Virtual Network Mapping in Cloud Computing: A Graph Pattern Matching Approach

Citation for published version:
Cao, Y, Fan, W & Ma, S 2017, 'Virtual Network Mapping in Cloud Computing: A Graph Pattern Matching Approach', *The Computer Journal*, vol. 60, no. 3, pp. 287-307. https://doi.org/10.1093/comjnl/bxw063

Digital Object Identifier (DOI):
10.1093/comjnl/bxw063

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
The Computer Journal

Virtual Network Mapping in Cloud Computing: A Graph Pattern Matching Approach

YANG CAO [1,2] WENFEI FAN [1,2] SHUAI MA [2]

[1] *University of Edinburgh* [2] *RCBD & SKLSDE Lab, Beihang University*
Email: {yang.cao@, wenfei@inf.}ed.ac.uk mashuai@buaa.edu.cn

Virtual network mapping (VNM) is to build a network on demand by deploying virtual machines in a substrate network, subject to constraints on capacity, bandwidth and latency. It is critical to data centers for coping with dynamic cloud workloads. This paper shows that VNM can be approached by graph pattern matching, a well-studied database topic. (1) We propose to model a virtual network request as a graph pattern carrying various constraints, and treat a substrate network as a graph in which nodes and edges bear attributes specifying their capacity. (2) We show that a variety of mapping requirements can be expressed in this model, such as virtual machine placement, network embedding and priority mapping. (3) In this model, we formulate VNM and its optimization problem with a mapping cost function. We establish complexity bounds of these problems for various mapping constraints, ranging from PTIME to NP-complete. For intractable problems, we show that their optimization problems are approximation-hard, *i.e.,* NPO-complete in general and APX-hard even for special cases. (4) We also develop heuristic algorithms for priority mapping, an intractable problem. (5) We experimentally verify that our algorithms are efficient and are able to find high-quality mappings, using real-life and synthetic data.

Keywords: Graph Pattern Matching; Cloud Computing; Virtual Network Mapping

1. INTRODUCTION

Virtual network mapping (VNM) is also known as virtual network embedding or assignment. It takes as input (1) a *substrate network* (SN, a physical network), and (2) a *virtual network* (VN) specified in terms of a set of virtual nodes (machines or routers, denoted as VMs) and their virtual links, along with constraints imposed on the capacities of the nodes (*e.g.,* CPU and storage) and on the links (*e.g.,* bandwidth and latency). VNM is to deploy the VN in the SN such that virtual nodes are hosted on substrate nodes, virtual links are instantiated with physical paths in the SN, and the constraints on the virtual nodes and links are satisfied.

VNM is critical to managing big data. Big data is often distributed to data centers [1, 2]. However, data center networks often become *the bottleneck* for dynamic cloud workloads of querying and managing the data. In traditional networking platforms, network resources are manually configured with static policies, and new workload provisioning often takes days or weeks [3]. This highlights the need for VNM, to automatically deploy virtual networks in a data center network in response to real-time requests. Indeed, VNM is increasingly employed in industry, *e.g.,* Amazon's EC2 [4], VMware Data Center [5] and Big Switch Networks [3]. It has proven effective in increasing server utilization, and in reducing server provisioning time (from days or weeks to minutes), server capital expenditures and operating expenses [3]. There has also been a host of work on virtualization techniques for big data [1, 2] and database systems [6–10].

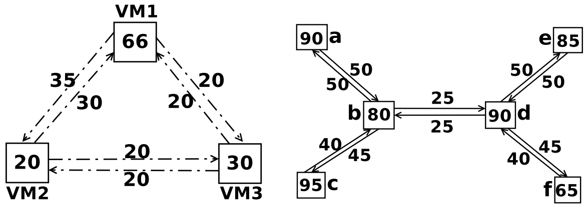
Several models have been proposed to specify VNM in various settings (see notations summarized in Table 1):

(1) *Virtual machine placement* (VMP): it is to find a mapping f from virtual machines in a VN to substrate nodes in an SN such that for each VM v, its capacity is no greater than that of $f(v)$, *i.e.,* $f(v)$ is able to conduct the computation of the VM v that it hosts [11].

(2) *Single-path* VN *embedding* ($\mathsf{VNE}_{\mathsf{SP}}$): it is to find

(a) an injective mapping f_v that maps nodes in VN to nodes in SN, subject to node capacity constraints; and

(b) a function that maps a virtual link (v, v') in VN to a path from $f_v(v)$ to $f_v(v')$ in SN that satisfies a bandwidth constraint, *i.e.,* the bandwidth of each link in the SN is no smaller than the sum of the bandwidth requirements of all those virtual links that are mapped to a path containing it [12–14].



(a) VN Request (b) SN

FIGURE 1. VN requests found in practice

(3) *Multi-path* VN *embedding* ($\mathsf{VNE_{MP}}$): it is to find a node mapping f_v as in $\mathsf{VNE_{SP}}$ and a function that maps each virtual link (v, v') to *a set* of paths from $f_v(v)$ to $f_v(v')$ in SN, subject to bandwidth constraints [15,16].

However, there are a number of VN requests that are commonly found in practice, but cannot be expressed in any of these models, as illustrated by the following.

EXAMPLE 1. Consider a VN request and an SN, depicted in Figures 1(a) and 1(b), respectively. The VN has three virtual nodes $\mathsf{VM_1}$, $\mathsf{VM_2}$ and $\mathsf{VM_3}$, each specifying a capacity constraint, along with a constraint on each virtual link. In the SN, each substrate node bears a resource capacity and each connection (edge) has an attribute, indicating either bandwidth or latency. Consider the following cases.

(1) Mapping with latency constraints ($\mathsf{VNM_L}$). Assume that the numbers attached to the virtual nodes and links in Fig. 1(a) denote requirements on CPUs and latencies for SN, respectively. Then the VNM problem, denoted by $\mathsf{VNM_L}$, aims to map each virtual node to a substrate node with sufficient computational power, and to map each virtual link (v, v') in the VN to a path in the SN such that its latency, *i.e., the sum* of the latencies of the edges on the path, does not exceed the latency specified for (v, v'). The need for studying VNM_L arises from latency sensitive applications such as multimedia transmitting networks [17], where constraints on virtual links concern latency rather than bandwidth.

(2) Priority mapping ($\mathsf{VNM_P}$). Assume that the constraints on the nodes in Fig. 1(a) indicate CPU capacities, and constraints imposed on the edges denote bandwidth capacities. Then the VNM problem, denoted by $\mathsf{VNM_P}$, is to map each virtual node to a node in SN with sufficient CPU capacity, and each virtual link (v, v') in the VN to a path in SN such that the *minimum* bandwidth of all edges on the path is no less than the bandwidth specified for (v, v'). The need for this is evident in many applications [18,19], when we want to give different priorities at run time to virtual links that share some physical links, and require the mapping only to provide bandwidth guarantee for the connection with the highest priority.

(3) Mapping with node sharing ($\mathsf{VNE_{SP(NS)}}$). Assume that the numbers attached to the virtual nodes and links in Fig. 1(a) denote requirements on CPUs and bandwidths for SN, respectively. Then $\mathsf{VNE_{SP(NS)}}$ is an extension of the single-path VN embedding ($\mathsf{VNE_{SP}}$) by supporting node sharing, *i.e.,* by allowing multiple virtual nodes to be mapped to the same substrate node, as needed by, *e.g.,* X-Bone [20].

Similarly, there is also practical need for extending other mappings with node sharing, such as virtual machine placement (VMP), latency mapping ($\mathsf{VNM_L}$), priority mapping $\mathsf{VNM_P}$ and multi-path VN embedding ($\mathsf{VNE_{MP}}$). We denote such an extension by adding a subscript NS (see Table 1). □

TABLE 1. Notations and various VNM cases

Notation	Description
VNM	virtual network mapping
VN	virtual network
SN	substrate network
VMs	virtual nodes (machines or routers)
VMP ($\mathsf{VMP_{(NS)}}$)	VM Placement (node sharing (NS))
$\mathsf{VNM_P}$ ($\mathsf{VNM_{P(NS)}}$)	priority mapping (with NS)
$\mathsf{VNE_{SP}}$ ($\mathsf{VNE_{SP(NS)}}$)	single-path embedding (with NS)
$\mathsf{VNE_{MP}}$ ($\mathsf{VNE_{MP(NS)}}$)	multi-path embedding (with NS)
$\mathsf{VNM_L}$ ($\mathsf{VNM_{L(NS)}}$)	latency constrained mapping (NS)

Observe the following. (a) VNM varies from practical requirements, *e.g.,* when latency, high-priority connections and node sharing are concerned. (b) Existing models are not capable of expressing such requirements; indeed, none of them is able to specify $\mathsf{VNM_L}$, $\mathsf{VNM_P}$ or $\mathsf{VNE_{SP(NS)}}$. (c) It would be an overkill to develop a model for each of the large variety of requirements, and to study it individually.

As suggested by the example, we need a generic model to express virtual network mappings in various practical settings, including both those already studied (*e.g.,* VMP, $\mathsf{VNE_{SP}}$ and $\mathsf{VNE_{MP}}$) and those that have been overlooked (*e.g.,* $\mathsf{VNM_L}$, $\mathsf{VNM_P}$ and $\mathsf{VNE_{SP(NS)}}$). The uniform model allows us to characterize and compare VNM in different settings, and better still, to study generic properties that pertain to all the variants. Among these are the complexity and approximation analyses of VNM, which are obviously important but have not yet been systematically studied by and large.

Contributions & Roadmap. This work takes a step toward providing a uniform model to characterize VNM. We show that VNM, an important problem for managing big data, can actually be tackled by graph pattern matching techniques, a database topic that has been well studied. We also provide complexity and approximation bounds for VNM. Moreover, for intractable VNM cases, we develop effective heuristic methods to find high-quality mappings.

(1) We propose a generic model to express VNM in terms of graph pattern matching [21] (Section 2). In this model a VN request is specified as a graph pattern, bearing various constraints on nodes and links defined

with aggregation functions, and an SN is simply treated as a graph with attributes associated with its nodes and edges. The decision and optimization problems for VNM are then simply graph pattern matching problems. We show that the model is able to express VNM commonly found in practice, including all the mappings we have seen so far (all the cases in Table 1).

(2) We establish complexity and approximation bounds for VNM (Section 3). We give a uniform upper bound for the VNM problems expressed in this model, by showing that all these problems are in NP. We also show that VNM is polynomial time (PTIME) solvable if only node constraints are present (VMP), but it becomes NP-complete when either node sharing is allowed or constraints on edges are imposed (all the other cases in Table 1). Moreover, we propose a VNM cost function and study optimization problems for VNM based on the metric. We show that the optimization problems are intractable in most cases and worse still, are NPO-complete in general and APX-hard [22] for special cases. To the best of our knowledge, these are among the first complexity and approximation results on VNM.

(3) These results tell us that it is beyond reach in practice to find PTIME algorithms for VNM with edge constraints such as $\mathsf{VNM_P}$ and $\mathsf{VNE_{SP}}$, or to find efficient approximation algorithms with decent performance guarantees. In light of these, we develop heuristic algorithms for priority mapping $\mathsf{VNM_P}$, with node sharing or not (Section 4). We focus on $\mathsf{VNM_P}$ since it is needed in, *e.g.,* internet-based virtualized infrastructure computing platform (iVIC [18]) and prioritized polling for virtual network interfaces [19]. Our algorithm reduces unnecessary computation by minimizing VNs requests and utilizing auxiliary graphs of SNs. While several algorithms are available for VN embedding (*e.g.,* [12–14]), no previous work has studied algorithms for $\mathsf{VNM_P}$.

(4) Finally, we experimentally verify the effectiveness and efficiency of our algorithm by providing a simulation study (Section 5). We evaluate our algorithm for priority mapping and VN embedding (with node sharing or not). We find that our algorithm is able to find high-quality mappings and is efficient on large VNs and SNs. In particular, it is able to find high-quality mappings, and has higher acceptance ratio than the previous mapping model ($\mathsf{VNE_{SP}}$), typically from 11% to 39%. Furthermore, it took 420 seconds for SNs with 10^6 nodes, and substantially outperforms previous algorithms for $\mathsf{VNE_{SP}}$ (SubIso [13], ViNE [16], RW-SP [23]) that took at least 912 seconds.

We contend that these results are useful for developing virtualized cloud data centers for querying and managing big data, among other things. By modeling VNM as graph pattern matching, we are able to characterize various VN requests with different classes of graph patterns, and study the expressive power and complexity of these graph pattern languages. Furthermore, techniques developed for graph pattern matching can be leveraged to study VNM. Indeed, the proofs of some of the results in this work capitalize on graph pattern techniques. On the other hand, the results of this work are also of interest to *the study of graph pattern matching* [21].

Related Work. This paper is an extension of our earlier work [24] by adding (a) the proofs for the complexity and approximation analyses of VNM (Section 3), (b) a heuristic algorithm for computing the minimum cost priority mapping ($\mathsf{VNM_P}$), with node sharing or not (Section 4), and (c) an extensive experimental study of the algorithm for computing $\mathsf{VNM_P}$ using real-life and synthetic data (Section 5).

Virtualization techniques have been investigated for big data processing [1, 2] and database applications, such as database appliance deployment and virtualized resources management for database systems [6–10, 25]. However, none of these has provided a systematic study of VNM, by modeling VNM as graph pattern matching. The only exception is [13], which adopted subgraph isomorphism for VNM, a special case of the generic model proposed in this work. Moreover, complexity and approximation analyses associated with VNM have not been studied in database applications.

Several models have been developed for VNM. (a) The VM placement problem (VMP) was studied in [11], which is similar to the bin packing problem and aims to map a set of VMs onto an SN in the presence of constraints on node capacities. (b) Single-path VN embedding ($\mathsf{VNE_{SP}}$) was investigated in [14, 26, 27], which is to map a VN to an SN by a node-to-node injective function and an edge-to-path function, subject to constraints on the CPU capacities of nodes and constraints on the bandwidths of physical connections. (c) Different from $\mathsf{VNE_{SP}}$, multi-path embedding ($\mathsf{VNE_{MP}}$) was studied in [15, 16], which allows an edge of a VN to be mapped to multiple parallel paths of an SN such that the sum of the bandwidth capacities of those paths is no smaller than the bandwidth of that edge. (d) Graph layout problems, while they are similar to VN mapping, do not have bandwidth constraints on edges but instead, impose certain topological constraints (see [28] for a survey).

In contrast to this work, the prior models are studied for specific domains. No previous work has studied generic models to support various VN requests that commonly arise in practice. Moreover, no prior work has considered newly emerging settings such as priority mapping, mappings with only latency constraints on links, and mappings with node sharing, which are tackled in this paper.

Very few complexity results are known for VNM. The only work we are aware of is [29], which claimed that the testbed mapping problem is NP-hard in the presence of node types and some links with infinite capacity. Several complexity and approximation results are established for graph pattern matching (see [21, 30] for surveys). However, those results are for edge-to-edge mappings, whereas VNM typically needs to map virtual links to physical paths. There have been recent extensions to support edge-to-path mappings for graph pattern matching [31–34], with several intractability and approximation bounds established there. Those differ from this work in that either no constraints on links are considered [31, 33], or graph simulation is adopted [32, 34], which does not work for VNM. The complexity and approximation bounds developed in this work are among the first results that have been developed for VNM in cloud computing.

A number of algorithms have been developed for VNM. There are greedy algorithms for the VM placement problem [11]. When considering bandwidth constraints on links, [27] provided a heuristic algorithm to find mappings with load balance with infinite SN resources. A special case of mapping to SNs of a backbone-star shape was studied in [14], allowing constraints on both nodes and links. A path-splitting assumption was proposed in [15], to rectify limitations of mapping an edge to a single path. Based on this assumption, [16] developed an MIP model and corresponding algorithms for finding such mappings. However, none of these algorithms works for the priority mappings studied in this paper.

2. A GENERIC MODEL BASED ON GRAPH PATTERN MATCHING

In this section we first represent virtual networks (VNs) and substrate networks (SNs) as weighted directed graphs. We then introduce a generic model to express virtual network mapping (VNM) in terms of graph pattern matching [21, 30].

2.1. Substrate and Virtual Networks

An SN consists of a set of substrate nodes connected with physical links, in which the nodes and links are associated with resources of a certain capacity, *e.g.,* CPU and storage capacity for nodes, and bandwidth and latency for links. A VN is specified in terms of a set of virtual nodes and a set of virtual links, along with requirements on the capacities of the nodes and the capacities of the links. Both VNs and SNs can be naturally modeled as weighted directed graphs.

Weighted directed graphs. A *weighted directed graph* is defined as $G = (V, E, f_V, f_E)$, where (1) V is a finite set of nodes; (2) $E \subseteq V \times V$ is a set of edges, in which (v, v') denotes an edge from v to v'; (3) f_V is a function defined on V such that for each node $v \in V$, $f_V(v)$ is a positive rational number; and similarly, (4) f_E is a function defined on E.

Substrate networks. A *substrate network* (SN) is a weighted directed graph $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$, where (1) V_S and E_S denote sets of substrate nodes and (directly connected) physical links, respectively; and (2) the functions f_{V_S} and f_{E_S} denote resource capacities on the nodes (*e.g.,* CPU) and links (*e.g.,* bandwidth and latency), respectively.

Virtual networks. A *virtual network* (VN) is specified as a weighted directed graph $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$, where (1) V_P and E_P denote virtual nodes and links, and (2) f_{V_P} and f_{E_P} are functions defined on V_P and E_P in the same way as in substrate networks, respectively.

EXAMPLE 2. The SN depicted in Fig. 1(b) is a weighted graph G_S, in which (1) the node set is $\{\mathsf{a}, \mathsf{b}, ..., \mathsf{f}\}$; (2) the edges include the directed edges in the graph; (3) the weights associated with nodes indicate CPU capacities; and (4) the weights of edges denote bandwidth or latency capacities.

Figure 1(a) shows a VN, where (1) the node set is $\{\mathsf{VM_1}, \mathsf{VM_2}, \mathsf{VM_3}\}$; (2) the edge set is $\{(\mathsf{VM_i}, \mathsf{VM_j}) \mid \mathsf{i}, \mathsf{j} = 1, 2, 3\}$; (3) $f_{V_P}(\mathsf{VM_1}) = 66$, $f_{V_P}(\mathsf{VM_2}) = 20$, $f_{V_P}(\mathsf{VM_3}) = 30$; and (4) the function f_{E_P} is defined on the edge labels. As will be seen when we define the notion of VN requests, the labels indicate requirements on deploying the VN in an SN. □

Paths. A *path* ρ from node u_0 to u_n in an SN G_S is denoted as $(u_0, u_1, \ldots, u_n)$, where (a) $u_i \in V_S$ for each $i \in [0, n]$, (b) there exists an edge $e_i = (u_{i-1}, u_i)$ in E_S for each $i \in [1, n]$, and moreover, (c) for all i, $j \in [0, n]$, if $i \neq j$, then $u_i \neq u_j$. We write $e \in \rho$ if e is an edge on ρ. When it is clear from the context, we also use ρ to denote the set of edges on the path, *i.e.,* $\{e_i \mid i \in [1, n]\}$.

2.2. Virtual Network Mapping

Virtual network mapping (VNM) from a VN G_P to an SN G_S is specified in terms of a node mapping, an edge mapping and a VN request. The VN request imposes constraints on the node mapping and edge mapping, defining their semantics. We next define these notions.

A *node mapping* from G_P to G_S is a pair (g_V, r_V) of functions, where g_V maps the set V_P of virtual nodes in G_P to the set V_S of substrate nodes in G_S, and for each v in V_P, if $g_V(v) = u$, $r_V(v, u)$ is a positive number. Intuitively, function r_V specifies the amount of resource of the substrate node u that is allocated to the node v.

For each edge (v, v') in G_P, we use $P(v, v')$ to denote the set of paths from $g_V(v)$ to $g_V(v')$ in G_S. An *edge mapping* from G_P to G_S is a pair (g_E, r_E) of functions such that (i) for each edge $(v, v') \in E_P$, $g_E(v, v')$ is a subset of paths in $P(v, v')$ such that for any $\rho \in g_E(v, v')$, there exists an edge $e \in \rho$ that does not occur in any other path in $g_E(v, v')$, and (ii) r_E assigns a positive number to each pair (e, ρ) for $e \in E_P$ and $\rho \in g_E(e)$. Intuitively, $r_E(e, \rho)$ is the amount of resource of the physical path ρ allocated to virtual link e.

VN requests. A VN *request* to an SN G_S is a pair $(G_P, \mathcal{C})$, where G_P is a VN, and $\mathcal{C}$ is a set of constraints

TABLE 2. Various VN requests

Constraints	C1	C2	C3	C4	C5	C6	C7
VMP ($\text{VMP}_{(\text{NS})}$)	✓	✓	×	×	×	✓ (×)	×
VNM_P ($\text{VNM}_{\text{P(NS)}}$)	✓	✓	op:'≤'; agg:'max'	✓	op:'≤'; agg:'min'	✓ (×)	✓
VNE_SP ($\text{VNE}_{\text{SP(NS)}}$)	✓	✓	op:'≤'; agg:'sum'	✓	op:'≤'; agg:'min'	✓ (×)	✓
VNE_MP ($\text{VNE}_{\text{MP(NS)}}$)	✓	✓	op:'≤'; agg:'sum'	✓	op:'≤'; agg:'min'	✓ (×)	×
VNM_L ($\text{VNM}_{\text{L(NS)}}$)	✓	✓	op:'≥'; agg:'sum'	×	op:'≥'; agg:'sum'	✓ (×)	✓

such that for a pair $((g_V, r_V), (g_E, r_E))$ of node and edge mappings from G_P to G_S, each constraint in $\mathcal{C}$ has one of the following forms:

(1) for each $v \in V_P$, $f_{V_P}(v) \leq r_V(v, g_V(v))$;
(2) for all nodes $u \in V_S$, $f_{V_S}(u) \geq \mathsf{sum}(N(u))$, where $N(u)$ is $\{\!|r_V(v,u) \mid v \in V_P,\ g_V(v) = u|\!\}$, a bag (an unordered collection of elements with repetitions) determined by virtual nodes in G_P hosted by u;
(3) for all edges $e \in E_P$, $f_{E_P}(e)$ op $\mathsf{agg}(Q(e))$, where $Q(e)$ is $\{\!|r_E(e,\rho) \mid \rho \in g_E(e)|\!\}$, a bag collecting physical paths ρ that instantiate e; here op is comparison operator $\leq$ or $\geq$, and $\mathsf{agg}()$ is one of the aggregation functions min, max and sum;
(4) for all edges $e' \in E_S$, $f_{E_S}(e') \geq \mathsf{sum}(M(e'))$, where $M(e')$ is $\{\!|r_E(e,\rho) \mid e \in E_P,\ \rho \in g_E(e),\ e' \in \rho|\!\}$, a bag collecting those virtual links that are instantiated by a physical link ρ containing e'; and
(5) for all $e \in E_P$ and $\rho \in g_E(e)$, $r_E(e,\rho)$ op $\mathsf{agg}(U(\rho))$ where $U(\rho)$ is $\{\!|f_{E_S}(e') \mid e' \in \rho|\!\}$), a bag of all edges on a physical path that instantiate e.
(6) for all nodes $u \in V_S$, $|N(u)| \leq 1$, *i.e.,* node sharing is not allowed.
(7) for all $e \in E_P$, $|Q(e)| \leq 1$, *i.e.,* an virtual link is allowed to be mapped to one physical path in SN.

Constraints in a VN request are classified as follows.

Node constraints: Constraints of form (1), (2) or (6). Intuitively, a constraint of form (1) assures that when a virtual node v is hosted by a substrate node u, u must provide adequate resource. A constraint of form (2) asserts that when a substrate node u hosts (possibly multiple) virtual nodes, u must have sufficient capacity to accommodate all those virtual nodes. Constraint (6) specifies whether a substrate node u can host more than one virtual node, *i.e.,* if node sharing is not allowed, then constraint (6) is included in $\mathcal{C}$.

Edge constraints: Constraints of form (3), (4), (5) or (7). A constraint of form (3) assures that when a virtual link e is mapped to a set of physical paths in the SN, those physical paths taken together satisfy the requirements (on bandwidths or latencies) of e. We denote by $|Q(e)|$ the number of physical paths to which e is mapped. Those of form (4) assert that for each physical link e', it must have sufficient bandwidth to accommodate those of all the virtual links that are mapped to some physical path containing e'. Those of form (5) assure that when a virtual link e is mapped to a set of paths, for each ρ in the set, the resource of ρ allocated to e may not exceed the capacities of the physical links on ρ. Those of form (7) specify whether each virtual link is mapped to a set of paths or a single path in the SN.

VNM. We say that a VN request $(G_P, \mathcal{C})$ can be *mapped to* an SN G_S, denoted by $G_P \rhd_{\mathcal{C}} G_S$, if there exists a pair $((g_V, r_V), (g_E, r_E))$ of node and edge mappings from G_P to G_S such that all the constraints of $\mathcal{C}$ are satisfied, *i.e.,* the functions g_V and g_E satisfy all the inequalities in $\mathcal{C}$.

The VNM *problem* is to determine, given a VN request $(G_P, \mathcal{C})$ and an SN G_S, whether $G_P \rhd_{\mathcal{C}} G_S$.

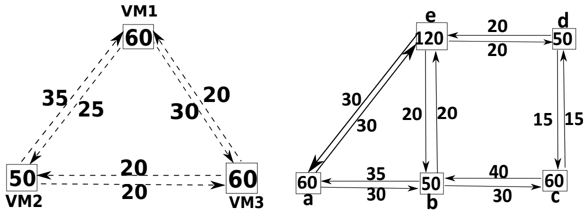
2.3. Case Study

As examples, below we examine VNM in various settings that we have seen in Section 1 (Table 1). All those VNM requirements can be expressed in this model, by treating VN request as a graph pattern and SN as a graph. These are summarized in Table 2 (✓and × indicate whether the corresponding constraints are needed or not, respectively). Below we illustrate a few cases.

Case 1: Virtual machine placement. VMP can be expressed as a VN request in which only node constraints are present. It is to find an injective mapping (g_V, r_V) from virtual nodes to substrate nodes (hence $|N| \leq 1$) that satisfies the node constraints, while imposing no constraints on edge mapping.

Case 2: Priority mapping. VNM_P can be captured as a VN request specified as $(G_P, \mathcal{C})$, where $\mathcal{C}$ consists of (a) node constraints of forms (1), (2) and (6), and (b) edge constraints of form (3) when op is $\leq$ and agg is max, form (5) when op is $\leq$ and agg is min, and form (6). It is to find an injective node mapping (g_V, r_V) and an edge mapping (g_E, r_E) such that for each virtual link e, $g_E(e)$ is a single path (hence $|Q(e)| = 1$). Moreover, it requires that the capacity of each virtual node v does not exceed the capacity of the substrate node that hosts v. When a virtual link e is mapped to a physical path ρ, the *bandwidth* of each edge on ρ is no less than that of e, *i.e.,* ρ suffices to serve any connection *individually*, including the one with the highest priority when ρ is allocated to the connection.

EXAMPLE 3. Consider the VN given in Fig. 1(a) and the SN of Fig. 1(b). Constraints for priority mapping can be defined as described above, using the node and edge labels (on bandwidths) in Fig. 1(a). There exists a priority mapping from the VN to the SN. Indeed, one can map VM_1, VM_2 and VM_3 to b, a and d, respectively,



(a) VN Request (b) SN

FIGURE 2. VN request and SN for case study

and map the virtual links to the shortest physical paths uniquely determined by the node mapping, *e.g.,* $(\mathsf{VM}_1, \mathsf{VM}_2)$ is mapped to (b, a). $\square$

<u>*Case 3: Single-path* VN *embedding.*</u> A $\mathsf{VNE_{SP}}$ request can be specified as $(G_P, \mathcal{C})$, where $\mathcal{C}$ consists of (a) node constraints of forms (1), (2) and (6), and (b) edge constraints of form (3) when op is $\leq$ and agg is sum, edge constraints of forms (4) and (5) when op is $\leq$ and agg is min, and constraints of form (7). It differs from $\mathsf{VNM_P}$ in that for each physical link e', it requires the *bandwidth* of e' to be no less than *the sum of* bandwidths of *all* those virtual links that are instantiated via e'. In contrast to VNM_P that aims to serve the connection with the highest priority at a time, $\mathsf{VNE_{SP}}$ requires that each physical link has enough capacity to serve *all connections* sharing the physical link at the same time.

Similarly, multi-path VN embedding (denoted by $\mathsf{VNE_{MP}}$) can be expressed as a VN request. It is the same as $\mathsf{VNE_{SP}}$ except that no constraints of form (7) are allowed, *i.e.,* a virtual link e can be mapped to *a set* $g_E(e)$ of physical paths, which, when taken together, provide sufficient bandwidth required by e.

When node constraints of form (6) are absent, *i.e.,* node sharing is allowed in $\mathsf{VNE_{SP}}$, *i.e.,* for single-path embedding with node sharing ($\mathsf{VNE_{SP(NS)}}$), a VN request is specified similarly. Here a substrate node u can host multiple virtual nodes (hence $|N(u)| \geq 0$) such that the sum of the capacities of all the virtual nodes does not exceed the capacity of u. Along the same lines, one can also specify multi-path VN embedding with node sharing ($\mathsf{VNE_{MP(NS)}}$).

Example 4. Consider the VN of Fig. 2(a), and the SN of Fig. 2(b). There exists a $\mathsf{VNE_{SP}}$ from the VN to the SN, by mapping $\mathsf{VM}_1, \mathsf{VM}_2, \mathsf{VM}_3$ to $\mathsf{a}, \mathsf{b}, \mathsf{e}$, respectively, and mapping the VN edges to the shortest paths in the SN determined by the node mapping. There is also a multi-path embedding $\mathsf{VNE_{MP}}$ from the VN to the SN, by mapping $\mathsf{VM}_1, \mathsf{VM}_2$ and VM_3 to a, c and e, respectively. For the virtual links, $(\mathsf{VM}_1, \mathsf{VM}_2)$ can be mapped to the physical path $(\mathsf{a}, \mathsf{b}, \mathsf{c})$, $(\mathsf{VM}_1, \mathsf{VM}_3)$ to (a, e), and $(\mathsf{VM}_3, \mathsf{VM}_2)$ to two paths $\rho_1 = (\mathsf{e}, \mathsf{b}, \mathsf{c})$ and $\rho_2 = (\mathsf{e}, \mathsf{d}, \mathsf{c})$ with $r_E((\mathsf{VM}_3, \mathsf{VM}_2), \rho_1) = 5$ and $r_E((\mathsf{VM}_3, \mathsf{VM}_2), \rho_2) = 15$; similarly for the other virtual links.

One can verify that the VN of Fig. 2(a) allows no more than one virtual node to be mapped to the same substrate node in Fig. 2(b). However, if we change the bandwidths of the edges connecting a and e in SN from 30 to $f_{V_S}(\mathsf{a}, \mathsf{e}) = 40$ and $f_{V_S}(\mathsf{e}, \mathsf{a}) = 50$, then there exists a mapping from the VN to the SN that supports node sharing. Indeed, in this setting, one can map both $\mathsf{VM}_1, \mathsf{VM}_2$ to e and map VM_3 to a; and map the virtual edges to the shortest physical paths determined by the node mapping; for instance, both $(\mathsf{VM}_1, \mathsf{VM}_3)$ and $(\mathsf{VM}_2, \mathsf{VM}_3)$ can be mapped to (e, a). $\square$

<u>*Case 4: Latency constrained mapping.*</u> A $\mathsf{VNM_L}$ request is expressed as $(G_P, \mathcal{C})$, where $\mathcal{C}$ consists of (a) node constraints of forms (1), (2) and (6), and (b) edge constraints of form (3) when op is $\geq$ and agg is min, of form (5) when op is $\geq$ and agg is sum, and of form (7). It is similar to $\mathsf{VNE_{SP}}$ except that when a virtual link e is mapped to a physical path ρ, it requires ρ to satisfy the *latency* requirement of e, *i.e.,* the sum of the latencies of the edges on ρ does not exceed that of e.

Example 5. One can verify that there is no latency mapping of the VN in Fig. 1(a) to the SN in Fig. 1(b).

However, if we change the constraints on the virtual links of the VN request to $(\mathsf{VM}_1, \mathsf{VM}_2) = 50$, $(\mathsf{VM}_2, \mathsf{VM}_1) = 55$, $(\mathsf{VM}_1, \mathsf{VM}_3) = (\mathsf{VM}_3, \mathsf{VM}_1) = 120$ and $(\mathsf{VM}_2, \mathsf{VM}_3) = (\mathsf{VM}_3, \mathsf{VM}_2) = 60$, then there is a mapping from the VN to the SN. We can map $\mathsf{VM}_1, \mathsf{VM}_2, \mathsf{VM}_3$ to $\mathsf{c}, \mathsf{b}, \mathsf{a}$, respectively, and map the edges to the shortest physical paths decided by the node mapping, *e.g.,* from $(\mathsf{VM}_1, \mathsf{VM}_3)$ to $(\mathsf{c}, \mathsf{b}, \mathsf{a})$. $\square$

3. COMPLEXITY AND APPROXIMATION

In this section we study fundamental issues associated with virtual network mapping. We first establish the complexity bounds of the VNM problem in various settings, from PTIME to NP-complete. We then introduce a cost metric for virtual network mapping, formulate optimization problems based on the function, and finally, give the complexity bounds and approximation hardness of the optimization problems.

3.1. The Complexity of VNM

We provide an upper bound for the VNM problem in the general setting, by showing it is in NP. We also show that the problem is in PTIME when only node constraints are present. However, when node sharing or edge constraints are imposed, it becomes NP-hard, even when both virtual and substrate networks are directed acyclic graphs (DAGs). That is, node sharing and edge constraints make our lives harder.

Theorem 3.1. *The* VNM *problem is*

(1) in NP regardless of what constraints are present;

(2) in PTIME when only node constraints are present, without node sharing, i.e., VMP *is in PTIME; however,*

(3) it becomes NP-complete when node sharing is requested, i.e., $\mathsf{VMP_{(NS)}}, \mathsf{VNM_{P(NS)}}, \mathsf{VNM_{L(NS)}}, \mathsf{VNE_{SP(NS)}}$

and $\mathsf{VNE}_{\mathsf{MP(NS)}}$ *are all NP-complete; and*

(4) it is NP-complete in the presence of edge constraints; i.e., $\mathsf{VNM}_{\mathsf{P}}, \mathsf{VNM}_{\mathsf{L}}, \mathsf{VNE}_{\mathsf{SP}}$ *and* $\mathsf{VNE}_{\mathsf{MP}}$ *are intractable.*

All the results hold when both VNs *and* SNs *are* DAGs*.*

Proof: (1) To show the upper bound, we give an NP algorithm for VNM in general case. Given a VN request $(G_P, \mathcal{C})$ and an SN G_S, the algorithm returns "Yes" if and only if $G_P \rhd_{\mathcal{C}} G_S$.

(i) Guess a node mapping function g_V and an edge mapping function g_E of VN on the SN.
(ii) Check whether there exist r_V and r_E such that (g_V, r_V) and (g_E, r_E) make node and edge mappings that satisfy the constraints in $\mathcal{C}$. If so, return "Yes".

The checking in step (ii) can be done in PTIME. Indeed, observe the following. (a) Both g_V and g_E are of size polynomial in $|G_P|$ and $|G_S|$. (b) The existence of r_V satisfying $\mathcal{C}$ can be checked in $O(|V_P|)$ time. (c) The existence of r_E satisfying $\mathcal{C}$ can be checked by formulating it as a linear (rational number) programming problem, where $r_E(e, \rho)$'s are variables for all paths ρ determined by g_E. For example, constraints of form (3) in the VN request in Section 2 with op as $=$ or $\leq$ and agg as min can be expressed as $f_{E_P}(e) \leq r_E(e, \rho)$, for all $e \in E_P$ and all $\rho \in g_E(e)$). As linear programming is in PTIME [35], so is the existence checking of r_E.

(2) We next propose a PTIME algorithm to check whether there exists a VMP from a VN request $(G_P, \mathcal{C})$ to an SN G_S with node constraints only and without node sharing, by reduction to the Maximum Bipartite Matching problem, which is in PTIME [36].

Given $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$ and $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$, the algorithm constructs a bipartite graph $G_B(V_L, V_R, E_B)$ as follows.

(i) Let V_L consist of $|V_P|$ nodes encoding V_P, and V_R consist of $|V_S|$ nodes encoding V_S.
(ii) For each pair of nodes $u \in V_P$ and $v \in V_S$, let u_L and v_R in V_L and V_R be the two nodes encoding u and v, respectively. We include (u_L, v_R) in E_B if $f_{V_P}(u) \leq f_{V_S}(v)$.

One can easily verify that G_B has a maximum bipartite match covering all nodes in V_L if and only if there exists a VMP from G_P to G_S. As the former can be checked in $O(|E_B|(|V_L| + |V_R|))$ time [36], VMP is in PTIME as well.

(3) To prove that all cases with node sharing are NP-complete, it suffices to show that $\mathsf{VMP}_{\mathsf{(NS)}}$ is NP-hard, for it is a special case of the other cases such as $\mathsf{VNM}_{\mathsf{P(NS)}}$, $\mathsf{VNM}_{\mathsf{L(NS)}}$, $\mathsf{VNE}_{\mathsf{SP(NS)}}$ and $\mathsf{VNE}_{\mathsf{MP(NS)}}$. We prove this by reduction from the Subset-Sum problem (SUBSUM). Given a set C of numbers $x_1, \ldots, x_k$ and a target number t, SUBSUM is to decide whether there exists a subset $C' \subseteq C$ such that $\sum_{x \in C'} x = t$. It is known that SUBSUM is NP-complete (cf. [37]).

Given an instance of SUBSUM, *i.e.,* $C = \{x_1, \ldots, x_k\}$ and t, we construct a VN request $G_P(V_P, E_P, f_{V_P}, f_{E_P})$ and an SN $G_S(V_S, E_S, f_{V_S}, f_{E_S})$, such that there is a $\mathsf{VMP}_{\mathsf{(NS)}}$ from G_P to G_S if and only if there exists $C' \subseteq C$ with $\sum_{x \in C'} x = t$. We give the reduction as follows.

(i) Let V_P of G_P be $\{v_1, \ldots, v_k\}$ and E_P be empty; moreover, for each $i \in [1, k]$, let $f_{V_P}(v_i) = x_i$. Intuitively, v_i of G_P is to encode x_i of C.
(ii) Let V_S of G_S consist of two nodes u and u' and E_S be empty; moreover, let $f_{V_S}(u) = t$ and $f_{V_S}(u') = (\sum_{x \in C} x) - t$. Intuitively, u is to encode t.

It is obvious that there exists a $\mathsf{VMP}_{\mathsf{(NS)}}$ from G_P to G_S if and only if there exists $C' \subseteq C$ with $\sum_{x \in C'} x = t$.

(4) In light of (1) above, we only need to show that $\mathsf{VNM}_{\mathsf{SP}}$, $\mathsf{VNM}_{\mathsf{MP}}$, $\mathsf{VNM}_{\mathsf{L}}$ and $\mathsf{VNM}_{\mathsf{P}}$ are NP-hard. First observe that $\mathsf{VNM}_{\mathsf{L}}$ is NP-hard since it subsumes the Subgraph Isomorphism problem, which is NP-complete (cf. [37]), as a special case where the latency requirements on virtual links and latency on physical links are all the same, *e.g.,* 1.

Below we first show that $\mathsf{VNM}_{\mathsf{SP}}$ and $\mathsf{VNM}_{\mathsf{MP}}$ are NP-hard by reduction from the SUBSUM problem. We then show that $\mathsf{VNM}_{\mathsf{P}}$ is NP-hard by reduction from the X3C problem, which is NP-complete [37].

(a) We first show that both $\mathsf{VNM}_{\mathsf{SP}}$ and $\mathsf{VNM}_{\mathsf{MP}}$ are NP-hard by reduction from SUBSUM (recall the statement of SUBSUM from the proof of (3)). Given an instance C and t of SUBSUM, we construct a VN $G_P(V_P, E_P, f_{V_P}, f_{E_P})$ and an SN $G_S(V_S, E_S, f_{V_S}, f_{E_S})$ such that there exists a $\mathsf{VNM}_{\mathsf{SP}}$ (resp. $\mathsf{VNM}_{\mathsf{MP}}$) from G_P to G_S if and only if there exists $C' \subseteq C$ with $\sum_{x \in C'} x = t$. We give the reduction as follows.

(i) Let V_P of G_P be $\{v_1, \ldots, v_k, v_o\}$ and E_P be $\{(v_o, v_1), \ldots, (v_o, v_k)\}$; $f_{V_P}(v_i) = 2$ for each $i \in [1, k]$ and $f_{V_P}(v_o) = 3$; moreover, let $f_{E_P}(v_o, v_i) = x_i$ for each $i \in [1, k]$. Intuitively, G_P is to encode C.
(ii) Let V_S of G_S be $\{u_1^l, u_1^r, \ldots, u_k^l, u_k^r, u_o, u_l, u_r\}$ and E_S be $\{(u_o, u_l), (u_o, u_r), (u_l, u_1^l), \ldots, (u_l, u_k^l), (u_r, u_1^r), \ldots, (u_r, u_k^r)\}$; let $f_{V_S}(u_o) = 3$, $f_{V_S}(u_l) = f_{V_S}(u_r) = 1$, $f_{V_S}(u_i^l) = f_{V_S}(u_i^r) = 2$ for all $i \in [1, k]$; in addition, let $f_{E_S}(u_o, u_l) = t$, $f_{E_S}(u_o, u_r) = \sum_{x \in C} x - t$, $f_{E_S}(u_l, u_i^l) = f_{E_S}(u_r, u_i^r) = \sum_{x \in C} x$. Here edge (u_o, u_l) is to encode t, and (u_o, u_r) is to encode $(\sum_{x \in C} x) - t$, and f_{V_S} and f_{V_P} together ensure that v_o of G_P must be mapped to u_o of G_S, and v_i of G_P must be mapped to u_j^l or u_j^r of G_S for some $j \in [1, m]$. These ensure $|g_V(v_o, v_i)| = 1$ for all $i \in [1, m]$. As a result, $\mathsf{VNM}_{\mathsf{SP}}$ and $\mathsf{VNM}_{\mathsf{MP}}$ coincide for G_P and G_S.

Observe that both G_P and G_S are DAGs. We next show that there exists a subset $C' \subseteq C$ such that

$\sum_{x\in C'} x = t$ if and only if there exists a $\mathsf{VNM_{SP}}$ (and thus $\mathsf{VNM_{MP}}$) from G_P to G_S.

(i) Assume first that there exists a subset $C' \subseteq C$ with $\sum_{x\in C'} x = t$. We show that there exists a $\mathsf{VNM_{SP}}$ from G_P to G_S. For each node v_i ($i \in [1, k]$) in G_P, g_V maps v_i to u_i^l if x_i is in C', and to u_i^r otherwise; moreover, $g_V(v_o) = u_o$. For each edge (v_o, v_i) in G_P, g_E maps it to the unique path that connects u_o and $g_V(v_i)$ in G_S. Let $r_V(v_o, u_o) = 3$, $r_V(v_i, g_V(v_i)) = 2$, and $r_E((v_o, v_i), g_E(v_o, v_i)) = x_i$. One can verify that (g_V, r_V) and (g_E, r_E) indeed form a $\mathsf{VNM_{SP}}$ ($\mathsf{VNM_{MP}}$) from G_P to G_S.

(ii) Conversely, assume that there exists a $\mathsf{VNM_{SP}}$ (and thus $\mathsf{VNM_{MP}}$) from G_P to G_S. We show that there exists a subset $C' \subseteq C$ with $\sum_{x\in C'} x = t$. Note that the node mapping g_V is fixed as discussed above, by the definition of f_{E_P} and f_{E_S}. In light of this, one can verify that $C' = \{x_i \mid g_V(v_i) = u_j^l, j \in [1, k]\}$ is a subset of C and moreover, $\sum_{x\in C'} x = f_{E_S}(u_o, u_l) = t$.

(b) We next show that $\mathsf{VNM_P}$ is NP-hard by reduction from the $\mathsf{X3C}$ problem. Given a finite set $S = \{x_1, x_2, \ldots, x_{3q}\}$, and a collection $C = \{C_1, C_2, \ldots, C_n\}$ of 3-element subsets of S, in which $C_i = \{x_{i_1 j_1}, x_{i_2 j_2}, x_{i_3 j_3}\}$ ($i_1, i_2, i_3 \in [1, q]$, $j_1, j_2, j_3 \in [1, 3]$), $\mathsf{X3C}$ is to determine whether C contains an *exact cover* for S, *i.e.,* whether there exists a subset $C' \subseteq C$ such that every element x_i of S occurs in exactly one member of C'. It is known that $\mathsf{X3C}$ is NP-complete (cf. [37]).

Given S and C of $\mathsf{X3C}$, we construct a VN $G_P(V_P, E_P, f_{V_P}, f_{E_P})$ and an SN $G_S(V_S, E_S, f_{V_S}, f_{E_S})$ such that C contains an exact cover for S if and only if there exists a $\mathsf{VNM_P}$ from G_P to G_S. Below we give the reduction.

(i) Let V_P consist of $4q$ nodes $\{v_{11}, v_{12}, v_{13}, \ldots, v_{q1}, v_{q2}, v_{q3}, v_1^C, \ldots, v_q^C\}$, and for any $i \in [1, q]$, $j \in [1, 3]$, let $f_{V_P}(v_{ij}) = 3i + (j-1)$ and $f_{V_P}(v_i^C) = 0.5$. Intuitively, nodes $\{v_{11}, v_{12}, v_{13}, \ldots, v_{q1}, v_{q2}, v_{q3}\}$ and nodes $\{v_1^C, \ldots, v_q^C\}$ are to encode S and to encode an exact cover of S, respectively.
We define E_P such that it consists of $3q$ edges, and for each $i \in [1, q]$, (v_i^C, v_{i1}), (v_i^C, v_{i2}) and (v_i^C, v_{i3}) are in E_P; in addition, for any $e \in E_P$, let $f_{E_P}(e) = 1$.

(ii) Let V_S consist of $|S| + |C| = 3q + n$ nodes $\{u_{11}, u_{12}, u_{13}, \ldots, u_{q1}, u_{q2}, u_{q3}, u_1^C, \ldots, u_n^C\}$, and for each $i \in [1, q]$, $j \in [1, 3]$, let $f_{V_S}(u_{ij}) = 3i + (j-1)$ and $f_{V_S}(u_i^C) = 0.5$. Intuitively, nodes $u_{11}, \ldots, u_{q3}$ are to encode S, while $u_1^C, \ldots, u_n^C$ are to encode C, respectively.
We define E_S such that it consists of $3n$ edges, and for each $C_i = \{x_{i_1 j_1}, x_{i_2 j_2}, x_{i_3 j_3}\}$ ($i \in [1, n]$), edges $(u_i^C, u_{i_1 j_1})$, $(u_i^C, u_{i_2 j_2})$ and $(u_i^C, u_{i_3 j_3})$ are included in E_S. In addition, for each $i \in [1, q]$, $j \in [1, 3]$, let $f_{V_S}(u_{ij}) = 3i + (j-1)$, and $f_{V_S}(v_i^C) = 0.5$. For each $e \in E_S$, let $f_{E_S}(e) = 0.5$.

Observe that both G_P and G_S are DAGs such that each v_i^C in V_P can be only mapped to one of $u_1^C, \ldots, u_n^C$ in G_S, and each v_{ik} in V_P can only be mapped to u_{ik} in V_S ($i \in [1, q]$ and $k \in \{1, 2, 3\}$), by the definition of f_{V_P} and f_{V_S}. Indeed, G_P encodes an exact cover in C since S and G_S encodes S and C, respectively.

We next show that there exists a priority mapping (g_V, r_V, g_E, r_E) if and only if there exists an exact cover in C for S.

(i) Assume first that there is a priority mapping (g_V, r_V, g_E, r_E) from G_P to G_S. Then there exists an exact cover $C' \subseteq C$ for S. More specifically, C' consists of the following: for each v_i^C in G_P with $g_V(v_i^C) = u_j^C$, C_j is included in C'. Then $C' \subseteq C$ is an exact cover of S. Indeed, suppose that C' is not an exact cover. Since each node v_{ik} in V_P can only be mapped to u_{ik} in V_S, we have that $|\{g_V(v_{ik}) \mid i \in \{1, 2, \cdots, q\}, k \in \{1, 2, 3\}\}| < |\{u_{ik} \mid \forall i \in \{1, 2, \cdots, q\}, k \in \{1, 2, 3\}\}|$, a contradiction to the definition of the injection g_V.

(ii) Conversely, assume that there exists an exact cover $C' \subseteq C$ for S. Let $C' = \{C_{j_1}, C_{j_2}, \ldots, C_{j_q}\}$, $j_1, \ldots, j_q \in [1, n]$. Consider the following mapping (g_V, r_V, g_E, r_E) from G_P to G_S. For each $C_{j_i} \in C'$, $g_V(v_{j_i}^C) = u_{j_i}^C$, $g_V(v_{j_i 1})$, $g_v(v_{j_i 2})$ and $g_V(v_{j_i 3})$ are the three nodes in G_S that are connected to $u_{j_i}^C$; g_E is uniquely determined by g_V; $r_V(v_{ik}, g_V(v_{ik})) = 1$, $r_V(v_i^C, g_V(v_i^C)) = 2$, for $i \in [1, q]$ and $k \in [1, 3]$; moreover, $r_E(e, \rho) = 1$ for $\rho = g_E(e)$. By the definition, (g_V, r_V, g_E, r_E) is a VNM_P mapping from G_P to G_S.

This completes the proof of Theorem 3.1. Note that G_P and G_S construed in the reductions of (3) and (4) above are all DAGs. As a consequence, all the results hold even when both VNs and SNs are DAGs. □

3.2. Approximation of Optimization Problems

In practice, we often want to find a VNM mapping with "the lowest cost". This highlights the need for introducing a function to measure the cost of a mapping and for studying its corresponding optimization problems.

A Cost Function. Consider an SN $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$, and a VN request $(G_P, \mathcal{C})$, where $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$. Assume a positive number associated with all nodes v and links e in G_S, denoted by $w(v)$ and $w(e)$, respectively, that indicates the price of the resources in the SN. Given a pair $((g_V, r_V), (g_E, r_E))$ of node and edge mappings from $(G_P, \mathcal{C})$ to G_S, its *cost* $c((g_V, r_V), (g_E, r_E))$ is defined as

$$c((g_V, r_V), (g_E, r_E)) = \sum_{v\in V_P} h_V(g_V, r_V, v) \cdot w(g_V(v)) + \sum_{e'\in E_S} h_E(g_E, r_E, e') \cdot w(e'),$$

where (1) $h_V(g_V, r_V, v) = r_V(v, g_V(v))/f_{V_S}(g_V(v))$, (2) $h_E(g_E, r_V, e') = \sum_{e\in E_P, \rho\in g_E(e), e'\in\rho} r_E(e, \rho)/f_{E_S}(e')$

when the resource of physical links is bandwidth, and (3) when latency is concerned, $h_E(g_E, r_V, e')$ is 1 if there exists $e \in E_P$ such that $e' \in g_E(e)$, and 0 otherwise.

Intuitively, h_V indicates that the more CPU resource is allocated, the higher the cost it incurs; similarly for h_E when bandwidth is concerned. When latency is considered, the cost of the edge mapping is determined only by g_E, whereas the resource allocation function r_E is irrelevant.

The cost function is motivated by economic models of network virtualization [38]. It is justified by Web hosting and cloud storage [39], which mainly sell CPU power or storage services of nodes. It is also motivated by virtual network mapping, which sells bandwidth of links [16]. In addition, it is to serve cloud provision in virtualized data center networks [40], for which dynamic routing strategy (latency) is critical while routing congestion (bandwidth allocation) is often considered secondary.

Minimum Cost Mapping. We now introduce optimization problems for virtual network mapping.

The *minimum cost mapping* problem is to find, given a VN request and an SN, a mapping $((g_V, r_V), (g_E, r_E))$ from the VN to the SN such that its cost based on the function above is minimum among all such mappings.

The decision problem for minimum cost mapping is to decide, given a number (bound) K, a VN request and an SN, whether there is a mapping $((g_V, r_V), (g_E, r_E))$ from the VN to the SN such that its cost is no larger than K.

We shall refer to the minimum cost mapping problem and its decision problem interchangeably in the sequel.

EXAMPLE 6. Consider the SN $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$ shown in Fig. 2(b), and the VN depicted in Fig. 2(a). Assume that the cost function $c()$ is set to be the same as f_{V_S} for the nodes and as f_{E_S} for the links in the SN, *i.e.,* the cost of a substrate node is the same as its CPU capacity, and the cost of a physical link is the same as its bandwidth capacity or latency.

Consider the multi-path embedding from the VN to the SN described in Example 4. Then the cost of the node mapping is $\frac{60}{60} \times 60 + \frac{50}{60} \times 60 + \frac{60}{120} \times 120 = 170$, while the cost of its edge mapping is $(\frac{25}{30} \times 30) \times 2 + (\frac{35}{40} \times 40 + \frac{35}{35} \times 35) + (\frac{20}{30} \times 30 + \frac{30}{30} \times 30) + ((\frac{5}{20} \times 20 + \frac{5}{30} \times 30) + (\frac{5}{40} \times 40 + \frac{5}{20} \times 20)) + ((\frac{15}{20} \times 20 + \frac{15}{15} \times 15) + (\frac{15}{15} \times 15 + \frac{15}{20} \times 20)) = 250$. Putting these together, the total cost is 420.

Consider the latency mapping given in Example 5. We can compute its cost along the same lines as above, except that the cost of each edge (h_E) is either 1 or 0. One can easily verify that the cost of this mapping is $(66 + 20 + 30) + (40 + 45 + 50 + 50) = 301$. □

Complexity and Approximation. We next study the minimum cost mapping problem for all the cases given in Table 1.

Having seen Theorem 3.1, it is not surprising that the optimization problem is intractable in most cases. This motivates us to study efficient approximation algorithms with performance guarantees. Unfortunately, the problem is hard to approximate in most cases. The results below tell us that when node sharing is requested or edge constraints are present, minimum cost mapping is beyond reach in practice for approximation.

THEOREM 3.2. *The minimum cost mapping problem is*

(1) in PTIME for VMP *without node sharing; however, when node sharing is requested,* i.e., *for* $\mathsf{VMP}_{(\mathsf{NS})}$, *it becomes NP-complete and is APX-hard even there always exists a* $\mathsf{VMP}_{(\mathsf{NS})}$ *mapping;*

(2) NP-complete and NPO-complete with edge constraints, i.e., VNM_P, $\mathsf{VNE}_{\mathsf{SP}}$, $\mathsf{VNE}_{\mathsf{MP}}$, $\mathsf{VNM}_{\mathsf{P(NS)}}$, VNM_L, $\mathsf{VNE}_{\mathsf{SP(NS)}}$, $\mathsf{VNE}_{\mathsf{MP(NS)}}$, $\mathsf{VNM}_{\mathsf{L(NS)}}$ *are all NPO-complete; and*

(3) APX-hard when there exists a unique node mapping in the presence of edge constraints. In particular, $\mathsf{VNM}_{\mathsf{P}}$ *does not admit* $c\ln(|V_P|)$*-approximation for some constant* $c > 0$*, unless P = NP.*

All the lower bounds hold when both VNs *and* SNs *are* DAGs.

Here NPO is the *class* of all NP optimization problems and APX is the *class* of problems that allow PTIME approximation algorithms with a constant approximation ratio (cf. [22]). (cf. [22]). An NPO-complete problem is NP-hard to optimize, and is among the hardest optimization problems.

Proof: (1) We first prove that VMP without node sharing is in PTIME by giving an cubic-time algorithm. Given a VN G_P and an SN G_S, the algorithm finds minimum VMP from G_P to G_S without node sharing by reducing the problem to the MINIMUM LINEAR ASSIGNMENT problem (MLA). MLA is to find a bijective assignment function from m objects $x_1, \ldots, x_m$ to another m objects $y_1, \ldots, y_m$ while minimizing the total assignment cost $\sum_{i,j} c(x_i, y_j)$. It can be solved in $O(m^3)$ time (cf. [41]).

Given G_P and G_S, the algorithm works as follows.

(i) Construct a set X of $|V_S|$ nodes such that for each node $v \in V_P$, there is an object x_v in X, and another $|V_S| - |V_P|$ dummy objects $x'_1, \ldots, x'_{|V_S|-|V_P|}$.
(ii) Construct a set Y of $|V_S|$ nodes such that for each node $u \in V_S$, there is an object y_u in Y.
(iii) For each $v \in V_P$ and $u \in V_S$, if $f_{V_P}(v) \leq f_{V_S}(u)$, then the assignment cost $c(x_v, y_u) = \frac{f_{V_P}(v)}{f_{V_S}(u)} w(u)$, and for any other pairs $(x, y) \in X \times Y$, $c(x, y) = M$, where $M = \sum_{u \in V_S}(w(u))$.
(iv) Assign objects in X to objects in Y by invoking an algorithm for MLA. If the total assignment cost is no less than $M(|V_S| - |V_P| + 1)$, then it returns "No"

since there is no VMP from G_P to G_S; otherwise it returns (g_V, r_V) as follows: for each $v \in V_P$, $g_V(v)$ is u if x_v is assigned to y_u, and $r_V(v,u) = f_{V_P}(v)$.

Observe that MLA is a generalization of the VMP problem. This ensures the correctness of the algorithm.

We next show that the problem becomes NP-complete and APX-hard to approximate when node sharing is requested, even for $\mathsf{VMP}_{(\mathsf{NS})}$ that always has valid mappings. This follows from the fact that the Generalized Minimum Bin Packing problem is a special case of $\mathsf{VMP}_{(\mathsf{NS})}$, and that the former is APX-hard and always has a feasible solution (cf. [42]).

(2) The NP-completeness follows from Theorem 3.1(4). We show that it is NPO-complete, *i.e.,* it is NPO-hard to approximate, by an AP-reduction from the Minimum Weighted 3SAT problem (MW3SAT). It is known that MW3SAT is NP-hard to approximate (cf. [22]). An instance of MW3SAT is a CNF formula $\phi = C_1 \wedge \cdots \wedge C_m$ defined over variables $x_1, \ldots, x_n$ with non-negative weights $w(x_1), \ldots, w(x_n)$, where each clause $C_j (j \in [1,m])$ is a Boolean formula of form $\ell_1^j \vee \ell_2^j \vee \ell_3^j$, in which each literal $\ell_i^j (i \in [1,3])$ is either x_k or $\bar{x}_k$ for $k \in [1,n]$. Given ϕ, MW3SAT is to find the minimum weight of truth assignment μ to the variables that satisfies ϕ, where the weight of a truth assignment μ is defined as $\sum_{i=1}^{n} w(x_i) \cdot \mu(x_i)$, and the Boolean values True and False of $\mu(x_i)$ are treated as 1 and 0, respectively.

We next present an AP-reduction from MW3SAT to VNM with edge constraints (a VN G_P and an SN G_S). We use I_P and $SOL_P(x)$ to denote instances and feasible solutions to an instance x of an optimization problem P, respectively, and use $R_P(x,s)$ to denote the relative approximation factor of solution s to instance x of P. An AP-reduction consists of two functions Γ and Λ, and a positive constant $\alpha \geq 1$ that satisfy the following constraints [22].

(i) For any instance $x \in I_{\mathsf{MW3SAT}}$ and any rational $r > 1$, $\Gamma(x,r) \in I_{\mathsf{VNM}}$.
(ii) For any instance $x \in I_{\mathsf{MW3SAT}}$ and any rational $r > 1$, if $SOL_{\mathsf{MW3SAT}}(x) \neq \emptyset$, then $SOL_{\mathsf{VNM}}(\Gamma(x,r)) \neq \emptyset$.
(iii) For any instance $x \in I_{\mathsf{MW3SAT}}$, any rational $r > 1$ and any $y \in SOL_{\mathsf{VNM}}(\Gamma(x,r))$, $\Lambda(x,y,r) \in SOL_{\mathsf{MW3SAT}}(x)$.
(iv) For any fixed rational r, functions Γ and Λ are computable in polynomial time.
(v) For any instance $x \in I_{\mathsf{MW3SAT}}$, any rational $r > 1$ and any $y \in SOL_{\mathsf{VMP}}(\Gamma(x,r))$, if $R_{\mathsf{VMP}}(\Gamma(x,r),y) \leq r$, then $R_{\mathsf{MW3SAT}}(x, \Lambda(x,y,r)) \leq 1 + \alpha(r-1)$.

We give the detailed reduction as follows.

Function Γ. Given an instance of MW3SAT described above, function Γ constructs $G_P(V_P, E_P, f_{V_P}, f_{E_P})$ and $G_S(V_S, E_S, f_{V_S}, f_{E_S})$ as follows.

(a) Construction of G_P. We define G_P such that

(i) the node set V_P of G_P consists of $2m + 2n$ nodes $X_1^P, \ldots, X_n^P, C_1^P, \ldots, C_m^P, S_1^P, \ldots, S_n^P, T_1^P, \ldots, T_m^P$; intuitively, $X_i^P (i \in [1,n])$ is to encode variable x_i, and $C_j^P (j \in [1,m])$ is to encode clause C_j;
(ii) for each variable x_i, if x_i or $\overline{x_i}$ occurs in clause C_j of ϕ, then edge (X_i^P, C_j^P) is included in E_P; for each $i \in [1,n]$, (S_i^P, X_i^P) is in E_P; moreover, for each $j \in [1,m]$, (C_j^P, T_j^P) is in E_P;
(iii) let $f_{V_P}(X_i^P) = 1$ and $f_{V_P}(S_i^P) = i + 2$ $(i \in [1,n])$; $f_{V_P}(C_j^P) = 2$ and $f_{V_P}(T_j^P) = j + 2 + n$ $(j \in [1,m])$; and
(iv) for each $e \in E_P$, let $f_{E_P}(e) = 1$.

(b) Construction of G_S. We define G_S such that

(i) the node set V_S of G_S contains $3n + 8m$ nodes: for each variable $x_i (i \in [1,n])$, we include three nodes X_{Ti}^S, X_{Fi}^S and S_i^S in V_S; for each clause $C_j (j \in [1,m])$ of ϕ, we add 8 nodes $0_j, \ldots, 7_j$, and T_j^S to V_S. Intuitively, nodes X_{Ti}^S and X_{Fi}^S are to encode truth values of variable x_i; nodes $0_j, \ldots, 7_j$ encode all possible truth assignments (three bits 0/1 digits, *e.g.,* 2_j encodes (*false*, *true*, *false*)) to variables in C_j.
(ii) For each clause $C_j = \ell_1^j \vee \ell_2^j \vee \ell_3^j$ in ϕ, if a truth assignment to variables in ℓ_1^j, ℓ_2^j and ℓ_3^j makes C_j true (suppose that node $p_j (p \in [0,7])$ in V_S encodes this truth assignment), then we add edges from the three corresponding nodes in V_S (encoding truth values of variables) to p_j in E_S.
For example, consider $C_j = x_1 \vee \overline{x_2} \vee x_3$, since (*true*, *false*, *false*) is an truth assignment to (x_1, x_2, x_3), edges $(X_{T1}^S, 4_j)$, $(X_{F2}^S, 4_j)$, $(X_{F3}^S, 4_j)$ are included in E_S.
In addition, for each $i \in [1,n]$, two edges (S_i^S, X_{Ti}^S) and (S_i^S, X_{Fi}^S) are included in E_S. Furthermore, let $f_{V_S}(X_{Ti}^S) = f_{V_S}(X_{Fi}^S) = 1$, and $f_{V_S}(S_i^S) = i + 2$.
For each $j \in [1,m], p \in [0,7]$, $(0_j, T_j^S), \ldots, (7_j, T_j^S)$ are also included in E_S. Moreover, let $f_{V_S}(p_j) = 2$ and $f_{V_S}(T_j^S) = j + 2 + n$.
For each $e \in E_S$, let $f_{E_S}(e) = 1$.
By the definition of $f_{V_S}(T_j^S)$, $f_{V_P}(T_j^P)$, $f_{V_S}(S_i^S)$ and $f_{V_P}(S_i^P)$, we know that there exists a unique node mapping from $T_1^P, \ldots, T_m^P$ and $S_1^P, \ldots, S_m^P$ in G_P to $T_1^S, \ldots, T_m^S$ and $S_1^S, \ldots, S_m^S$ that satisfies node constraints, *i.e.,* mapping T_j^P and S_i^P to T_j^S and S_i^S, respectively, for each $i \in [1,n]$ and $j \in [1,m]$.
(iii) For each $u \in V_S$, let $w(v) = 0$.
(iv) For each $(X_{Ti}^S, p_j) (i \in [1,n], j \in [1,m], p \in [0,7])$ in E_S, we let the weight $w(X_{Ti}^S, p_j) = w(x_i)$. For any other $e \in E_P$, let $w(e) = 0$.

Observe the following. (i) By the definition of $f_{V_S}(S_i^S)$, $f_{V_S}(T_j^S)$, $f_{V_P}(S_i^P)$ and $f_{V_P}(T_j^P)$, for each $i \in [1,n]$, S_i^P in G_P has to be mapped to S_i^S; and for each $j \in [1,m]$, T_j^P in G_P has to be mapped to T_j^S, no matter whether node sharing is allowed or not. (ii) Because of (i), each node C_j^P in G_P has to be mapped to one of 0_j, ..., 7_j, and similarly, each node X_i^P has to be mapped to either X_{Ti}^S or X_{Fi}^S, but not both. (iii) Because of (i) and (ii), $\mathsf{VNM_P}$, $\mathsf{VNE_{SP}}$, $\mathsf{VNE_{MP}}$, $\mathsf{VNM_L}$, $\mathsf{VNM_{P(NS)}}$, $\mathsf{VNE_{SP(NS)}}$, $\mathsf{VNE_{MP(NS)}}$ and $\mathsf{VNM_{L(NS)}}$ from G_P to G_S coincide. Hence below we only discuss $\mathsf{VNM_P}$.

<u>Function Λ.</u> We next present function Λ that converts VNM mappings from G_P to G_S constructed above back to a truth assignment μ to ϕ. Given a VNM mapping (g_V, r_V, g_E, r_E), function Λ produces a truth assignment μ for ϕ such that for each X_i^P (for $i \in [1, n]$), $\mu(x_i) = true$ if $g_V(X_i^P) = X_{Ti}^S$, and $\mu(x_i) = false$ otherwise.

<u>Constant α.</u> We simply let $\alpha = 1$. This completes the construction.

Below we verify that $(\Gamma, \Lambda, \alpha)$ is an AP-reduction from WM3SAT to the minimum VNM problem with edge constraints (*e.g.,* $\mathsf{VNM_P}$). Observe the following.

(i) It is obvious that the functions Γ and Λ are both computable in polynomial time.
(ii) For any instance ϕ of MW3SAT, $\Gamma(\phi)$ is an instance of the minimum cost mapping problem for $\mathsf{VNM_P}$.
(iii) Formula ϕ has a truth assignment μ if and only if there exists a $\mathsf{VNM_P}$ from G_P to G_S.
More specifically, suppose first that ϕ has a truth assignment μ such that $\mu(\phi) = true$. Then $\mu(C_j) = true$ for each $j \in [1, m]$. Thus there exists g_V such that $g_V(X_1^P)$, ..., $g_V(X_n^P)$ together ensure that, for each $j \in [1, m]$, at least one of 0_j, ..., 7_j is mapped by C_j^P. That is, there exists a node mapping g_V and an edge mapping that form a $\mathsf{VNM_P}$ from G_P to G_S. Conversely, suppose that there exists a $\mathsf{VNM_P}$ from G_P to G_S. By the construction above, the node mapping encodes a truth assignment μ to variables in ϕ. Moreover, since each C_j^P has to be mapped to one of 0_j, ..., 7_j, clause C_j is assured to be satisfied by the truth assignment. Therefore, $\mu(\phi) = true$.
(iv) Similar to (iii), it is easy to see that Λ transfers node mappings of the $\mathsf{VNM_P}$ from G_P to G_S into a truth assignment to variables of ϕ, as node mapping g_V always maps X_i^P to either X_{Ti}^S or X_{Fi}^S, but not both.
(v) By the construction above, one can verify that for any instance ϕ of MW3SAT, if μ is the optimum solution for ϕ (*i.e.,* minimum weight truth assignment), then $\sum_{i\in[1,n]} x_i w(x_i)$ equals the minimum weight of $\mathsf{VNM_P}$ from G_P to G_S. In addition, for any $\mathsf{VNM_P}$ mapping from G_P to G_S, its cost is equal to $\sum_{i\in[1,n]} x_i w(x_i)$.

That is, for any instance ϕ of MW3SAT, for any feasible mapping s of $f(\phi)$, $R_{\mathsf{VNM}}(f(\phi), s) = R_{\mathsf{MW3SAT}}(\phi, g(s))$. Thus $(\Gamma, \Lambda, \alpha)$ is an AP-reduction from MW3SAT to the minimum cost mapping problem for $\mathsf{VNM_P}$, and hence for all the other VNM cases with edge constraints.

TABLE 3. Summary of complexity results

Problems	Complexity	Approximation
VMP	PTIME	– –
$\mathsf{VMP_{(NS)}}$	NP-complete	APX-hard
$\mathsf{VNM_P}$, $\mathsf{VNM_{P(NS)}}$	NP-complete	NPO-complete
$\mathsf{VNM_L}$, $\mathsf{VNM_{L(NS)}}$	NP-complete	NPO-complete
$\mathsf{VNE_{SP}}$, $\mathsf{VNE_{SP(NS)}}$	NP-complete	NPO-complete
$\mathsf{VNE_{MP}}$, $\mathsf{VNE_{MP(NS)}}$	NP-complete	NPO-complete

Observe that the G_P and G_S construed in the reductions above are DAGs. Hence, all results hold even when both VNs and SNs are DAGs.

(3) We only need to show that $\mathsf{VNM_P}$ does not have a PTIME $\ln(|V_P|)$-approximation algorithm even when there exists a unique node mapping. Indeed, $\mathsf{VNM_P}$ contains the DIRECTED STEINER TREE problem (DST) as a special case, where the nodes in VN correspond to terminals of DST. Given a directed weighted graph $G(V, E)$, a specified root $r \in V$, and a set of terminals $T \subseteq V$, DST is to find the minimum cost arborescence that is rooted at r and spans all the nodes in T. Since DST is not approximable within $c \ln |T|$ for some $c > 0$ even when G is a DAG (cf. [22]), $\mathsf{VNM_P}$ is not approximable within $O(c \ln |V_P|)$. This verifies that minimum VNM is APX-hard even when there exists fixed node mapping, in the presence of edge constraints.

This completes the proof of Theorem 3.2. The lower bounds remain intact when VNs and SNs are DAGs, since all our reductions given above use DAGs only. □

We summarize the complexity results in Table 3.

4. COMPUTING MINIMUM COST VNM

Theorem 3.2 tells us that any efficient algorithms for computing minimum cost VNM are necessarily heuristic. We next develop a greedy algorithm to find minimum cost priority mappings ($\mathsf{VNM_P}$), with node sharing or not. Given a VN request $(G_P, \mathcal{C})$, an SN G_S, and a cost function $c()$, the algorithm finds a mapping $((g_V, r_V), (g_E, r_E))$ from G_P to G_S such that it satisfies the node and edge constraints in $\mathcal{C}$ and moreover, the cost $c((g_V, r_V), (g_E, r_E))$ is minimized, if such a mapping exists. To the best of our knowledge, this is the first algorithm for computing $\mathsf{VNM_P}$.

Previous algorithms for computing VNM (*e.g.,* [15]) typically consists of two stages. It first finds a candidate node mapping (g_V, r_V), and then checks whether it is *valid*, *i.e.,* whether it admits a corresponding edge mapping (g_E, r_E); if so, it computes (g_E, r_E) by traversing the entire SN. If the (g_V, r_V) is not valid, the entire process has to start all over again. Hence a mapping is often found only after repeated trials and failures. This hinders the scalability of the algorithms.

In contrast, we unify the processes of computing node mappings and edge mappings. During the process of building a node mapping, we check whether the (partial) mapping found so far is valid, *i.e.,* we do not wait for a node mapping to be completed before starting the validation process. To efficiently validate a partial node mapping and build its corresponding edge mapping, we use an auxiliary graph structure for SN G_S. In addition, we minimize the VN pattern G_P. Obviously, the smaller G_P is, the less costly the mapping process is.

In this section, we first present an auxiliary structure for SN (Section 4.1) and then develop an algorithm for minimizing VN patterns (Section 4.2). Finally, leveraging the auxiliary structure and minimization, we present our algorithm for minimum cost $\mathsf{VNM_P}$ (Section 4.3).

4.1. Auxiliary Graphs: Unifying Mappings

Given a weighted directed graph $G(V,\ E,\ f_V,\ f_E)$, its *auxiliary graph* $G_{aux}(V_a, E_a, f_{V_a}, f_{E_a}, P_{E_a})$ is a weighted directed graph such that (1) $V_a = V$, $f_{V_a} = f_V$, (2) edge $(u, v) \in E_a$ if and only if there exists a path from u to v in G, (3) $f_{E_a}(u, v) = \max\{\min\{\rho\} \mid \rho$ is a path from u to v in $G\}$, where $\min\{\rho\} = \min\{f_E(e) \mid e$ is an edge on $\rho\}$ in G, and (4) $P_{E_a}(u, v)$ is a path ρ in G with $\min\{\rho\} = f_{E_a}(u, v)$.

One can verify the following for priority mappings, which justifies the need for auxiliary graphs.

PROPOSITION 4.1. *Consider a* VN $G_P(V_P, E_P, f_{V_P}, f_{E_P})$ *and an* SN $G_S(V_S, E_S, f_{V_S}, f_{E_S})$. *For any node mapping* (g_V, r_V), *with the auxiliary graph of* G_S, *it takes* $O(|E_P|)$ *time to determine whether* (g_V, r_V) *is valid and to compute a corresponding edge mapping.*

Proof: Observe the following. (1) The auxiliary graph G_{aux} records the maximum bandwidth between any two nodes in SN. (2) For any node mapping (g_V, r_V), we can determine whether it admits an edge mapping by querying the maximum bandwidth between matched nodes from G_{aux}. This takes $O(|E_P|)$ time as there are at most $|E_P|$ node pairs that require edge mapping checks. From these the proposition follows. □

Algorithm. We next present an algorithm, referred to as compAuxGraph, for building auxiliary graphs. Given a weighted directed graph G, the algorithm returns the auxiliary graph G_{aux} of G, as shown in Fig. 3.

The algorithm starts from an empty G_{aux} (line 1) and iteratively adds nodes to G_{aux} by calling procedure updateAuxGraph (lines 2-3). As will be seen shortly, it may add an edge (u, u') to G_{aux}; when $f_{E_a}(u, u') = 0$, there exists no path from u to u' in G. Such edges are removed form G_{aux} (line 4), and, finally, the auxiliary graph is returned (line 5).

Given a node v in G and the auxiliary graph $G_{aux}(V_a,\ E_a,\ f_{V_a},\ f_{E_a},\ P_{E_a})$ of the subgraph of G such that $v \notin V_a$, procedure updateAuxGraph returns the auxiliary graph of the subgraph of G with nodes $V_a \cup \{v\}$. It works as follows. For each node u in G_{aux}, updateAuxGraph adds two new edges (v, u) and (u, v) to E_a, and assigns their weights $f_{E_a}(u, v)$, $f_{E_a}(v, u)$ and paths $P_{E_a}(u, v)$ and $P_{E_a}(v, u)$ by calling procedure assign (omitted; lines 1-3). For each new edge (v, u), weight $f_{E_a}(v, u)$ is either $f_E(v, u)$ (if there exists an edge (v, u) in G), or $\max\{\min\{f_{E_a}(v, u'),\ f_E(u', u)\}\}$ for all nodes u' in V_a such that (v, u') is an edge in G. Moreover, $P_{E_a}(v, u)$ is either edge (v, u), or a path consisting of (v, u') followed by $P_{E_a}(u', u)$; similarly for the new edge (u, v).

After these, the weights and paths of existing edges are updated (lines 4-8). For each edge (u, u'), the triangle with edges $(u, u'), (u, v)$ and (v, u') is considered to find weight h. If $h > f_{E_a}(u, u')$, $f_{E_a}(u, u')$ is changed to h (line 7), and $P_{E_a}(u, u')$ is changed to the concatenation of $P_{E_a}(u, v)$ and $P_{E_a}(v, u')$ (line 8). Finally, node v is added to G_{aux} (line 9), and the updated auxiliary graph is returned (line 10).

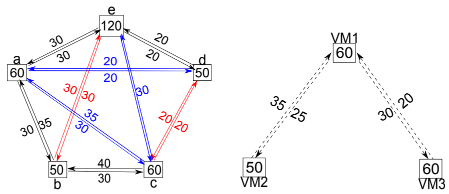
Input: A weighted directed graph $G(V, E, f_V, f_E)$.
Output: An auxiliary graph G_{aux}.
1. $G_{aux}(V_a, E_a, f_{V_a}, f_{E_a}, P_{E_a}) := (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$;
2. **for each** node v in G **do**
3. $\quad G_{aux} :=$ updateAuxGraph(G_{aux}, G, v);
4. Remove edges (u, u') from G_{aux} having $f_{E_a}(u, u') = 0$;
5. **return** G_{aux}.

Procedure updateAuxGraph (G_{aux}, G, v)
Input: Auxiliary graph G_{aux}, graph G, and node v.
Output: Updated G_{aux} by incorporating v.
1. **for each** node u in V_a **do**
2. $\quad E_a := E_a \cup \{(u, v), (v, u)\}$;
3. $\quad$ Assign(v, u, G_{aux}); Assign(u, v, G_{aux});
4. **for each** edge (u, u') in G_{aux} having $u, u' \in V_a$ **do**
5. $\quad h := \min\{f_{E_a}(u, v), f_{E_a}(v, u')\}$;
6. $\quad$ **if** $f_{E_a}(u, u') < h$ **then**
7. $\quad\quad f_{E_a}(u, u') := h$;
8. $\quad\quad P_{E_a}(u, u') := P_{E_a}(u, v) + P_{E_a}(v, u')$;
9. $V_a := V_a \cup \{v\}$; $f_{V_a}(v) := f_V(v)$;
10. **return** G_{aux};

FIGURE 3. Algorithm compAuxGraph

EXAMPLE 7. For the SN shown in Fig. 2(b), the auxiliary graph constructed by compAuxGraph is shown in Fig. 4(a). Note that the bandwidths on edges between b and e, c and d are larger than their counterparts in the SN of Fig. 2(b), since they are updated by procedure updateAuxGraph (lines 5–7). Moreover, there are new edges with positive bandwidth directly connecting a and d, a and c, b and d, c and e.

For each edge (u, v), the auxiliary graph also records the path with the maximum bandwidth among all paths connecting u and v in SN. Taking edges (b, e) and (c, d) as examples, $P(b,\ e) = (b, a, e)$, $P(e,\ b) = (e, a, b)$, $P(c, d) = (c, b, e, d)$ and $P(d, c) = (d, e, a, b, c)$. Note that paths (c, b, a, e, d) and (d, e, b, c) also carry the



(a) Auxiliary graph (b) Minimized VN

FIGURE 4. Auxiliary graphs and minimizing VNs

maximum bandwidth in the SN for edges (c, d) and (d, c), respectively, but compAuxGraph only records one of them since it already suffices to assure the existence of an edge mapping. □

Correctness & complexity. One can verify the following property about updateAuxGraph: (1) for any new edge (u, v), its weight and path are not affected by updating existing edges; and (2) for any existing edge (u, u'), it suffices to consider the triangle with edges $(u, u'), (u, v)$ and (v, u') for updating its weight and path. This shows that updateAuxGraph always produces an auxiliary graph $G_{aux}(V_a, E_a, f_{V_a}, f_{E_a}, P_{E_a})$ for the subgraph of G with nodes V_a only. From this the correctness of algorithm compAuxGraph follows.

Algorithm compAuxGraph is in $O(|V|^3)$ time since procedure updateAuxGraph takes $O(|V_a|^2)$ time, and it is called $|V|$ times in total. Note that $|V_a| \leq |V|$.

4.2. Minimizing Virtual Network Patterns

We next show how to minimize VNs.

Equivalence. Given two VNs $G_{P_1}(V_P, E_{P_1}, f_{V_P}, f_{E_{P_1}})$ and $G_{P_2}(V_P, E_{P_2}, f_{V_P}, f_{E_{P_2}})$, we say that G_{P_1} is *equivalent* to G_{P_2}, denoted by $G_{P_1} \equiv G_{P_2}$, if for any SN G_S and cost function $c()$, there exists a VNM from G_{P_1} to G_S if and only if there exists another VNM from G_{P_2} to G_S with the same cost.

We can minimize a VN G_P in cubic-time:

Theorem 4.1. *There exists a cubic-time algorithm that, given any* VN G_P*, finds an equivalent* VN G_P^m *of* G_p *such that for any* $G'_P \equiv G_P$, G_P^m *has no more edges than* G'_P.

We next present such an algorithm for minimizing VNs, denoted by minVN and shown in Fig. 5. Given a VN G_P, it returns a minimized equivalent VN G_P^m.

Given G_P, algorithm minVN first computes the auxiliary graph G'_P of G_P (line 1), with an empty path set since the path information is not needed here. Starting from an empty VN G_P^m (line 2), the algorithm iteratively adds nodes to G_P^m, one at a time by calling procedure updateVN (lines 3-4). Finally, the minimized VN G_P^m is returned (line 5).

We next present procedure updateVN. Given a node v in G'_P and the minimized VN $G_P^m(V_P^m, E_P^m, f_{V_P^m}, f_{E_P^m})$ of the subgraph of G with nodes V_P^m, where $v \notin G'_P$, it returns the minimized VN G_P^m of the subgraph of G with nodes $V_P^m \cup \{v\}$. More specifically, procedure updateVN

Input: A virtual network $G_P(V_P, E_P, f_{V_P}, f_{E_P})$.
Output: A minimized equivalent VN G_P^m.

1. $G'_P(V_P, E'_P, f_{V_P}, f_{E'_P}, \emptyset) :=$ compAuxGraph(G_P);
2. $G_P^m(V_P^m, E_P^m, f_{V_P^m}, f_{E_P^m}) := (\emptyset, \emptyset, \emptyset, \emptyset)$;
3. **for each** node v in G'_P **do**
4. $\quad G_P^m :=$ UpdateVN (G_P^m, v, G'_P);
5. **return** G_P^m.

Procedure UpdateVN (G_P^m, v, G'_P).
Input: VN G_P^m, node v, and auxiliary graph G'_P.
Output: Updated G_P^m by incorporating v.

1. $V_P^m := V_P^m \cup \{v\}$; $f_{V_P^m}(v) := f_{V_P}(v)$;
2. **for each** node $u \neq v$ in V_P^m **do**
3. $\quad$ **if** $(v, u) \in E'_P$ **and** there is no $u' \in V_P^m$ such that $(u', u) \in E'_P$ and $(v, u') \in E_P^m$
4. $\quad$ **then** $E_P^m := E_P^m \cup \{(v, u)\}$; $f_{E_P^m}(v, u) := f_{E'_P}(v, u)$;
5. $\quad$ **if** $(u, v) \in E'_P$ **and** there is no $u' \in V_P^m$ such that $(u, u') \in E'_P$ and $(u', v) \in E_P^m$
6. $\quad$ **then** $E_P^m := E_P^m \cup \{(u, v)\}$; $f_{E_P^m}(u, v) := f_{E'_P}(u, v)$;
7. **return** G_P^m;

FIGURE 5. Algorithm minVN for minimizing VNs

first adds node v to G_P^m (line 1). It then adds edges to G_P^m that connect node v with other nodes in G_P^m (lines 2-6). An edge (v, u) is added to E_P^m only if there exists no node u' such that there is a path from u' to u in G_P^m ($(u', u) \in E'_P$) and (v, u') is an edge in G_P^m (lines 3-4); similarly for edge (u, v) (lines 5-6). Finally, the updated G_P^m is returned (line 7).

Example 8. Consider the VN in Fig. 2(a) for priority mapping. Given the VN, procedure minVN derives from it an equivalent yet simpler VN, as shown in Fig. 4(b). Observe the following. (1) There exist no edges (VM_2, VM_3) and (VM_3, VM_2) in Fig. 4(b), as opposed to Fig. 2(a). This is because (VM_2, VM_3) (resp. (VM_3, VM_2)) is entailed by edges (VM_2, VM_1) and (VM_1, VM_3) (resp. edges (VM_3, VM_1) and (VM_1, VM_3)), and hence, can be left out. (2) The edge constraints in Fig. 4(b) differ from their counterparts in Fig. 2(a). □

Correctness & complexity. To show the correctness, one can first verify the following.

Lemma 4.1. *For any* VN G_P*, procedure* updateVN *returns an* VN G_P^m *such that there exists a unique path from node* u *to* v *in* G_P^m *if and only if there exists a path from node* u *to* v *in the* VN G_P.

Proof: (1) Assume first that there exists a path from nodes u to v in VN G_P. We then show that there must exist a unique path from nodes u to v in G_P^m, which is returned by UpdateVN.

From the definition of auxiliary graph, we know that there must be an edge (u, v) in G'_P. From lines 3 and 5 of UpdateVN, one can see that if (u, v) is in G'_P, then there must be a path that carries the same bandwidth in G_P^m. This path is either the edge (u, v) in G_P^m, or a path via an intermediate node u', as stated in lines 4 and 6. This verifies the existence of a path from u to v to G_P^m. Such a path is unique. Indeed, once the path connecting u to v is added to G_P^m at some point of the **for** loop in UpdateVN, no new paths from u to v will be added since UpdateVN finds that there exists u' such that (u, u') is in G'_P and (u', v) is in G_P^m (lines 3 and 5).

(2) Conversely, assume that there exists a path from u to v in G_P^m. Then it must be introduced by the **for** loop in UpdateVN invoked by u. Since there exists a path that connects u to v found by UpdateVN, edge (u, v) must be included in G'_P; hence there is a path that connects u to v in VN G_P (by the definition of the auxiliary graph for VN). □

By Lemma 4.1, we can show that procedure updateVN produces a minimized VN $G_P^m(V_P^m, E_P^m, f_{V_P^m}, f_{E_P^m})$ for the subgraph of G_P with nodes V_P^m only. From this the correctness of algorithm minVN immediately follows.

Observe the following. (1) Algorithm compAuxGraph runs in $O(|V_P|^3)$ time. (2) Procedure updateVN takes $O(|V_P^m|^2)$ time, and it is called $|V_P|$ times in total. Hence, algorithm minVN runs in $O(|V_P|^3)$ time.

4.3. Finding Minimum Cost Priority Mappings

We are now ready to present our algorithm for computing priority mappings, denoted by compVNM and shown in Fig. 6. Given a VN request $(G_p, \mathcal{C})$, an SN G_S, and a cost function $c()$, the algorithm finds a low cost VNM $((g_V, r_V), (g_E, r_E))$ from G_P to G_S if there exists one. As will be seen shortly, it uses a predefined non-negative integer k to control the level of backtracks, which is typically small, *e.g.,* 2 or 3.

As remarked earlier, the algorithm employs two optimization strategies to reduce search space. (1) It removes redundant edge constraints from VN G_P, via algorithm minVN (line 2). (2) It constructs the auxiliary graph G_{aux} of SN G_S by using algorithm compAuxGraph, to validate a node mapping without traversing the entire G_S (line 3). The algorithm builds a low cost node mapping by inspecting nodes one by one, via procedure backTrackMap (lines 4-6). It unifies the processes of building node mappings and edge mappings: it checks whether the partial node mapping found so far is valid ($(g_V, r_V, S) \neq$ null, line 6). If so, it finds the corresponding edge mapping by calling procedure identifyEdgeMap (omitted; line 7). With G_{aux}, the edge mapping can be found in $O(|E_P^m|)$ time (Proposition 4.1). A VNM is returned if there exists one (line 8).

We next present procedure backTrackMap. Given a new node v, a node set S for which mappings are already identified, a node set *backS*, and non-negative integers i and k, it expands the mapping for S by including v. If v cannot be mapped to a substrate node, it backtracks and searches other nodes, along the same lines as [13]. The backtrack depth is bounded by k. It uses i to keep track of the current backtrack depth, and *backS* to record the set of nodes backtracked. In contrast to [13] that has to traverse the entire G_s, we reduce the search space by inspecting only virtual nodes in the minimized VN G_P^m, and by checking edge constraints using auxiliary graph G_{aux}.

Input: An SN G_S, a VN request $(G_P, \mathcal{C})$, a cost function $c()$, and a positive integer k.
Output: A low cost mapping from G_P to G_S.

1. $(g_V, r_V) := (\emptyset, \emptyset)$; $S := \emptyset$;
2. $G_P^m(V_P^m, E_P^m, f_{V_P^m}, f_{E_P^m}) :=$ minVN(G_P);
3. $G_{aux}(V_a, E_a, f_{V_a}, f_{E_a}) :=$ compAuxGraph(G_S);
4. **for each** v in V_{P_m} **do**
5. $\quad (g_V, r_V, S) :=$ backTrackMap$(v, S, \emptyset, 0, k)$;
6. $\quad$ **if** $(g_V, r_V, S) =$ null **then return** null;
7. $(g_E, r_E) :=$ identifyEdgeMap(g_V, r_V, G_{aux});
8. **return** $((g_V, r_V), (g_E, r_E))$.

Procedure backTrackMap$(v, S, backS, i, k)$
Input: Node v, node sets S and *backS*, non-negative integers i and k.
Output: Updated node mapping (g_V, r_V).

1. **if** $i > k$ **then return** null;
2. **if** there exists u in G_{aux} with Valid$(v, u, S) =$ true **then**
3. $\quad g_V(v) := u$; $r_V(v) := f_{V_P^m}(v)$; $S := S \cup \{v\}$;
4. $\quad$ **return** (g_V, r_V, S);
5. **for each** $v' \in S \setminus backS$ **do**
6. $\quad$ **if** Valid$(v, g_V(v'), S \setminus \{v'\})$ **then**
7. $\quad\quad g_V(v) := g_V(v')$; $r_V(v) := f_{V_P^m}(v)$; $S := S \cup \{v\} \setminus \{v'\}$;
8. $\quad$ **if** backTrackMap$(v', S, backS \cup \{v\}, i+1, k)$ **then**
9. $\quad\quad$ **return** (g_V, r_V, S);
10. $\quad S := S \setminus \{v\} \cup \{v'\}$; $g_V(v') := g_V(v)$;
11. **return** null;

FIGURE 6. Algorithm compVNM for priority mappings

More specifically, if the current backtrack depth $i > k$, then the procedure returns null (line 1). Otherwise, it checks whether there is a node u to which node v can be mapped (lines 3-4). It uses procedure Valid (omitted), which checks whether the (partial) node mapping admits an edge mapping by inspecting the edge constraints in G_{aux}. If not, node v may be mapped to a node $g_V(v')$ to which node v' is already mapped (line 6), and procedure backTrackMap is called recursively to find a mapping node for node v' (line 8). Such nodes v' are checked (lines 5-9), with their information backed up (line 7) and restored later (line 10). If a valid node mapping cannot be found, null is returned (line 11).

Example 9. Consider the VN request and SN of Fig. 2. Assume a cost function $c()$ for the SN such that (1) for nodes a, b and c, their costs are the same as their node capacities; (2) for d and e, their costs are ten times of their node capacities; and (3) the cost of each physical link in the SN is its edge capacity.

We show below how compVNM finds a priority mapping from the VN to the SN. Algorithm compVNM first computes the minimized VN and the auxiliary graph G_{aux} of the SN, as shown in Fig. 4. It then finds mappings for nodes VM_1, VM_2 and VM_3 in the minimized VN by calling procedure backTrackMap and by leveraging G_{aux}. It starts with VM_1 and maps it to SN node a via backTrackMap such that (VM_1, a) is a valid node mapping for the subgraph of V_P^m with node VM_1 only. It then invokes backTrackMap and maps VM_2 to SN node c such that (VM_1, a) and (VM_3, c) make a valid node mapping for the the subgraph of V_P^m with nodes VM_1 and VM_3. Similarly, a candidate mapping node b is found for VM_2. No backtrack is needed in backTrackMap for all these nodes. Then compVNM identifies edge mappings by using the auxiliary graph G_{aux}. It maps virtual edges to those paths recorded in G_{aux}, *e.g.,* $(\mathsf{VM}_1, \mathsf{VM}_2)$ is mapped to $P(a, b)$ (see Example 7). Finally the mapping is found and returned. □

Complexity. Algorithm compVNM is in $O(|V_S|^3 + |V_P|^{(k+1)}\ |E_P|(|V_P| + |V_S|) + |V_P|^3)$ time, where $|V_S|$, $|V_P|$, $|E_P|$ are the number of nodes in G_S, the number of nodes in G_P and the number of edges in G_P, respectively. Indeed, procedures compAuxGraph, minVN and backTrackMap take $O(|V_S|^3)$ time, $O(|V_P|^3)$ time and $O(|V_P|^k|E_P|(|V_P| + |V_S|))$ time, respectively. Here k is a *predefined constant*. We found that a small k (usually no more than 3) typically suffices, as will be verified in the experimental study (Section 5).

Remark. One can extend algorithm compVNM for priority mappings with node sharing, denoted by $\mathsf{compVNM}_{\mathsf{NS}}$, by simply allowing multiple virtual nodes in G_P to be mapped to the same node in G_S in Valid.

5. EXPERIMENTAL STUDY

In this section we present an experimental study of our techniques for computing virtual network priority mappings (VNM_P). We conducted two sets of experiments to evaluate (1) the effectiveness of VNM_P versus conventional virtual network embedding ($\mathsf{VNM}_{\mathsf{SP}}$) and (2) the efficiency of our algorithms.

Experimental setting. Following the tradition of virtual network topology research (*e.g.,* [12–14]), we used the following datasets that simulate real-life virtual networks.

*Substrate networks (*SNs*).* We used three types of substrate networks, as found in real life. (a) Directed-tree networks, in which for any two nodes u and v, there exists an edge (u, v) if and only if there exists an edge (v, u), and the network becomes a tree if the two edges between any two nodes are merged into one. (b) Full-mesh networks, in which for any two nodes u and v, there exist two edge (u, v) and (v, u). (c) Random networks, in which for any pair of nodes u and v, there exists an edge (u, v) with probability p. Directed-tree networks and full-mesh networks were constructed by adopting real-life network topologies (http://en.wikipedia.org/wiki/Network_topology).

TABLE 4. Summary of testing parameters

	VN Requests	SNs
Number of nodes	n_P	n_S
Edge probability	p_P	p_S
Node capacity	w_{V_P}	w_{V_S}
Edge capacity	w_{E_p}	w_{E_S}

We developed a graph generator to produce these networks, controlled by the following parameters: (a) the number n_S of nodes, (b) the node capacity w_{V_S}, (c) the edge capacity w_{E_S}, and (d) the probability p_S (for random networks only).

VN *requests.* VN requests arrive in a Poisson process with an average of λ requests per time unit, as commonly adopted by network community [13, 15, 16]. Each one has a lifespan with an average of l time units. The VNs were randomly produced by the same graph generator for substrate networks, controlled by four parameters: (a) the number n_P of nodes, (b) the virtual node capacity w_{V_P}, (c) the edge capacity w_{E_P}, and (d) the probability p_P.

Algorithms. We have implemented the following algorithms, all in C++. (a) Algorithms compVNM and $\mathsf{compVNM}_{\mathsf{NS}}$ for computing VNM_P (Section 4), without node sharing and with node sharing, respectively. (b) Algorithms SubIso [13], ViNE [16] and RW-SP [23] for computing $\mathsf{VNE}_{\mathsf{SP}}$ (single-path embedding without node sharing; see Sections 1 and 2). (c) Algorithm $\mathsf{ViNE}_{\mathsf{NS}}$ that extends ViNE for computing $\mathsf{VNE}_{\mathsf{SP}}$ with node sharing. We compared algorithms compVNM and $\mathsf{compVNM}_{\mathsf{NS}}$ with those for $\mathsf{VNE}_{\mathsf{SP}}$ because there are no previous algorithms for VNM_P, and $\mathsf{VNE}_{\mathsf{SP}}$ is the VN model closest to VNM_P, with or without node sharing.

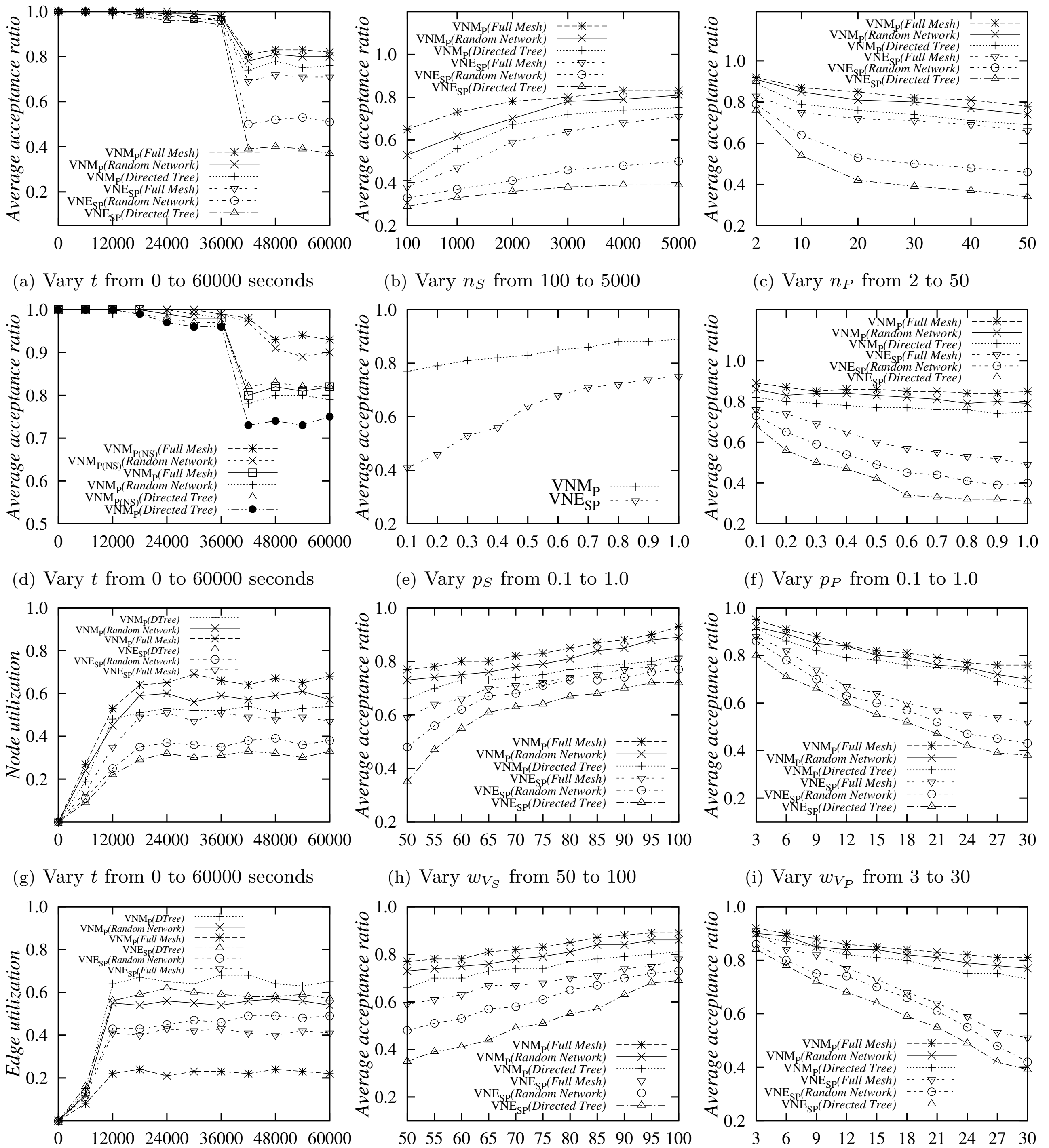
The experiments were run on a machine with Intel Core i7 860 CPU and 16GB of memory. All the experiments were repeated over 5 times and the average is reported here.

Experimental results. We next report our findings. In all the experiments, for VN requests, we fixed $\lambda = 0.02$ and $l = 1000$, which were decided based on the substrate networks considered, and had little impact on the quality and efficiency tests. We also fixed the backtrack depth $k = 3$ for compVNM and $\mathsf{compVNM}_{\mathsf{NS}}$. We adopted algorithm ViNE for $\mathsf{VNE}_{\mathsf{SP}}$ when comparing the mapping quality of VNM_P with $\mathsf{VNE}_{\mathsf{SP}}$. We summarize the tested factors in Table 4.

Exp-1: Mapping quality. In the first set of experiments, we evaluated (1) the mapping quality of VNM_P vs. $\mathsf{VNE}_{\mathsf{SP}}$, (2) the impact of node sharing, and (3) the resource utilization on nodes and edges.

We used the *average acceptance ratio* (AR), a quality measure commonly adopted by the network community [13,15,16], to evaluate the mapping quality. Given a time stamp t, AR is defined as:

$$\mathsf{AR}(t) = \#\mathsf{validVNs}(t)\ /\ \#\mathsf{arrivedVNs}(t),$$



(a) Vary t from 0 to 60000 seconds (b) Vary n_S from 100 to 5000 (c) Vary n_P from 2 to 50

(d) Vary t from 0 to 60000 seconds (e) Vary p_S from 0.1 to 1.0 (f) Vary p_P from 0.1 to 1.0

(g) Vary t from 0 to 60000 seconds (h) Vary w_{V_S} from 50 to 100 (i) Vary w_{V_P} from 3 to 30

(j) Vary t from 0 to 60000 seconds (k) Vary w_{E_S} from 50 to 100 (l) Vary w_{E_P} from 3 to 30

FIGURE 7. Mapping quality

where #validVNs(t) denotes the number of VN requests that are fulfilled until time t, and #arrivedVNs(t) denotes the total number of VN requests arrived until time t, respectively. Intuitively, AR(t) is the ratio of VNs successfully mapped during time interval $[0, t]$.

(1) We first evaluated the impact of time t on AR. For VN requests, we fixed $p_P = 0.5$, n_P in [2, 50], and w_{V_P} and w_{E_P} in [3, 30]. For SNs, we fixed $n_S = 5000$, and w_{V_S} and w_{E_S} in [50, 100]. Since *medium-size ISPs* have about *500* nodes only [13], $n_S = 5000$ suffices. We varied t from 0 to $60,000$ seconds.

Figure 7(a) shows the AR of VNM$_P$ and VNE$_{SP}$ over directed-tree, full-mesh and random networks. We found the following. (i) In all the cases, the AR decreases *w.r.t.* t, and becomes stable when t is about $42,000s$. This is because initially there exists no workload in the SNs; the SNs are fully loaded when

t reaches $42,000$ seconds, since only a certain amount of work can be handled by the SNs. (ii) The AR of $\mathsf{VNM_P}$ is consistently higher than that of $\mathsf{VNE_{SP}}$ (in the range of $[11\%, 39\%]$) in all the cases. (iii) The impact of network topologies on the AR of $\mathsf{VNM_P}$ is much smaller than that of $\mathsf{VNE_{SP}}$. Indeed, the stable AR for $\mathsf{VNM_P}$ is in the range of [76%, 82%], while for $\mathsf{VNE_{SP}}$, it is around 37% and 71% on directed-tree and full-mesh networks, respectively. This is because $\mathsf{VNM_P}$ has weaker capacity constraints on edges of SN compared to $\mathsf{VNE_{SP}}$.

Figure 7(d) shows the AR of $\mathsf{VNM_P}$ with node sharing versus its counterpart without node sharing, over directed-tree, full-mesh and random networks. The results show the following. (i) Node sharing consistently improves the AR for priority mappings (in the range of [8%, 11%]). Indeed, the AR on full-mesh networks is over 93% with node sharing, as opposed to 82% without node sharing. (ii) Node sharing also improves the AR for $\mathsf{VNE_{SP}}$ (not shown). This verifies that the idea of node sharing is generic, and can be employed by other virtual network mapping models.

Note that the AR of all algorithms had a significant drop around t = 40000s in both Figures 7(a) and 7(d). This is because at that time, all the resource of the SNs were almost already allocated to the VN requests that were posed on the SNs and were still in their life span.

(2) To evaluate the impact of SNs on AR, we fixed $t = 60,000s$, and VN with $n_P = 50$, $p_P = 0.5$ and $w_{V_P} = w_{E_P} = 30$. We varied one of the four factors of SNs: n_S from 100 to 5,000, p_S from 0.1 to 1.0, and w_{V_S} and w_{E_S} from 50 to 100, while fixing the other three factors of SNs with default values n_S = 5,000, p_S = 0.5, $w_{V_s} = w_{E_s} = 100$. The test of p_S can be conducted on SNs of random networks only since edges in full-mesh and directed-tree networks cannot be randomly generated, *e.g.*, p is always 1 for full-mesh SNs.

The results are reported in Figures 7(b), 7(e), 7(h) and 7(k), which tell us the following. (i) The AR increases *w.r.t.* n_S, p_S, w_{V_S} and w_{E_S}. This is because of the following. (a) The larger n_S is, there are more nodes in the SNs, and the larger p_S is, there are more links in the SNs. (b) The lager w_{V_P} and w_{E_P} are, there are larger capacities in the nodes and links of the SNs, respectively. Hence, the SNs can handle more requests when any of these four factors is increased, and therefore, their AR gets larger. (ii) The AR of $\mathsf{VNM_P}$ is consistently higher than the AR of $\mathsf{VNE_{SP}}$ in all the cases, up to 37%. (iii) The AR of $\mathsf{VNM_P}$ is less sensitive to network topologies than the AR of $\mathsf{VNE_{SP}}$, which is consistent with the results reported in Figure 7(a).

(3) To evaluate the impact of VN requests on AR, we fixed $t = 60,000s$, and SNs with $n_S = 5000$, $p_S = 0.5$ and $w_{V_S} = w_{E_S} = 100$. We varied the four factors of VNs: n_P from 2 to 50, p_P from 0.1 to 1.0, and w_{V_S}, w_{E_S} from 3 to 30, while fixing the other three factors of VNs with default values $n_P = 50$, $p_p = 0.5$, $w_{V_P} = w_{E_P} = 30$. Again the test of p_P is conducted on the VNs of random networks only.

As shown in Figures 7(c), 7(f), 7(i) and 7(l), the results tell us the following. (i) The AR decreases *w.r.t.* n_P, p_P, w_{V_P} and w_{E_P}. Indeed, (a) the larger n_P is, the more machines are requested by the VNs; (b) the larger p_P is, the more links are demanded; and (c) the lager w_{V_P} and w_{E_P} are, the more capacities are required. As a result, AR decreases with the increase of any of these four factors, which makes the VN requests harder to fulfill. (ii) The AR of $\mathsf{VNM_P}$ is consistently higher than the AR of $\mathsf{VNE_{SP}}$ in all the cases, up to 33%. (iii) The AR of $\mathsf{VNM_P}$ is less sensitive to network topologies than that of $\mathsf{VNE_{SP}}$, as we have seen earlier.

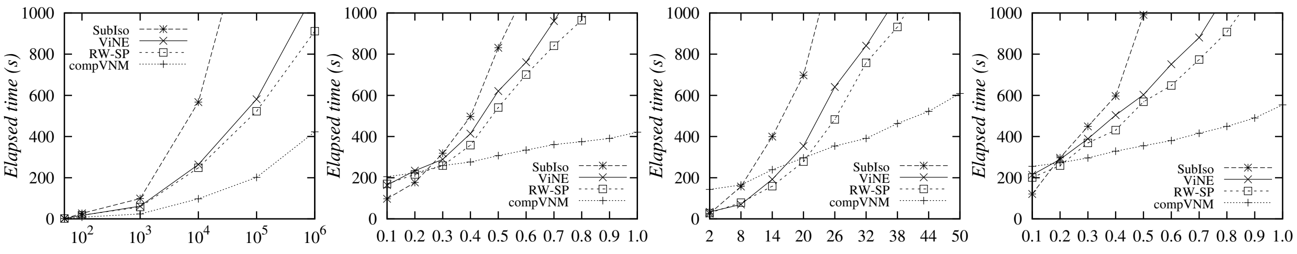
(4) We also evaluated the impact of time t on the resource utilization of nodes and edges, in the same settings as (1).

(i) The average resource utilization of substrate nodes is shown in Fig. 7(g). It shows the following. (a) Node utilization of SNs becomes stable after $t = 24000s$. This is because after 24000s, the total number of hosted VNs becomes stable as there is no more resource for new requests, unless existing VN requests expire. (b) Node utilization of full-mesh networks is higher than that of random networks, followed by directed trees, for both $\mathsf{VNM_P}$ mappings and $\mathsf{VNE_{SP}}$ mappings. Intuitively, the denser an SN is, the fewer VN requests will be denied by the SN due to edge capacity constraints. Therefore, they can host more VNs with the same node capacities than sparser SNs. (c) For each type of the three SN topologies, the node utilization of $\mathsf{VNM_P}$ is higher than that of $\mathsf{VNE_{SP}}$, which demonstrates the benefit of priority mappings.

(ii) Figure 7(j) shows the average edge utilization. It tells us the followings. (a) After $t = 12000s$, the average edge utilization becomes stable, no matter what topological structures SNs have. This is analogous to node utilization. (b) Priority mapping over directed trees gains the highest edge mapping, but gets the lowest over full-mesh networks. This is because $\mathsf{VNM_P}$ requires more on node capacities due to its weak edge capacity constraints. Therefore, on full-mesh networks, the bottleneck is the node mapping, which leads to lower edge utilization. (c) Generally, the impact of network topologies on the edge utilization for $\mathsf{VNM_P}$ is larger than that for $\mathsf{VNE_{SP}}$. This is consistent with node utilization.

Exp-2: Mapping efficiency. In this set of experiments, we evaluated the efficiency of our algorithm compVNM for $\mathsf{VNM_P}$ versus algorithms SubIso [13], ViNE [16] and RW-SP [23] for $\mathsf{VNE_{SP}}$. We used large random networks in the experiments. We do not report the impact of node and edge capacities w_{V_P} and w_{E_P} on VNs, and w_{V_S} and w_{E_S} on SNs, since these factors have little impact on the efficiency, as shown by the corresponding complexity analysis (see Section 4).

(1) To evaluate the impact of SNs, we fixed VN requests



(a) Vary n_S from 10^2 to 10^6 (b) Vary p_S from 0.1 to 1.0 (c) Vary n_P from 2 to 50 (d) Vary p_P from 0.1 to 1.0

FIGURE 8. Mapping efficiency and scalability

with $n_V = 25$, $p_P = 0.5$, $w_{V_P} = w_{E_P} = 30$, we varied n_S from 10^2 to 10^6 (while fixing $p_S = 0.5$, $w_{V_S} = w_{E_S} = 100$) and p_S from 0.1 to 1.0 (while fixing $n_S = 10^6$), respectively. The results are shown in Figures 8(a) and 8(b), respectively.

(2) To evaluate the impact of VNs, we fixed SNs with $n_S = 500{,}000$, $p_S = 0.5$, $w_{V_S} = w_{E_S} = 100$, we varied n_V from 2 to 50 (while fixing $p_V = 0.5$, $w_{V_P} = w_{E_P} = 30$) and p_V from 0.1 to 1.0 (while fixing $n_V = 50$, $w_{V_P} = w_{E_P} = 30$), respectively. The results are reported in Figures 8(c) and 8(d), respectively.

These results tell us the following. (i) As expected, the running time of all these algorithms increases with the increase of n_S, p_S, n_P and p_P. (ii) Algorithm compVNM is efficient: it took only around 420s for SNs with *1 million* nodes. (ii) It outperforms the other three algorithms for $\mathsf{VNE_{SP}}$ in almost all the cases. Indeed, compVNM is about twice faster than the other algorithms. While it took compVNM less than 600s for $n_S = 10^6$, $p_S = 1.0$, $n_P = 50$ or $p_P = 1.0$ in Figures 8(a), 8(b), 8(c) and 8(d), respectively, the other algorithms took at least 912s, or could not run to completion.

Summary. From these experimental results we find the following. (1) Priority mapping ($\mathsf{VNM_P}$) proposed in this work is able to find high-quality mappings, and has higher acceptance ratio than the previous mapping model ($\mathsf{VNE_{SP}}$), typically from 11% to 39%. (2) Priority mapping is less sensitive to network topologies. (3) Node sharing improves the mapping quality, typically from 8% to 11%. (4) The average node and edge utilization of $\mathsf{VNM_P}$ is much higher than $\mathsf{VNE_{SP}}$. (5) Our algorithm for computing priority mapping is efficient, *e.g.,* it took 420 seconds for SNs with 10^6 nodes, and it substantially outperforms previous algorithms for $\mathsf{VNE_{SP}}$ that took more than 912 seconds.

6. CONCLUSION

We have proposed a generic model to express various VN requests found in practice, based on graph pattern matching. We have also established several intractability and approximation hardness results in various practical VNM settings. These are among the first efforts to settle fundamental problems for virtual network mapping. For intractable VNM cases, we have developed algorithms for priority mapping, a VNM problem identified in this work that is important in emerging applications. We have experimentally verified that the algorithms are effective and efficient, using real-life and synthetic data. These results not only provide foundation for developing virtualized cloud data centers, but are also useful to the study of graph pattern matching in the presence of constraints.

Several extensions are targeted for future work. First, we are currently evaluating the techniques with large SNs, and developing optimization techniques for VNM. Second, to simplify the discussion we have only presented constraints on CPU, storage, bandwidth and latency in this work. It is possible to extend our model to incorporate factors such as storage locality, data placements requirements and security policies. Third, incremental VNM methods need to be explored to adapt to peak and off-peak cloud workloads. Fourth, we are also studying other practical quality functions for VNM beyond mapping costs. Finally, we are exploring techniques for processing VN requests in the uniform model for different applications, as well as their use in graph pattern matching in real-life applications.

ACKNOWLEDGEMENTS

Fan and Cao are supported in part by 973 Program 2014CB340302 and 2012CB316200, NSFC 61133002, the Foundation for Innovative Research Groups of NSFC, Shenzhen Peacock Program 1105100030834361, Guangdong Innovative Research Team Program 2011D005, EPSRC EP/J015377/1 and EP/M025268/1, ERC-2014-AdG 652976, NSF III 1302212, and a Google Faculty Research Award. Ma is supported in part by 973 Program 2014CB340304, NSFC 61322207 and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Trelles, O., Prins, P., Snir, M., and Jansen, R. C. (2011) Big data, but are we ready? *Nature reviews Genetics*, **12**.

[2] Agrawal, D., Das, S., and El Abbadi, A. (2011) Big data and cloud computing: current state and future opportunities. *EDBT*.

[3] http://www.bigswitch.com/.

[4] http://aws.amazon.com/ec2/.

[5] http://www.vmware.com/solutions/datacenter/.

[6] Xiong, P., Chi, Y., Zhu, S., Moon, H. J., Pu, C., and Hacigümüs, H. (2011) Intelligent management of virtualized resources for database systems in cloud environment. *ICDE*.

[7] Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., and Kamath, S. (2010) Automatic virtual machine configuration for database workloads. *TODS*, **35**.

[8] Aboulnaga, A., Salem, K., Soror, A., Minhas, U., Kokosielis, P., and Kamath, S. (2009) Deploying database appliances in the cloud. *IEEE Data Eng. Bull*, **32**, 13–20.

[9] Aboulnaga, A., Amza, C., and Salem, K. (2008) Virtualization and databases: state of the art and research challenges. *EDBT*.

[10] Shivam, P., Demberel, A., Gunda, P., Irwin, D. E., Grit, L. E., Yumerefendi, A. R., Babu, S., and Chase, J. S. (2007) Automated and on-demand provisioning of virtual machines for database applications. *SIGMOD*.

[11] Bobroff, N., Kochut, A., and Beaty, K. (2007) Dynamic placement of virtual machines for managing sla violations. *IM*.

[12] Houidi, I., Louati, W., and Zeghlache, D. (2008) A distributed virtual network mapping algorithm. *ICC*.

[13] Lischka, J. and Karl, H. (2009) A virtual network mapping algorithm based on subgraph isomorphism detection. *SIGCOMM workshop VISA*.

[14] Lu, J. and Turner, J. (2006) Efficient mapping of virtual networks onto a shared substrate. *Washington University, TR WUCSE-2006*, **35**.

[15] Yu, M., Yi, Y., Rexford, J., and Chiang, M. (2008) Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM CCR*, **38**, 17–29.

[16] Chowdhury, N., Rahman, M., and Boutaba, R. (2009) Virtual network embedding with coordinated node and link mapping. *INFOCOM*.

[17] Reinhardt, W. (1994) Advance reservation of network resources for multimedia applications. *IWACA*.

[18] http://frenzy.ivic.org.cn/.

[19] Schlansker, M. S., Collard, J.-f., and Kumar, R. (2013). Prioritized polling for virtual network interfaces. US Patent 8,364,874.

[20] http://www.isi.edu/xbone/.

[21] Gallagher, B. (2006) Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, **6**, 45–53.

[22] Ausiello, G. (1999) *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Verlag.

[23] Cheng, X., Su, S., Zhang, Z., Wang, H., Yang, F., Luo, Y., and Wang, J. (2011) Virtual network embedding through topology-aware node ranking. *SIGCOMM CCR*, **41**, 38–47.

[24] Cao, Y., Fan, W., and Ma, S. (2015) Virtual network mapping: A graph pattern matching approach. *BICOD*.

[25] Hsu, W. and Shieh, Y. (2013) Virtual network mapping algorithm in the cloud infrastructure. *J. NCA*, **36**, 1724–1734.

[26] Ricci, R., Alfeld, C., and Lepreau, J. (2003) A solver for the network testbed mapping problem. *SIGCOMM CCR*, **33**, 81.

[27] Zhu, Y. and Ammar, M. (2006) Algorithms for assigning substrate network resources to virtual network components. *INFOCOM*.

[28] Díaz, J., Petit, J., and Serna, M. (2002) A survey of graph layout problems. *CSUR*, **34**, 313–356.

[29] Andersen, D. G. (2002). Theoretical approaches to node assignment.

[30] Fan, W. (2012) Graph pattern matching revised for social network analysis. *ICDT*, pp. 8–21.

[31] Fan, W., Li, J., Ma, S., Wang, H., and Wu, Y. (2010) Graph homomorphism revisited for graph matching. *VLDB*, **3**, 1161–1172.

[32] Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., and Wu, Y. (2010) Graph pattern matching: From intractable to polynomial time. *VLDB*.

[33] Fan, W. and Bohannon, P. (2008) Information preserving XML schema embedding. *TODS*, **33**.

[34] Fan, W., Li, J., Ma, S., Tang, N., and Wu, Y. (2011) Adding regular expressions to graph reachability and pattern queries. *ICDE*.

[35] Megiddo, N. (1987) On the complexity of linear programming. *Advances in Economic Theory: Fifth World Congress*.

[36] Cormen, T. (2001) *Introduction to algorithms*. The MIT press.

[37] Sipser, M. (2012) *Introduction to the Theory of Computation*. Cengage Learning.

[38] Chowdhury, N. M. K. and Boutaba, R. (2010) A survey of network virtualization. *Computer Networks*, **54**, 862–876.

[39] Bavier, A. C., Feamster, N., Huang, M., Peterson, L. L., and Rexford, J. (2006) In VINI veritas: realistic and controlled network experimentation. *SIGCOMM*.

[40] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009) Bcube: A high performance, server-centric network architecture for modular data centers. *SIGCOMM*.

[41] Munkres, J. (1957) Algorithms for the assignment and transportation problems. *SIAM*, **5**, 32–38.

[42] Anily, S., Bramel, J., and Simchi-Levi, D. (1994) Worst-case analysis of heuristics for the bin packing problem with general cost structures. *Operations research*, **42**, 287–298.