

# Virtual Network Mapping: A Graph Pattern Matching Approach

Yang Cao<sup>1,2</sup>   Wenfei Fan<sup>1,2</sup>   Shuai Ma<sup>1</sup>

<sup>1</sup>RCBD&SKLSDE Lab, Beihang University, China

<sup>2</sup>University of Edinburgh, UK

{caoyang, mashuai}@act.buaa.edu.cn   wenfei@inf.ed.ac.uk

**Abstract.** 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 optimization problems, we further show that these problems are approximation-hard, *i.e.*, NPO-complete in general and APX-hard even for special cases.

## 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 [9, 38]. However, data center networks 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 [1]. This highlights the need for VNM, to automatically deploy virtual networks in a data center network in

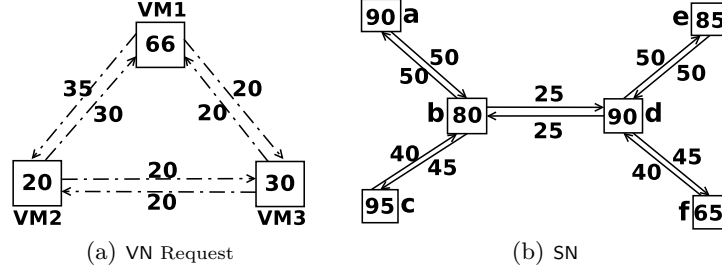


Fig. 1. VN requests found in practice

response to real-time requests. Indeed, VNM is increasingly employed in industry, *e.g.*, Amazon’s EC2 [2], VMware Data Center [3] and Big Switch Networks [1]. It has proven effective in increasing server utilization and reducing server provisioning time (from days or weeks to minutes), server capital expenditures and operating expenses [1]. There has also been a host of work on virtualization techniques for big data [9, 38] and database systems [7, 8, 35, 37, 39].

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 [13].
- (2) *Single-path VN embedding* (VNE<sub>SP</sub>): 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 [26, 28, 29].
- (3) *Multi-path VN embedding* (VNE<sub>MP</sub>): it is to find a node mapping  $f_v$  as in VNE<sub>SP</sub> 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 [18, 40].

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 VM<sub>1</sub>, VM<sub>2</sub> and VM<sub>3</sub>, 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* (VNM<sub>L</sub>). 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  $\text{VNM}_L$ , 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  $\text{VNM}_L$  arises from latency sensitive applications such as multimedia transmitting networks [32], where constraints on virtual links concern latency rather than bandwidth.

(2) *Priority mapping* ( $\text{VNM}_P$ ). Assume that the constraints on the nodes in Fig. 1(a) are CPU capacities, and constraints imposed on edges are bandwidth capacities. Here the VNM problem, denoted by  $\text{VNM}_P$ , 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 [4, 34], 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* ( $\text{VNE}_{\text{SP}(\text{NS})}$ ). 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  $\text{VNE}_{\text{SP}(\text{NS})}$  is an extension of the single-path VN embedding ( $\text{VNE}_{\text{SP}}$ ) 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 [6].

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

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  $\text{VNM}_L$ ,  $\text{VNM}_P$  or  $\text{VNE}_{\text{SP}(\text{NS})}$ . (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,  $\text{VNE}_{\text{SP}}$  and  $\text{VNE}_{\text{MP}}$ ) and those that have been overlooked (*e.g.*,  $\text{VNM}_L$ ,  $\text{VNM}_P$  and  $\text{VNE}_{\text{SP}(\text{NS})}$ ). The uniform model allows us to characterize and compare VNMs 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 VNMs, 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 VNMs. We show that VNMs, 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

**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 ( $VMP_{(NS)}$ )	VM Placement (node sharing (NS))
$VNM_P$ ( $VNM_{P(NS)}$ )	priority mapping (with NS)
$VNE_{SP}$ ( $VNE_{SP(NS)}$ )	single-path embedding (with NS)
$VNE_{MP}$ ( $VNE_{MP(NS)}$ )	multi-path embedding (with NS)
$VNM_L$ ( $VNM_{L(NS)}$ )	latency constrained mapping (NS)

and approximation bounds for VNMs. Moreover, for intractable VNM cases, we develop effective heuristic methods to find high-quality mappings.

(1) We propose a generic model to express VNMs in terms of graph pattern matching [24] (Section 3). 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 VNMs are then simply graph pattern matching problems. We show that the model is able to express VNMs 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 VNMs (Section 5). 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 [11] for special cases. To the best of our knowledge, these are among the first complexity and approximation results on VNMs.

(3) These results tell us that it is beyond reach in practice to find PTIME algorithms for VNMs with edge constraints such as  $VNM_P$  and  $VNE_{SP}$ , or to find efficient approximation algorithms with decent performance guarantees. In light of these, we develop heuristic algorithms for priority mapping  $VNM_P$ , with node sharing or not (Section 6). We focus on  $VNM_P$  since it is needed in, *e.g.*, internet-based virtualized infrastructure computing platform (iVIC [4]) and prioritized polling for virtual network interfaces [34]. 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.*, [26, 28, 29]), no previous work has studied algorithms for  $VNM_P$ .

(4) Finally, we experimentally verify the effectiveness and efficiency of our algo-

rithm by providing a simulation study (Section 7). We evaluate our algorithm for priority mapping and VN embedding (with or without node sharing). We find that our algorithm is able to find high-quality mappings and is efficient on large VNs and SNs. In particular, it outperforms previous algorithms for VN embedding in the cost of mappings, with better or comparable efficiency.

All the proofs of the results can be found in [5].

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 VNMs. 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* [24].

**Related Work.** Virtualization techniques have been investigated for big data processing [9, 38] and database applications, such as database appliance deployment and virtualized resources management for database systems [7, 8, 27, 35, 37, 39]. However, none of these has provided a systematic study of VNM, by modeling VNM as graph pattern matching. The only exception is [28], 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 for cloud computing in database applications.

Several models have been developed for VNM. (a) The VM placement problem (VMP) was studied in [13], which is similar to the bin packing problem and aims to map a set of VMs onto an SN with constraints on node capacities. (b) Single-path VN embedding (VNE<sub>SP</sub>) was investigated in [29, 33, 41], which is to map a VN to an SN by a node-to-node injection 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 VNE<sub>SP</sub>, multi-path embedding (VNE<sub>MP</sub>) was studied in [18, 40], 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 [20] for a survey). In contrast to our work, these 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 [10], 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 [24] for a survey). 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 [21–23], with several intractability and approximation bounds established there. Those differ from this work in that either no constraints on links are considered [23], or graph simulation is adopted [21, 22], 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 [13]. When considering bandwidth constraints on links, [41] 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 [29], allowing constraints on both nodes and links. A path-splitting assumption was proposed in [40], to rectify limitations of mapping an edge to a single path. Based on this assumption, [18] developed an MIP model and algorithms for finding such mappings. Unfortunately none of these works for priority mappings studied in this paper.

## 2 Related Work

Virtualization techniques have been investigated for big data processing [38] and database applications [7, 8, 39]. However, none of these has provided a systematic study of VNM, by modeling VNM as graph pattern matching. The only exception is [28], 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 for cloud computing in database applications.

Several models have been developed for VNM. (a) The VM placement problem (VMP, [13]) is to map a set of VMs onto an SN with constraints on node capacities. (b) Single-path VN embedding ( $VNE_{SP}$ , [33]) is to map a VN to an SN by a node-to-node injection 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  $VNE_{SP}$ , multi-path embedding ( $VNE_{MP}$ , [18, 40]) 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) While graph layout problems are similar to VN mapping, they do not have bandwidth constraints on edges but instead, impose certain topological constraints (see [20] for a survey). In contrast to our work, these models are studied for specific domains, and no previous work has studied generic models to support various VN requests that commonly arise in practice.

Very few complexity results are known for VNM. The only work we are aware of is [10], 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 [24]

for a survey). 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 [22, 23], with several intractability and approximation bounds established there. Those differ from this work in that either no constraints on links are considered [23], or graph simulation is adopted [22], 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.

### 3 Graph Pattern Matching Model

Below 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 [24].

#### 3.1 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 physical links (directly connected), 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  $G_S$ , where (1) the node set is  $\{a, b, \dots, f\}$ ; (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  $\{VM_1, VM_2, VM_3\}$ ; (2) the edge set is  $\{(VM_i, VM_j) \mid i, j = 1, 2, 3\}$ ; (3)  $f_{V_P}(VM_1) = 66$ ,  $f_{V_P}(VM_2) = 20$ ,  $f_{V_P}(VM_3) = 30$ ; 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*  $\rho$  from node  $u_0$  to  $u_n$  in an SN  $G_S$  is denoted as  $(u_0, u_1, \dots, u_n)$ , 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  $\rho$ . When it is clear from the context, we also use  $\rho$  to denote the set of edges on the path, i.e.,  $\{e_i \mid i \in [1, n]\}$ .

### 3.2 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 for each edge  $(v, v') \in E_P$ ,  $g_E(v, v')$  is a subset of  $P(v, v')$ , and  $r_E$  attaches a positive number to each pair  $(e, \rho)$  if  $e \in E_P$  and  $\rho \in g_E(e)$ . Intuitively,  $r_E(e, \rho)$  is the amount of resource of the physical path  $\rho$  allocated to the virtual link  $e$ .

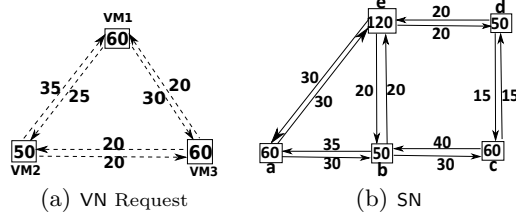
**VN requests.** A VN *request* to an SN  $G_S$  is a pair  $(G_P, \mathcal{C})$ , where  $G_P$  is a VN, and  $\mathcal{C}$  is a set of constraints such that for a pair  $((g_V, r_V), (g_E, r_E))$  of node and edge mappings from  $G_P$  to  $G_S$ , each constraint in  $\mathcal{C}$  has one of the forms below:

- (1) for each  $v \in V_P$ ,  $f_{V_P}(v) \leq r_V(v, g_V(v))$ ;
- (2) for each  $u \in V_S$ ,  $f_{V_S}(u) \geq \text{sum}(N(u))$ , 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 each  $e \in E_P$ ,  $f_{E_P}(e) \text{ op } \text{agg}(Q(e))$ , where  $Q(e)$  is  $\{\{r_E(e, \rho) \mid \rho \in g_E(e)\}\}$ , a bag collecting physical paths  $\rho$  that instantiate  $e$ ; here **op** is either the comparison operator  $\leq$  or  $\geq$ , and **agg**() is one of the aggregation functions **min**, **max** and **sum**;
- (4) for each  $e' \in E_S$ ,  $f_{E_S}(e') \geq \text{sum}(M(e'))$ , 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  $\rho$  containing  $e'$ ; and
- (5) for each  $e \in E_P$  and  $\rho \in g_E(e)$ ,  $r_E(e, \rho) \text{ op } \text{agg}(U(\rho))$  where  $U(\rho)$  is  $\{\{f_{E_S}(e') \mid e' \in \rho\}\}$ , a bag of all edges on a physical path that instantiate  $e$ .

Constraints in a VN request are classified as follows.

**Node constraints:** Constraints of form (1) or (2). 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. When  $u$  hosts at most one virtual node, i.e., if node sharing is not allowed, then  $|N(u)| \leq 1$ , where we use  $|N(u)|$  to denote the number of virtual nodes hosted by  $u$ .





**Fig. 2.** VN request and SN for case study

*Edge constraints:* Constraints of form (3), (4) or (5). Constraints of form (3) assure that when a virtual link  $e$  is mapped to a set of physical paths in the SN, those physical paths 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  $\rho$  in the set, the resource of  $\rho$  allocated to  $e$  must be consistent with the capacities of the physical links on  $\rho$ , *e.g.*, may not exceed the minimum bandwidth of the physical links on  $\rho$ .

**VNM.** We say that a VN request  $(G_P, \mathcal{C})$  can be *mapped to* an SN  $G_S$ , denoted by  $G_P \triangleright_{\mathcal{C}} G_S$ , 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  $G_P \triangleright_{\mathcal{C}} G_S$ .

## 4 Case study

All the VNM requirements in the Introduction (Section 1) can be expressed in our model, by treating VN request as a pattern and SN as a graph. Below we present a case study.

*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.*  $\text{VNM}_P$  can be captured as a VN request specified as  $(G_P, \mathcal{C})$ , where  $\mathcal{C}$  consists of (a) node constraints of forms (1) and (2), and (b) edge constraints of form (3) when **op** is  $\leq$  and **agg** is **max**, and form (5) when **op** is  $\leq$  and **agg** is **min**. 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  $\rho$ , the *bandwidth* of each edge on  $\rho$  is no less than that of  $e$ , *i.e.*,  $\rho$  suffices to serve any connection *individually*, including the one with the highest priority when  $\rho$  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, and map the virtual links to the shortest physical paths uniquely determined by the node mapping, *e.g.*,  $(VM_1, VM_2)$  is mapped to  $(b, a)$ .

*Case 3: Single-path VN embedding.* A  $VNE_{SP}$  request can be specified as  $(G_P, \mathcal{C})$ , where  $\mathcal{C}$  consists of (a) node constraints of forms (1) and (2), and (b) edge constraints of form (3) when  $op$  is  $\leq$  and  $agg$  is  $sum$ , and edge constraints of forms (4) and (5) when  $op$  is  $\leq$  and  $agg$  is  $min$ . It differs from  $VNM_P$  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,  $VNE_{SP}$  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  $VNE_{MP}$ ) can be expressed as a VN request. It is the same as  $VNE_{SP}$  except that a virtual link  $e$  can be mapped to a *set*  $g_E(e)$  of physical paths. When taken together, the paths in  $g_E(e)$  provide sufficient bandwidth required by  $e$ .

When node sharing is allowed in  $VNE_{SP}$ , *i.e.*, for single-path embedding with node sharing ( $VNE_{SP(NS)}$ ), 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  $u$ . Similarly, one can also specify multi-path VN embedding with node sharing ( $VNE_{MP(NS)}$ ).

*Example 4.* Consider the VN of Fig. 2(a), and the SN of Fig. 2(b). There is a  $VNE_{SP}$  from the VN to the SN, by mapping  $VM_1, VM_2, VM_3$  to  $a, b, e$ , 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  $VNE_{MP}$  from the VN to the SN, by mapping  $VM_1, VM_2$  and  $VM_3$  to  $a, c$  and  $e$ , respectively. For the virtual links,  $(VM_1, VM_2)$  can be mapped to the physical path  $(a, b, c)$ ,  $(VM_1, VM_3)$  to  $(a, e)$ , and  $(VM_3, VM_2)$  to two paths  $\rho_1 = (e, b, c)$  and  $\rho_2 = (e, d, c)$  with  $r_E((VM_3, VM_2), \rho_1) = 5$  and  $r_E((VM_3, VM_2), \rho_2) = 15$ ; 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  $f_{VS}(a, e) = 40$  and  $f_{VS}(e, a) = 50$ , then there exists a mapping from the VN to the SN that supports node sharing. Indeed, in this setting, one can map both  $VM_1, VM_2$  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  $(VM_1, VM_3)$  and  $(VM_2, VM_3)$  can be mapped to  $(e, a)$ .

*Case 4: Latency constrained mapping.* A  $VNM_L$  request is expressed as  $(G_P, \mathcal{C})$ , where  $\mathcal{C}$  consists of (a) node constraints of forms (1) and (2), and (b) edge constraints of form (3) when  $op$  is  $\geq$  and  $agg$  is  $min$ , and of form (5) when  $op$  is  $\geq$  and  $agg$  is  $sum$ . It is similar to  $VNE_{SP}$  except that when a virtual link  $e$  is

mapped to a physical path  $\rho$ , it requires  $\rho$  to satisfy the *latency* requirement of  $e$ , *i.e.*, the sum of the latencies of the edges on  $\rho$  does not exceed that of  $e$ .

*Example 5.* One can verify that there is no latency mapping of the VN shown 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:  $(VM_1, VM_2) = 50$ ,  $(VM_2, VM_1) = 55$ ,  $(VM_1, VM_3) = (VM_3, VM_1) = 120$  and  $(VM_2, VM_3) = (VM_3, VM_2) = 60$ , then there is a mapping from the VN to the SN. We can map  $VM_1, VM_2, VM_3$  to  $c, b, a$ , respectively, and map the edges to the shortest physical paths determined by the node mapping.

## 5 Complexity and Approximation

We next 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. Due to the space constraint, we defer the detailed proofs to [5].

### 5.1 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 1.** *The virtual network mapping 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.,  $VMP_{(NS)}$ ,  $VNM_{P(NS)}$ ,  $VNM_{L(NS)}$ ,  $VNE_{SP(NS)}$  and  $VNE_{MP(NS)}$  are NP-complete; and*
- (4) *it is NP-complete in the presence of edge constraints; i.e.,  $VNM_P$ ,  $VNM_L$ ,  $VNE_{SP}$  and  $VNE_{MP}$  are intractable.*

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

### 5.2 Approximation of Optimization Problems

In practice, one typically wants 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 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)  $h_E(g_E, r_E, e') = \sum_{e \in E_P, \rho \in g_E(e), e' \in \rho} r_E(e, \rho) / f_{E_S}(e')$  when the resource of physical links is bandwidth, and

(3) when latency is concerned,  $h_E(g_E, r_E, e')$  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 [17]. It is justified by Web hosting and cloud storage [12], which mainly sell CPU power or storage services of nodes, and by virtual network mapping, which also sells bandwidth of links [18]. It is also to serve cloud provision in virtualized data center networks [25], for which dynamic routing strategy (latency) is critical while routing congestion (bandwidth allocation) is 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.

**Complexity and Approximation.** We next study the minimum cost mapping problem for all cases given before. Having seen Theorem 1, it is not surprising that the optimization problem is intractable in most cases. This motivates us to study their 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 2.** *The minimum cost mapping problem is*

- (1) *in PTIME for VMP without node sharing; however, when node sharing is requested, i.e., for VMP<sub>(NS)</sub>, it becomes NP-complete and is APX-hard;*
- (2) *NP-complete and NPO-complete for VNM<sub>P</sub>, VNE<sub>SP</sub>, VNE<sub>MP</sub>, VNM<sub>L</sub>, VNM<sub>P(NS)</sub>, VNE<sub>SP(NS)</sub>, VNE<sub>MP(NS)</sub>, VNM<sub>L(NS)</sub>; And*
- (3) *APX-hard when there is a unique node mapping in the presence of edge*

constraints. In particular,  $\text{VNM}_P$  does not admit  $\ln(|V_P|)$ -approximation, unless  $P = NP$ .

The NPO-hardness results remain intact even when both VNs and SNs are DAGs.

Here NPO is the *class* of all NP optimization problems (cf. [11]). An NPO-complete problem is NP-hard to optimize, and is among the hardest optimization problems. APX is the *class* of problems that allow PTIME approximation algorithms with a constant approximation ratio (cf. [11]).

## 6 Computing Minimum Cost VNM

Theorem 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 ( $\text{VNM}_P$ ), 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  $\text{VNM}_P$ .

Prior algorithms for computing VNM (e.g., [40]) 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.

Below we first present auxiliary structures for SN (Section 6.1) and then develop an algorithm for minimizing VN patterns (Section 6.2). Finally, leveraging the auxiliary structures and minimization, we present our algorithm for minimum cost  $\text{VNM}_P$  (Section 6.3).

### 6.1 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  $(u, v) \in E_a$  iff there exists a path from  $u$  to  $v$  in  $G$ , (3)  $f_{E_a}(u, v) = \max\{\min\{\rho\} \mid \rho \text{ is a path from } u \text{ to } v \text{ in } G\}$ , where  $\min\{\rho\} = \min\{f_E(e) \mid e \text{ is an edge on } \rho\}$  in  $G$ , and (4)  $P_{E_a}(u, v)$  is a path  $\rho$  in  $G$  with  $\min\{\rho\} = f_{E_a}(u, v)$ .

---

*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.    $G_{aux} := \text{updateAuxGraph}(G_{aux}, G, v)$ ;
4. Remove edges  $(u, u')$  from  $G_{aux}$  having  $f_{E_a}(u, u') = 0$ ;
5. **return**  $G_{aux}$ .

**Procedure**  $\text{updateAuxGraph}(G_{aux}, G, v)$

*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.    $E_a := E_a \cup \{(u, v), (v, u)\}$ ;
3.    $\text{Assign}(v, u, G_{aux}); \text{Assign}(u, v, G_{aux})$ ;
4. **for each** edge  $(u, u')$  in  $G_{aux}$  having  $u, u' \in V_a$  **do**
5.    $h := \min\{f_{E_a}(u, v), f_{E_a}(v, u')\}$ ;
6.   **if**  $f_{E_a}(u, u') < h$  **then**
7.      $f_{E_a}(u, u') := h$ ;
8.      $P_{E_a}(u, u') := P_{E_a}(u, v) + P_{E_a}(v, u')$ ;
9.  $V_a := V_a \cup \{v\}$ ;  $f_{V_a}(v) := f_V(v)$ ;
10. **return**  $G_{aux}$ ;

---

**Fig. 3.** Algorithm  $\text{compAuxGraph}$

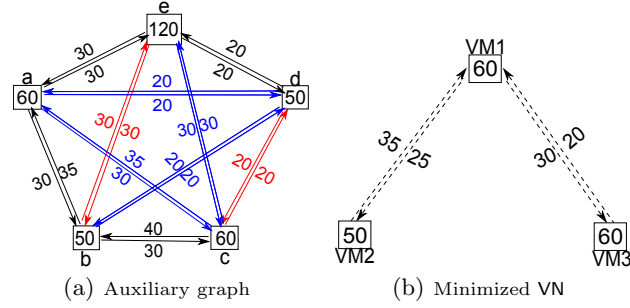
One can verify the following for priority mappings, which justifies the need for auxiliary graphs (see [5] for a proof).

**Proposition 3:** *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.  $\square$*

**Algorithm.** We next present an algorithm, referred to as  $\text{compAuxGraph}$ , 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  $\text{updateAuxGraph}$  (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 from  $G_{aux}$  (line 4) and 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  $\text{updateAuxGraph}$  returns the auxiliary graph of the subgraph of  $G$  with nodes  $V_a \cup \{v\}$ . It works as follows. For each node  $u$  in  $G_{aux}$ ,  $\text{updateAuxGraph}$  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  $\text{assign}$  (omitted due to the lack of space; lines



**Fig. 4.** Examples of auxiliary graphs and minimizing VNs

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).

*Example 6.* 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 they are in the SN of Fig. 2(b), since they are updated by procedure `updateAuxGraph` (lines 5-7). Moreover, there are now 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 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 derive the existence of an edge mapping.

**Correctness & complexity.** One can verify the following.

**Lemma 1.** *In `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.*

Lemma 1 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 immediately.

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  $|V_a| \leq |V|$ .

## 6.2 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  $G_S$  iff there exists another VNM from  $G_{P_2}$  to  $G_S$  with the same cost.

We can minimize an VN  $G_P$  in cubic-time:

**Theorem 4.** *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), which has 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` 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  $G_P^m$  ( $(u', u) \in E_P^m$ ) 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 7.* Consider the VN shown 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 shown in Fig. 2(a).

**Correctness & complexity.** One can verify the following.

**Lemma 2.** *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  $G_P$ .*



---

*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.  $G'_P(V_P, E'_P, f_{V_P}, f_{E'_P}, \emptyset) := \text{compAuxGraph}(G_P)$ ;
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.      $G_P^m := \text{UpdateVN}(G_P^m, v, G'_P)$ ;
5. **return**  $G_P^m$ .

**Procedure**  $\text{UpdateVN}(G_P^m, v, G'_P)$ .

*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.     **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.         **then**  $E_P^m := E_P^m \cup \{(v, u)\}$ ;  $f_{E_P^m}(v, u) := f_{E'_P}(v, u)$ ;
5.     **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.         **then**  $E_P^m := E_P^m \cup \{(u, v)\}$ ;  $f_{E_P^m}(u, v) := f_{E'_P}(u, v)$ ;
7. **return**  $G_P^m$ ;

---

**Fig. 5.** Algorithm minVN for minimizing VNs

By Lemma 2, we can show that procedure  $\text{updateVN}$  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  $\text{compAuxGraph}$  runs in  $O(|V_P|^3)$  time. (2) Procedure  $\text{updateVN}$  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.

These complete the proof of Theorem 4 and provide an algorithm for minimizing VN patterns.

### 6.3 Computing Minimum Cost Priority Mappings

We are now ready to present our algorithm for computing priority mappings, denoted by  $\text{compVNM}$  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 (no more than 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  $G_S$  with algorithm  $\text{compAuxGraph}$ , to validate a node mapping without traversing the entire  $G_S$  (line 3). The algorithm builds a low cost node mapping by

---

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

**Procedure**  $\text{backTrackMap}(v, S, \text{backS}, i, k)$

*Input:* Node  $v$ , node sets  $S$  and  $\text{backS}$ , 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  $\text{Valid}(v, u, S) = \text{true}$  **then**
  3.    $g_V(v) := u; \quad r_V(v) := f_{V_P^m}(v); \quad S := S \cup \{v\};$
  4.   **return**  $(g_V, r_V, S);$
  5. **for each**  $v' \in S \setminus \text{backS}$  **do**
  6.   **if**  $\text{Valid}(v, g_V(v'), S \setminus \{v'\})$  **then**
  7.      $g_V(v) := g_V(v'); \quad r_V(v) := f_{V_P^m}(v); \quad S := S \cup \{v\} \setminus \{v'\};$
  8.   **if**  $\text{backTrackMap}(v', S, \text{backS} \cup \{v\}, i + 1, k)$  **then**
  9.     **return**  $(g_V, r_V, S);$
  10.    $S := S \setminus \{v\} \cup \{v'\}; \quad g_V(v') := g_V(v);$
  11. **return null**;
- 

**Fig. 6.** Algorithm  $\text{compVNM}$  for priority mappings

inspecting nodes one by one, via procedure  $\text{backTrackMap}$  (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  $((g_V, r_V, S) \neq \text{null}$ , line 6). If so, it finds the corresponding edge mapping by calling procedure  $\text{identifyEdgeMap}$  (omitted; line 7). With  $G_{aux}$ , the edge mapping can be found in  $O(|E_P^m|)$  time (Lemma 1). A VNM is returned if there exists one (line 8).

We next present procedure  $\text{backTrackMap}$ . Given a new node  $v$ , a node set  $S$  for which mappings are already identified, a node set  $\text{backS}$ , 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 [28]. The backtrack depth is bounded by  $k$ . It uses  $i$  to keep track of the current backtrack depth, and  $\text{backS}$  to record the set of nodes backtracked. In contrast to [28] 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  $G_{aux}$ .

More specifically, if the current backtrack depth  $i > k$ , then the procedure returns **null** since a node mapping cannot be found (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 8.* 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  $(VM_2, c)$  make a valid node mapping for the the subgraph of  $V_P^m$  with nodes  $VM_1$  and  $VM_2$ . Similarly, a candidate mapping node  $b$  is found for  $VM_3$ . 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.*,  $(VM_1, VM_2)$  is mapped to  $P(a, b)$  (see Example 6). 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 and 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 by the experimental study presented in the next section.

**Remark.** One can extend algorithm **compVNM** for priority mappings with node sharing, denoted by **compVNM<sub>NS</sub>**, by simply allowing multiple virtual nodes in  $G_P$  to be mapped to the same node in  $G_S$  in **Valid**.

## 7 Experimental Study

We next present an experimental study of our techniques for computing virtual network priority mappings (**VNM<sub>P</sub>**). We conducted two sets of experiments to evaluate: (1) the effectiveness of **VNM<sub>P</sub>** vs. conventional virtual network embedding (**VNM<sub>SP</sub>**) and (2) the efficiency of our algorithms.

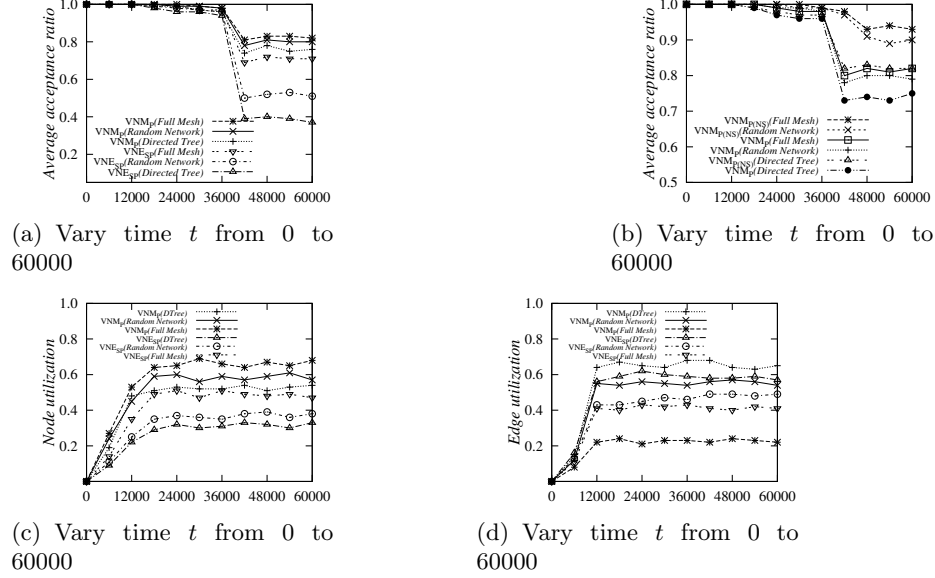
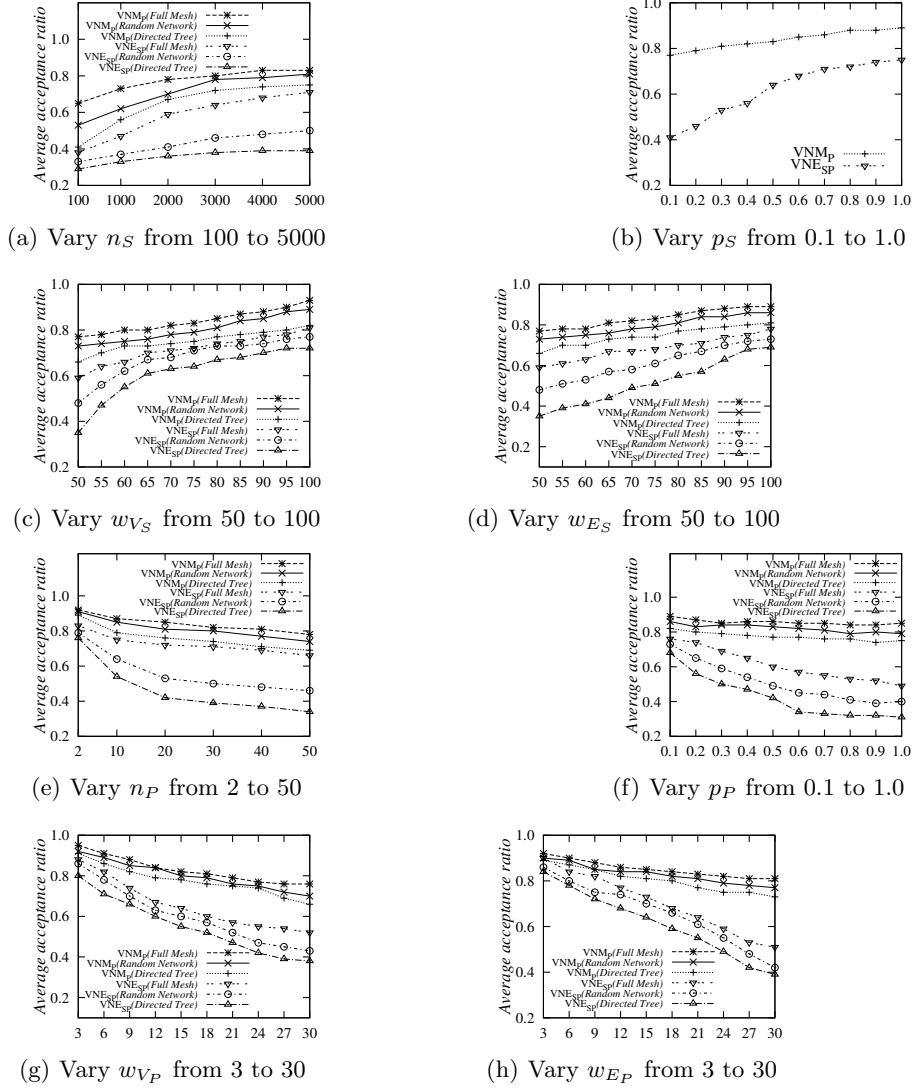


Fig. 7. Mapping quality over time

**Experimental setting.** Following the tradition of virtual network topology research (*e.g.*, [26,28,29]), we used the following datasets that simulates 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  $(u, v)$  iff 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 exists an edge  $(u, v)$ . (c) Random networks, in which for any two 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](http://en.wikipedia.org/wiki/Network_topology)).

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  $\lambda$  requests per time unit, as commonly adopted by network community [18, 28, 40]. Each one has exponentially many distributed 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 compVNM<sub>N5</sub> for computing VNM<sub>P</sub>, without node sharing



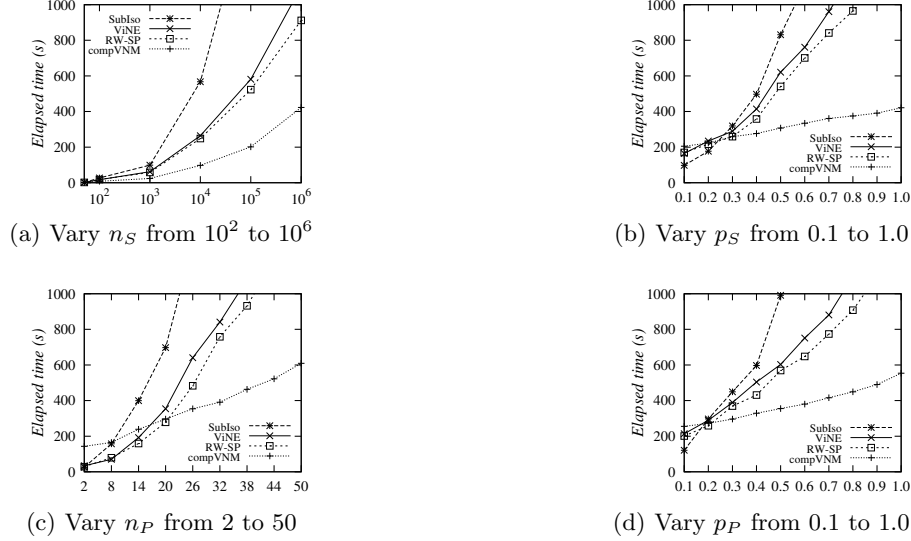
**Fig. 8.** Mapping quality over SN and VN

and with node sharing; (b) Algorithms Sublso [28], ViNE [18] and RW-SP [16] for computing  $VNE_{SP}$ ; (c) Algorithm ViNE<sub>NS</sub> that extends ViNE for computing  $VNE_{SP}$  with node sharing; and (d) the graph generator for producing VNs and SNs.

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.

**Table 2.** Experimental 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	$w_{E_P}$	$w_{E_S}$



**Fig. 9.** Mapping efficiency and scalability evaluation

**Experimental results.** We next report our findings. In all 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 **compVNM<sub>NS</sub>**. We adopted algorithm **ViNE** for **VNE<sub>SP</sub>** when comparing the mapping quality of **VNM<sub>P</sub>** with **VNE<sub>SP</sub>**. We summarize the tested factors in Table 2.

**Exp-1: Mapping quality.** In the first set of experiments, we evaluated (1) the mapping quality of **VNM<sub>P</sub>** vs. **VNE<sub>SP</sub>**; (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 [18, 28, 40], to evaluate the mapping quality. Given a time stamp  $t$ , AR is defined as:

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

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  $w_{V_P}, w_{E_P}$  in  $[3, 30]$ . For SNs, we fixed  $n_S = 5000$ , and  $w_{V_S}, w_{E_S}$  in  $[50, 100]$ . We took  $n_S = 5000$  since *medium-size ISPs* have about 500 nodes only [28]. We varied  $t$  from 0 to 60,000.

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 42,000. This is because initially there exists no workload in the SNs; the SNs are fully loaded when  $t$  reaches 42,000, since only a certain amount of work could be handled by the SNs. (ii) The AR of  $VNM_P$  is consistently higher than that of  $VNE_{SP}$  (in the range of [11%, 39%]) in all the cases. (iii) The impact of network topologies on the AR of  $VNM_P$  is much smaller than that of  $VNE_{SP}$ . Indeed, the stable AR for  $VNM_P$  is in the range of [76%, 82%], while for  $VNE_{SP}$ , it is around 37% and 71% on directed tree and full mesh networks, respectively.

Figure 7(b) shows the AR of  $VNM_P$ , with node sharing or not, 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  $VNE_{SP}$  (we did not report it here due to the space constraint). This shows 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  $t = 40000$ s in both Figures 7(a) and 7(b). 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  $t = 60,000$ , and VN with  $n_P = 50$ ,  $p_P = 0.5$  and  $w_{V_P} = w_{E_P} = 30$ , and we varied  $n_S$  from 100 to 5,000,  $p_S$  from 0.1 to 1.0, and  $w_{V_S}, w_{E_S}$  from 50 to 100. The test of  $p_S$  can be conducted on SNs of random networks only.

The results are reported in Figures 8(a), 8(b), 8(c) and 8(d). (i) These figures show that 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 larger  $w_{V_P}$  and  $w_{E_P}$  are, there are larger capacities in the nodes and links of the SNs, respectively. Hence, the SN can handle more requests when any of these four factors is increased, and therefore, so is the AR. (ii) The AR of  $VNM_P$  is consistently higher than the AR of  $VNE_{SP}$  in all the cases, up to 37%. (iii) The AR of  $VNM_P$  is less sensitive to network topologies than the AR of  $VNE_{SP}$ .

(3) To evaluate the impact of VN requests on AR, we fixed  $t = 60,000$ , and SNs with  $n_S = 5000$ ,  $p_S = 0.5$  and  $w_{V_S} = w_{E_S} = 100$ , while varying  $n_P$  from 2 to 50,  $p_P$  from 0.1 to 1.0, and  $w_{V_P}, w_{E_P}$  from 3 to 30. Again the test of  $p_P$  can be conducted on the VNs of random networks only.

The results are shown in Figures 8(e), 8(f), 8(g) and 8(h). (i) The results tell us that AR decreases *w.r.t.*  $n_P$ ,  $p_P$ ,  $w_{V_P}$  and  $w_{E_P}$ . Indeed, (a) the larger  $n_P$  is,

more machines are requested by the VNs; (b) the larger  $p_P$  is, more links are demanded; and (c) the larger  $w_{V_P}$  and  $w_{E_P}$  are, 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  $\text{VNM}_P$  is consistently higher than the AR of  $\text{VNE}_{SP}$  in all the cases, up to 33%. (iii) The AR of  $\text{VNM}_P$  is less sensitive to network topologies than that of  $\text{VNE}_{SP}$ .

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

(i) The average resource utilization of substrate nodes is shown in Fig. 10(b). It shows the following. (a) Node utilization of SNs becomes consistent after  $t = 24000$ . (b) Node utilization of full mesh is higher than that of random network, followed by directed tree, for both  $\text{VNM}_P$  mappings and  $\text{VNE}_{SP}$  mappings. (c) For each type of the three SN topologies, the node utilization of  $\text{VNM}_P$  is higher than that of  $\text{VNE}_{SP}$ .

(ii) Figure 10(c) shows the average edge utilization. It tells us the followings. (a) After  $t = 12000$ , the average edge utilization becomes consistency, no matter on which kind of SNs. (b) Priority mapping over directed tree gains the highest edge mapping; however, it is the lowest of all cases while over full mesh. (c) Generally, the impact of network topologies on the edge utilization for  $\text{VNM}_P$  is larger than that for  $\text{VNE}_{SP}$ .

**Exp-2: Mapping efficiency.** In this set of experiments, we evaluated the efficiency of our algorithm  $\text{compVNM}$  for  $\text{VNM}_P$  vs. algorithms  $\text{Sublso}$  [28],  $\text{ViNE}$  [18] and  $\text{RW-SP}$  [16] for  $\text{VNE}_{SP}$ . We used large random networks in the experiments. Due to the space constraint, 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 complexity analysis.

(1) To evaluate the impact of SNs, we fixed VN requests with  $n_V = 25$ ,  $p_P = 0.5$ ,  $w_{V_P} = w_{E_P} = 30$ , and varied  $n_S$  from  $10^2$  to  $10^6$  and  $p_S$  from 0.1 to 1.0, respectively. The results are shown in Figures 9(a) and 9(b), respectively.

(2) To evaluate the impact of VNs, we fixed VNs with  $n_S = 500,000$ ,  $p_S = 0.5$ ,  $w_{V_S} = w_{E_S} = 100$ , and varied  $n_V$  from 2 to 50 and  $p_V$  from 0.1 to 1.0, respectively. The results are reported in Figures 9(c) and 9(d), respectively.

These results show 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  $\text{compVNM}$  is efficient: it took only around 420s for SNs with 1 million nodes. (iii) It outperforms the other three algorithms for  $\text{VNE}_{SP}$  in almost all the cases. Indeed,  $\text{compVNM}$  is about twice faster than the other algorithms. While it took  $\text{compVNM}$  less than 600s for  $n_S = 10^6$ ,  $p_S = 1.0$ ,  $n_P = 50$  or  $p_P = 1.0$  in Figures 9(a), 9(b), 9(c) and 9(d), respectively, the other algorithms took at least 900s, or could not run to completion.

**Summary.** From these experimental results we find the following. (a) Priority mapping ( $\text{VNM}_P$ ) proposed in this work is able to find high-quality mappings, and has higher acceptance ratio than previous models (*e.g.*,  $\text{VNE}_{SP}$ ), typically



up from 11% to 39%. (b) Priority mapping is less sensitive to network topologies. (c) Node sharing improves mapping quality, typically in the range [8%, 11%]. (d) The average node and edge utilization of  $\text{VNM}_P$  is much higher than that of  $\text{VNE}_{SP}$ . (e) Our algorithm for computing priority mapping is efficient, *e.g.*, it took 400s for SNs with  $10^6$  nodes, and it substantially outperforms previous algorithms for  $\text{VNE}_{SP}$ .

## 8 Conclusion

We have proposed a model to express various VN requests found in practice, based on graph pattern matching, and we have shown that the model is able to express VNMs commonly found in practice. We have also established a number of intractability and approximation hardness results in various practical VNM settings. These are among the first efforts to settle fundamental problems for virtual network mapping. A few topics are targeted for future work. We are developing practical heuristic algorithms and optimization techniques for VNM. We are also exploring techniques for processing VN requests for different applications, as well as their use in graph pattern matching.

**Acknowledgments.** Fan and Cao are supported in part by NSFC 61133002, 973 Program 2014CB340302, Shenzhen Peacock Program 1105100030834361, Guangdong Innovative Research Team Program 2011D005, EPSRC EP/J015377/1 and EP/M025268/1, 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. <http://www.bigswitch.com/>.
2. <http://aws.amazon.com/ec2/>.
3. <http://www.vmware.com/solutions/datacenter/>.
4. <http://frenzy.ivic.org.cn/>.
5. <http://homepages.inf.ed.ac.uk/s1165433/papers/vnm-full.pdf>.
6. <http://www.isi.edu/xbone/>.
7. A. Abounaga, C. Amza, and K. Salem. Virtualization and databases: state of the art and research challenges. In *EDBT*, 2008.
8. A. Abounaga, K. Salem, A. Soror, U. Minhas, P. Kokosielis, and S. Kamath. Deploying database appliances in the cloud. *IEEE Data Eng. Bull.*, 32(1):13–20, 2009.
9. D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: current state and future opportunities. In *EDBT*, 2011.
10. D. Andersen. Theoretical approaches to node assignment. *Unpublished Manuscript*, 2002.
11. G. Ausiello. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Verlag, 1999.
12. A. C. Bavier, N. Feamster, M. Huang, L. L. Peterson, and J. Rexford. In VINI veritas: realistic and controlled network experimentation. In *SIGCOMM*, 2006.

13. N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *IM*, 2007.
14. M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. In *SODA*, 1998.
15. C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *SODA*, 2000.
16. X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang. Virtual network embedding through topology-aware node ranking. *SIGCOMM CCR*, 41(2):38–47, 2011.
17. N. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.
18. N. Chowdhury, M. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM*, 2009.
19. T. Cormen. *Introduction to algorithms*. The MIT press, 2001.
20. J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *CSUR*, 34(3):313–356, 2002.
21. W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *ICDE*, 2011.
22. W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. In *VLDB*, 2010.
23. W. Fan, J. Li, S. Ma, H. Wang, and Y. Wu. Graph homomorphism revisited for graph matching. *VLDB*, 2010.
24. B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS.*, 2006.
25. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.
26. I. Houidi, W. Louati, and D. Zeglache. A distributed virtual network mapping algorithm. In *ICC*, 2008.
27. W.-H. Hsu and Y.-P. Shieh. Virtual network mapping algorithm in the cloud infrastructure. *J.NCA*, 2013.
28. J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *SIGCOMM workshop VISA*, 2009.
29. J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. *Washington University, TR WUCSE-2006*, 35, 2006.
30. N. Megiddo. On the complexity of linear programming. In *Advances in Economic Theory: Fifth World Congress*, 1987.
31. J. Munkres. Algorithms for the assignment and transportation problems. *SIAM*, 5:32–38, 1957.
32. W. Reinhardt. Advance reservation of network resources for multimedia applications. In *IWACA*, 1994.
33. R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *SIGCOMM CCR*, 33:81, 2003.
34. M. S. Schlansker, J.-f. Collard, and R. Kumar. Prioritized polling for virtual network interfaces, Jan. 29 2013. US Patent 8,364,874.
35. P. Shivam, A. Demberel, P. Gunda, D. E. Irwin, L. E. Grit, A. R. Yumerefendi, S. Babu, and J. S. Chase. Automated and on-demand provisioning of virtual machines for database applications. In *SIGMOD*, 2007.
36. D. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1), 1993.

37. A. A. Soror, U. F. Minhas, A. Abounaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. *TODS*, 35(1), 2010.
38. O. Trelles, P. Prins, M. Snir, and R. C. Jansen. Big data, but are we ready? *Nature reviews Genetics*, 12(3), 2011.
39. P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigümüs. Intelligent management of virtualized resources for database systems in cloud environment. In *ICDE*, 2011.
40. M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM CCR*, 38(2):17–29, 2008.
41. Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, 2006.

## Appendix A: Proofs

### Proof of Theorem 1

Given a VN request  $(G_P, \mathcal{C})$  in which  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$  and an SN  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$ . We prove results of theorem 1 as follows.

**Proof:** (1) To show the upper bound, we first provide a specific NP algorithm for  $\text{VNE}_{\text{MP}}$  and  $\text{VNE}_{\text{MP}(\text{NS})}$ . Given a VN request and an SN, the specific algorithm first guesses a  $g_V$ , and then checks whether there exist  $r_V, g_E$  and  $r_E$  such that  $(g_V, r_V)$  and  $(g_E, r_E)$  satisfy the constraints carried by VN request, by reduction to the fractional multi-commodity flow problem which is in PTIME [19].

For cases other than  $\text{VNE}_{\text{MP}}$  and  $\text{VNE}_{\text{MP}(\text{NS})}$ , we give a unified algorithm consisting of three steps as follows.

- (i) Guess a node mapping function  $g_V$  and an edge mapping function  $g_E$  of VN on the SN.
- (ii) Check whether there exists  $r_V$  and  $r_E$  such that  $(g_V, r_V)$  and  $(g_E, r_E)$  make node and edge mappings that satisfy the constraints set  $\mathcal{C}$ . It is sure that the existence of  $r_V$  can be checked in  $O(|V_P|)$ -time. We now claim that the checking of edge constraints can be formulated as a linear (rational number) programming problem with  $r_E(e, \rho)$  as its variables.
  - (a) For constraints of form 3, if  $\text{agg} = \text{"max"}$ , then the constraints can be formulated as a set of inequality with each one in the form  $f_{E_S}(e) \text{ op } r_E(e, \rho)$  ( $\forall e \in g_E(e)$ ). While if  $\text{agg} = \text{"sum"}$ , the constraints can be formulated as one inequality in the form  $f_{E_S}(e) \text{ op } \sum_{\forall e \in g_E(e)} \{r_E(e, \rho)\}$ .
  - (b) For constraints of form 4, 5, the formulations are similar to constraints of form 3. For example, constraints in form 4 can be formulated similarly to form 3 with settings  $\text{op} = \text{"\ge"}$  and  $\text{agg} = \text{"sum"}$ .

As linear (rational number) programming problem can be done in time polynomial in  $|V_P|$  [30], the existence of edge mapping can be check in PTIME.

Thus the checking can be done in PTIME, *i.e.*, the above procedures form into an NP algorithm for VNM problem.

(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 only node constraints and without node sharing. Let  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$  and  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$ . The algorithm consists of three steps:

- (a) It builds a bipartite graph  $G_B = (N_L, N_R, E_B)$ , where  $N_L = V_P$ ,  $N_R = V_S$  and  $E = \{(u, u') \mid u \in V_P, u' \in V_S, f_{V_P}(u) \leq f_{V_S}(u')\}$ .
- (b) It finds a maximum bipartite matching of  $G_B$  (*i.e.*, an injective function  $f_B$

from  $N_L$  to  $N_R$ ) in  $O(|E_B|(|N_L|+|N_R|))$  time [19]. If such  $f_B$  does not exist, it returns ‘NO’, as one can verify that there is no VMP from the VN request to the SN in this case.

(c) Otherwise, it constructs a VMP from the VN request to the SN satisfying constraints specified by  $\mathcal{C}$ , such that  $g_V(v) = f_B(v)$ ,  $r_V(v, g_V(v)) = f_{V_P}(v)$  (for  $\forall v \in V_P$ ). The calculated functions  $g_V$  and  $r_V$  satisfy constraints of form 1 for  $f_{V_P}(v) = r_V(v, g_V(v))$ , and those of form 2 since  $\text{sum}(N(u))$  is 0 or  $f_{V_P}(v)$  (where  $g_V(v) = f_B(v) = u$ ).

(3) To prove all cases with node sharing are NP-complete, we have only to show that the virtual machine placement with node sharing ( $\text{VMP}_{(\text{NS})}$ ) is NP-complete, as the latter one is a special case of the other cases such as  $\text{VNM}_{\text{P}(\text{NS})}$ ,  $\text{VNM}_{\text{L}(\text{NS})}$ ,  $\text{VNE}_{\text{SP}(\text{NS})}$  and  $\text{VNE}_{\text{MP}(\text{NS})}$ , when edge constraints are absent.

We conduct a reduction from the Bin Packing problem. Recall that the instance of Bin Packing problem consists of a finite set  $U = \{u_1, u_2, \dots, u_n\}$  of items, a size  $s(u) \in \mathbb{Z}^+$  for each  $u \in U$ , a positive integer bin capacity  $B$  and a positive integer  $K$  which denotes the number of bins. The Bin Packing problem is to ask whether there is a partition of  $U$  in to disjoint sets  $U_1, \dots, U_k$  such that the sum of the sizes of the items in each  $U_i$  is  $B$  or less. We called a partition of  $U$  a valid partition, if the constraints are satisfied, *i.e.*, items can be partitioned and packed by the  $K$  bins. Given an arbitrary instance  $\text{I}_B$  of the Bin Packing problem, we construct an instance  $\text{I}_M$  of  $\text{VMP}_{(\text{NS})}$  such that there exists a valid partition of  $U$  in  $\text{I}_B$  for instance  $\text{I}_B$  iff there exists an mapping for  $\text{I}_M$ . We give constructions of  $\text{I}_M$  below:

(a) VN request  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$  and  $\mathcal{C}$ :

$$\begin{aligned} V_P &: \{v_1, v_2, \dots, v_n\}; \\ E_P &: \emptyset; \\ f_{V_P} &: \forall i \in \mathbb{N}^+, i \leq n, f_{V_P}(v_i) = s(u_i); \\ f_{E_P} &: \emptyset; \\ \mathcal{C} &: \text{Constraints of form 1 and form 2;} \end{aligned}$$

(b) SN  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$ :

$$\begin{aligned} V_S &: \{v'_1, v'_2, \dots, v'_K\} \\ E_S &: \emptyset; \\ f_{V_S} &: \forall j \in \mathbb{N}^+, j \leq K, f_{V_S}(v'_j) = B; \\ f_{E_S} &: \emptyset; \end{aligned}$$

It is obviously that the above construction of  $\text{I}_M$  can be finished in PTIME, and there is a valid partition of  $U$  of instance  $\text{I}_B$ , iff there is a VMP of instance  $\text{I}_M$ , from the VN request to the SN with satisfying the constraints.

For one thing, if there is a valid partition of  $U$  in  $\mathbf{l}_B$ , there must be a VMP for  $\mathbf{l}_M$ . We present the VMP  $(g_V, r_V)$  as follows:

$$\begin{aligned} g_V &: \forall i \in \mathbb{N}^+, i \leq n, \text{ if } u_i \text{ is partitioned into } U'_k, \text{ then } g_V(v_i) = v'_{k'}; \\ r_V &: \forall i \in \{1, 2, \dots, n\}, r_V(v_i) = f_{V_P}(v_i); \end{aligned}$$

As in  $\mathbf{l}_B$ ,  $\sum_{\forall i, u_i \rightarrow U_{k'}} \{s(u_i)\} \leq B$ , thus the VMP for  $\mathbf{l}_M$  satisfies the constraints of form 1 and 2.

For another thing, if there is a VMP for instance  $\mathbf{l}_M$ , *i.e.*, there exists a node mapping  $(g_V, r_V)$  from the VN to the SN constructed above and satisfies the constraints, then there must exist a valid partition of  $U$  in instance  $\mathbf{l}_B$ . We present the partition as follows:  $\forall i \in \{1, 2, \dots, n\}$ , if  $g_V(v_i) = v'_{k'}$  ( $k' \in \{1, 2, \dots, K\}$ ), we partition the item  $u_i$  to the set  $U_{k'}$  which will be packed by the  $k'$ th bin. It is obviously that the partition of  $U$  is a valid partition for  $\mathbf{l}_B$  for a substrate node to which some virtual nodes are mapped must have enough capacity to accommodate those virtual nodes, which leads the validity of the partition for  $\mathbf{l}_B$ .

Thus VMP with node sharing is NP-hard as Bin Packing is NP-complete [30]. Because it is already proved in NP, it is a NP-complete problem.

(4) We next prove that  $\text{VNM}_{SP}$ ,  $\text{VNM}_L$  and  $\text{VNM}_P$  are NP-complete problem by reducing from PARTITION, Subgraph Isomorphism and X3C problems respectively.

(a) We first show that the  $\text{VNE}_{SP}$  is NP-complete by reducing from PARTITION, which is a NP-complete problem [30].

The PARTITION problem is to judge, when given a finite set  $C$  and a “size”  $s(a) \in \mathbb{Z}^+$  for each  $a \in C$ , whether there exists a subset  $C' \subseteq C$  such that  $\sum_{a \in C'} s(a) = \sum_{a \in C - C'} s(a)$ . Given an instance  $\mathbf{l}_P$  of the PARTITION problem, *i.e.*, a finite set  $C = \{a_1, a_2, \dots, a_n\}$  and weight function  $s(a_i) \in \mathbb{Z}^+$ , we construct an instance  $\mathbf{l}_M$  of  $\text{VNM}_{SP}$ , such that there exists a partition of  $\mathbf{l}_P$  iff there is a mapping of  $\mathbf{l}_M$ . We present  $\mathbf{l}_M$  corresponding to  $\mathbf{l}_P$ , as follows:

(1) The VN request  $(G_P, \mathcal{C})$  in which  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$  is defined as follows:

$$\begin{aligned} V_P &: \{v_1, v_2, \dots, v_n, T_P\}; \\ E_P &: \{(T_P, v_i) \mid \forall i \in \mathbb{Z}^+, i \leq n\}; \\ f_{V_P} &: \forall v_i \in V_P, f_{V_P}(v_i) = z_i, f_{V_P}(T_P) = z_{n+1}, \text{ in which } \forall i, j \in \{1, 2, \dots, n\}, \text{ if } \\ &\quad i > j, \text{ then } z_i > z_j \text{ } (z_i, z_j \in \mathbb{Z}^+); \\ f_{E_P} &: \forall i \in \{1, 2, \dots, n\}, f_{E_P}(T_P, v_i) = s(a_i); \end{aligned}$$

(2) The SN  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$  is defined as follows:

$$\begin{aligned}
V_S &: \{u_1^L, u_2^L, \dots, u_n^L, u_1^R, u_2^R, \dots, u_n^R, T_S\}; \\
E_S &: \{(u_{i+1}^L, u_i^L) \mid \forall i \in \{1, \dots, n-1\}\} \cup \{(T_S, u_n^L)\} \cup \{(T_S, u_n^R)\} \cup \{(s_{i+1}^R, s_i^R) \mid \forall i \in \{1, 2, \dots, n-1\}\}; \\
f_{V_S} &: f_{V_S}(u_i^L) = f_{V_S}(u_i^R) = f_{V_P}(v_i) = z_i \ (\forall i \in \{1, 2, \dots, n\}), f_{V_S}(T_S) = z_{n+1}; \\
f_{E_S} &: f_{E_S}(T_S, u_n^L) = f_{V_S}(T_S, u_n^R) = \frac{A_0}{2}, \text{ in which } A_0 = \sum_{i=1}^n s(a_i). \text{ For any other} \\
&\text{edge } e \in E_S, f_{E_S}(e) = +\infty;
\end{aligned}$$

It is obviously that the above construction can be finished in PTIME. We now verify that instance  $\mathbf{l_B}$  has a partition iff  $\mathbf{l_M}$  has a single-path embedding  $(g_V, r_V, g_E, r_E)$ .

In fact, if there is a partition of  $\mathbf{l_B}$ , then one can verify that the following mapping is a valid single-path embedding:

$$\begin{aligned}
g_V &: \text{for each item } a_i \in C, \text{ if } a_i \in C' \text{ in the partition of } \mathbf{l_B}, \text{ then } g_V(a_i) = u_i^L; \\
&\text{otherwise } g_V(v_i) = u_i^R; \\
r_V &: \text{for each } v \in V_P, r_V(v) = f_{V_P}(v); \\
g_E &: \forall i \in \{1, 2, \dots, n\}, g_E(T_P, v_i) = (T_S, g_V(v_i)); \\
r_E &: \text{for each edge } (T_P, v_i) \text{ in } E_P, r_E((T_P, v_i), g_E(T_P, v_i)) = f_{E_S}(T_P, v_i);
\end{aligned}$$

For the other direction, when there exists a single-path embedding  $(g_V, r_V, g_E, r_E)$  for instance  $\mathbf{l_M}$ , one can verify that the following subset  $C'$  of  $C$  in instance  $\mathbf{l_B}$  forms a valid partition:  $\forall i \in \{1, 2, \dots, n\}$ , if  $g_V(v_i) = u_i^L$ , then  $a_i \in C'$ .

Thus the single-path embedding problem ( $\mathbf{VNE_{SP}}$ ) is NP-hard. As we've already proved that it is in NP, thus  $\mathbf{VNE_{SP}}$  is NP-complete.

(b) We then show that  $\mathbf{VNM_L}$  is NP-complete. In fact, the **Subgraph Isomorphism** problem is a special case of  $\mathbf{VNM_L}$  when the latency requirements on virtual links and latency on physical links are all the same, *e.g.*, a constant  $a$ . As **Subgraph Isomorphism** is NP-hard [30], we get the result that  $\mathbf{VNM_L}$  is NP-hard. Thus it NP-complete as we've already showed that it is in NP.

(c) We next conduct a reduction from the NP-complete problem **X3C** (.cf [30]) to show that  $\mathbf{VNM_P}$  is NP-hard.

Recall that the instance  $I_C$  of the **X3C** problem is: given a finite set  $S = \{x_1, x_2, \dots, x_{3q}\}$ , and a collection  $C$  of 3-element subsets of  $X$ , *i.e.*,  $C = \{C_1, C_2, \dots, C_n\}$  of which  $C_i = \{x_{i1}, x_{i2}, x_{i3}\}$  ( $\forall j \in \{1, 2, 3\}, x_{ij} \in S$ ). The **X3C** problem is to determine whether  $C$  contains an *exact cover* for  $X$ , *i.e.*, whether there exists a subset  $C' \subseteq C$  such that every element  $x_i$  of  $X$  occurs in exactly one member of  $C'$ .

Given a instance  $\mathbf{l_C}$  of the **X3C** problem, we now construct an instance of  $\mathbf{VNM_P}$  problem, denoted by  $\mathbf{l_P}$ , *i.e.*, a **VN** and an **SN**, such that there is a

mapping  $(g_V, r_V, g_E, r_E)$  of VN to SN with satisfying the priority mapping constraints of  $\mathcal{C}$  iff there is an *exact cover* of  $\mathsf{I}_{\mathcal{C}}$ . We present the constructed instance  $\mathsf{I}_{\mathcal{P}}$  by the following:

(1) VN  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$ :

$$\begin{aligned} V_P &: \{v_{11}, v_{12}, v_{13}, v_{21}, \dots, v_{q1}, v_{q2}, v_{q3}, v_1^C, v_2^C, \dots, v_q^C\}; \\ E_P &: \{(v_i^C, v_{ij}) \mid \forall i \in \{1, 2, \dots, q\}, j \in \{1, 2, 3\}\}; \\ f_{V_P} &: \forall v \in V_P, f_{V_P}(v) = 0; \\ f_{E_P} &: \forall e \in E_P, f_{E_P}(e) = a, \text{ in which } a \text{ is a positive constant;} \end{aligned}$$

(2) SN  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$ :

$$\begin{aligned} V_S &: \{u_{11}, u_{12}, u_{13}, u_{21}, \dots, u_{q1}, u_{q2}, u_{q3}, u_1^C, u_2^C, \dots, u_n^C\}; \\ E_S &: \{(u_i^C, u_{i[1,2,3]}) \mid \forall i \in \{1, 2, \dots, q\}\}; \\ f_{V_S} &: \forall u \in V_S, f_{V_S}(u) = +\infty; \\ f_{E_S} &: \forall e \in E_S, f_{E_S}(e) = a; \end{aligned}$$

It is clear that the construction of  $\mathsf{I}_{\mathcal{P}}$  can be finished in PTIME. One can find that,  $v_i^C$  in  $V_P$  can be only mapped to one node in the set  $\{u_j^C \mid \forall j \in \{1, 2, \dots, n\}\}$ , and  $v_{ik}$  in  $V_P$  can be only mapped to  $u_{jl}$  in  $V_S$  ( $k, l \in \{1, 2, 3\}$ ). In fact,  $G_P$  encodes an *exact cover* of  $\mathsf{I}_{\mathcal{C}}$  and  $G_S$  encodes  $\mathsf{I}_{\mathcal{C}}$ . We next show that there exists a priority mapping  $(g_V, r_V, g_E, r_E)$  iff there exists an *exact cover* of  $\mathsf{I}_{\mathcal{C}}$ .

On one hand, if there is a priority mapping  $(g_V, r_V, g_E, r_E)$  of VN to the SN in instance  $\mathsf{I}_{\mathcal{P}}$ , then we claim that  $\{g_V(v_i^C) \mid \forall i \in \{1, 2, \dots, q\}\}$  is an *exact cover* of  $\mathsf{I}_{\mathcal{C}}$ . For if it is not an *exact cover*, we have  $|\{g_V(v_{ik}^C) \mid \forall i \in \{1, 2, \dots, q\}, k \in \{1, 2, 3\}\}| < |\{v_{ik}^C \mid \forall i \in \{1, 2, \dots, q\}, k \in \{1, 2, 3\}\}|$ . This is against the definition of the injection  $g_V$ .

On the other hand, if there is an *exact cover* of  $\mathsf{I}_{\mathcal{C}}$ , we claim that the *exact cover* also gives an priority mapping  $(g_V, r_V, g_E, r_E)$  for instance  $\mathsf{I}_{\mathcal{P}}$ . We use  $S^C = \{v_{j_k}^C \mid k \in \{1, 2, \dots, q\} \text{ and } j_k \in \{1, 2, \dots, n\}\} \subseteq C = \{C_1, C_2, \dots, C_n\}$  to denote the *exact cover* of  $\mathsf{I}_{\mathcal{C}}$ . Then we have the priority mapping as follows:

$$\begin{aligned} g_V &: \forall i \in \{1, 2, \dots, q\}, k \in \{1, 2, 3\}, g_V(v_i^C) = u_{j_i}^C, g_V(v_{ik}) = u_{j_i k}; \\ r_V &: \forall v \in V_P, r_V(v) = 0; \\ g_E &: \forall (v_1, v_2) \in E_P, g_E(v_1, v_2) = (g_V(v_1), g_V(v_2)); \\ r_E &: \forall e \in E_P, r_E(e) = f_{V_P}(e). \end{aligned}$$

It is clear that  $(g_V, r_V, g_E, r_E)$  defines a priority mapping of VN on the SN of instance  $\mathsf{I}_{\mathcal{P}}$ .

Thus the instance  $\mathsf{I}_{\mathcal{P}}$  of  $\mathsf{VNM}_{\mathcal{P}}$  has a priority mapping iff the instance  $\mathsf{I}_{\mathcal{C}}$ , which is NP-complete [30], has an *exact cover*. This follows that the  $\mathsf{VNM}_{\mathcal{P}}$  is NP-hard. Along with the result that it is in NP,  $\mathsf{VNM}_{\mathcal{P}}$  is NP-complete.  $\square$



## Proof for theorem 2

Given a VN request  $(G_P, \mathcal{C})$  in which  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$ , an SN  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$  and a cost function  $c : V_S \cup E_S \rightarrow \mathbb{Z}^+$  associates a positive number with each substrate node and link in  $G_S$ . We then prove results of theorem 2 as follows.

### Proof:

(1) We first prove VMP without node sharing is in NP.

We provide an polynomial time (cubic-time) algorithm to find minimum cost VMP of a VN request on an SN, without node sharing and the edge constraints are absent. The algorithm consists three steps:

- (a) Add  $|V_S| - |V_P|$  meta nodes to the set  $V_P$  and denote the augmented virtual nodes set as  $V'_P$ . (If  $|V_P| \geq |V_S|$ , then return 'NO').
- (b) Attach costs  $x(u, v)$  to each node pair  $(u, v) \in V'_P \times V_S$ : when  $u \in V_P$ ,  $v \in V_S$  and  $f_{V_P}(v) \leq f_{V_S}(u)$ ,  $x(u, v) = \frac{f_{V_P}(u)}{f_{V_S}(v)} c(v)$ ; else  $x(u, v) = M$ , where  $M = \sum_{v \in V_S} (c(v))$ .
- (c) Assign the  $|V_S|$  meter nodes and virtual nodes to the  $|V_S|$  substrate nodes under the settings of the linear assignment problem, which aims to assign  $m$  object to another  $m$  object with minimizing the total assignment cost and can be solved in  $O(|V_S|^3)$  time [31].
- (d) Assume the the minimum assignment of the assignment problem in (c) is denoted by an injection  $f_B$ , if the total cost of assignment  $f_B$  is larger than  $M(|V_S| - |V_P| + 1)$ , then outputs 'NO' for there is no VMP of the VN on SN, otherwise outputs the minimum VMP denoted by a node mapping  $(g_V, r_V)$  as:  $g_V(v) = f_B(v)$  and  $r_V(v) = f_{V_P}(v)$  ( $\forall v \in V_P$ ).

Thus the above algorithm finds a minimum VMP of a VN on an SN in  $O(|V_S|^3)$ -time if there exists one.

We next show that it becomes NP-complete and APX-hard to approximate when node sharing is requested.

The NP-complete result is obviously. First, as proved in theorem 1, it is NP-hard to decide whether there exists a VMP of a VN on an SN, thus it is NP-hard to decide whether there exists a VMP such that its cost is smaller than a constant  $K$  when consider costs on the substrate nodes and links. As it is in NP as stated in theorem 1, it is NP-complete.

We next prove that it is APX-hard to approximate. We conduct a L-reduction from minimum RGAP (restricted generalized assignment problem), which can be proved to be APX-hard by adopting the work about the GAP (generalized

assignment problem) in [15, 36], to verify that the minimum cost for  $\text{VMP}_{(\text{NS})}$  is APX-hard.

We give the description of minimum GAP as follows. Given a pair  $(\mathcal{B}, \mathcal{S})$  where  $\mathcal{B} = \{b_1, b_2, \dots, b_m\}$  is a set of  $m$  bins and  $\mathcal{S} = \{a_1, a_2, \dots, a_n\}$  is a set of  $n$  items. Each bin  $b_j \in \mathcal{B}$  has a capacity  $c_B(b_j)$ , and for each item  $a$  and bin  $b$ , we are given a size  $s(a, b)$  and a budget  $w(a, b)$  specifying the capacity requirement of items and the cost of packing  $a$  by bin  $b$ . The minimum GAP is to find a feasible packing for  $\mathcal{S}$  in  $\mathcal{B}$  and minimize the budget of the packing while packing. Here the budget of a packing is the sum of the budget of each item and consider instances that possess a feasible packing. RGAP is a restricted version of GAP satisfying:

- (a)  $\forall i \leq m, j \leq n, \frac{w(a_i, b_j)}{s(a_i, b_j)}$  depends only on  $j$ ; (We use  $w_s(b_j)$  to denote  $\frac{w(a_i, b_j)}{s(a_i, b_j)}$ .)
- (b)  $\forall i, j \leq n, a \in \mathcal{S}, s(a, b_i) = s(a, b_j)$ . (We use  $s(a)$  to denote  $s(a, b_i)$ .)

We below L-reduce from RGAP to prove that  $\text{VMP}_{(\text{NS})}$  is APX-hard.

Recall a L-reduction from A and B (two optimization problems) consists of two polynomial time computable functions  $R$  and  $S$  such that:

- i) ( $R: I_A \rightarrow I_B$ ) If  $x \in I_A$  (set of instances of A), and  $\text{OPT}(x)$  is the measurement of the optimum solution of instance  $x$ , then  $R(x)$  is an instance of B with its measurement of the optimum solution satisfying  $\text{OPT}(R(x)) \leq \alpha \text{OPT}(x)$ , in which  $\alpha$  is a positive constant;
- ii) ( $S: \text{SOL}_A(R(x)) \rightarrow \text{SOL}_B(x)$ ) If  $s$  is any feasible solution of  $R(x)$ , then  $S(s)$  is a feasible solution of  $x$  such that  $|\text{OPT}(x) - c(S(s))| \leq \beta |\text{OPT}(R(x)) - c(s)|$ , in which  $c$  denotes the measurement of instances and  $\beta$  is a positive constant.

The key property of L-reduction is that it preserves approximability: If there is a L-reduction  $(R, S)$  from  $A$  to  $B$  with constance  $\alpha$  and  $\beta$ , and there is polynomial time  $\epsilon$ -approximation algorithm for  $B$  ( $\epsilon > 1$ ), then there is a PTIME  $\alpha\beta(\epsilon-1)$ -approximation algorithm for  $A$ . We'll use the contrapositive proposition to prove approximation-hardness.

Then we present the L-reduction  $(R, S, \alpha, \beta)$  from minimum RGAP to minimum cost mapping for  $\text{VMP}_P$  as follows:

- (a)  $R: I_{\text{RGAP}} \rightarrow I_{\text{VMP}}$ . We present function  $R$  by construct an instance of  $\text{VMP}_{(\text{NS})}$  corresponding to an arbitrary instance of RGAP in the form above:

$$\begin{aligned} \text{VN} : G_P &= (V_P, E_P, f_{V_P}, f_{E_P}) \\ &- V_P = \{v_1, \dots, v_n\}; \\ &- f_{V_P}(v_i) = s(a_i) \ (\forall i \leq n); \\ &- E_P = \emptyset; f_{E_P} = \text{NULL}; \end{aligned}$$

$SN : G_S = (V_S, E_S, f_{V_S}, f_{E_S})$   
 $- V_S = \{u_1, u_2, \dots, u_m\}$   
 $- f_{V_S}(u_i) = c_B(b_i) \ (\forall i \in \{1, 2, \dots, m\}), ;$   
 $- E_S = \emptyset; f_{E_P} = \text{NULL}.$   
 $c : c(u_i) = w_s(b_i) \cdot c_B(b_i) \ (\forall b_i \in \mathcal{B});$

(b)  $S: SOL_{VMP}(f(I_{P_A})) \rightarrow SOL_{P_A}(I_{P_A})$ . We define the function  $S$  by the following. For any feasible solution  $(g'_V, r'_V)$ , function  $S$  derives a packing for instance of RGAP by packing item  $a_i$  to bin  $b_j$  if  $g'_V(v_i) = u_j \ (\forall i \in \{1, 2, \dots, n\}, j \in \{1, 2, \dots, m\})$ ;

Intuitively,  $R$  and  $S$  constructs a bijection from instance of minimum RGAP to instance of minimum cost mapping for VMP. Moreover, the cost of mapping a node of VN to the SN is equal to the budge of packing the corresponding item to the corresponding bin. Specifically, leveraging the additional property of RGAP, the cost of mapping node  $v_i$  in  $V_P$  is mapped to  $u_j$  in  $V_S$  is  $c_M(v_i) = \frac{f_{V_P}(v_i)}{f_{V_S}(u_j)} \cdot c(u_j) = \frac{s(a_i)}{c_B(b_j)} \cdot w_s(b_j) c_B(b_j) = w(a_i, b_j)$ , *i.e.*, the cost of the node mapping in the instance of minimum cost mapping for VMP is equal to the budget of bin packing in the instance of minimum RGAP. Thus the above reduction ensures the following:

1. function  $R$  asserts that for each feasible solution of  $I_{RGAP}$ , there is a corresponding feasible solution for  $I_{VMP}$ ; Moreover, the optimum solution of  $I_{RGAP}$  corresponds to the optimum solution of  $I_{VMP}$  by  $f$ ;
2. function  $S$  ensures that for each feasible solution of  $I_{VMP}$ , there is a corresponding feasible solution for  $I_{RGAP}$ . If  $y$  is the optimum (minimum) solution of  $f(I_{VMP})$ , then  $S(y)$  is the optimum (maximum) solution of  $I_{RGAP}$ ;
3. For each feasible solution  $s_1$  of  $I_{RGAP}$ , the budget of  $s_1$  is equal to the mapping cost of the corresponding feasible solution  $s_2$  for  $I_{VMP}$ .

Thus  $R$  and  $S$  satisfy the constraints of the L-reduction with  $\alpha=\beta=1$ . As minimum RGAP is APX-hard, so is the minimum cost mapping for  $VMP_P$ .

(2) We first show the NP-complete results of minimum cost mapping with edge constraints by reduction from X3C problem. Then we propose an AP-reduction from minimum weight 3-SAT problem to show the NPO-completeness of those optimization problems.

As showed in theorem 1, it is NP-hard to deciding whether there exists a VNM with edge constraints, thus it is also NP-hard to deciding whether there exists a VNM with bounded weight (decision version of the minimum cost mapping problem). Besides, it is in NP for the NP algorithm for deciding whether there exists a VNM with edge constraints, when given a VN and an SN, can be naturally adopted here as the checking of the weight of a VNM can be finished in polynomial time. Thus the minimum cost mapping problem is NP-complete

for  $\text{VNM}_P$ ,  $\text{VNE}_{SP}$ ,  $\text{VNE}_{MP}$ ,  $\text{VNM}_L$ ,  $\text{VNM}_{P(NS)}$ ,  $\text{VNE}_{MP(NS)}$  and  $\text{VNM}_{L(NS)}$ , which are all  $\text{VNM}$  cases with edge constraints.

Moreover, we next show that minimum cost mapping problem is NPO-complete in the sense of approximation-hardness, which means that there exists no polynomial time approximation algorithm when consider the edge constraints. We verify it by conducting a AP-reduction from the MW-3SAT (short for minimum weight 3SAT) problem which is a known NPO-complete problem.

First, recall that an instance  $\phi$  of MW-3SAT is of the form  $C_1 \wedge C_2 \cdots \wedge C_n$  where all variables in  $\phi$  are  $x_1, \dots, x_m$  with nonnegative weights  $w_1, w_2, \dots, w_n$ , each clause  $C_j$  ( $j \in [1, n]$ ) is a Boolean formula of the form  $y_{j1} \vee y_{j2} \vee y_{j3}$ , and moreover, for  $i \in [1, 3]$ ,  $y_{ji}$  is either  $x_{p_{ji}}$  or  $\overline{x_{p_{ji}}}$  for  $p_{ji} \in [1, m]$ , in which  $x_{p_{ji}}$  denotes the occurrence of a variable in the literal  $i$  of clause  $C_j$ . A solution is a truth assignment  $\tau$  to the variables  $x_i (i \in [1, m])$  that satisfies  $\phi$ . The MW-3SAT problem is to find the minimum solution, *i.e.*, minimize  $\sum_{\forall i \in [1, m]} x_i \cdot w(x_i)$  where the Boolean values TRUE and FALSE are identified with 1 and 0, respectively.

We next briefly introduce the AP-reduction. Given two optimization problems  $P_1$  and  $P_2$  in NPO, recall that an AP-reduction consists of two functions  $f$  and  $g$ , and a positive constant  $\alpha > 1$  subject to the following constraints (We use  $I_P$  to denote the set of all instances of NPO problem  $P$ , use  $\text{SOL}_P(x)$  to denote the set of feasible solutions of instance  $x$  of  $P$ , use  $R_P(x, s)$  to denote the relative approximation factor of solution  $s$  of instance  $x$ , *i.e.*,  $\max(\frac{m(x, s)}{\text{OPT}(x)}, \frac{\text{OPT}(x)}{m(x, s)})$  where  $m(x, s)$  denotes the value of feasible solution  $s$  of instance  $x$  and  $\text{OPT}(x)$  denotes the value of optimum value of instance  $x$ .) :

1. For any instance  $x \in I_{P_1}$  and any rational  $r > 1$ ,  $R(x, r) \in I_{P_2}$ .
2. For any instance  $x \in I_{P_1}$  and for any rational  $r > 1$ , if  $\text{SOL}_{P_1}(x) \neq \emptyset$ , then  $\text{SOL}_{P_2}(R(x, r)) \neq \emptyset$ .
3. For any instance  $x \in I_{P_1}$ , for any rational  $r > 1$ , and for any  $y \in \text{SOL}_{P_2}(R(x, r))$ ,  $S(x, y, r) \in \text{SOL}_{P_1}(x)$ .
4.  $R$  and  $S$  are computable in time polynomial for any fixed rational  $r$ .
5. For any instance  $x \in I_{P_1}$ , for any rational  $r > 1$ , and for any  $y \in \text{SOL}_{P_2}(R(x, r))$ ,  $R_{P_2}(R(x, r), y) \leq r$  implies  $R_{P_1}(x, S(x, y, r)) \leq 1 + \alpha(r - 1)$ .

We now construct an AP-reduction from the MW-3SAT problem to the minimum cost mapping problem with edge constraints.

(I) function  $S$ . Given an instance  $\phi$  of the MW-3SAT problem, we construct VN and SN as follows:

- VN:  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$  is defined as follows.
- $V_P = \{C_1^P, \dots, C_n^P, X_1, \dots, X_m^P\}$ ;
  - $E_P = \{(X_{p_{j1}}^P, C_j), (X_{p_{j2}}^P, C_j), (X_{p_{j3}}^P, C_j) \mid \forall i \in [1, m], \forall j \in [1, n]\}$ ;

- $f_{V_P} : V_P \rightarrow \{0\}$ , *i.e.*, there is node capacity constraints on virtual nodes of the VN;
  - $f_{E_P}$ : For each edge  $e \in E_P$ ,  $f_{E_P}(e) = a$  where  $a$  is a constant.
- SN:  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$  is defined as follows.
- $V_S = \{X_{T1}^S, X_{F1}^S, \dots, X_{Tm}^S, X_{Fm}^S, 0_1, \dots, 7_1, \dots, 0_n, \dots, 7_n\}$ ;
  - $E_S$  contains  $7 \times 3$  edges for each clause  $C_j = y_{j1} \vee y_{j2} \vee y_{j3}$  of  $\phi$  ( $j \in [1, n]$ ), and there are in total  $21n$  edges in  $E_S$ . We represent the truth assignments of clause  $C_j$  in terms of 8 nodes  $C_j(\rho)$ , where  $\rho$  ranges over all truth assignment of variables  $x_{p_{ji}}$  ( $i = 1, 2, 3$ ). Each node  $C_j^S(\rho)$  is a three-bit constant  $y_{j1}y_{j2}y_{j3}$ . For each truth assignment  $\rho$  of  $x_{p_{j1}}$ ,  $x_{p_{j2}}$  and  $x_{p_{j3}}$  that makes  $C_j$  true,  $E_P$  consists of the following edges:  $(X_{Tp_{jk}}, C_j^S(\rho))$  if  $\rho(X_{p_{jk}}) = \text{true}$ , or  $(X_{Fp_{jk}}, C_j^S(\rho))$  if  $\rho(X_{p_{jk}}) = \text{false}$ , where  $k \in [1, 3]$ .
  - $f_{V_S} : V_S \rightarrow +\infty$ ;
  - $f_{E_S}$ : For each edge  $e \in E_S$ ,  $f_{E_S}(e) = a$ .
- c: For each edge  $e \in E_S$ , if  $e = (X_{Tj}^S, C_j^R(\rho))$  for some  $j \in [1, m]$ , then  $f_{E_S}(e) = w(x_j)$ , otherwise ( $e = (X_{Fj}^S, C_j^R(\rho))$ )  $f_{E_S}(e) = 0$ . For each node  $v \in V_P$ ,  $c(v) = 0$ .

Intuitively, VN encodes the instance  $\phi$  of MW-3SAT. Node  $X_i^P$  ( $i \in [1, m]$ ) denotes variable  $x_i$ , and node  $C_j^P$  ( $j \in [1, n]$ ) denotes clause  $C_j^P$ . Edge  $(X_i^P, C_j^P)$  encodes that variable  $x_i$  appears in clause  $C_j$ , *i.e.*,  $x_i$  is one of the three variables  $x_{p_{ji}}$  ( $i \in [1, 3]$ ). SN encodes the truth assignments of the variables that satisfy the clauses in the instance  $\phi$  of MW-3SAT. Node  $X_{Ti}^S$  ( $i \in [1, m]$ , resp.  $X_{Fi}^S$ ) means assigning variable  $x_i$  a **true** (resp. **false**) value. Nodes  $\{0_j, \dots, 7_j\}$  denotes  $C_j(\rho)$ , which are denoted as a three-bit constant w.r.t the truth assignments of the three variables in clause  $C_j$ .

The above construction of  $R$  guarantees that (a) node  $X_i^P$  ( $i \in [1, m]$ ) in VN is mapped to either node  $X_{Ti}^S(\text{true})$  or  $X_{Fi}^S(\text{false})$  of  $G_2$  and (b) node  $C_j^P$  ( $j \in [1, n]$ ) in VN is mapped to one of the nodes  $\{0_j, \dots, 7_j\}$  of SN.

(II) *function S*: We next present function  $S$  which maps a solution of instance  $R(x)$  for the MW-3SAT problem to a solution of instance  $x$  for the minimum cost mapping problem with edge constraints.

A solution  $s$  of instance  $R(x)$  for the MW-3SAT problem is a mapping  $(g_V, g_E, r_V, r_E)$  from the VN to SN constructed above. We define a truth assignment for instance  $x$  with respect to the mapping as follows. For each variable  $x_i$  ( $i \in [1, m]$ ),  $\rho(x_i) = \text{true}$  if  $g_V(X_i^P) = X_{Ti}^S$  and  $\rho(x_i) = \text{false}$  if  $g_V(X_i^P) = X_{Fi}^S$ .

(III)  $\alpha$ : The constant  $\alpha$  here is defined to be 1.

We below verify that  $(R, S, \alpha)$  defines an AP-reduction from the NPO-complete problem MW-3SAT to the minimum cost mapping problem with edge constraints.

- 1). It is obviously that the functions  $R$  and  $S$ , *i.e.*, the above constructions, are finished in polynomial time.
- 2). For any instance  $\phi$  of MW-3SAT,  $R(\phi)$  is an instance of the minimum cost mapping problem with edge constraints (*i.e.*, a VN and SN, with additional cost and constraints on edges) as constructed above.
- 3). For any instance  $\phi$  of MW-3SAT, if  $SOL_{MW-3SAT}(\phi) \neq \emptyset$ , *i.e.*, there is a truth assignment  $\rho$  that makes  $\phi$  true, then  $SOL_{VNM}(R(\phi)) \neq \emptyset$ . In fact we can present a valid VNM  $(g_V, g_E, r_V, r_E)$  for  $R(\phi)$  as follows: (1) for each  $i \in [1, m]$ ,  $g_V(X_i^R) = X_{T_i}^S$  if  $\rho(x_i) = \text{true}$ , and  $g_V(X_i^R) = X_{F_i}^S$  if  $\rho(x_i) = \text{false}$ , (2) for each  $j \in [1, n]$ ,  $g_V(C_j^R) = C_j^S(\rho)$  defined as above, (3) for each edge  $e = (u, v)$  in VN,  $g_E(e)$  is defined as the unique path (edge) in SN from  $g_V(u)$  to  $g_V(v)$ , (4) for each node  $v \in V_P$ ,  $r_V(v, g_V(v)) = f_{V_P}(v)$ , and (5) for each edge  $e \in E_P$ ,  $r_E(e, g_E(e)) = f_{E_P}(e)$ . It is easy to verify that the VNM  $(g_V, r_V, r_V, r_E)$  is a feasible VNM from VN to SN.
- 4). As the SN constructed by  $R$  encodes all the truth assignments of instance of  $\phi$  for the MW-3SAT problem, function  $S$  indeed transfers solution of  $R(\phi)$  to solution of  $\phi$ , which means  $S(s) \in SOL_{MW-3SAT}(\phi)$ .
- 5). From the definition of cost function for SN, one can verify that (1) for any instance  $\phi$  of MW-3SAT, if  $\rho$  is the optimum solution for  $\phi$  (*i.e.*, minimum weight truth assignment), then the weight of the  $\rho$  is equal to the optimum solution of  $R(\phi)$ , *i.e.*, the minimum cost mapping, and (2) for any feasible solution  $s$  of  $R(\phi)$ , the mapping cost of  $s$  is equal to the weight of the truth assignment of  $S(s)$ .

With the two observations, for any instance  $\phi$  of MW-3SAT, for any feasible mapping  $s$  of  $R(\phi)$ ,  $R_{VNM}(R(\phi), s) = R_{MW-3SAT}(\phi, S(s))$ .

Thus  $(R, S, 1)$  forms an AP-reduction from MW-3SAT to the minimum cost mapping problem with edge constraints. As MW-3SAT problem is NPO-complete and VNM is in NP, minimum cost mapping problem with edge constraints is NPO-complete for cases  $VNM_P$ ,  $VNE_{SP}$ ,  $VNE_{MP}$ ,  $VNM_L$ ,  $VNM_{P(NS)}$ ,  $VNE_{MP(NS)}$  and  $VNM_{L(NS)}$ ,

(3) The APX-hard result is obvious as finding the minimum cost mapping with determined node mapping contains the minimum Steiner Tree problem as a special case, which naturally implies a L-reduction. As the minimum Steiner Tree is APX-hard, so is the minimum cost mapping problem for this special case. More specifically, by conducting L-reductions from the minimum directed steiner tree [14] problem to the minimum cost mapping with determined node mapping for  $VNM_P$ , we show get an approximation bound for the latter problem.

We next show the approximation-hardness of  $VNM_P$  with determined node mapping by a L-reduction from the minimum DST (directed steiner tree) problem.

Recall that an instance  $I_{DST}$  consists of a directed graph  $G_D = (V_D, E_D)$  (assume  $V_D = \{v_1, v_2, \dots, v_m\}$ ,  $E_D = \{e_1, e_2, \dots, e_n\}$ ), a root node  $r \in V_D = \{v_1,$

$v_2, \dots, v_m\}$ , a set of terminal nodes  $X = \{t_1, t_2, \dots, t_k\}$ , and a cost function  $w$  that attaches a positive number to each of the edges in  $E_D$ . The minimum DST problem is to find the minimum weight arborescence rooted at  $r$  and spanning all the vertices in  $X$ , *i.e.*,  $r$  should have a path to every vertex in  $X$ .

The L-reduction from the minimum DST problem to the minimum cost mapping problem for  $\text{VNM}_P$  with node mapping determined consists of function  $R$ , function  $S$  and two positive constant  $\alpha, \beta$  defined as follows:

I) function  $R$ :  $R$  maps instance of DST to instance of minimum cost mapping problem for  $\text{VNM}_P$  with node mapping uniquely determined. Given any arbitrary instance  $I_{\text{DST}}$  of the minimum DST problem, we present  $R$  by constructing corresponding instance  $I_{\text{VNM}_P}$  of minimum cost mapping problem for  $\text{VNM}_P$  with node mapping uniquely determined, *i.e.*, a VN, an SN, and a cost function  $c$  of  $I_{\text{VNM}_P}$ .

- 1) VN:  $G_P = (V_P, E_P, f_{V_P}, f_{E_P})$ , in which
  - $V_P = \{r^P, t_1^P, \dots, t_k^P\}$ ;
  - $E_P = \{(r^P, t_1^P), (r^P, t_2^P), \dots, (r^P, t_k^P)\}$ ;
  - $f_{V_P}: f_{V_P}(r^P) = 1, f_{V_P}(t_i^P) = i \ (\forall i \in [1, k])$ ;
  - $f_{E_P}: \forall e \in E_P, f_{E_P}(e) = a, a$  is a constant;
- 2) SN:  $G_S = (V_S, E_S, f_{V_S}, f_{E_S})$ , in which
  - $V_S = \{v_1^S, v_2^S, \dots, v_m^S\}$ , in which  $r^S \in V_S$  and  $\{t_1^S, t_2^S, \dots, t_k^S\} \subseteq V_S$ ;
  - $E_S = \{e_1^S, e_2^S, \dots, e_n^S\}$ ;
  - $f_{V_S}: f_{V_S}(r^S) = 1, f_{V_S}(t_i^S) = i \ (\forall i \in [1, k])$ ;
  - $f_{E_S}: \forall e \in E_S, f_{E_S}(e) = a$ ;
- 3)  $c: \forall i \in [1, n], c(e_i^S) = w(e_i)$ .

II) function  $S$ : For any instance  $I_{\text{DST}}$ ,  $R(I_{\text{DST}})$  is an instance of the minimum cost mapping problem for  $\text{VNM}_P$  with node mapping uniquely determined. Here function  $S$  maps an arbitrary feasible solution  $s$  of instance  $R(I_{\text{DST}})$  to a feasible solution of instance  $I_{\text{DST}}$ .

For any feasible solution  $s$  of  $R(I_{\text{DST}})$ , *i.e.*, an mapping  $(g_V, g_E, r_V, r_E)$  in which  $g_V$  and  $r_V$  are uniquely determined, we present  $S$  by constructing the corresponding feasible solution of  $I_{\text{DST}}$  as follows: for any  $e = (r^P, t_i^P) \in E_P$  ( $\forall i \in [1, k]$ ), if  $g_E(e) = \rho_e$  in which  $\rho_e = \{r^S, v_{p_1}^S, \dots, v_{p_q}^S, t_i^S\}$  is a path in VN from  $r^S$  to  $t_i^S$ , then for instance  $I_{\text{DST}}$ , the path connecting root node  $r$  to terminal node  $t_i$  is the path  $(r, v_{p_1}, \dots, v_{p_q}, t_i)$ .

III) positive number  $\alpha$  and  $\beta$ :

- 1).  $\alpha = 1$ ;
- 2).  $\beta = 1$ ;

We next verify that  $(R, S, \alpha, \beta)$  forms a L-reduction from minimum DST problem to minimum cost mapping problem for  $\text{VNM}_P$  with node mapping uniquely determined.

- (i). From the construction above, we know that  $R$  and  $S$  are polynomial time computable.
- (ii). For any instance  $I_{\text{DST}}$  of the minimum DST problem, the weight of the optimum solution of  $\text{OPT}(I_{\text{DST}})$  is equal to cost of the optimum solution (i.e.,  $\text{VNM}_P$ ) of instance  $R(I_{\text{DST}})$  for a) As edges of  $G_D$  and  $\text{SN}$  are one-to-one correspondent, along with the cost (weight) on them, and b) the uniquely determined node mapping  $g_V$  in the instance constructed by  $R$  ensures that  $\text{SN}$  encodes  $G_D$  and  $\text{VN}$  encodes  $r$  and the terminal nodes set  $X$  of the instance  $I_{\text{EDP}}$ . Thus the restriction i) in the definition of L-reduction established with  $\alpha = 1$ ;
- (iii). Similar to (ii), if  $s$  is any feasible solution of  $R(I_{\text{DST}})$ , one can easily verify that  $S(s)$  is a feasible solution of  $I_{\text{DST}}$  such that  $|\text{OPT}(x) - w(S(s))| = |\text{OPT}(R(x)) - c(s)|$ , which admits the restriction ii) in the definition of L-reduction with  $\beta = 1$ ; In fact, here  $w(S(s)) = c(s)$ .

As it is NP-hard to approximate minimum DST problem to a factor better than  $\ln k$  where the  $k$  is the number of the terminals unless  $P \neq NP$ , the minimum cost mapping problem for  $\text{VNM}_P$  does not admit  $\ln |V_P|$ -approximation algorithms even with node mapping uniquely determined.  $\square$

## Appendix B: Algorithms

### Proof for proposition 3

This is intuitive. One can observe the followings.

- 1) The auxiliary graph  $G_{aux}$  used by `compVNM` records the maximum bandwidth between any two nodes in  $\text{SN}$ .
- 2) For any node mapping  $(g_V, r_V)$ , `compVNM` determines whether it admits a edge mapping by querying the maximum bandwidth between matched nodes from  $G_{aux}$ . This takes  $O(|E_P|)$  time as there are at most  $|E_P|$  pairs of nodes that requires edge mapping checks.
- 3) For any node mapping  $(g_V, r_V)$  found by `compVNM`, there exists a valid compatible edge mapping  $(g_E, r_E)$ .

### Proof for lemma 1

**Proof:** In order to show the correctness of lemma 1, we first present the *augmentation condition* that will be used in the proof.

**Augmentation Condition.** *In the auxiliary graph constructed by `compAuxGraph`, for any two nodes  $u$  and  $v$ , if there exists an*



This condition is necessary for the following two reasons. a) If the bandwidth of any two sides of a triangle are different, then the least one (*e.g.*, carried by edge  $(u, v)$ ) should be update to the lesser one of the remaining two sides (*e.g.*,  $(u, w)$  and  $(w, v)$ ), for  $u \rightarrow w \rightarrow v$  forms a valid path connecting  $u$  and  $v$  while possessing larger bandwidth than edge  $(u, v)$ . b) If the bandwidth carried by the equal sides (*e.g.*,  $(u, w)$  and  $(w, v)$ ) is larger than that carried by the remaining side (*e.g.*,  $(u, v)$ ), then it turns out that the latter should be updated to that larger bandwidth carried by the equal sides.

The proof consists of three blocks.

(a) *Necessary edges to be considered when updating the newly introduced edges.* It is obviously that the new edges, which are introduced by connecting the new node  $m$  to existing partial auxiliary graph  $G_{aux}$ , may not carry the largest bandwidth. To update their bandwidth, one only needs to reconfigure the bandwidths on the new edges such that any triangles containing those new edges satisfy the augmentation condition. As those triangles must contain the new node  $m$ , those newly introduced edges that connecting  $m$  the the partial constructed auxiliary graph  $G_{aux}$  is necessary to be considered.

(b) *Necessary and sufficient edges to be considered when updating existing edges in  $G_{aux}$  after bandwidths on those new edges involved with  $m$  are up to date.* After the bandwidths on the newly introduced edges are updated, those on the old edges in  $G_{aux}$  may be outdated. We next show that to update the bandwidth carried on any old edge  $(i', j')$  in  $G_{aux}$ , one only needs to concern about the triangle that consisting of  $i', j'$  and  $m$ .

Obviously, it is necessary to check whether the triangle that containing nodes  $i', j'$  and  $m$  satisfies the augmentation condition, as bandwidths on  $(m, i')$  (or  $(i', m)$ ) may be changed. We then show that after updating those triangles consisting of the new node  $m$  and old edges in  $G_{aux}$ , all triangles in  $G_{aux}$  already meet the augmentation condition. Consider an arbitrary triangle consisting of three nodes  $i', j'$  and  $k'$  in  $G_{aux}$ . For simplicity, we consider undirected graphs here, things remains similar for directed graph. We use  $b(i', j')$  (or  $b(e)$ ) to denote the bandwidth on edge  $(i', j')$  (or  $e$ ). After updating the bandwidth on edges  $(m, i')$ ,  $(m, j')$  and  $(m, k')$ ,  $b(i', j') = \min\{b(m', i'), b(m', j')\}$ ,  $b(j', k') = \min\{b(m', j'), b(m', k')\}$ ,  $b(k', i') = \min\{b(m', k'), b(m', i')\}$ . Without lose of generality, we assume that  $b(m', i') \geq b(m', j') \geq b(m', k')$ . Then  $b(i', j') = b(m', j')$ ,  $b(j', k') = b(m', k')$ ,  $b(k', i') = b(m', k')$ . Thus the triangle consisting of nodes  $i', j'$  and  $k'$  in  $G^{Aug}$  satisfies the augmentation condition. This follows that there is no need to update the old edges in  $G_{aux}$  after checking and updating all triangles consisting of nodes  $i, j$  and  $m$  (for any nodes  $i$  and  $j$  in  $G_{aux}$ ).

(c) *Sufficient edges to be consider for the newly introduced edges.* With (b), we learn that all triangles in  $G_{aux}$  along with the newly introduced node  $m$  and associated edges already satisfy the augmentation condition, revealing that those edges concerned in (a) when updating edges connecting  $m$  to  $G_{aux}$  are sufficient.

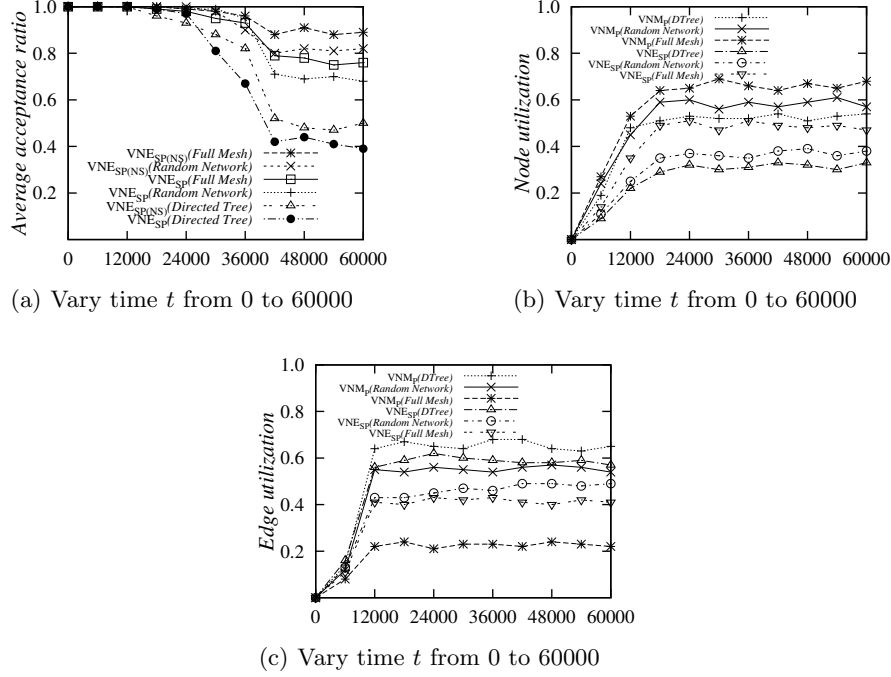


Fig. 10. Additional experiments

From (a) and (c), conjecture (1) in lemma 1 is established. The correctness of conjecture (2) is ensured by (b).  $\square$

#### Proof for theorem 4

**Proof:** Consider the scenario that adding the new node  $v$  to the subgraph  $G^k$  of the VN  $G_P$ . We denote the auxiliary graph of  $G^k$  by  $G_{aux}^k$ .

(1) We first show that the updates of existing edges in  $G_{aux}^k$  will not affect the determination of the weight and recorded path on the new edge  $(u, v)$  (or  $(v, u)$ ). In fact, existing edge  $(u, u')$  needs to be updated if and only if the newly introduced edges  $(u, v)$  and  $(v, u')$  form into a directed path connecting  $u$  and  $u'$  and moreover, the weight (bandwidth) carried this path is larger than that on existing edge  $(u, u')$ . This is because for any edge  $(u, u')$  in  $G_{aux}^k$ , it carries the maximum weight (bandwidth) from  $u$  to  $u'$  in  $G^k$ . Once the updates of  $(u, u')$  w.r.t.  $(u, v)$  and  $(v, u')$  finished, along with the conjecture (2) of the lemma (For any existing edge  $(u, u')$ , it suffices to consider the triangle with edges  $(u, u')$ ,  $(u', v)$  and  $(v, u)$  for updating its weights and path), the new edge  $(u, v)$  need

not to be updated. We next verify the conjecture (2) of the lemma (without leveraging conjecture (1)).

(2) Consider three nodes in  $G_{aux}^k$ ,  $u_1$ ,  $u_2$  and  $u_3$ . We use  $w(e)$  and  $w(\rho)$  to denote the weight (bandwidth) of an edge  $e$  and a path  $\rho$ . After the new edges connecting  $v$  and  $u_1$ ,  $u_2$  and  $u_3$  are updated,  $w(u_1, u_2) = \min\{w(u_1, v), w(v, u_2)\}$ ,  $w(u_2, u_3) = \min\{w(u_2, v), w(v, u_3)\}$ ,  $w(u_1, u_3) = \min\{w(u_1, v), w(v, u_3)\}$ . We next show that  $w(u_1, u_3) \geq \min\{w(u_1, u_2), w(u_2, u_3)\}$ . This is intuitive as  $\min\{w(u_1, u_2), w(u_2, u_3)\} = \min\{\min\{w(u_1, v), w(v, u_2)\}, \min\{w(u_2, v), w(v, u_3)\}\} = \min\{w(u_1, v), w(v, u_2), w(u_2, v), w(v, u_3)\} \leq \min\{w(u_1, v), w(v, u_3)\}$ . Thus it suffices to consider the triangle with edges  $(u, u')$ ,  $(u, v)$  and  $(v, u')$ , in order to update the weight and path of the existing edge  $(u, u')$ .

From (1) and (2), we have the establishment of lemma 4.  $\square$

## Proof for lemma 2

**Proof:** (1) First we show that if there exists a path from nodes  $u$  to  $v$  in VN  $G_P$ , then 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 exist an edge  $(u, v)$  in  $G'_P$ . From line 3 and line 5 of **UpdateVN**, one can find 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  $G'_P$ , or a path that transits a intermediate node  $u'$ , as stated in line 4 and line 6. This verifies the existence of a path from nodes  $u$  to  $v$  in  $G_P^m$ . We need to show the uniqueness of the path by the following. Once the path that connecting  $u$  to  $v$  is added to  $G_P^m$  at some round of the for blocks in line 2 in **UpdateVN**, paths that from  $u$  to  $v$  will not be introduced as in line 3 and line 5, **UpdateVN** finds that there exists  $u'$  such that  $(u, u')$  is in  $G'_P$  and  $(u', v)$  is in  $G_P^m$ . Thus the path from  $u$  to  $v$  is unique.

(2) We next show the reverse direction. If there exists a path from  $u$  to  $v$  in  $G_P^m$ , then it must be introduced in some round of the for block in **UpdateVN** invoked with parameter  $u$ . As there exists a path connecting  $u$  to  $v$  in **UpdateVN**, thus the edge  $(u, v)$  must be included in  $G'_P$ , which means that there is a path connecting  $u$  to  $v$  in VN  $G_P$  (following the definition of auxiliary graph for VN).  $\square$

## Appendix C: Additional Experiments

We conduct two sets of additional experiments here, to study the impacts of node sharing and resource utilization over time. The settings are the same to

the first set of experiments in mapping quality (**Exp-1**), *i.e.*, fixing  $p_P = 0.5$ ,  $n_P$  in  $[2, 50]$ ,  $w_{V_P}$  and  $w_{E_P}$  in  $[3, 30]$  for VN requests,  $n_S = 5000$ ,  $w_{V_S}$  and  $w_{E_S}$  in  $[50, 100]$  for SNs, and varying  $t$  from 0 to 60,000. Results are shown in Fig. 10.

(1) In the first additional experiment, we evaluate the impacts of node sharing on  $VNE_{SP}$ . Figure 10(a) shows the AR of  $VNE_{SP}$ , with node sharing or not, over directed tree, full mesh and random networks. The results reveals that node sharing consistently improves the AR for  $VNE_{SP}$  (in the range of  $[12\%, 15\%]$ ). Indeed, the AR over directed tree is over 53%, as opposed to 38% without node sharing.

(2) In the second set of additional experiments, we evaluate the utilization of nodes and edges. (a) The average resource utilization of substrate nodes is shown in Fig. 10(b). It shows the following. (i) Node utilization of SNs becomes consistent after  $t = 24000$ . (ii) Node utilization of full mesh is higher than that of random network, followed by directed tree, for both  $VNM_P$  mappings and  $VNE_{SP}$  mappings. (iii) For each type of the three SN topologies, the node utilization of  $VNM_P$  is higher than that of  $VNE_{SP}$ . (b) Figure 10(c) shows the average edge utilization. It tells the followings. (i) After  $t = 12000$ , the average edge utilization becomes consistency, no matter on which kind of SNs. (ii) Priority mapping over directed tree gains the highest edge mapping, however, it is the lowest of all cases while over full mesh. (iii) Generally, the impact of network topologies on the edge utilization for  $VNM_P$  is larger than that for  $VNE_{SP}$ .

**Conclusion.** The additional experiments show the following. (a) Node sharing is also helpful to  $VNE_{SP}$ . It improves the AR of  $VNE_{SP}$  on various SNs. (b) The average node utilization of  $VNM_P$  is much higher than that of  $VNE_{SP}$ . (c) The impact of network typologies on average edge utilization for  $VNM_P$  is higher than that for  $VNE_{SP}$ .