Public Review for

Parametrized Complexity of Virtual Network Embeddings: Dynamic & Linear Programming Approximations

M. Rost, E. Döhne, S. Schmid

Network virtualization requires efficient mapping of virtual networks onto the physical network. Unfortunately, this problem is NP-complete, and solving it approximately also poses serious computational challenges. Existing solutions tend to be heuristics that do not provide formal performance guarantees.

Matthias Rost, Elias Döhne, and Stefan Schmid explore a parameterized-complexity approach and first consider the Valid Mapping Problem (VMP) that asks for a valid minimal-cost mapping of a virtual network onto the physical network. Specifically, the authors consider the virtual-network treewidth as a parameter that captures closeness of the virtual-network graph to a tree. While virtual networks typically have bounded treewidth, the paper develops a DynVMP algorithm that decomposes the virtual-network graph into trees and applies dynamic programming to solve VMP starting from the leaves of the tree decomposition. The computational complexity of the solution belongs to the XP class, and DynVMP runs in polynomial time for virtual networks with bounded treewidth. Then, using a Fully Polynomial-Time Approximation Scheme (FPTAS) for minimum-cost Latency-Constrained Shortest Paths (LCSP), the paper adapts DynVMP to support latency constraints. Based on their successful solution for VMP, the authors tackle the Virtual Network Embedding Problem (VNEP) where the virtual-network mapping should be not only valid but also feasible with respect to capacity constraints. In particular, Rost, Döhne, and Schmid extend their previous column-generation linear-programming approximation for the offline VNEP and generalize the solution for an arbitrary virtual-network graph. Again, the proposed solution runs in polynomial time for virtual networks with bounded treewidth and is adapted to accommodate latency constraints.

Supplementing the theoretical results on achieved runtime and solution-quality guarantees, the paper experimentally evaluates its approach against existing heuristics. First, the evaluation assesses the treewidth of random graphs and shows that their tree decomposition can be done on the order of seconds. Then, the paper examines the runtime and performance of its solution for the offline VNEP with four ViNE-heuristic variants.

Public review written by
Sergey Gorinsky
IMDEA Networks Institute, Spain

Parametrized Complexity of Virtual Network Embeddings: Dynamic & Linear Programming Approximations

Matthias Rost
TU Berlin, Germany
mrost@inet.tu-berlin.de

Elias Döhne
TU Berlin, Germany
edoehne@inet.tu-berlin.de

Stefan Schmid
University of Vienna, Austria
stefan_schmid@univie.ac.at

ABSTRACT

This paper makes the case for a parametrized complexity approach to tackle the fundamental but notoriously hard Virtual Network Embedding Problem. In particular, we show that the structure of the to-be-embedded virtual network requests can be exploited toward fast (i.e., fixed-parameter tractable) approximation algorithms, using dynamic as well as linear programming algorithms.

Our approach does provide formal guarantees on the runtime and solution quality and can safeguard also latency constraints. Using extensive computational experiments we demonstrate the practical relevance of our novel approach.

CCS CONCEPTS

• **Networks → Network resources allocation**; • **Theory of computation → Fixed parameter tractability**;

KEYWORDS

Virtual Network Embedding, Approximation, Fixed-Parameter Tractability, Dynamic Programming, Linear Programming

1 INTRODUCTION

The Virtual Network Embedding Problem (VNEP) captures the essence of many resource allocation problems in networks [5]: Given are a substrate network, representing the physical infrastructure, and a virtual request graph, representing a customer's workload; the task is to map each virtual request node to a physical substrate node and to realize each virtual request edge as a path in the substrate connecting the respective servers while safeguarding, among others, capacity constraints. The VNEP has attracted much interest over the last years and is closely related to other embedding problems, e.g., the embedding of service function chains [8], virtual clusters [1], or virtual datacenters [13]. Indeed, in all of these cases a request topology is to be embedded in the provider's physical *substrate* network. Figure 1 gives an example.

Alas, the VNEP is algorithmically very challenging: it is $\mathcal{NP}$-complete and *inapproximable* under any objective [10]. Even more, the VNEP remains $\mathcal{NP}$-complete in the absence of capacity constraints. Concretely, given restrictions on the mapping of request nodes and edges, the respective *Valid Mapping Problem* (VMP) asking to determine a *valid* mapping respecting only the given restrictions, is $\mathcal{NP}$-complete, even for planar and degree-bounded request graphs. However, the VMP is not only an elementary problem, solving the VMP was recently also shown to be of crucial importance for the development of approximations for the VNEP: to approximate the offline variant of the VNEP using randomized rounding the computation of (convex combinations of) valid mappings using Linear Programming (LP) is necessary [11].

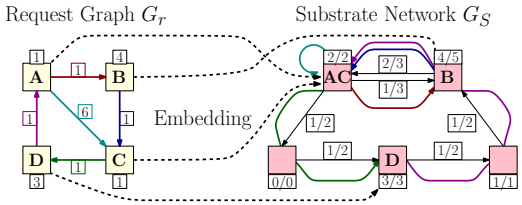

Figure 1: Exemplary embedding of a request on a substrate. The numbers represent demands and allocations/capacities.

Contributions. This paper initiates the study of a *parametrized complexity* approach to solve the fundamental Valid Mapping Problem, which in turn leads to novel solutions for the VNEP. The motivation behind a parametrized complexity approach is that many $\mathcal{NP}$-hard problems become polynomial-time tractable when considering input parameters beyond the input size: the parametrized complexity classes $\mathcal{FPT}$ and $\mathcal{XP}$ contain problems that can be solved in time $O(2^{\text{poly}(k)} \cdot \text{poly}(|X|))$ and $O(|X|^{\text{poly}(k)} + \text{poly}(|X|))$ for a problem instance X with parameter k, respectively [6]. We employ the treewidth of the request graph as parametrization, which measures the closeness of the request graph to a tree [2] and derive a number of new results:

(1) We develop the dynamic programming algorithm DynVMP to solve the VMP, which runs in $\mathcal{XP}$-time $O(n_S^{\text{poly}(k)} \cdot \text{poly}(n_S \cdot n_r))$, where n_S and n_r denote the number of substrate and request nodes, respectively, and k denotes the treewidth of the request graph. Thus, for graphs of bounded treewidth DynVMP runs in polynomial-time.
(2) Based on Linear Programming duality, we show that the DynVMP algorithm can be used as *separation oracle* and derive an efficient *column generation* approach for solving Linear Programming relaxations of the VNEP. Accordingly, the previous (polynomial-time) approximation result of [11] for cactus graphs is generalized to graphs of bounded treewidth, while yielding $\mathcal{XP}$-approximations for all other graph classes.
(3) For the VNEP with per-edge latency constraints, we derive a novel approximation result based on computing approximate latency-observing mappings using the DynVMP algorithm.
(4) To demonstrate the applicability of our approach in practice, we study the treewidth of random graphs, and evaluate randomized rounding heuristics with state-of-the-art heuristics.

Novelty & Related Work. The VNEP has received much attention over the last years and we refer to the survey [5] for an overview. Most existing works consider heuristics which do not provide any formal performance guarantees.

Much less is known about polynomial-time approximation algorithms. Even et al. [4] present an approximation algorithm for linear chain requests. A first more general approximation of the

offline VNEP (cf. Definition 2.5) was recently proven for the class of cactus request graphs [11], i.e. graphs in which cycles intersect in at most a single node. In particular, it was shown that the previously known Multi-Commodity Flow (MCF) LP formulation [3] yields *invalid* mappings. Hence, the integrality gap of the MCF formulation is unbounded, rendering it useless for approximations. Accordingly, a novel LP formulation was proposed which always returns valid mappings, but grows in size compared to the MCF LP.

In this paper, we present a parametrized column generation approach to compute optimal LP solutions for *arbitrary* request graphs based on dynamic programming. Leveraging this parametrized complexity perspective, we not only generalize the previously known approximation to arbitrary graphs, but also obtain approximations for the VNEP under latency constraints. We are not aware of any VNEP approximations respecting latencies.

Structure. Section 2 introduces our formal model, and Section 3 presents the idea of decomposing requests into trees. Our dynamic program is given in Section 4 and our approximations in Section 5. In Section 6 rounding heuristics are discussed. Our evaluation is presented in Section 7. We conclude our work in Section 8.

2 FORMAL MODEL

Within this section the VNEP and the VMP are formally introduced. As latency constraints play an important role in Service Function Chaining [8], we (optionally) incorporate these here.

Substrate Network. We refer to the physical network as substrate network and model it as directed graph $G_S = (V_S, E_S)$. Capacities of the substrate are given via the function $d_S : G_S \to \mathbb{R}_{\geq 0}$. The capacity $d_S(u)$ of node $u \in V_S$ may represent the number of (virtual) CPUs while the capacity $d_S(u,v)$ of edge $(u,v) \in E_S$ represents the available bandwidth. We denote by $\mathcal{P}_S$ the set of all simple paths in G_S. Each substrate element $x \in G_S$ may be attributed with costs $c_S(x) \in \mathbb{R}_{\geq 0}$ for using it (per unit capacity). Substrate edges may be attributed with latencies via $l_S : E_S \to \mathbb{R}_{\geq 0}$.

Request Graphs. A request is similarly modeled as directed graph $G_r = (V_r, E_r)$ together with node and edge demands $d_r : G_r \to \mathbb{R}_{\geq 0}$. Per edge latency bounds are given by $l_r : E_r \to \mathbb{R}_{\geq 0}$, when latencies are considered. Each request $r \in \mathcal{R}$ may be attributed with a profit $b_r \in \mathbb{R}_{\geq 0}$ that the provider obtains when embedding the request, subject to the following restrictions.

Mapping Restrictions. Virtual nodes and edges can only be mapped on substrate nodes and edges of sufficient capacity. Furthermore, the customer or provider may restrict the mapping of request nodes $i \in V_r$ and edges $(i,j) \in E_r$ by providing sets of *forbidden* substrate nodes $\overline{V}_S^i \subseteq V_S$ and edges $\overline{E}_S^{i,j} \subseteq E_S$. The set $\overline{V}_S^i$ may for example include substrate nodes too distant to the customer or servers not suited to host the functionality of request node i. Similarly, the set $\overline{E}_S^{i,j}$ contains edges which must be avoided due to security or other technical policies. Accordingly, we denote the set of suitable substrate nodes for $i \in V_r$ by $V_S^{r,i} = \{u \in V_S \setminus \overline{V}_S^i \mid d_S(u) \geq d_r(i)\}$ while the set of suitable substrate edges for $(i,j) \in E_r$ is denoted by $E_S^{r,i,j} = \{(u,v) \in E_S \setminus \overline{E}_S^{i,j} \mid d_S(u,v) \geq d_r(i,j)\}$. The maximal demand $d_{\max}(r,x)$ of any request element for a single substrate resource $x \in G_S$ is defined as $d_{\max}(r,u) = \max(\{0\} \cup \{d_r(i) \mid i \in V_r : u \in V_S^{r,i}\})$ and $d_{\max}(r,u,v) = \max(\{0\} \cup \{d_r(i,j) \mid (i,j) \in E_r : (u,v) \in E_S^{r,i,j}\})$ for substrate nodes $u \in V_S$ and edges $(u,v) \in E_S$, respectively.

Problem Definitions. According to the above introduction of mapping restrictions, a valid mapping is defined as follows.

Definition 2.1 (Valid Mapping). A *valid* mapping of request $r \in \mathcal{R}$ to the substrate G_S is a tuple $m_r = (m_r^V, m_r^E)$ of functions that map nodes and edges, respectively, s.t. the following holds:

- The function $m_r^V : V_r \to V_S$ maps virtual nodes *validly* to substrate nodes, such that $m_r^V(i) \in V_S^{r,i}$ holds for $i \in V_r$.
- The function $m_r^E : E_r \to \mathcal{P}_S$ maps virtual edges $(i,j) \in E_r$ to *valid* simple paths in G_S connecting $m_r^V(i)$ to $m_r^V(j)$, such that $m_r^E(i,j) \subseteq E_S^{r,i,j}$ holds for $(i,j) \in E_r$.
- When latencies are considered $\sum_{(u,v) \in m_r^E(i,j)} l_S(u,v) \leq l_r(i,j)$ must hold for $(i,j) \in E_r$.

The set of *all* valid mappings of request r is denoted by $\mathcal{M}_r$. □

Hence, a valid mapping enforces the validity of each *single* virtual element with respect to mapping restrictions and resource capacities. Cumulative resource allocations are defined as follows:

Definition 2.2 (Allocations). We denote by $A(m_r, x) \in \mathbb{R}_{\geq 0}$ the resource allocation induced by the valid mapping $m_r = (m_r^V, m_r^E)$ on substrate element $x \in G_S$. For $u \in V_S$ and $(u,v) \in E_S$ the following holds, respectively: $A(m_r, u) = \sum_{i \in V_r : m_r^V(i) = u} d_r(i)$ and $A(m_r, u, v) = \sum_{(i,j) \in E_r : (u,v) \in m_r^E(i,j)} d_r(i,j)$. We denote the maximal allocation that a valid mapping may impose on a substrate resource $x \in G_S$ by $A_{\max}(r,x) = \max_{m_r \in \mathcal{M}_r} A(m_r, x)$. □

A set of mappings is *feasible* if it respects resource capacities:

Definition 2.3 (Feasible Embedding). A set of mappings $\{m_r\}_{r \in \mathcal{R}}$ over a set of requests $\mathcal{R}$ is feasible, if and only if $\sum_{r \in \mathcal{R}} A(m_r, x) \leq d_S(x)$ holds for $x \in G_S$. A single mapping m_r is feasible, if this holds for the singleton set $\{m_r\}$. □

The online and offline VNEP are defined as follows:

Definition 2.4 (Online Virtual Network Embedding Problem). The online VNEP asks for a feasible embedding m_r of a single request r minimizing the cost $c(m_r) = \sum_{x \in G_S} c_S(x) \cdot A(m_r, x)$. □

Definition 2.5 (Offline Virtual Network Embedding Problem). The offline VNEP asks for a feasible embedding $\{m_r\}_{r \in \mathcal{R}'}$ of a subset of requests $\mathcal{R}' \subseteq \mathcal{R}$ maximizing the profit $\sum_{r \in \mathcal{R}'} b_r$. □

As the feasibility of an embedding implies the validity of the respective mappings, the computation of valid mappings is a prerequisite for both VNEP variants. We also introduce the (online) Valid Mapping Problem as follows:

Definition 2.6 (Valid Mapping Problem (VMP)). Given a request r, the VMP asks for finding the valid mapping m_r minimizing the cost function $c(m_r) = \sum_{x \in G_S} c_S(x) \cdot A(m_r, x)$. □

We note that when request demands are small compared to the substrate capacities, the online VNEP reduces to the VMP:

Observation 2.7. *Given a request for which any valid mapping $m_r \in \mathcal{M}_r$ is* feasible*, i.e. $A_{\max}(r,x) \leq d_S(x)$ holds for all substrate resources $x \in G_S$, then the online VNEP reduces to the VMP: an optimal solution to the VMP is an optimal solution to the VNEP.*

3 REQUEST GRAPH TREE DECOMPOSITIONS

In the following, we revisit the notion of tree decompositions [2, 6] and apply it to request graphs. Tree decompositions are used to represent arbitrary graphs as trees (cf. Figure 2). The definition of tree decompositions ensures that (i) all nodes and edges of the request graph are *covered* while (ii) preserving crucial *structural information* of the original graph. Combinatorial optimization problems as the VMP can then be solved on this tree representation by using dynamic programming [6]. As tree decompositions are defined for undirected graphs, we consider undirected request graphs:

Definition 3.1 (Undirected Request Graph $\overline{G}_r$). For a request graph $G_r = (V_r, E_r)$ its undirected interpretation $\overline{G}_r = (V_r, \overline{E}_r)$ is given by $\overline{E}_r = \{\{i,j\} | (i,j) \in E_r\}$ on the original node set V_r. □

Note that directed, antiparallel edges $(i,j), (j,i) \in E_r$ of the original request graph are accordingly represented using only a single undirected edge $\{i,j\} \in \overline{E}_r$. Tree decompositions, here concerning the request graphs, are then defined as follows [6].

Definition 3.2 (Tree Decomposition $\mathcal{T}_r = (T_r, \mathcal{B}_r)$). Given an undirected request $\overline{G}_r = (V_r, \overline{E}_r)$, a tree decomposition of $\overline{G}_r$ is a pair $\mathcal{T}_r = (T_r, \mathcal{B}_r)$ consisting of an undirected tree $T_r = (V_T, E_T)$ and a family $\mathcal{B}_r = \{B_t\}_{t \in V_T}$ of subsets $B_t \subseteq V_T$, also referred to as the node bags, for which the following conditions hold:

(1) For all request nodes $i \in V_r$, the set $V_T^{-1}(i) = \{t \in V_T \mid i \in B_t\}$ of tree nodes containing node i is connected in T_r.
(2) Each request node and each (undirected) request edge is contained in at least one of the bags: $\forall i \in V_r . \exists t \in V_T : i \in B_t$ and $\forall \{i,j\} \in \overline{E}_r . \exists t \in V_T : \{i,j\} \subseteq B_t$ hold. □

The treewidth is then defined as follows (cf. [6]):

Definition 3.3 (Width of a Tree Decomposition and Treewidth). The width $\mathrm{tw}(\mathcal{T}_r) \in \mathbb{N}$ equals the maximal bag size minus one, i.e. $\mathrm{tw}(\mathcal{T}_r) = \max_{t \in V_T} |B_t| - 1$. The treewidth of an undirected graph equals the minimal width among all tree decompositions. □

Finding tree decompositions of minimal width is itself a challenging optimization problem and known to be $\mathcal{NP}$-hard [6]. However, if the treewidth of a graph G is known to be $k \in \mathbb{N}$, the problem of finding a tree decomposition is fixed-parameter tractable. Several important graph classes (including many request topologies usually considered in the literature) are known to have small treewidths (cf. Table 1). The example requests of Figure 2, a service chain [8] and a virtual cluster [1], have treewidths 1 and 2 respectively, as the service chain is outerplanar and the virtual cluster is a tree. However, even if a request graph does not belong to a graph class of bounded treewidth, recent exact algorithms can compute optimal tree decompositions in a matter of seconds (see Section 7).

A tree decomposition naturally groups request nodes together into node bags. As the size of each bag is bounded for graphs

Graph Class	tw	Description
trees	1	connected graph without cycle
cacti	2	cycles intersect only in a single node
series-parallel	2	source-terminal graphs; generated using parallel and serial composition
(1-)outerplanar	2	planar graph; nodes lie on outer face
k-outerplanar	$k+1$	planar graph; removal of outer face nodes yields $(k-1)$-outplanar graph

Table 1: Graph Classes of Bounded Treewidth [2]

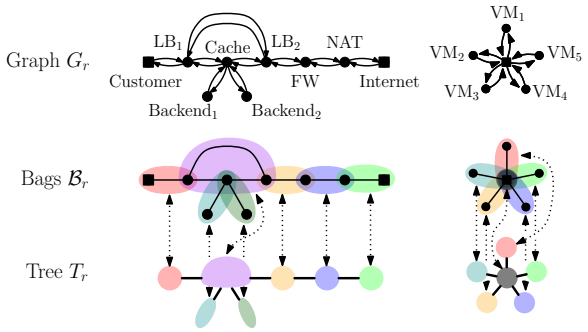

Figure 2: Depicted are two exemplary virtual network request graphs together with corresponding tree decompositions: a load-balancing service chain and a virtual cluster with 5 VMs. The *covering* node bags are depicted in the middle, while the resulting trees are depicted on the bottom. The widths of the decompositions are 2 (left) and 1 (right).

of bounded treewidth, this allows to perform more complex operations on the whole bag *in polynomial-time*. In particular, instead of mapping single virtual nodes, we will consider the joint mappings of all request nodes contained in the bags. While the number of mapping possibilities grows exponentially in the node bag's size, it is *polynomial* for graphs of bounded treewidth. Concretely, the number of mapping possibilities for a node bag B_t equals $\prod_{i \in B_t} |V_S^{r,i}| \in O(|V_S|^{\mathrm{tw}(\mathcal{T}_r)+1})$. We mathematically represent the space of node bag mappings as follows. We denote by $\mathcal{M}(B_t) = [B_t \to V_S]$ the set of *all* functions from B_t to V_S, i.e. $m_t^V \in \mathcal{M}(B_t)$ maps all virtual nodes of B_t. Given a specific bag mapping $m_t^V \in \mathcal{M}(B_t)$, a cost-optimal valid mapping of the subgraph $G_r[B_t] = (B_t, E_r[B_t])$ induced by B_t, i.e. $E_r[B_t] = \{(i,j) \in E_r \mid i,j \in B_t\}$, is computable in polynomial-time:

Lemma 3.4 (Computation of optimal induced mappings). *Given a node bag mapping $m_t^V \in \mathcal{M}(B_t)$, one can check in time $O(\mathrm{poly}(|B_t| \cdot |G_S|))$ if a valid edge mapping extension m_t^E exists, such that $m_t = (m_t^V, m_t^E)$ is a valid mapping of the induced subgraph $G_r[B_t]$. Furthermore, if such an induced valid mapping exists, the least cost one can be computed in time $O(\mathrm{poly}(|B_t| \cdot |G_S|))$.*

Proof. The validity of the given node mapping m_t^V can be checked by testing whether $m_t^V(i) \in V_S^{r,i}$ holds for each virtual node $i \in B_t$. As the node mappings are fixed, one can compute a shortest *valid* path for each edge $(i,j) \in E_r[B_t]$ by applying e.g. Dijkstra's algorithm, albeit only considering substrate edges contained in $E_S^{r,i,j}$. If valid paths exist for all induced edges $E_r[B_t]$ under the node mapping m_t^V, a cost-optimal edge mapping m_t^E is obtained and otherwise no valid mapping can exist. □

Besides this, we employ the following facts for our algorithm.

Fact 3.5 ([6]). *Let $\mathcal{N}(t) \subseteq V_T$ denote the neighboring tree nodes of $t \in V_T$. For any tree node $t \in V_T$ and any pair $t_1, t_2 \in \mathcal{N}(t)$ of neighbors of t with $t_1 \neq t_2$, the following holds: $(B_{t_1} \cap B_{t_2}) \setminus B_t = \emptyset$.*

Fact 3.6 ([6]). *Any tree decomposition can be transformed into a small one for which $B_{t_1} \not\subseteq B_{t_2}$ holds for all $t_1, t_2 \in V_T$ with $t_1 \neq t_2$. For any small tree decomposition $|V_T| = |\mathcal{B}_r| \leq |V_r|$ holds.*

The first fact states that node bags *separate* neighboring node bags from each other, while the second allows to bound the size of the tree $|V_T|$ by the number of original request nodes $|V_r|$.

The following additional notation will be used throughout this work. We employ $B_{t_1 \cap t_2}$ to denote the intersection of the corresponding node bags, i.e. $B_{t_1 \cap t_2} = B_{t_1} \cap B_{t_2}$. Given a node bag mapping m_t^V, we denote by $\langle m_t^V | V_r' \rangle : V_r' \to V_S$ the restriction of m_t^V to a subset $V_r' \subseteq B_t$, such that $\langle m_t^V | V_r' \rangle(i) = m_t^V(i)$ for $i \in V_r'$.

4 DYNAMIC PROGRAM DynVMP

We now present the $\mathcal{XP}$-algorithm DynVMP for solving the VMP. We first consider the VMP without latency constraints and afterwards present a minor extension to cater for latencies. The algorithm uses the tree decomposition $\mathcal{T}_r$ of the request graph and applies dynamic programming: starting from the leaves of the tree decomposition, (partial) cost-optimal valid mappings are constructed bottom-up. This is facilitated by Lemma 3.4. Starting at the leaves, these cost-optimal valid mappings are combined in a bottom-up fashion. Concretely, the algorithm stores for each tree node $t \in V_T$ and each node bag mapping $m_t^V \in \mathcal{M}(B_t)$ the optimal mapping costs in the table $\mathbb{C}[t][m_t^V]$ (infinite costs indicate infeasibility) together with the node mappings in table $\mathbb{M}[t][m_t^V]$ (see Lines 2-4).

The nodes of the tree decomposition are then traversed bottom-up (post-order traversal). Considering a specific tree node $t \in V_T$ with node bag B_t, all node bag mappings $m_t^V \in \mathcal{M}(B_t)$ are enumerated (Line 7). Only if the induced mapping is valid, the mapping is considered and otherwise the corresponding cost $\mathbb{C}[t][m_t^V]$ stays infinite (indicating invalidity). Considering leaves, the (induced) mapping costs of locally valid mappings can be readily computed using Lemma 3.4 by the `InducedCost` function. For nodes having children, the current mapping m_t^V is sought to be extended as cheaply as possible. To this end, all suitable child mappings $m_{t_c}^V \in \mathcal{M}(B_{t_c})$ agreeing with the current mapping m_t^V are considered and according to the cost-optimal one the mapping table $\mathbb{M}[t][m_t^V]$ is updated. Importantly, the different children will never set a mapping of a virtual node $i \in V_r$ twice by Fact 3.5: a request node i is either contained in only a single child bag or in multiple; however, if it is contained in multiple bags, then it must be contained in B_t. Accordingly, if $i \in B_t$ holds, then the mapping of i is already explicitly set by m_t^V and the child mappings cannot disagree on the mapping of i, as only matching mappings were selected in Line 12. Only if for all children valid mappings exist, the cost is updated and otherwise the mapping is considered to be invalid (cf. Lines 23 and 24). Having processed the whole tree, the optimal valid mapping is retrieved at the root node $\hat{t}_r$ or $\perp$ is returned to indicate that none exists.

Algorithm 1: DynVMP: Computing Optimal Valid Mappings

Input : substrate G_S, request G_r, tree decomposition $\mathcal{T}_r$
Output: valid mapping of minimal cost or $\perp$ if none exists

```
1  PrecomputeShortestValidPaths(G_r,G_S)
2  foreach t ∈ V_T do                                   // initialize tables
3      foreach m_t^V ∈ M(B_t) do
4          C[t][m_t^V] ← ∞ and M[t][m_t^V] ← (i ↦ ⊥ | i ∈ V_r \ B_t)
5  set Q_T ← PostOrderTraversal(T_r, t̂_r)
6  for t ∈ Q_T do                                       // traverse tree in post-order
7      for m_t^V ∈ M(B_t) do                            // consider node bag mappings
8          if InducedMappingLocallyValid(m_t^V) then
9              set children_valid ← True
10             for (t, t_c) ∈ δ⁺(t) do // find best child mapping m̂_{t_c}^V
11                 set m̂_{t_c}^V ← ⊥
12                 for ( m_{t_c}^V ∈ M(B_{t_c}) with
                         ⟨m_{t_c}^V | B_{t_c ∩ t}⟩ = ⟨m_t^V | B_{t_c ∩ t}⟩ ) do
13                     if m̂_{t_c}^V = ⊥ or C[t_c][m_{t_c}^V] < C[t_c][m̂_{t_c}^V] then
14                         m̂_{t_c}^V ← m_{t_c}^V
15             if m̂_{t_c}^V ≠ ⊥ then                    // if valid mapping exists
16                 for i ∈ V_r \ B_t do                 // as m_t^V fixes B_t mapping
17                     if i ∈ B_{t_c} then              // as m_{t_c}^V maps i
18                         M[t][m_t^V](i) ← m̂_{t_c}^V(i)
19                     else if M[t_c][m̂_{t_c}^V](i) ≠ ⊥ then
20                         M[t][m_t^V](i) ← M[t_c][m̂_{t_c}^V](i)
21             else                                     // induced valid mapping cannot exist
22                 set children_valid ← False and exit for-loop
23         if children_valid then
24             C[t][m_t^V] ← InducedCost(m_t^V ∪ M[t][m_t^V])
25 choose m̂_{t̂_r}^V ∈ M(B_{t̂_r}) s.t. ĉ ← C[t̂_r][m̂_{t̂_r}^V] is minimal
26 if ĉ < ∞ then return InducedMapping(m̂_{t̂_r}^V ∪ M[t̂_r][m̂_{t̂_r}^V])
27 else return ⊥
```

Theorem 4.1. *The DynVMP algorithm correctly determines whether a valid mapping exists and if so, returns a cost-optimal one. Its runtime is bounded by* $O(|V_r|^3 \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+3})$.

Proof. By the above description of the algorithm, the algorithm returns an optimal valid mapping, if one exists. With respect to the runtime, we first note that $|V_T| \leq |V_r|$ holds when considering small tree decompositions (cf. Fact 3.6). The pre-computation of all shortest valid paths can be implemented in time $O(|V_r|^2 \cdot |V_S|^3)$ by applying Dijkstra's algorithm for each of the $O(|V_r|^2)$ request edges for each potential substrate start node. On the other hand, the runtime of the Lines 12 to 22 dominate the main algorithm's runtime. Here, for each of the at most $|V_r|$ tree nodes at most $|V_S|^{\mathrm{tw}(\mathcal{T}_r)+1}$ many mappings m_t^V are considered, for which again at most $|V_r| \cdot |V_S|^{\mathrm{tw}(\mathcal{T}_r)+1}$ many mappings of its children must be considered while adapting the mappings in Lines 17 to 20 may again take $O(|V_r|)$ time, yielding the claimed overall runtime. □

Lastly, we show that the DynVMP algorithm can be used to approximate the cost of valid mappings *under latency constraints*. While computing minimum-cost latency-constrained shortest paths (LCSP) is itself an $\mathcal{NP}$-hard problem, a fully polynomial-time approximation scheme (FPTAS) exists:

Theorem 4.2 (LCSP FPTAS, Lorenz & Raz [9]). *For any* $\varepsilon' > 0$, *a* $(1+\varepsilon')$*-optimal path satisfying the latency bound can be computed in* $O\Big(|E_S| \cdot |V_S| \cdot (\log\log|V_S| + 1/\varepsilon')\Big) = \mathrm{time}_{LCSP}(\varepsilon')$.

The FPTAS for the LCSP can be used in the DynVMP algorithm to compute approximate latency respecting valid paths in Line 1. As each computed path is $(1+\varepsilon')$-optimal, the resulting mapping is also $(1+\varepsilon')$-optimal and we obtain the following result:

Theorem 4.3. *Using the LCSP FPTAS, the DynVMP algorithm finds a* $(1+\varepsilon')$*-optimal valid mapping, if one exists. Its runtime is bounded by* $O(|V_r|^2 \cdot (|V_r| \cdot |V_S|^{2 \cdot \mathrm{tw}(\mathcal{T}_r)+2} + \mathrm{time}_{LCSP}(\varepsilon')))$.

5 $\mathcal{XP}$-APPROXIMATIONS FOR THE VNEP

We now present a novel Linear Programming (LP) approach for the VNEP that allows us to generalize the previously obtained approximation result for the offline VNEP of [11]. Our approach also enables the first approximations under latency constraints.

Concretely, in [11] it was shown that approximations can be obtained if the *fractional* offline VNEP can be solved. In the fractional variant several valid mappings can be selected and weighed to obtain a convex combination of mappings. The LP Formulation 1 naturally models this by enumerating *all* valid mappings. Using randomized rounding (see Algorithm 2), the following was obtained:

Theorem 5.1 (VNEP Approximation for Cacti [11]). *Algorithm 2 returns an (α, β, γ)-approximate solution for the VNEP of at least an $\alpha = 1/3$ fraction of the optimal profit, and allocations on nodes and edges within factors of β and γ of the original capacities, respectively, with high probability, where $\beta, \gamma \geq 1$ are defined as*

$$\beta = 1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(V_S) \cdot \log(|V_S|)} \text{ and } \gamma = 1 + \varepsilon \cdot \sqrt{2 \cdot \Delta(E_S) \cdot \log(|E_S|)}$$

with $\Delta(X) = \max_{x \in X} \sum_{r \in \mathcal{R}: d_{\max}(r,x) > 0} (A_{\max}(r,x)/d_{\max}(r,x))^2$ being the maximal sum of squared maximal allocation-to-capacity ratios over the resource set X and the maximum demand-to-capacity ratio $\varepsilon = \max_{r \in \mathcal{R}, x \in G_S} d_{\max}(r,x)/d_S(x)$.

In the following, we show how the LP Formulation 1 can be solved efficiently for *arbitrary request graphs* by using the DynVMP algorithm. While the primal formulation uses exponentially many variables, its dual (cf. Formulation 2) uses a polynomial number of variables $\vec{\lambda}$ (corresponding to Constraint 2) and $\vec{\mu}$ (corresponding to Constraint 3) while employing exponentially many constraints. However, it is known that such LP formulations can be solved in polynomial-time, as long as *violated* constraints can be identified in polynomial-time by a 'separation oracle' [7].

Considering the case without latencies first, Constraints 6 can be separated using DynVMP as follows. First, we interpret the $\vec{\mu}$ variables as resource costs $c_{S,\mu} : G_S \to \mathbb{R}_{\geq 0}$ with $c_{S,\mu}(x) = \mu_x$. Accordingly, the mapping cost $c_{S,\mu}(m_r)$ of a valid mapping m_r equals $\sum_{x \in G_S} \mu_x \cdot A(m_r, x)$, i.e. the second term of the left-hand side of Constraint 6. Accordingly, the DynVMP algorithm can be used to compute cost-optimal mappings $\hat{m}_r$ for each request $r \in \mathcal{R}$. If $c_{S,\mu}(\hat{m}_r) \geq b_r - \lambda_r$ holds, all valid mappings of request r satisfy Constraint 6. On the other hand, if $c_{S,\mu}(\hat{m}_r) < b_r - \lambda_r$ holds, then the constraint corresponding to the mapping $\hat{m}_r$ is added to the Linear Program 2. By initializing $\vec{\lambda} = \vec{\mu} = \vec{0}$ and iteratively separating the violated constraints as long as one exists, an optimal LP solution can be computed. For practical applications, the following lemma is helpful in terminating the separation process once a solution of sufficient quality has been found:

Lemma 5.2. *Let $\vec{\mu}, \vec{\lambda}$ be the dual variables of a primal LP solution and let $\epsilon > 0$. If $c_{S,\mu}(m_r) \cdot (1 + \epsilon) \geq b_r - \lambda_r$ holds for all $m_r \in \mathcal{M}_r$ and each $r \in \mathcal{R}$, then the primal LP solution is $(1 + \epsilon)$-optimal.*

Proof. This follows from *weak duality* [7], as scaling the $\vec{\mu}$ variables by a factor of $(1 + \epsilon)$ yields a feasible dual solution while increasing the objective by at most a factor $(1 + \epsilon)$. □

As the separation of violated constraints equals the introduction of new variables ('columns') in the primal, this approach is generally referred to as 'column generation'. As the runtime of these approaches is polynomially bounded in the runtime of the separation oracle [7], the following $\mathcal{XP}$-result is obtained.

Formulation 1: Primal Enumerative LP for the Offline VNEP

$$\max \sum_{r \in \mathcal{R}, m_r^k \in \mathcal{M}_r} f_r^k \cdot b_r \tag{1}$$

$$\sum_{m_r^k \in \mathcal{M}_r} f_r^k \leq 1 \qquad \forall r \in \mathcal{R} \tag{2}$$

$$\sum_{r \in \mathcal{R}, m_r^k \in \mathcal{M}_r} f_r^k \cdot A(m_r^k, x) \leq d_S(x) \qquad \forall x \in G_S \tag{3}$$

$$f_r^k \in [0,1] \qquad \forall r \in \mathcal{R}, m_r^k \in \mathcal{M}_r \tag{4}$$

Formulation 2: Dual Enumerative LP for the Offline VNEP

$$\min \sum_{r \in \mathcal{R}} \lambda_r + \sum_{x \in G_S} \mu_x \cdot d_S(x) \tag{5}$$

$$\lambda_r + \sum_{x \in G_S} \mu_x \cdot A(m_r^k, x) \geq b_r \qquad \forall r \in \mathcal{R}, m_r^k \in \mathcal{M}_r \tag{6}$$

$$\lambda_r \geq 0 \qquad \forall r \in \mathcal{R} \tag{7}$$

$$\mu_x \geq 0 \qquad \forall x \in G_S \tag{8}$$

Algorithm 2: Randomized Rounding Approximation (cf. [11])

1 **foreach** $r \in \mathcal{R}$ **do** // preprocess requests
2 **compute** solution to LP Formulation 1 for request r
3 **remove** r from $\mathcal{R}$ if solution's profit is less than b_r
4 **compute** solution to LP Formulation 1 for all requests $\mathcal{R}$
5 **do** // perform randomized rounding
6 **foreach** $r \in \mathcal{R}$ **embed** r **using** m_r^k **with probability** f_r^k
   // request r is rejected with prob. $1 - \sum_k f_r^k$
7 **while** solution is *not* (α, β, γ)-approximate

Theorem 5.3. *LP Formulations 1 and 2 can be solved in time $O\left(\text{poly}\left(\sum_{r \in \mathcal{R}} |V_r|^3 \cdot |V_S|^{2 \cdot \text{tw}(\mathcal{T}_r)+3}\right)\right)$ by using DynVMP as oracle.*

As the approximation framework developed in [11] only depends on the ability to solve the LP Formulation 1 optimally, the approximation result readily carries over to arbitrary request graphs.

Theorem 5.4. *Using the DynVMP algorithm to solve LP Formulation 1 and applying the rounding procedure of [11], a tri-criteria (α, β, γ)-($\mathcal{XP}$-)approximation for the offline VNEP is obtained (with α, β, γ as defined in Theorem 5.1), with high probability. Accordingly, polynomial-time approximations are obtained when considering requests of bounded treewidth, as for example outerplanar graphs.*

We note that parametrized approximations are indeed the best one can hope for, as the VNEP is $\mathcal{NP}$-complete for planar request graphs [10] and – unless $\mathcal{P} = \mathcal{NP}$ holds – no polynomial-time algorithms can exist. Furthermore, we note that planar graphs have unbounded treewidth: a $k \times k$ grid has a treewidth of k [6].

Lastly, we turn towards approximations under latency constraints. In this case, the Constraints 6 can only be separated approximatively (cf. Theorem 4.3): for each request $r \in \mathcal{R}$ a $(1 + \varepsilon')$-optimal mapping $\tilde{m}_r$ is computed and respective columns are added as long as $c_{S,\mu}(\tilde{m}_r) < b_r - \lambda_r$ holds. After the separation process, some of the Constraints 6 might still be violated. In fact, only $c_{S,\mu}(m_r^k) \cdot (1 + \varepsilon') \geq b_r - \lambda_r$ holds for all mappings $m_r^k \in \mathcal{M}_r$ and by Lemma 5.2 the respective solution is $(1 + \varepsilon')$-optimal.

To obtain an approximation, Algorithm 2 must be adapted to this approximative setting as follows. In Line 3, a request r is only removed when their achieved profit is less than $b_r/(1 + \varepsilon')$, as this indicates that the request can never be fully embedded. Furthermore, the analysis of [11] requires that the LP's profit $\hat{b}$ is larger than $b_{\max} = \max_{r \in \mathcal{R}} b_r$. First, note that $b_{\max}/(1 + \varepsilon') \leq \hat{b}$ always holds [11]. Accordingly, in the case that $b_{\max}/(1 + \varepsilon') \leq \hat{b} < b_{\max}$

holds, an additional profit of $b_{\max} - \hat{b}$ must be ensured. This can be achieved by adding a fractional embedding of the request $r_{\max} \in \mathcal{R}$ having the largest profit. In particular, given the initial LP computation for $r_{\max}$ of Line 2, the returned solution can be scaled to fully embed $r_{\max}$ while exceeding capacities by at most a factor $(1+\varepsilon')$. Adding a $(b_{\max} - \hat{b})/b_{\max} \leq \varepsilon'/(1+\varepsilon')$ fraction of this embedding to LP solution of Line 4, the condition $\hat{b} \geq b_{\max}$ holds while the LP solution exceeds capacities by at most a factor $(1+\varepsilon')$. Rounding this solution as before, the following approximation is obtained:

Theorem 5.5. *For the offline VNEP with latency constraints, a tri-criteria* $(\alpha/(1+\varepsilon'), \beta+\varepsilon', \gamma+\varepsilon')$*-(*$\mathcal{XP}$*-)approximation is obtained for any* $\varepsilon' > 0$ *and* α, β, γ *as defined in Theorem 5.1,* with high probability. *The runtime of the algorithm lies in* $O\left(\text{poly}\left(\sum_{r\in\mathcal{R}} |V_r|^2 \cdot \left(|V_r| \cdot |V_S|^{2\cdot \text{tw}(\mathcal{T}_r)+2} + \text{time}_{LCSP}(\varepsilon')\right)\right)\right)$.

6 RANDOMIZED ROUNDING HEURISTICS

The approximations come at the cost of violating resource capacities. However, randomized rounding can be easily adapted to obtain $\mathcal{XP}$-heuristics not violating resource capacities. In [11] a first heuristic was proposed, which works as Algorithm 2 but discards selected mappings whose addition would exceed capacities.

As an improvement over this heuristic and facilitated by the column generation approach, we propose a novel rounding heuristic that a priori removes mappings whose addition would lead to resource violations and recomputes the LP before applying the rounding (see Algorithm 3). Therefore, the addition of any rounded mapping is *feasible* while also better guiding the rounding process by providing *currently optimal* rounding probabilities. Specifically, in Lines 4 and 5 first all infeasible mappings are 'removed' by setting the respective LP variables to 0. To reflect made rounding decisions in the LP, either the respective mapping variable is set to 1 (Line 9), or all mappings of a rejected request are disabled (Line 11).

Besides this novel rounding heuristic, we also consider different orders to round request mappings in: randomized (R) as proposed in [11], and either sorting the requests in descending fashion by their static (S) profits or their actual achieved profit (A) in the LP.

Algorithm 3: Heuristical Rounding with LP Recomputation

1 **compute solution to LP 1** (using column generation)
2 **set** sol $\leftarrow \emptyset$ **and** $\mathcal{R}' \leftarrow \mathcal{R}$
3 **foreach** $r \in \mathcal{R}$ **do**
4 **foreach** $r' \in \mathcal{R}'$ **and each** $m^k_{r'} \in \mathcal{M}_{r'}$ **do**
5 **if** sol $\cup \{m^k_r\}$ is infeasible **then set** $f^k_{r'} = 0$
6 **resolve Linear Program** (without column generation)
7 **choose** $\hat{m}_r \leftarrow m^k_r$ **with probability** f^k_r
8 **if** $\hat{m}_r \neq \emptyset$ **then**
9 **set** sol $\leftarrow$ sol $\cup \{\hat{m}_r\}$ **and** $f^k_r = 1$ `// accept request r`
10 **else**
11 **set** $f^k_r = 0$ **for all** $m^k_r \in \mathcal{M}_r$ `// reject request r`
12 $\mathcal{R}' \leftarrow \mathcal{R} \setminus \{r\}$
13 **return** sol

7 EVALUATION

In this section we present two types of evaluation to validate our approach. Firstly, we present a study of the treewidth of random graphs to grasp for which graphs our approach may be reasonable. Secondly, we generate a large set of offline VNEP instances in accordance with the methodology presented in [11] and compare the performance of the randomized rounding heuristics to the performance of the well-known ViNE heuristics [3]. All experiments have been conducted on a server equipped with 4 Intel XEON E5-4627v3 CPUs and 256 GB RAM. Our Python 2.7 source code, which uses Gurobi 8.0 to solve the LPs, is publicly available[1].

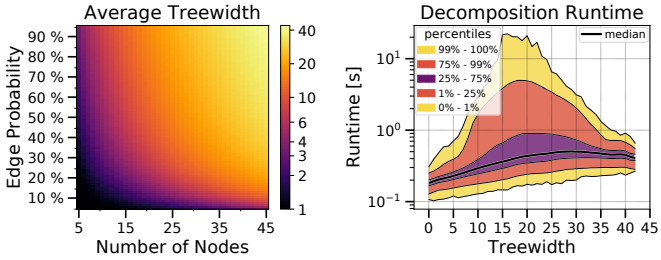

Figure 3: Study of the treewidth of random graphs using Tamaki's algorithm [12]. Note the logarithmic axes.

Qualitative and Quantitative Analysis of the Treewidth. We have generated 1,200 undirected graphs with $\{5, \ldots, 45\}$ nodes and edge creation probabilities in the range of $\{0.05, 0.06, \ldots, 0.95\}$, yielding 4.47M graphs overall. We have then run the exact algorithm by Tamaki [12] to compute the optimal treewidth. Our results are presented in Figure 3. Notably, the (average) treewidth is less than 6 for most graphs with fewer than 15 nodes and a connection probability of less than 50%. The runtime for computing the tree decompositions of width less than 10 lies vastly below two seconds with a median computation time of only 200ms, enabling the application of our approach in the first place.

Instance Generation. We have generated 6,000 offline VNEP instances (without latencies) according to the methodology presented in [11], also using the same 40-node substrate topology GÉANT from the Topology Zoo. Instances of $\{40, 60, 80, 100\}$ requests are generated having a treewidth of exactly $\{1, 2, 3, 4\}$. The number of nodes per request is drawn uniformly from $\{5, \ldots, 15\}$. For treewidth 1, i.e. trees, the request graphs are generated randomly by adding edges until the graph is a tree (discarding edges creating cycles). For generating graphs of treewidth 2, 3, 4, we employ the graphs generated to evaluate the performance of Tamaki's algorithm. To this end we have stored all generated undirected graphs and uniformly at random select graphs of the respective treewidth and number of nodes. As directed requests are considered, edge orientations are chosen uniformly at random.

As in [11] the request demands are drawn from an exponential distribution according to node resource factors (NRF) $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ and edge resource factors (ERF) $\{0.25, 0.5, 1.0, 2.0, 4.0\}$. Intuitively, a NRF of 0.6 implies that the sum of generated node resource demands equals 60% of the available substrate node resources, while an ERF of 0.5 implies that if each virtual edge spans exactly 0.5 substrate edges, then the average edge utilization equals exactly 100%. As in [11], the mapping of virtual nodes is constrained to 25% of substrate nodes and the profits of requests are set to equal the (optimal) minimum

[1] https://github.com/vnep-approx/evaluation-acm-ccr-2019

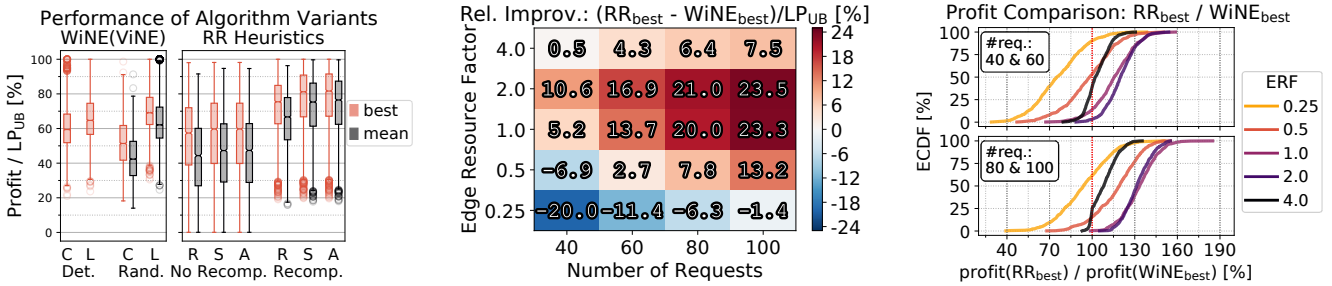

Figure 4: Performance of the WiNE algorithms and the randomized rounding heuristics. Left: Performance of the different studied algorithms compared to the upper bound (LP_{UB}). Center: Relative improvement of randomized rounding over WiNE (a cell averages 300 results). Right: Direct comparison of the best profits achieved (an ECDF represents 600 results).

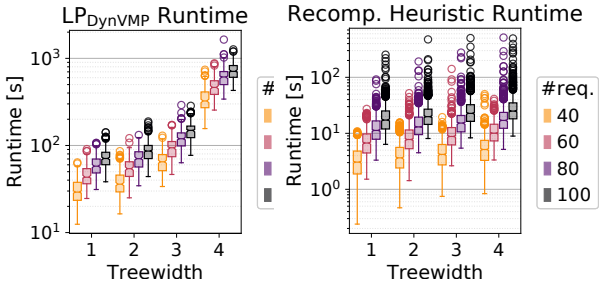

Figure 5: Runtime of the novel LP (left) and of the novel rounding heuristic (right) as a function of the treewidth and the number of requests. Note the logarithmic y-axes.

embedding costs. For each parameter combination (number of requests, treewidth, NRF, ERF) 15 instances are considered.

Studied Algorithms and Implementation Details. We compare the performance of the randomized rounding heuristics and the offline ViNE heuristics for unsplittable edge embeddings [3]. The ViNE algorithms use the Multi-Commodity Flow LP to guide the embedding of single requests: node mappings are performed either randomly or deterministically according to the LP node mapping variables while request edges are embedded using shortest-paths. Two different LP objectives were proposed in [3]: one minimizing resource usage and another also performing load-balancing. For the offline setting, the authors of [3] have proposed the window-based heuristic (WiNE) that orders requests descendingly according to their profits and greedily embeds each request using ViNE.

Our DynVMP implementation employs several optimizations. Most importantly, cost computations are realized using matrix multiplications. For the novel LP, the separation is terminated upon 1.001-optimality (cf. Lemma 5.2). For each randomized algorithm several executions are considered: 20 for WiNE, 50 for randomized rounding with recomputations and 500 without.

Results. We first report on the performance of the different algorithms. In Figure 4 (left) the best and mean solution quality relative to the maximum attainable LP profit is depicted. For WiNE, the load-balancing (L) objective outperforms the cost (C) one. Considering the randomized rounding (RR) heuristics, the ones with recomputations significantly outperform the ones without. Ordering the requests according to the profit (S / A) yields the best solutions.

As we are mostly interested in the potential improvement over greedy heuristics, the center and right plots of Figure 4 compare the best solution computed by any WiNE execution to the best solution of any randomized rounding execution. The mean relative improvement over WiNE significantly depends on the number of requests and the edge resource factor: when edge resources are scarce (ERFs 0.25 and 0.5) WiNE performs better, while for ERFs 1.0, 2.0, and 4.0 randomized rounding consistently yields better solutions (86.7% of scenarios). Even more, for 80 and 100 requests and ERFs of 1.0 and 2.0, randomized rounding finds better solutions in 99.9% of the scenarios, improving the best WiNE solution by more than 30% in 57.5% of the scenarios. The performance drop for low ERFs may be due to fewer generated mappings being feasible.

The runtime of the column generation LP lies in the order of 100 seconds for treewidths below 3 and several hundred seconds for treewidth 4 (see Figure 5). The runtime of the recomputation heuristics mainly ranges between few seconds and 60 seconds (see Figure 5). The average runtimes of the rounding without recomputations and the WiNE heuristics was 0.03s and 6.38s, respectively.

8 CONCLUSION

This work has presented the first $\mathcal{XP}$-approximations for the VNEP for *arbitrary* request graphs and allowing for edge latencies. As shown in the evaluation, applying randomized rounding heuristics can yield significantly better solutions in practice compared to greedy heuristics while coming at the cost of higher runtimes.

REFERENCES

[1] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards predictable datacenter networks. In *ACM SIGCOMM CCR*, Vol. 41.

[2] Hans L. Bodlaender. 1998. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science* 209, 1 (1998), 1 – 45.

[3] M. Chowdhury, M. R. Rahman, and R. Boutaba. 2012. ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping. *IEEE/ACM Transactions on Networking* 20, 1 (2012).

[4] Guy Even, Matthias Rost, and Stefan Schmid. 2016. An Approximation Algorithm for Path Computation and Function Placement in SDNs. In *Proc. SIROCCO*.

[5] A. Fischer, J.F. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. 2013. Virtual Network Embedding: A Survey. *Comm. Surveys Tutorials, IEEE* 15, 4 (2013).

[6] Jörg Flum and Martin Grohe. 2006. *Parameterized complexity theory.* Springer.

[7] Martin Grötschel, László Lovász, and Alexander Schrijver. 1988. *Geometric algorithms and combinatorial optimization.* Springer-Verlag Berlin Heidelberg.

[8] J. Gil Herrera and J. F. Botero. 2016. Resource Allocation in NFV: A Comprehensive Survey. *IEEE TNSM* 13, 3 (2016).

[9] Dean H. Lorenz and Danny Raz. 2001. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters* 28, 5 (2001).

[10] Matthias Rost and Stefan Schmid. 2018. NP-Completeness and Inapproximability of the Virtual Network Embedding Problem and Its Variants. In *Proc. IFIP Networking 2018.*

[11] Matthias Rost and Stefan Schmid. 2018. Virtual Network Embedding Approximations: Leveraging Randomized Rounding. In *Proc. IFIP Networking 2018.*

[12] Hisao Tamaki. 2017. Positive-Instance Driven Dynamic Programming for Treewidth. In *Proc. 25th Annual European Symposium on Algorithms (ESA).*

[13] M. F. Zhani, Q. Zhang, G. Simona, and R. Boutaba. 2013. VDC Planner: Dynamic migration-aware Virtual Data Center embedding for clouds. In *Proc. IFIP/IEEE International Symposium on Integrated Network Management.*