using a hopfield neural network," *IEEE Transactions on Vehicular Technology*, vol. 53, no. 1, pp. 106–117, Jan 2004.

[29] A. Hiramatsu, "Integration of atm call admission control and link capacity control by distributed neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, pp. 1131–1138, Sep 1991.

[30] R.-G. Cheng and C.-J. Chang, "Neural-network connection-admission control for ATM networks," *IEE Proceedings - Communications*, vol. 144, no. 2, p. 93, 1997.

[31] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning Theory and Kernel Machines*. Springer, 2003, pp. 129–143.

[32] G. Li, M. Semerci, B. Yener, and M. J. Zaki, "Effective graph classification based on topological and label attributes," *Stat. Anal. Data Min.*, vol. 5, no. 4, pp. 265–283, Aug. 2012.

[33] G. H. Golub and H. A. van der Vorst, "Eigenvalue computation in the 20th century," *Journal of Computational and Applied Mathematics*, vol. 123, no. 1-2, pp. 35–65, nov 2000.

[34] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Netsimile: A scalable approach to size-independent network similarity," *CoRR*, vol. abs/1209.2684, 2012. [Online]. Available: http://arxiv.org/abs/1209.2684

[35] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, pp. 47–97, Jan 2002.

[36] 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.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[38] Tijmen Tieleman and Geoffrey Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude."

[39] Justin Bayer, Christian Osendorfer, Sarah Diot-Girard, Thomas Rueckstiess, and Sebastian Urban, "climin - A pythonic framework for gradient-based function optimization," TUM, Tech. Rep., 2015. [Online]. Available: https://github.com/BRML/climin

[40] Justin Bayer, Christian Osendorfer, Max Karl, Maximilian Soelch, and Sebastian Urban, "breze," TUM. [Online]. Available: https://github.com/breze-no-salt/breze

[41] 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.

[42] M. R. Rahman and R. Boutaba, "SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 105–118, Jun. 2013.

[43] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com

[44] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for Software Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 655–685, Jan. 2016.

[45] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with sdn network hypervisors: The cost of virtualization," *IEEE Transactions on Network and Service Management*, vol. PP, no. 99, pp. 1–1, 2016.