Computer Networks 56 (2012) 1797–1813

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Virtual network embedding through topology awareness and optimization

Xiang Cheng [a], Sen Su [a,*], Zhongbao Zhang [a], Kai Shuang [a], Fangchun Yang [a], Yan Luo [b], Jie Wang [b]

[a] *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China*
[b] *University of Massachusetts, Lowell, MA 01854, United States*

ARTICLE INFO

Article history:
Received 30 May 2011
Received in revised form 13 October 2011
Accepted 30 January 2012
Available online 7 February 2012

Keywords:
Network virtualization
Virtual network embedding
Topology awareness
Random walk
Meta-heuristic
Particle swarm optimization

ABSTRACT

Embedding a sequence of virtual networks (VNs) into a given physical network substrate to accommodate as many VN requests as possible is known to be NP-hard. This paper presents a new approach to studying this problem. In particular, we devise a topology-aware measure on node resources based on random walks and use it to rank a node's resources and topological attributes. We then devise a greedy algorithm that matches nodes in the VN to nodes in the substrate network according to node ranks. In most situations there exist multiple embedding solutions, and so we want to find the best embedding that increases the possibility of accepting future VN requests and optimizes the revenue for the provider of the substrate network. We present an integer linear programming formulation for this optimization problem when path splitting is not allowed. We then devise a fast-convergent discrete Particle Swarm Optimization algorithm to approximate this problem. Extensive simulation results show that our algorithms produce near optimal solutions and significantly outperform existing algorithms in terms of the ratio of the long-term average revenue over the VN request acceptance.

1. Introduction

The Internet has become a critical infrastructure in the modern society. We would like to use the Internet to evaluate and deploy innovative network designs. This, however, has proven challenging, which has seriously hindered the advancement of network technologies, and thus calls for fundamental redesigns of the Internet architecture [1,2].

Network virtualization, proposed recently to address the ossification problem of the current Internet, allows multiple heterogeneous virtual networks (VNs) to coexist on a shared network substrate. Each VN gets a *slice* of the substrate network resources for meeting the resource requirements of the VN. Leveraging the programmability of the underlying hardware resources, a VN may run non-IP protocol (e.g., AIP [3]) on a chosen network topology. Supporting VNs is therefore a desirable attribute for experimenting and deploying novel Internet architecture [4].

Infrastructure providers (InPs) and service providers (SPs) play different roles in network virtualization: InPs manage the physical infrastructure while SPs create VNs and offer end-to-end services [2,4–6]. Since the substrate network resources are limited, to increase the revenues for the InPs, it is important to make efficient use of the underlying resources and handle each online VN request on-demand, where different VNs may have different topologies, resource requirements, and lifetime.

Embedding VN requests of the SPs into the substrate network (a.k.a. VN embedding) is NP-hard because of node and link constraints. Even if all the virtual nodes are mapped, it is still NP-hard to embed virtual links without violating the

* Corresponding author.
E-mail addresses: .chengxiang@bupt.edu.cn (X. Cheng), susen@bupt.edu.cn (S. Su), zhongbaozb@bupt.edu.cn (Z. Zhang), shuangk@bupt.edu.cn (K. Shuang), fcyang@bupt.edu.cn (F. Yang), yan_luo@uml.edu (Y. Luo), wang@cs.uml.edu (J. Wang).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2012.01.022

bandwidth constraints into the substrate paths. Thus, to reduce the hardness of the VN embedding problem and enable efficient heuristics, early studies on VN embedding typically follow the following four approaches:

1. Assume that VN requests are known in advance (i.e., an offline version) [7,8].
2. Ignore certain resource constraints of the VN requests (e.g., CPU, bandwidth or location) [7–11].
3. Perform no admission control when the resource of the substrate network is insufficient [7–9].
4. Focus only on specific VN topologies (e.g., backbone-star) [8].

Existing embedding algorithms typically ignore the topological structure (i.e., the quality of neighbors of the nodes) of the VNs and the underlying substrate network. We note that the topological attributes of nodes could have significant impact on the success and efficiency of embedding VNs. This calls for a new measure that measures a node's resources and its topological attributes at the same time.

To address this issue, we devise a Markov random walk model, inspired by Google's PageRank measure [12], for computing topology-aware resource ranking of a node to reflect both the resource and the quality of connections of the node in the network. We denote this model by NodeRank [11]. PageRank considers a link from web page A to web page B as a vote. A web page is considered important if a number of important web pages all vote for it. In so doing, it allows the topology of the web to influence the PageRank of a web page. In the context of VN embedding, we consider a node to be important if a node links forward to a number of nodes with relatively high resources available. Treating the connectivity between two nodes as a Markov chain transition with certain probability, we calculate the relative resource quality of a node with a Markov chain model based on the topology of the network.

Using the notion of NodeRank, we present a two-stage greedy VN embedding algorithm and denote it by *RW-MaxMatch*. In the first stage, it maps a virtual node with the highest rank to a substrate node with the highest rank, a virtual node with the second highest rank to a substrate node with the second highest rank, and continues in this manner for the rest of the virtual nodes. We call such mapping a *topology-aware large-to-large and small-to-small (TL2S2) mapping*. In the second stage, it embeds virtual links using the shortest path algorithm. Since NodeRank reflects not only the resource requirements or availability of a network node but also its topological characteristic (i.e., the quality of its neighbors), leveraging NodeRank to construct the VN embedding solution can benefit both the node mapping and link mapping and thus increases the possibility of accepting the given VN request.

Depending on the policies of InPs, paths in a VN may or may not be split into multiple paths in the substrate network. Under the assumption that path splitting is supported by the substrate network, Chowdhury et al. [13] present a mixed integer programming (MIP) formulation for the VN embedding problem and propose MIP-based online VN embedding algorithms. When path splitting is not supported by the substrate network, however, these algorithms would no longer be appropriate. Moreover, the linear programming (LP) relaxation and rounding techniques used by the algorithms may result in infeasible VN embedding solutions. Even if a feasible solution can be obtained, it may still be far from being optimal [14].

To address the second issue, we first note that different VN embedding solutions could result in different resource costs to the substrate network. Thus, reducing such cost may help increase the possibility of accepting more future VN requests. We present an integer linear programming (ILP) formulation for the VN embedding problem to minimize such cost. We devise an approximation algorithm to this model while maintaining a high acceptance rate of VN requests. Leveraging our topology-aware node resource ranking measure, our approximation algorithm follows Particle Swarm Optimization (PSO), a meta-heuristic optimization method, and we denote our algorithm by *RW–PSO*. The main idea of RW–PSO is as follows: We consider the position of each particle in PSO as a possible VN embedding solution, where each particle updates its position to achieve a better position according to the local and global information. A near-optimal solution of VN embedding can be obtained through the evolution process of the particles. In particular, we propose a *TL2S2 preferred local selection strategy* based on NodeRank to accelerate the convergence speed of the algorithm. We show that RW–PSO not only helps accept the current VN request but also provides more substrate resources for future VN requests.

RW–PSO may be viewed as an enhanced version of RW-MaxMatch, which allows a tradeoff of algorithm running time for a better quality of VN embedding. Simulation results show that RW-MaxMatch and RW–PSO significantly outperform the previous approaches in terms of the ratio of the long-term average revenue over the VN request acceptance.

The rest of the paper is organized as follows. We present the network model, the definition of VN embedding problem and its common objectives in Section 3. In Section 4 we present the method of computing the topology-aware resource ranks of nodes using the random walk model. Sections 5 and 6 describe RW-MaxMatch and RW–PSO respectively. We evaluate the efficiency of RW-MaxMatch and RW–PSO in Section 7. In Section 2 we discuss the related work. Section 8 concludes the paper.

2. Related work

The VN embedding problem is similar to the virtual private network (VPN) provisioning problem [15] and the network testbed mapping problem. The major difference between them is on resource constraints. For a VPN request, the resource constraints are typically just bandwidth requirements from sources to destinations specified by a traffic matrix rather than node constraints (e.g., CPU). The Assign algorithm [16] used in the Emulab testbed considers constraints on both nodes and links, where the node constraint is provided as the exclusive use of nodes, i.e., different VNs cannot share the same substrate node. VN embedding, however, allows substrate nodes and links to be shared by multiple VNs. In this section, we first review related work on VN embedding. We then discuss previous work using Markov random walk and PSO.

To obtain VN embedding solutions that satisfy the flow constraints of VN request and to make efficient use of substrate network resources, Lu et al. [8] propose a VN embedding algorithm based on flow constraints. However, the algorithm neglects resource constraints of virtual nodes and works only on the backbone-star topology for VNs.

Zhu and Ammar in [7] develop a basic algorithm for VN embedding, and present subdividing heuristics and adaptive optimization strategies to improve the performance. They also develop a selective VN reconfiguration scheme that prioritizes the reconfiguration for the critical VNs. Their algorithms assume that substrate network resources are unlimited, and focus on load balancing in the substrate network without the need for admission control.

Yu et al. [10] advocate a different approach on the design of the substrate network to enable simpler embedding algorithms and more efficient use of resources. They simplify virtual link embedding by allowing the substrate network to support path splitting and migration and propose VN embedding algorithms without restricting the VN embedding problem space. Their algorithms deal with resource constraints of virtual nodes and virtual links, VN access control, online VN request, and VN topology diversity.

While considering both online VN embedding problem space as in [10] and location requirements of virtual nodes, Chowdhury et al. [13] formulate a MIP model for VN embedding problem under the assumption that the substrate network supports path splitting. They use LP relaxation and rounding techniques to obtain approximations to their MIP formulation. Unfortunately, the rounding results of their algorithms that correspond to the node mapping stage may result in an infeasible link embedding solution. Even if the link embedding is feasible, the solution may still be far away from the optimal solution. In addition, their MIP formulation is inappropriate when path splitting is not supported by the substrate network and consequently the corresponding algorithms they proposed would not perform well.

Lischka and Karl [17] model the topology of the substrate network and the VN as a directed graph, and propose a VN embedding algorithm based on subgraph isomorphism which maps nodes and links during the same phase. Their algorithm can be seen as an extended version of the classic VF graph matching algorithm [18], where link-on-link mapping has been relaxed. However, this algorithm cannot work when the location constraints on the virtual nodes are taken into consideration.

Houidi et al. [19] present a distributed offline VN embedding algorithm that achieves embedding through communicating and exchanging messages between agent-based substrate nodes. Although no centralized algorithms can avoid a single point of failure, the performance and scalability of this distributed algorithm does not achieve comparable performance with the centralized ones. Besides, their algorithm assumes that substrate network resources are unlimited to accept and handle all VN requests.

To maximize the aggregate performance across the VNs, He et al. [20] propose an architecture called DaVinci to dynamically adapt VNs for a customized network substrate, where each substrate link periodically reassigns bandwidth shares among its virtual links. A distributed protocol is run on each VN to optimize the VN's own performance objective independently, such as minimizing the delay or maximizing the throughout. While it makes efficient use of substrate bandwidth resource, DaVinci does not take the node resource constraints into consideration, and the stability of the system is a concern since it adjusts the bandwidth resource periodically.

Since network conditions change over time due to the arrival and departure of VNs, resources in the substrate network may become fragmented. Butt et al. [21] present a topology-aware measure using scaling factors for the substrate network, which identifies the bottleneck nodes and links in the substrate network. They then propose a set of algorithms for re-optimizing and re-embedding initially-rejected VN requests. Since these algorithms can also be applied when our algorithms fail to map a VN request, this work is complementary to ours.

VN embedding across multiple domains (i.e., InPs) has been studied in recent years. For example, Chowdhury et al. [22] present a policy-based inter-domain VN embedding framework that embeds end-to-end VNs in a decentralized manner. Houidi et al. [23] present an optimization problem and a heuristics algorithm for the problem.

Our work is on VN embedding in the same substrate domain, which differs from the existing publications in several ways. First, different from existing studies [7–11], we address the online VN embedding problem with admission control, and do not need to reduce the problem space. Second, we consider both available resources and topology properties of a node to rank the relative importance of a node, which can be leveraged in the mapping procedure. Different from existing work that only consider the resource (e.g., CPU, bandwidth, or both) of a node without considering its topology property in computing the resource availability, our work mends this gap. Third, our topology-aware node ranking measure is leveraged to benefit the current VN embedding process, rather than identifying the bottlenecks of the substrate nodes and links for the VN embedding re-optimization process proposed in [21]. Different from the previous work [13], we formulate an ILP model for the VN embedding problem when path splitting is not supported by the substrate network. We devise a meta-heuristic based VN embedding algorithm to obtain near-optimal solutions, which offers much better performance than algorithms based on path splitting presented in [13] in terms of the ratio of the long-term average revenue over the VN request acceptance.

Page et al. [12] use random walk model to rank the relative importance of web pages, where the rank of a page depends on the topological properties of the weighted links between the pages, regardless of their content. A more general framework for this scheme was proposed in [24]. Another example of using random walk model is to compute a set of topological signatures for each node in a graph, and it is also shown to be effective for exact (and approximate) graph matching (see [25]). In a typical graph matching problem, there are either no weights on the nodes or links, or there are only visual features (e.g., in the RGB color space) contained by the nodes without links. Unlike PageRank and graph matching, however, in the VN embedding problem there are weights on both nodes and links, and the weights are typically non-uniform.

PSO is an emerging population-based optimization method, first introduced by Eberhart and Kennedy in 1995 [26], which is inspired by the flocking behavior of many species, like birds or school of fish, in food hunting. It is a random search algorithm that simulates an evolutionary process. Easily implementable, efficient, and with stable convergence, PSO has a large number of successful applications in finding near-optimal solutions to difficult optimization problems. PSO was proposed to handle continuous nonlinear optimization problems, which cannot be used directly on discrete optimization problems such as the optimal VN embedding problem. To tackle this problem, several variants of PSO for discrete optimization problems such as [27] and [28] are proposed. However, they are problem-specific and thus cannot directly be used for the optimal VN embedding problem. Different from the existing discrete PSO algorithms, in RW–PSO, the parameters and operations of the particles (e.g., position, velocity and update) are redefined according to the VN embedding problem. In addition, a local selection strategy is devised to initialize and update positions of particles to accelerate convergence.

3. Network model and problem description

3.1. Substrate network

We denote the topology of a substrate network by a weighted undirected graph $G_S = \left(N_S, L_S, A_S^n, A_S^l\right)$, where N_S is the set of the substrate nodes, L_S the set of the substrate links, A_S^n the attributes of the substrate nodes, and A_S^l the attributes of the substrate links. The attributes of a node may include processing capacity, storage, and location. The attributes of a link may include the bandwidth and delay. In this paper, we consider the available CPU capacity and location constraint as node attributes and the available bandwidth as link attribute. Fig. 1(b) is a sample substrate network, where the numbers in rectangles represent available CPU resources at the nodes and the numbers over the links represent available bandwidths.

3.2. Virtual network request

Denote the topology of a VN by a weighted undirected graph $G_V = \left(N_V, L_V, R_V^n, R_V^l\right)$, where the N_V is the set of the virtual nodes and L_V the set of the virtual links. The resource requirements on virtual nodes and virtual links are denoted by R_V^n and R_V^l, respectively. Denote each VN request by $VNR^{(i)}(G_V, t_a, t_d, W)$, where t_a and t_d are variables, denoting, respectively, the arrival time of the VN request and the duration of the VN staying in the substrate network. Similar to [13], W is a non-negative value indicating how far a virtual node $n_V \in N_V$ can be placed from the location specified by $Loc(n_V)$. When the ith VNR arrives, the substrate network allocates resources to the VN that satisfy the constraints of the virtual nodes and links. If there are no sufficient substrate resources available, the VNR is either rejected or postponed. The allocated substrate resources are released when the VN departs. Fig. 1(a) and (c) are two VN requests with node and link constraints.

3.3. Virtual network embedding problem description

The VN embedding problem is defined by a mapping

$$M : G_V(N_V, L_V) \to G_S(N'_S, P'_S)$$

from G_V to a subset of G_S, where $N'_S \subset N_S$, $P'_S \subset P_S$ and P_S denotes the set of all loop-free paths of the substrate network. The mapping can be decomposed into node mapping followed by link mapping. Node mapping places virtual nodes to different substrate nodes that satisfy the node resource constraints, and link mapping assigns virtual links to loop-free paths on the substrate that satisfy the link resource requirements.

Fig. 1(a) and (b) show a VN embedding solution for VN request 1. For example, the node mapping solution of VN request 1 is $\{a \to A, b \to B, c \to F\}$ and the link mapping solution is $\{(a,b) \to (A,B), (a,c) \to (A,F), (b,c) \to (B,F)\}$. After the substrate network allocates resources to VN request 1, the residual capacities of the substrate nodes and links are shown in Fig. 1(d). Fig.1(c) and (d) show a VN embedding solution for VN request 2. Note that the virtual nodes of different VN requests can be mapped onto the same substrate node, but the virtual nodes in the same VN request cannot share the same substrate node.

3.4. Objectives

Similar to the early work in [7,10,13], the revenue of serving a VN request at time t is defined by the following formula:

$$R(G_V, t) = \sum_{n_V \in N_V} CPU(n_V) + \sum_{l_V \in L_V} BW(l_V), \tag{1}$$

where $CPU(n_V)$ and $BW(l_V)$ are, respectively, the CPU and bandwidth requirements for the virtual node n_V.

Define the cost of serving a VN request at time t to be the sum of the total resources in the substrate network allocated to the VN as

$$C(G_V, t) = \sum_{n_V \in N_V} CPU(n_V) + \sum_{l_V \in L_V} \sum_{l_S \in L_S} a(l_S, l_V) \cdot BW(l_S), \tag{2}$$

where $BW(l_S)$ is the bandwidth of l_S, $a(l_S, l_V) \in \{0,1\}$, $a(l_S, l_V) = 1$ iff substrate link l_S is allocated to virtual link l_V. Note that for $a(l_S, l_V) = 1$ it is necessary to have $BW(l_S) \geqslant BW(l_V)$. The cost of serving a VN request depends on the chosen substrate paths. This formula neglects the costs of processing packets.

An effective online VN embedding algorithm should maximize the revenue of InPs and accept the largest possible number of VN requests in a given period of time. The long-term average revenue, denoted by R_S, is defined by [10]

$$R_S = \lim_{T \to \infty} \frac{\sum_{t=0}^{T} R(G_V, t)}{T}. \tag{3}$$

The VN request acceptance ratio of the substrate network, denoted by R_A, can be defined as:

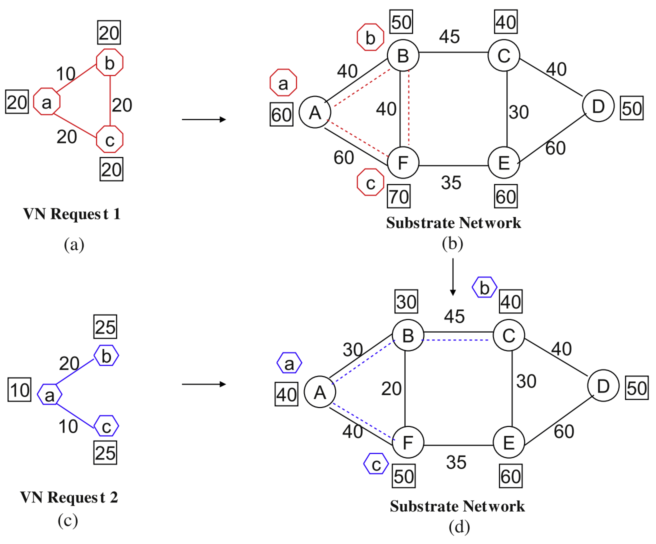


Fig. 1. Examples of VN embedding.

$$R_A = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} VNR_S}{\sum_{t=0}^{T} VNR}, \tag{4}$$

where $VNR_S \in \{0,1\}$, $VNR_S = 1$ iff the given VN request is successfully accepted by the substrate network, and $\sum_{t=0}^{T} VNR$ is the total number of VN requests.

We measure the efficiency of substrate resource usage in terms of the long-term revenue to cost (R/C) ratio, denoted by R_c, which is defined as:

$$R_C = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} R(G_V,t)}{\sum_{t=0}^{T} C(G_V,t)}. \tag{5}$$

Our objective is to maximize the long-term average revenue R_S. When the long-term average revenues of multiple VN embedding solutions are nearly the same, we seek the one that offers the highest ratio of the VN request acceptance ratio over the long-term R/C ratio.

4. Noderank: a ranking for every node in the network topology

VN embedding involves a mapping of nodes and a mapping of links. Node mapping can be achieved by selecting substrate nodes with sufficient CPU resources. Link mapping requires sufficient link resources on the path between selected nodes in the substrate networks assigned to a path in a VN. Most early publications (e.g., [10]) perform node mappings and link mappings at two different stages, where nodes are selected first at the node-mapping stage, and link allocation and path selection are done at the link-mapping stage. Incorporating topology attributes during the node mapping stage, we take a different approach to link mapping, aiming to improve the success rate and efficiency of link mapping. A motivational example is illustrated in Fig. 2, where larger nodes are nodes with more CPU resources and the wider lines are links with more bandwidth resources. Nodes $n1$ and $n2$ seem to have the same resource available if they are considered alone. However, $n1$ is a "better" node because the neighbors of $n1$, namely, $A1$, $B1$, and $C1$, have more resources than those of $n2$'s neighbors, and so mapping a virtual node to $n1$ has a higher chance to achieve a successful link mapping.

We define the notion of *node rank* to measure the resource availability of a node and denote by *NodeRank* the value of the node rank of a node. Intuitively, the rank of a given node u is determined by its CPU power and its collective bandwidth of outgoing links. It is also affected by the ranks of the nodes that can be reached from u. We model the first using the product of its CPU and collective bandwidth of outgoing links as in [10]. We model the second by dividing reachable nodes into two groups, that is, the nodes that are incident to the outgoing links of u and the nodes that can be reached from u via multiple hops. Modeling connectivity is challenging. In this paper, we define a jumping probability to model the likelihood of a node that is reachable from u via multiple hops and define a forward probability to model the influence of the neighboring nodes from u's forward links. In particular, let

$$H(u) = CPU(u) \sum_{l\in L(u)} BW(l), \tag{6}$$

where, on a substrate network, $L(u)$ is the set of all the outgoing links of u, $CPU(u)$ is the remaining CPU resource of u, and $BW(l)$ is the unoccupied bandwidth resource of link l. On a VN, $CPU(u)$ and $BW(l)$ are the capacity constraints of the node u, respectively. The initial NodeRank value for node u can be computed by

$$NR^{(0)}(u) = \frac{H(u)}{\sum_{v\in V} H(v)}. \tag{7}$$

Let u, $v \in V$ be two different nodes. Let

$$p^J_{uv} = \frac{H(v)}{\sum_{w\in V} H(w)}, \tag{8}$$

$$p^F_{uv} = \frac{H(v)}{\sum_{w\in nbr_1(u)} H(w)}, \tag{9}$$

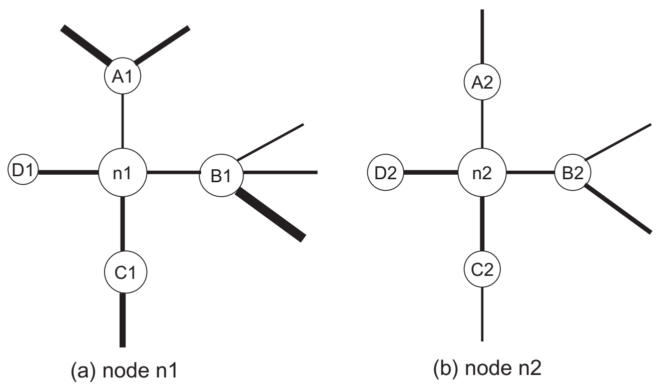


Fig. 2. Motivational example.

where p^J_{uv} denotes the jumping probability from node u to land on node v, $nbr_1(u) = \{v|(u,v) \in E\}$, and p^F_{uv} denotes the forward probability from node u to node v with $(u,v) \in E$. Clearly,

$$\sum_{v\in V} p^J_{uv} = 1, \qquad \sum_{v\in nbr_1(u)} p^F_{uv} = 1.$$

The probabilities p^J_{uv} and p^F_{uv} may be viewed as voting for resources of node u by nodes reachable from u and their neighboring nodes. Voting by a non-neighboring node implies that there should exist a multi-hop path between u and v. Thus, the topology information of the two nodes is also embedded in the probabilities.

For any node $v \in V$, let

$$NR^{(t+1)}(v) = \sum_{u\in V} p^J_{uv} \cdot p^J_u \cdot NR^{(t)}(u) + \sum_{u\in nbr_1(v)} p^F_{uv} \cdot p^F_u \cdot NR^{(t)}(u), \quad (10)$$

where $p^J_u + p^F_u = 1$, $p^J_u \geqslant 0$, $p^F_u \geqslant 0$, and $t = 0,1, \ldots$ The p^J_u and p^F_u are bias factors, and we will typically want to set p^J_u to 0.15 and p^F_u to 0.85 (for details how these values are determined please see Section 7).

For a network of n nodes with $V = \{v_1, v_2, \ldots, v_n\}$, let $NR^{(t)}_i = NR^{(t)}(v_i)$ and denote the vector of NodeRanks at iteration t by $NR^{(t)} = (NR^{(t)}_1, NR^{(t)}_2, \ldots, NR^{(t)}_n)^T$, where $t = 0,1,\ldots$ We have

$$NR^{(t+1)} = \mathbf{T} \cdot NR^{(t)},$$

where $\mathbf{T}$ is a one-step transition matrix of the Markov chain defined by

$$\mathbf{T} = \begin{pmatrix} p^J_{11} & p^J_{12} & \cdots & p^J_{1n} \\ p^J_{21} & p^J_{22} & \cdots & p^J_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p^J_{n1} & p^J_{n2} & \cdots & p^J_{nn} \end{pmatrix} \cdot \begin{pmatrix} p^J_1 & 0 & \cdots & 0 \\ 0 & p^J_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p^J_n \end{pmatrix} + \begin{pmatrix} 0 & p^F_{12} & \cdots & p^F_{1n} \\ p^F_{21} & 0 & \cdots & p^F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p^F_{n1} & p^F_{n2} & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} p^F_1 & 0 & \cdots & 0 \\ 0 & p^F_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p^F_n \end{pmatrix}. \quad (11)$$

Note that $\mathbf{T}$ is a stochastic matrix with 1 as its maximum eigenvalue. This guarantees that the above recurrence relation converges to

$$NR^{(*)} = \left(NR^{(*)}_1, NR^{(*)}_2, \ldots, NR^{(*)}_n\right)^T,$$

which is referred to as the steady state distribution [29] and can be computed using a classic iterative scheme [12], given by Algorithm 1.

Algorithm 1: The NodeRank computing method.

```
1: Given a positive value $\epsilon$, $i \leftarrow 0$
2: repeat
3:     $NR^{(i+1)} \leftarrow \mathbf{T} \cdot NR^{(i)}$
4:     $\delta \leftarrow \|NR^{(i+1)} - NR^{(i)}\|$
5:     $i++$
6: until $\delta < \epsilon$
```

We devise two efficient VN embedding algorithms based on the measure of NodeRanks, denoted by RW-MaxMatch and RW–PSO, respectively. In the following two sections, we will describe the details of these two algorithms.

5. RW-MaxMatch: mapping with NodeRank

Leveraging the benefits of NodeRank, we devise RW-MaxMatch, a two-stage VN embedding algorithm. During the first phase, it first computes the NodeRank for each virtual node and substrate node. It then sorts virtual nodes in non-increasing order according to their NodeRanks, and does the same thing on substrate nodes. Let $m = |N_V|$ and $n = |N_S|$. Without loss of generality, let the sorted virtual nodes be

$$N^1_V, N^2_V, \ldots, N^m_V, \text{ where } NR_1 \geqslant NR_2 \geqslant \cdots \geqslant NR_m,$$

and let the sorted substrate nodes be

$$N^1_S, N^2_S, \ldots, N^n_S, \text{ where } NR_1 \geqslant NR_2 \geqslant \cdots \geqslant NR_n.$$

RW-MaxMatch maps N^1_V to N^1_S, provided that the CPU capacity, the bandwidth capacity, and the location requirement of N^1_S satisfy what node N^1_V asks for. It then maps N^i_V to N^i_S in a similar way, for $i = 2, \ldots, m$. For convenience we call this mapping a *TL2S2 mapping* (which stands for "topology-aware large-to-large and small-to-small" mapping). Such mapping has the following two advantages:

1. It helps to satisfy the resource requirement of the current VN request especially for the link mapping stage.

2. It helps to balance the substrate network loads.

Algorithm 2 executes the first phase.

In the second phase, RW-MaxMatch maps virtual links to substrate links (see Algorithm 3). For a given virtual link with bandwidth resource constraint $BW(l_V)$, we first remove the substrate links with residual bandwidth smaller than $BW(l_V)$. We then find a shortest path on the substrate network between these two mapped virtual nodes.

Algorithm 2: RW-MaxMatch: The node mapping stage

```
1: Compute NodeRanks of all nodes in both $G_S$ and $G_V$
   using Algorithm 1 with a given value of $\epsilon$.
2: Sort the nodes in $G_S$ in non-increasing order of
   NodeRanks.
3: Sort the nodes in $G_V$ in non-increasing order of
   NodeRanks.
4: Map virtual nodes to substrate nodes using the
   TL2S2 mapping procedure.
```

Algorithm 3: RW-MaxMatch: The link mapping stage

```
1: Enqueue each virtual link $l_V$ of the VN request to a
   queue Q.
2: Dequeue a virtual link $l_V$ from Q. Remove those
   substrate links that cannot satisfy the bandwidth
   requirement of $l_V$. Use the shortest path algorithm
   to find a link mapping solution for $l_V$. If it cannot
   find a path for $l_V$,then return FAILED.
3: Goto Step 2 if Q is not empty, and return SUCCESS
   otherwise.
```

It has been shown that the iterative scheme (i.e., Algorithm 1) yields any desired precision ϵ with a number of iterations proportional to $\max\{1, -\log\epsilon\}$ [30]. Finding a shortest path can be done in polynomial time. Thus, RW-MaxMatch is a polynomial-time algorithm in terms of $|G_S|$, $|G_V|$, and $\max\{1, -\log\epsilon\}$.

6. Mapping with NodeRank: RW–PSO

A given VN request may be embedded in the substrate network in a number of different ways. Different embedding solutions could result in different substrate resource costs. This phenomenon can be seen in the examples of VN embedding presented in Section 3 (Fig. 1). Suppose substrate nodes B and F satisfy all the requirements of the virtual nodes b and c in VN request 2. We can construct a VN embedding solution for VN request 2 differently from the one presented in Section 3, with node mapping: $\{a \to A, b \to B, c \to F\}$ and link mapping: $\{(a,b) \to (A,B), (a,c) \to (A,F)\}$. It can be verified that this VN embedding solution consumes less substrate network resources than the solution presented in Fig. 1(d) and thus increases the possibility of accepting more VN requests.

RW-MaxMatch, however, does not consider substrate cost. To address this issue we formulate an ILP model to minimize the substrate resource cost for a VN embedding. We present RW–PSO, a meta-heuristic algorithm, to obtain an approximation to the ILP model. Leveraging the meta-heuristic optimization and NodeRanks, RW–PSO can reduce the VN embedding cost while maintaining a high acceptability of VN requests. Compared to RW-Maxmatch, RW–PSO may run a little longer time, but can reduce substrate resource cost significantly.

We first present in Section 6.1 the ILP model and discuss the design choice of using PSO as the optimizer to obtain an approximation of this ILP. Next in Section 6.2, we briefly introduce PSO, and discuss challenging issues when employing PSO to the optimal VN embedding problem. Sections 6.3 and 6.4 describe how to deal with these challenges. We present RW–PSO in Section 6.5.

6.1. ILP for optimal VN embedding

Recall that $a((i,j),(u,v))$ is a binary variable, where $a((i,j),(u,v)) = 1$ if virtual link (u,v) is routed on physical link (i,j) and 0 otherwise.

Let x_i^u be a binary variable such that $x_i^u = 1$ if virtual node u is mapped to the substrate node i and 0 otherwise.

The MIP model formulated by Chowdhury et al. [13] assumes that the substrate network supports path splitting. We formulate an ILP model for the VN embedding problem without this assumption. The ILP formulation is given below, where W is a pre-determined positive number.

Objective:

$$\mathit{Minimize} \sum_{(u,v)\in L_V} \sum_{(i,j)\in L_S} a((i,j),(u,v)) \cdot BW((u,v)) \quad (12)$$

Capacity Constraints:

$$(\forall u \in N_V)(\forall i \in N_S) : \begin{cases} x_i^u \cdot CPU(u) \leqslant CPU(i) \\ x_i^u \cdot Dis(Loc(i), Loc(u)) \leqslant W \end{cases} \quad (13)$$

$$(\forall (i,j) \in L_S)(\forall (u,v) \in L_V) :$$
$$a((i,j),(u,v)) \cdot BW((u,v)) \leqslant BW((i,j)) \quad (14)$$

Connectivity Constraints

$$(\forall i \in N_S)(\forall (u,v) \in L_V) :$$
$$\sum_{(i,j)\in L_S} a((i,j),(u,v)) - \sum_{(j,i)\in L_S} a((i,j),(u,v)) = \begin{cases} 1, & \text{if } x_i^u = 1 \\ -1, & \text{if } x_i^v = 1 \\ 0, & \text{otherwise} \end{cases} \quad (15)$$

Variable Constraints:

$$(\forall i \in N_S) : \sum_{u\in N_V} x_i^u \leqslant 1,$$
$$(\forall u \in N_V) : \sum_{i\in N_S} x_i^u = 1 \quad (16)$$
$$(\forall i \in N_S)(\forall u \in N_V) : x_i^u \in \{0,1\},$$
$$(\forall (i,j) \in L_S)(\forall (u,v) \in L_V) : a((i,j),(u,v)) \in \{0,1\} \quad (17)$$

Remarks

- For a VN request, the CPU costs of different VN embedding solutions, neglecting the CPU costs of processing packets at intermediate nodes on a communication path, are the same. (The CPU costs of processing packets, if not negligible, may be incorporated in the bandwidth of corresponding nodes in our model.) Thus, we only consider the cost of bandwidth resource in the objective function.

- Constraints (13) and (14) specify node constraints and link constraints, respectively. For the node constraints, the CPU capacity of the substrate node i must satisfy the CPU request of the virtual node u, and its location must be within the range of the virtual node specified by W, which indicates how far the virtual node can be placed from the location specified by $Loc(u)$. The distance function Dis denotes the Euclidean distance of two nodes. For example, suppose node n_1 is located at (x_1,y_1) and node n_2 at (x_2,y_2), then the value of $Dis(Loc(i),Loc(u))$ is equal to $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$. For the virtual link constraints, the substrate link (i,j) must meet the bandwidth requirement for the virtual link (u,v).
- Constraint (15) is the flow conservation constraint for routing one unit of traffic from substrate node u to substrate node v. It requires that for any substrate path that serves virtual link (u,v), if the starting node is not for u and the ending node is not for v, then the flow coming into u is the same as that going out from v. (This constraint specifies that no path can be split.) Furthermore, node u has an exogenous input of 1 unit of traffic that has to find its way to the substrate node corresponding to node v.
- Constraint (16) ensures that a virtual node must correlate with just one substrate node.
- Constraint (17) denotes the binary domain constraints for the variables $a((i,j),(u,v))$ and x_i^u.

Solving ILP is in general an NP-hard problem [14], and so we turn our attention to finding a feasible solution that is near optimal. Techniques to solve ILP can be roughly classified into two main categories: exact algorithms and approximation algorithms. Exact algorithms such as Branch and Bound (BB) and Cutting Plane (CP) are guaranteed to find an optimal solution. These algorithms, however, incur exponential running time, and so only instances of a moderate size could be practically solved [31]. For approximation algorithms, one often use a local optimal strategy to achieve a near-optimal feasible solution. The technique of meta heuristics has been shown useful in practice, including Genetic Algorithm (GA)[32], Simulated Annealing (SA) [33], Evolutionary Programming (EP) [34], and Particle Swarm Optimization (PSO) [26]. These are iterative search techniques inspired from biological and physical phenomena, which have been successfully applied to a wide range of optimization problems. In particular, PSO is a population-based stochastic global optimizer that can, compared to other population-based methods, generate better optimal solution in lesser computing time with stable convergence [35]. It is also easy to implement with smaller number of parameters to adjust. We choose PSO as our optimizer to find a feasible solution to the ILP model.

6.2. PSO basics

In PSO, a swarm of particles, represented as potential solutions, are flying through the problem space following the current optimum particles. Each particle i is associated with two vectors: the position vector

$$X_i = (x_i^1, x_i^2, \ldots, x_i^D)$$

and the velocity vector

$$V_i = (v_i^1, v_i^2, \ldots, v_i^D),$$

where D denotes the dimensions of the solution space. The position and velocity of each particle can be initialized randomly within the corresponding ranges.

Let w denote the inertia weight, c_1 the cognition weight, and c_2 the social weight. Let r_1^d and r_2^d denote two random variables uniformly distributed in the range of $[0,1]$ for the dth dimension. Let $pBest_i$ be the position with the best fitness found so far for the ith particle, and $gBest_i$ the best position in the swarm. During the evolutionary process, the position and velocity of particle i on the dth dimension are updated as follows:

$$v_i^d = w \cdot v_i^d + c_1 \cdot r_1^d \left(pBest_i^d - x_i^d\right) + c_2 \cdot r_2^d \left(gBest^d - x_i^d\right) \quad (18)$$

$$x_i^d = x_i^d + v_i^d. \quad (19)$$

We need to address the following two issues before we can adopt PSO to minimize the cost of embedding VN requests.

1. Standard PSO only deals with continuous optimization problem, and so it is not directly applicable to the optimal VN embedding problem, for it is a discrete optimization problem.
2. The randomness of PSO may result in slow convergence. In addition, it may also fragment the substrate network resources and hinder the substrate network from accepting larger VN requests.

We first devise a variant of discrete PSO by redefining the parameters and operations of the particles. We then present a *TL2S2 preferred selection strategy* to initialize and update positions of the particles, which helps accelerate the convergence speed and balance the substrate network loads.

6.3. Discrete PSO for VN embedding

Label the virtual nodes and substrate nodes, respectively. Define the position and velocity parameters for discrete PSO as follows:

Position (X): Let $X_i = (x_i^1, x_i^2, \ldots, x_i^D)$, a position vector of a particle, denote a possible VN embedding solution, where x_i^d is the ith substrate node selected from a candidate node list of the dth virtual node, and D the total number of virtual nodes in the VN request. The candidate node list of the virtual node consists of substrate nodes that satisfy the resource constraints of this node. Note that the position vector only represents a node mapping. We can use Algorithm 3 to map links after nodes are mapped. If a link mapping solution is found, we call this position a *feasible position* for the current VN request.

Velocity (V): The velocity vector

$$V_i = (v_i^1, v_i^2, \ldots, v_i^D)$$

of the particle is used to guide the current VN embedding solution to achieve a better solution, where v_i^d is a binary variable. If $v_i^d = 0$, then the corresponding virtual node mapping in the current VN embedding solution should be adjusted by selecting another substrate node from its candidate node list; otherwise, the current choice remains.

The subtraction, addition, and multiplication operations of the particles are redefined as follows:

Subtraction ($\ominus$): $X_i \ominus X_j$ returns the difference of the two VN embedding solutions X_i and X_j. The resulted value of the corresponding dimension is 1 if X_i and X_j have the same values at the same dimension, and 0 otherwise. For example,

$$(1,7,3,5,6) \ominus (1,9,3,2,6) = (1,0,1,0,1).$$

Addition ($\oplus$): $P_iV_i \oplus P_jV_j$ means to keep V_i with probability P_i and keep V_j with probability P_j in the corresponding dimension, where $P_i + P_j = 1$. For example,

$$0.2(0,1,0,1,0) \oplus 0.8(0,1,1,0,0) = (0,1,*,*,0),$$

where $*$ denotes the probability of being 0 or 1. In this example, the first $*$ is equal to 0 with probability 0.2, and the second $*$ is equal to 1 with probability 0.8.

Multiplication ($\otimes$): $x_i^d \otimes v_i^d$ indicates the position update process of the particles. The result of this operation is a new position that corresponds to a new VN embedding solution. Its operation rule is this: If the value of V_i in the dth dimension equals 1, the value of X_i in the corresponding dimension will be kept; otherwise, the value of X_i in the corresponding dimension should be adjust by selecting another substrate node from its candidate list. Taking $(1,5,7,3,8) \otimes (0,1,1,1,0)$ as an example, the first and fifth virtual node embedding solutions should be adjusted.

The position and velocity of particle i on the dth dimension are determined according to the following velocity and position update recurrence relations:

$$v_i^d = P_1 v_i^d \oplus P_2\left(pBest_i^d \ominus x_i^d\right) \oplus P_3\left(gBest^d \ominus x_i^d\right)\Big), \tag{20}$$

$$x_i^d = x_i^d \otimes v_i^d, \tag{21}$$

where P_1 is the inertia weight, P_2 the cognition weight, and P_3 the social weight. Typically, P_1, P_2, and P_3 are set to constant values that satisfy the inequality

$$0 < P_1 \leqslant P_2 \leqslant P_3 < 1(P_1 + P_2 + P_3 = 1).$$

6.4. TL2S2 preferred local selection strategy

For the basic PSO, it is common to generate and update the position parameters of the particles randomly within the corresponding range with equal probability during the evolutionary process. In VN embedding, however, if we overuse the bottleneck resources of the substrate network, it may fragment the substrate network resources and thus prevent the substrate network from accepting a larger VN network request. Moreover, since the position vector of a particle corresponds to a node mapping solution, if we randomly generate one or more dimensions of this vector, it may violate the capacity constraint (see Eq. 14) or the connectivity constraint (see Eq. 15) of virtual links in the linking mapping stage, which may slow down the convergence speed of our algorithm. We present a TL2S2 preferred selection strategy for position initialization and update of the particles to achieve quick convergence and balance substrate network loads.

The TL2S2 preferred selection strategy shares the same idea with TL2S2 mapping presented in Section 5, where a virtual node with larger NodeRank has higher probability to be mapped to a substrate node with larger NodeRank. Such a strategy can help produce a larger number of VN embedding solutions during the iterative process of our PSO-based algorithm, and thus can help improve the convergence speed of the process. Moreover, since the algorithm tends to choose substrate nodes with more available resources to map virtual nodes, it can balance the substrate network loads to some extent.

For a VN request consisting of n virtual nodes, the TL2S2 preferred selection strategy for position initialization is presented in Algorithm 4.

Algorithm 4: TL2S2 preferred selection strategy for position initialization

```
1: Construct a candidate node list for every virtual
   node.
2: Enqueue all virtual nodes to a priority queue PQ in
   non-increasing order according to NodeRanks. The
   virtual node with larger NodeRank has priority to
   select a substrate node from their candidate node
   list.
3: Dequeue a virtual node $v$ from PQ. Select a
   substrate node from its candidate node list
   according to NodeRanks of the substrate nodes. The
   possibility of a substrate node $i$ being selected is
   equal to
       $\frac{NR_S^i}{\sum_{j=1}^{m} NR_S^j}$,
   where $m$ is the total number of candidates in the
   candidate node list of $v$.
4: Remove the substrate node which has just been
   selected from the candidate node list of all the
   virtual nodes.
5: If PQ is not empty, goto Step3.
```

Similar to the TL2S2 preferred selection strategy for position initialization, for a VN embedding solution that needs to be adjusted, the TL2S2 preferred selection strategy for position update is presented in Algorithm 5.

Algorithm 5: TL2S2 preferred selection strategy for position update

```
1: Construct a candidate node list for each virtual
   node needed to be re-selected.
2: Enqueue these virtual nodes to a priority queue PQ
   in non-increasing order according to their
   NodeRanks. The virtual node with larger NodeRanks
   has priority to select a substrate node from its
   candidate node list.
3: Dequeue a virtual node $v$ from PQ. Select substrate
   node from its candidate node list according to
   NodeRanks of the substrate nodes. The possibility of
   a substrate node $i$ being selected is equal to
   $\frac{NR_S^i}{\sum_{j=1}^{m} NR_S^j}$,
   where $m$ is the total number of nodes in the
   candidate node list of $v$.
4: Remove the substrate node which has just been
   selected from the candidate node list of all the
   virtual nodes.
5: If PQ is not empty, goto Step3.
```

6.5. RW–PSO algorithm description

The RW–PSO algorithm takes the substrate network and a VN embedding request as input, takes the objective function (12) as fitness function $f(X)$, and outputs a feasible VN embedding solution. Note that the feasibility of the current position for each particle will be checked by Algorithm 3. If it is an unfeasible position, the fitness value $f(X)$ of this particle will be set to $+\infty$. The details of this algorithm are presented in Algorithm 6. The flowchart of RW–PSO algorithm is presented in Fig. 3.

Algorithm 6: RW–PSO algorithm

```
1: Construct a candidate substrate node list for each
   virtual node in the VN request.
2: Initialize a population of particles. Use the TL2S2
   preferred selection strategy to initialize the position
   vector $X$ and the velocity vector $V$ for each particle.
   This corresponds to node mapping. Ensure that the
   dimension values of the position vector $X$ are distinct
   pairwise, to guarantee that no two virtual nodes in
   the same VN request will be mapped to the same
   substrate node.
3: Initialize $pBest$ and $gBest$. For each particle, check
   the feasibility of its position. This corresponds to
   link mapping. Set the local best position $pBest$ and
   global best position $gBest$ to these particles
   according to their fitness values of $f(X)$.
4: Update position vector $X$ and velocity vector $V$ for
   each particle. If $f(X)$ of the particle is $< +\infty$, then
   using Recurrence Relations (20) and (21) to update
   the vectors following the TL2S2 preferred selection
   strategy. Otherwise, re-initialize its position vector
   $X$ and its velocity vector $V$ uniformly at random, also
   using the TL2S2 preferred selection strategy.
5: Get the fitness values of these particles. If
   $f(X_i) < f(pBest)$, then set $X_i$ to be the local best
   position of the particle $i$. If $f(pBest) < f(gBest)$, then
   set $pBest$ to be the global best position.
6: If the fitness function $f(X)$ has not reached the preset
   value, then goto Step 4, otherwise goto Step 7.
7: Output the VN embedding solution and stop. The
   fitness function of the final $gBest$ being $< +\infty$
   indicates that a near-optimal VN embedding
   solution has been obtained. Otherwise, there is no
   feasible solution found.
```

7. Performance evaluation

To evaluate the feasibility and efficiency of RW-MaxMatch and RW–PSO, we design and implement a VN embedding simulator, based on the simulator [36] we previously constructed [11]. We will describe our evaluation settings in Section 7.1. After that we will evaluate our algorithms in the following two settings:

1. For a single VN request, we will primarily compare RW–PSO and the existing algorithms on the cost of VN embedding and the running time. Evaluation results are presented in Section 7.2.
2. For a sequence of online VN requests, we will compare the performance between our algorithms and the existing algorithms. The advantage of the topology-aware node resource ranking measure used in our algorithms is also evaluated. Evaluation results are presented in Section 7.3.

7.1. Evaluation settings

Our numeric evaluations consider two types of inputs, namely, inputs of regular sizes and inputs of small sizes. We use inputs of regular sizes to study performance with a single substrate network and various VNs. We use inputs of small sizes to compare approximation solutions with the optimal solutions.

1. For inputs of regular sizes, as in previous work [10,11], we generate substrate networks of 100 nodes and around 500 links, corresponding to an ISP of a medium size. The number of virtual nodes is generated uniformly at random between 2 and 20. This setting is used in Sections 7.2.1 and 7.3.
2. For inputs of small sizes, the number of substrate nodes varies from 10 and 50 nodes. The number of VN nodes is generated uniformly at random between 2 and 3 (The reason why this setting is chosen is given in Section 7.2.2).

The CPU and bandwidth resources of the substrate nodes and links are real numbers uniformly distributed between 50 and 100. The CPU and bandwidth requirements of virtual nodes and links are real numbers uniformly distributed between 0 and 50. The VN connectivity is fixed on average at 50%, which means that an n-node VN has on average $n(n-1)/4$ links. Similar to the work in [13], we use the GT-ITM tool [37] to generate network topology and location attributes of the substrate nodes and virtual nodes. The location coordinates (x,y) of these nodes are real numbers uniformly distributed between 0 and 100. Other evaluation settings for a single VN request and online requests will be presented, respectively, in Sections 7.2 and 7.3.

The parameters used in our algorithms are configured as follows. The combination of the two atomic actions $x(p,j)$ and $x(p,f)$ presented in Formula (10) may help balance between local and global node resources. Our experiments show that setting $x(p,j)$ to 0.15 and $x(p,f)$ to 0.85, the same values used in Google's Page rank search engine, will achieve the best performance. The value for parameter ϵ presented in the iterative scheme is set to 0.0001. The population size in PSO is set to 5 and the maximum number of iterations (a stopping criterion) will be given later. The parameters P_1, P_2, and P_3 are set to 0.1, 0.2, and 0.7, respectively.

All of the simulation experiments are performed on a server with Intel 3 GHz dual-core CPU, 2 GB memory, 160 GB disk, and Linux 2.6 OS. We evaluate eight algorithms, listed in Table 1.

To evaluate the performance of the meta-heuristic technique, we compare RW–PSO with the algorithms presented in [13]. For simplicity, we use MM to denote MaxMatch. Thus, RW-MM is a shorthand of RW-MaxMatch. To evaluate the benefit of the topology awareness technique, we introduce CB-MM, CB-PSO, and R-PSO, which are variants of RW-MM and RW–PSO. Moreover, to quantify the difference between the VN embedding solution obtained by RW–PSO and the optimal VN embedding solution, we compare the approximation solutions obtained by RW–PSO with the optimal solutions obtained by the GNU Linear Programming Kit (GLPK) [38], a standard ILP solver.

Other algorithms such as RW-BFS [11], the VN embedding algorithm based on subgraph isomorphism presented in [17], and the re-optimization mechanism for VN embedding presented in [21] have been studied. They are not included in Table 1 for comparisons for the following reasons:

1. RW-BFS is a local search algorithm with sub-optimal performance on VN requests with location constraints on virtual nodes. Thus, there is no need to compare with this algorithm.
2. The VN embedding algorithm based on subgraph isomorphism is limited to substrate networks and VNs that are modeled as directed graphs. Thus, a fair comparison is not feasible.
3. The re-optimization mechanism for VN embedding presented is complimentary to our algorithms, for it is applied when a VN request cannot be mapped, and so there is no need to compare with this algorithm.

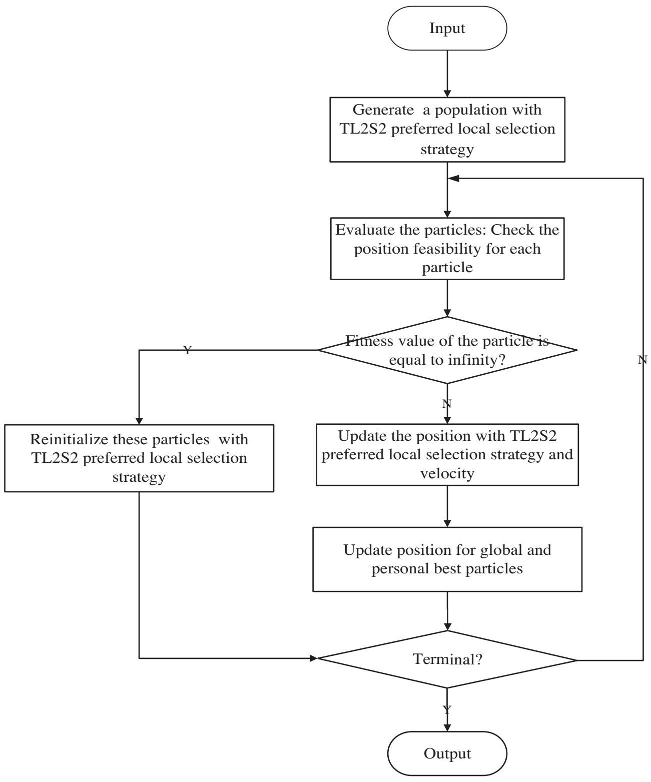

Input
Generate a population with TL2S2 preferred local selection strategy
Evaluate the particles: Check the position feasibility for each particle
Fitness value of the particle is equal to infinity?
Y
N
N
Reinitialize these particles with TL2S2 preferred local selection strategy
Update the position with TL2S2 preferred local selection strategy and velocity
Update position for global and personal best particles
Terminal?
Y
Output

Fig. 3. The flowchart of RW–PSO.

Table 1
Comparisons of Algorithms.

Notation	Algorithm Description
RW-MM	It is the short hand of RW-MaxMatch. It first maps virtual nodes with larger NodeRanks to the substrate nodes with larger NodeRanks, then embed the virtual links using shortest path algorithm when the substrate does not support path splitting
CB-MM	It simply uses Formula (6) to compute node resource similar to [10], where CB denotes the product of CPU and bandwidth. The other steps of CB-MM are the same as RW-MaxMatch
RW–PSO	It uses the TL2S2 preferred preferred selection strategy to initialize and update the positions of the particles
CB-PSO	It simply uses Formula (6) to compute node resource rank similar to [10]. The other steps of CB-PSO are the same as RW–PSO
R-PSO	It is the same as RW–PSO except the use of the TL2S2 preferred selection strategy
D-ViNE-SP	It is a VN embedding algorithm presented in [13]. It uses deterministic node mapping with unsplittable link mapping on shortest path when path splitting is not supported
D-ViNE-LB	It is the best performed VN embedding algorithm presented in [13]. It uses deterministic node mapping with splittable link mapping and load balance using MCF when path splitting is supported
GLPK	It uses the standard ILP solver of GNU Linear Programming Kit (GLPK) [38] to solve the ILP model of the optimal VN embedding problem without any relaxation

7.2. Evaluation for single VN request

We first compare RW–PSO with the existing heuristic-based algorithms presented in [13] on embedding cost and running time. We then compare RW–PSO with the optimal solutions on embedding cost and running time. We use GLPK to solve the ILP model for the optimal solutions on inputs of small sizes.

7.2.1. Comparisons with existing heuristic algorithms

To evaluate embedding cost and running time of the algorithms presented in Table 1 (except GLPK) for a single VN request, we use a regular-sized substrate network and VNs as input. We generate 50 different pairs of substrate networks and virtual networks, and run these algorithms on them. We count the VN requests accepted by these algorithms and record the arithmetic mean as the final results. In these numerical experiments, the maximum number of iterations of RW–PSO is set to 100. Notice that RW-MM and CB-MM may be viewed as special cases of RW–PSO and CB-PSO (i.e., there is only one swarm member without any iteration) to certain extent.

We summarize the major results as follows.

1. *RW–PSO achieves substantially lower cost on VN embedding in shorter running time* Fig. 4 depicts the comparisons of substrate cost on VN embedding between our PSO-based algorithms (i.e., RW–PSO, CB-PSO, and R-PSO) and the existing algorithms. It shows that PSO-based algorithms can achieve substantially lower VN embedding cost than that of D-ViNE-SP and D-ViNE-LB in a reasonable number of iterations. For example, when the number of iterations is 20, the cost of RW–PSO VN embedding is 19% lower than that of D-ViNE-SP and 7% lower than that of D-ViNE-LB. Each PSO-based algorithm can achieve a lower cost on VN embedding than that of D-ViNE-LB in 15 iterations or less. RW–PSO provides the lowest cost with slightly more time. The running time of each PSO-based algorithm increases linearly in terms of the number of iterations, as shown in Fig. 5. Meantime, the cost of VN embedding declines slowly as shown in Fig. 4. When the number of iterations is less than 35, the PSO-based algorithms can produce solutions close to being optimal while incurring less running time. Note that the revenue of the VN request is also shown in Fig. 4. Due to the resource constraints (e.g., the location constraints on virtual nodes), the VN embedding cost is usually higher than the revenue from serving the VN request (note that the formulation of revenue is simply a resource measure, which does not reflect actual monetary values).
2. *RW–PSO gets better convergence due to adopting the TL2S2 preferred preferred selection strategy*. As shown in Fig. 4, RW–PSO incurs lower embedding cost than R-PSO with the same number of iterations. It means that the TL2S2 preferred selection strategy does play an important role in accelerating convergence. The reason is that mapping virtual nodes to the substrate nodes with adequate resources increases the probability of satisfying the resource requirements of the VN request, and consequently helps to accelerate convergence. The running time of RW–PSO increases a little more than that of R-PSO because of the calculations of NodeRank and the sorting procedures (see Fig. 5). In addition, we can also observe that RW–PSO incurs even lower embedding cost than CB-PSO (see Fig. 4). It implies that using a topology-aware NodeRank as a node resource measure is better than simply using the previous method presented in Eq. (6). This is because leveraging NodeRank values of network nodes to initialize or update the position parameters of particles, we can further increase the possibility of meeting resource constraints of the given VN request and therefore it benefits the particle swarm iteration process to find more near-optimal VN embedding solution.

7.2.2. Comparisons with exact algorithm on inputs of small sizes

To quantify how close RW–PSO is to the optimal on VN embedding costs, we compare the solutions obtained by RW–PSO with the solutions obtained by GLPK's ILP solver. Due to the inherent complexity of the optimal VN embedding problem, the time complexity of the ILP solver turned out to be exponential. For example, for a VN with 4 nodes and substrate network with 50 nodes, GLPK's ILP solver takes several days to compute the optimal VN embedding solution. However, it only takes about 5 s for RW–PSO (with 100 iterations) to find a near-optimal VN embedding solution. Thus, we can only carry out comparisons on inputs of small sizes. We set the number of substrate nodes

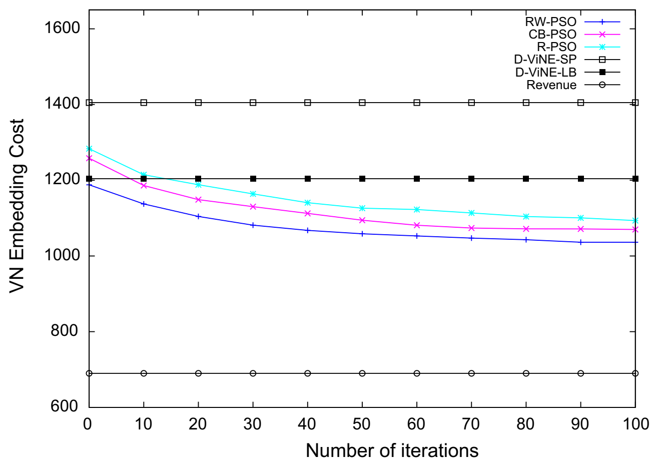


Fig. 4. VN embedding cost comparison with the number of virtual nodes from 2 to 20.

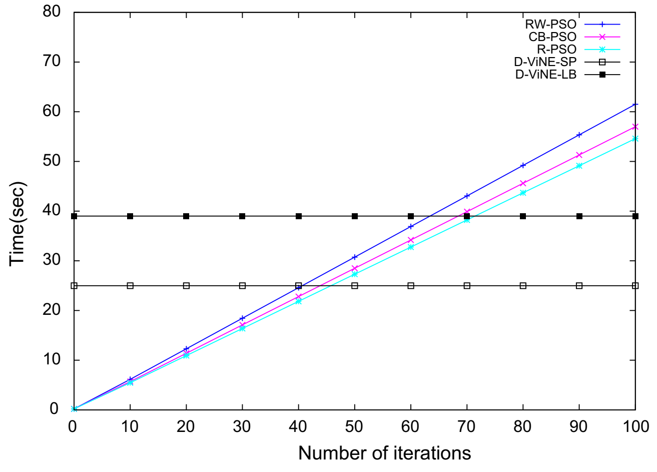


Fig. 5. Running time comparison with the number of virtual nodes from 2 to 20.

between 10 and 50 and the number of virtual nodes from between 2 and 3. Other settings are the same as those in Section 7.2.1. In addition, we also compare the running time of these two algorithms.

Our experiments show that the VN embedding costs obtained by RW–PSO are very close to the minimum and the performance of RW–PSO is stable. In particular, as shown in Fig. 6, for substrate networks with 30 nodes, the VN embedding cost obtained by RW–PSO is 108 and the actual optimal VN embedding cost is 105. Thus the approximation ratio of RW–PSO to the optimal solution is less than 1.03. For substrate network with 40 nodes, the approximation ratio becomes 1.02. It indicates that RW–PSO not only can produce near-optimal solutions but also have stable performance. For the running time comparison, when the substrate nodes is 50, RW–PSO only takes 3 s on average to find a near-optimal solution for a VN request, as shown in Fig. 8. However, the GLPK's ILP solver takes about 500 s on average to get the optimal VN embedding solution, as shown in Fig. 7. Thus, even for small-size input, using standard ILP solver to solve the optimal VN embedding problem would not be practical.

7.3. Evaluation for online VN requests

To evaluate the performance of the algorithms presented in Table 1 for online VN requests, we assume that VN requests arrive following Poisson process with an average arrival rate of 5 VNs per 100 time units, and each one has an exponentially distributed lifetime with an average of 500 time units. We run our simulation for about 50,000 time units, which corresponds to about 2,500 VN requests. The performance metrics include the long-term average revenue defined by Eq. (3), the VN requests acceptance ratio defined by Eq. (4), and the long-term R/C ratio defined by Eq. (5). Notice that we consider the regular-sized input for evaluation of the online VN requests, i.e., the number of the virtual nodes is generated by a uniform distribution between 2 and 20. The time complexity of the GLPK's ILP solver on inputs of this size is too high to compute an optimal solution.

Since the resource condition of the substrate changes over time due to the arrival and departure of VNs, if we set the maximum number of iterations of our PSO-based algorithms too large or too small, it may consume

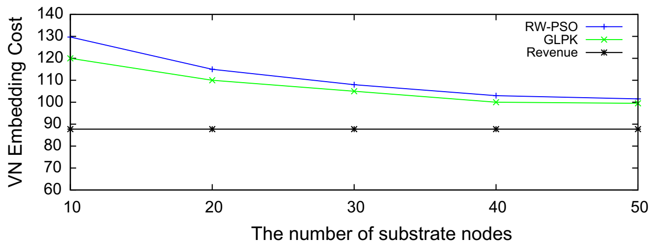


Fig. 6. VN embedding cost comparison with number of virtual nodes between 2 and 3.

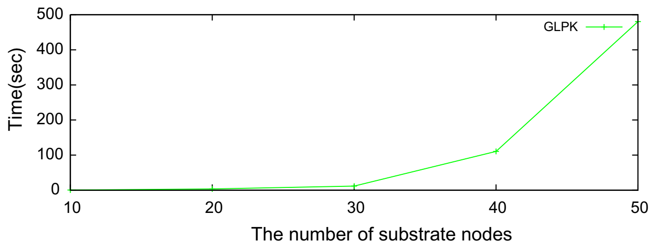


Fig. 7. Running time of RW–PSO with number of virtual nodes between 2 and 3.

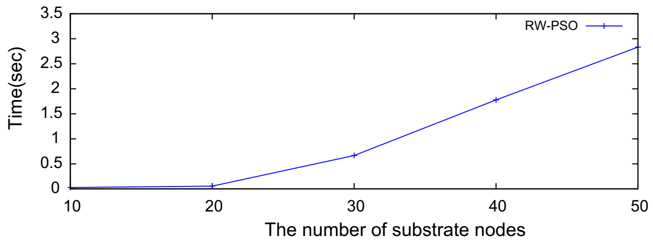
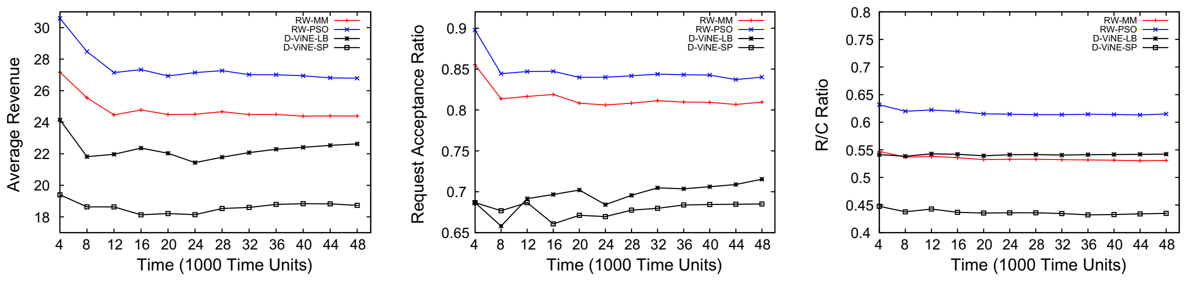


Fig. 8. Running time of RW–PSO with number of virtual nodes between 2 and 3.

excessive running time to obtain a near-optimal VN embedding solution or obtain a suboptimal VN embedding solution in very short running time. Therefore, according to Figs. 4 and 5, we set the maximum number of iterations to 20 to tradeoff between performance and running time.

We summarize the major results as follows:

1. *Both RW-MM and RW–PSO significantly outperform the existing algorithms in term of the long-term average revenue*. As shown in Fig. 9(a), RW-MM and RW–PSO obtain higher long-term average revenues than the existing node and link coordinated VN embedding algorithms D-ViNE-SP and D-ViNE-LB. For example, at the time unit of 48,000, the long-term revenue of RW–PSO is 38% and 16% higher than D-ViNE-SP and D-ViNE-LB, respectively. Since the LP relaxation and rounding techniques used by D-ViNE-SP and D-ViNE-LB may incur infeasible VN embedding solution, they suffer poor performance in term of the VN request acceptance ratio shown in Fig. 9(b) and consequently result in lower long-term average revenues. This may also be the reason why the VN request acceptance ratio of RW-MM is higher than that of the two algorithms. Moreover, as shown in Fig. 9(c), RW–PSO obtains the highest long-term R/C ratio compared to other algorithms, for it produces solutions closer to the minimum than any other algorithms in comparison.
2. *Our topology-aware node resource ranking measure NodeRank can help to increase the VN acceptance ratio*. As shown in Fig. 10(b), the VN request acceptance ratio of RW-MM is higher than that of CB-MM. Correspondingly, RW-MM achieves a larger long-term average revenue than CB-MM, as shown in Fig. 10(a). This is because NodeRank of a network node reflects not only its CPU and bandwidth resource quality but also the resource quality of its neighbors. Thus, leveraging such a topology-aware node resource rank to construct VN embedding solutions can increase the possibility of satisfying the resource requirements of the VN request. We can also notice that the long-term R/C ratios in RW-MM and CB-MM are nearly the same. It implies that using NodeRank alone to construct VN embedding solutions cannot help to reduce VN embedding cost. This observation motivates RW–PSO.



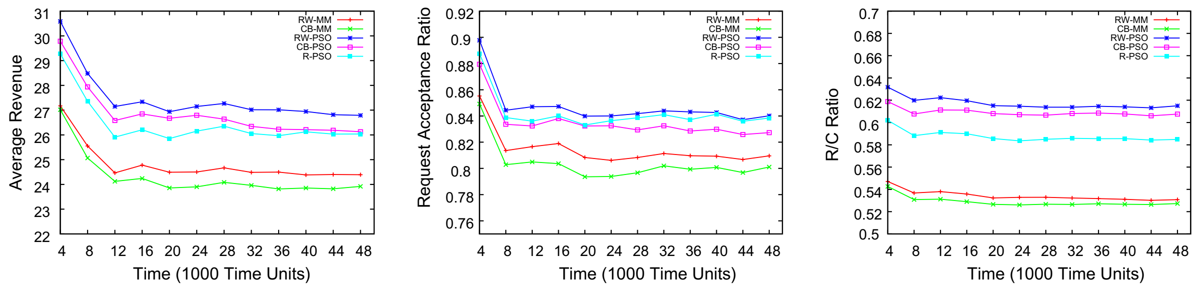
(a) Comparisons of long-term average revenues (b) Comparisons of the VN request acceptance ratio over time (c) Comparison of the long-term R/C ratio

Fig. 9. Comparison between the RW-based and the existing algorithms with number of virtual nodes between 2 and 20.

3. *RW–PSO achieves better performance than RW-MM.* Fig. 9(b) and (a) show that the VN request acceptance ratio and the long-term average revenue of RW–PSO are higher than those of RW-MM. For example, at the time unit of 40,000, the VN request acceptance ratio and the long-term average revenue of RW–PSO are 4% and 8% higher than those of RW-MM, respectively. The reason is that RW–PSO can make more efficient use of substrate resources by minimizing the VN embedding cost for each VN request as shown in Fig. 9(c). The saved substrate resources can be used to accept more future VN requests and consequently generate more revenues. Note that RW–PSO trades off running time for better performance, which means that it consumes more running time than RW-MM. The flexibility of PSO, however, allows us to balance between the desired level of optimization and the running time through the control knobs such as population size and the maximum number of iterations.
4. *RW–PSO gets better performance due to adopting the TL2S2 preferred preferred selection strategy*. As shown in Fig. 10(b), after time unit of 20,000, the VN acceptance ratio of RW–PSO and R-PSO are nearly the same, but the long-term average revenue of R-PSO is much lower than that of RW–PSO, as shown in Fig. 10(a). This is because R-PSO does not consider the load balance of the substrate network, which may easily lead to fragmentation of substrate network resources. Thus, it may accept VN requests of smaller size but reject those of large size. The TL2S2 preferred selection strategy adopted by RW–PSO can help to balance the substrate resource loads and thus avoid this problem. In addition, since adopting the TL2S2 preferred strategy allows RW–PSO to achieve faster convergence, it obtains a higher long-term R/C ratio than that of R-PSO (see Fig. 10(c)). In addition, we notice that the long-term average revenue of RW–PSO is higher than that of CB-PSO (see Fig. 10(a)). This is because that RW–PSO converges faster than CB-PSO under the same number of iterations. Consequently it can save more substrate resources for the current VN request and generate more revenues from accepting more future VN requests.

8. Conclusion

VN embedding is one of the major problems in network virtualization. Efficient online VN embedding algorithms play an important role in generating more revenues for InPs. In general, increasing the VN request acceptance ratio is an effective way to increase revenues. In this paper, we devise VN embedding algorithms through topology awareness and optimization to increase the possibility of accepting the current VN request and more future VN requests. Inspired by Google's PageRank, we formulate a Markov random walk model for computing topology-aware node resource rank of a node, referred to as NodeRank, which can reflect both the resource and the quality of connections of a node. Based on NodeRank, we first propose a two-stage VN embedding algorithm denoted by RW-MaxMatch. We show that leveraging such a node ranking to construct VN embedding can increase the possibility of accepting the current VN request. We devise RW–PSO to increase



(a) The long-term average revenue comparison (b) The VNR acceptance ratio over time comparison (c) The long-term R/C ratio comparison

Fig. 10. Comparison between RW-based, CB-based and other algorithms with number of virtual nodes between 2 and 20.

the possibility of accepting future VN requests while maintaining high acceptability for the current VN request. At the heart of RW–PSO is an enhanced discrete PSO with a biased local selection strategy based on NodeRank. The simulation results show that NodeRank is a better node resource measure and our algorithms outperform the previous approaches in terms of the long-term average revenue over VN request acceptance ratio.

In this paper, we define the amount of node resources presented in Eq. (6) as the same as that defined previous work [10]. But we also find that using other methods to mark the node resources value, such as $\alpha \cdot \text{CPU}+(1-\alpha)\cdot \text{BW}$ $(0<\alpha<1)$, appears to achieve better results in our preliminary experiments. The value of α can be changed dynamically to adapt to the substrate network resource environment when mapping a new virtual network request, and such study is left for further development. Besides, how to leverage the path splitting feature of the substrate network to further improve the performance of RW–PSO can also be a subject of future research.

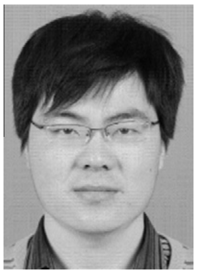
Acknowledgments

The work was supported in part by the following funding agencies of China: National Natural Science Foundation under Grant 61170274, the National Basic Research Program (973 Program) under Grant 2011CB302506, the Innovative Research Groups of the National Natural Science Foundation under Grant 61121061, and the Important National Science & Technology Specific Projects: Research about Architecture of Mobile Internet (2011ZX03002-001-01). This work was also supported in part by the following funding agencies of the United States: a subaward from the NSF/BBN GENI project, the National Science Foundation under Grants CCF-0830314 and CNS-1018422.

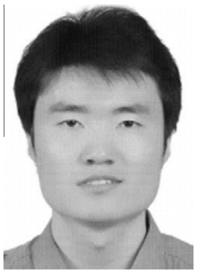
References

[1] T. Anderson, L. Peterson, S. Shenker, J. Turner, Overcoming the Internet impasse through virtualization, IEEE Computer 38 (4) (2005) 34–41.
[2] J. Turner, D. Taylor, Diversifying the internet, in: Proceedings of IEEE GLOBECOM, vol. 2, 2005.
[3] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, S. Shenker, Accountable internet protocol (aip), in: Proceedings of the ACM SIGCOMM, 2008, pp. 339–350.
[4] N. Feamster, L. Gao, J. Rexford, How to lease the Internet in your spare time, ACM SIGCOMM Computer Communication Review 37 (1) (2007) 61–64.
[5] N. Chowdhury, R. Boutaba, Network virtualization: state of the art and research challenges, IEEE Communications magazine 47 (7) (2009) 20–26.
[6] N. Chowdhury, R. Boutaba, A survey of network virtualization, Computer Networks 54 (5) (2010) 862–876.
[7] Y. Zhu, M. Ammar, Algorithms for assigning substrate network resources to virtual network components, in: Proceedings of IEEE INFOCOM, 2006.
[8] J. Lu, J. Turner, Efficient mapping of virtual networks onto a shared substrate, Department of Computer Science and Engineering, Washington University in St. Louis, Technical Report WUCSE-2006 35.
[9] J. Fan, M. Ammar, Dynamic topology configuration in service overlay networks: a study of reconfiguration policies, in: Proceedings of IEEE INFOCOM, 2006.
[10] M. Yu, Y. Yi, J. Rexford, M. Chiang, et al., Rethinking virtual network embedding: Substrate support for path splitting and migration, Computer Communication Review 38 (2) (2008) 17–29.
[11] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, J. Wang, Virtual network embedding through topology-aware node ranking, ACM SIGCOMM Computer Communication Review 41 (2) (2011) 39–47.
[12] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: bringing order to the web, technical report, Stanford digital library technologies project.
[13] N. Chowdhury, M. Rahman, R. Boutaba, Virtual network embedding with coordinated node and link mapping, in: Proceedings of IEEE INFOCOM, 2009.
[14] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons Inc, 1998.
[15] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, B. Yener, Provisioning a virtual private network: A network design problem for multicommodity flow, in: Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing, 2001, pp. 389–398.
[16] R. Ricci, C. Alfeld, J. Lepreau, A solver for the network testbed mapping problem, ACM SIGCOMM Computer Communication Review 33 (2) (2003) 65–81.
[17] J. Lischka, H. Karl, A virtual network mapping algorithm based on subgraph isomorphism detection, in: Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures, 2009, pp. 81–88.
[18] L. Cordella, P. Foggia, C. Sansone, M. Vento, An improved algorithm for matching large graphs, in: Proceedings of the third IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, 2001, pp. 149–159.
[19] I. Houidi, W. Louati, D. Zeghlache, A distributed virtual network mapping algorithm, in: Proceedings of IEEE ICC, 2008, pp. 5634–5640.
[20] J. He, R. Zhang-Shen, Y. Li, C. Lee, J. Rexford, M. Chiang, Davinci: Dynamically adaptive virtual networks for a customized internet, in: Proceedings of the 2008 ACM CoNEXT Conference, ACM, 2008, pp. 1–12.
[21] N. Butt, M. Chowdhury, R. Boutaba, Topology-Awareness and Re-optimization Mechanism for Virtual Network Embedding, in: Proceeding of IFIP Networking, 2010, pp. 27–39.
[22] M. Chowdhury, F. Samuel, R. Boutaba, Polyvine: policy-based virtual network embedding across multiple domains, in: Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, ACM, 2010, pp. 49–56.
[23] I. Houidi, W. Louati, W. Ameur, D. Zeghlache, Virtual network provisioning across multiple substrate networks, Computer Networks 54 (4) (2011) 1011–1023.
[24] M. Diligenti, M. Gori, M. Maggini, D. di Ingegneria dell'Informazione, A unified probabilistic framework for web page scoring systems, IEEE Transactions on knowledge and data engineering 16 (1) (2004) 4–16.
[25] M. Gori, M. Maggini, L. Sarti, Exact and approximate graph matching using random walks, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (7) (2005) 1100–1111.
[26] J. Kennedy, R. Eberhart, et al., Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks 4, 1995, pp. 1942–1948.
[27] J. Kennedy, R. Eberhart, A discrete binary version of the particle swarm algorithm, in: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 1997.'Computational Cybernetics and Simulation'., vol. 5, 1997, pp. 4104–4108.
[28] K. Wang, L. Huang, C. Zhou, W. Pang, Particle swarm optimization for traveling salesman problem, in: Proceedings of the IEEE International Conference on Machine Learning and Cybernetics, vol. 3, 2003, pp. 1583–1585.
[29] E. Seneta, Non-negative Matrices and Markov Chains, Springer-Verlag, 2006.
[30] M. Bianchini, M. Gori, F. Scarselli, Inside pagerank, ACM Transactions on Internet Technology (TOIT) 5 (1) (2005) 92–128.
[31] G. Raidl, J. Puchinger, Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization, Hybrid Metaheuristics (2008) 31–62.
[32] L. Davis, Handbook of genetic algorithms, Arden Shakespeare, 1991.
[33] S. Kirkpatrick, Optimization by simulated annealing: quantitative studies, J. Statistical Phys. 34 (5) (1984) 975–986.
[34] J. Kim, H. Myung, Evolutionary programming techniques for constrained optimization problems, IEEE Trans. Evol. Comput. 1 (2) (2002) 129–140.
[35] P. Bajpai, S. Singh, Fuzzy adaptive particle swarm optimization for bidding strategy in uniform price spot market, IEEE Trans. Power Syst. 22 (4) (2007) 2152–2160.

[36] VNE-RW Simulator. URL <http://int.bupt.edu.cn/~sensu/vne-rw.html>.
[37] E. Zegura, K. Calvert, S. Bhattacharjee, How to model an internetwork, in: Proceedings of IEEE INFOCOM, vol. 2, 1996, pp. 594–602.
[38] GNU Linear Programming Kit. URL <http://www.gnu.org/software/glpk/>.


Xiang Cheng is a Ph.D. Candidate from Beijing University of Posts and Telecommunications in China. His major is Computer Science. His research interests include network support for cloud computing and network virtualization.

Sen Su received the Ph.D. Degree in Computer Science from the University of Electronic Science and Technology, China, in 1998. He is currently a Professor at the Beijing University of Posts and Telecommunications. His research interests include distributed computing systems and the architecture of Internet service.


Zhongbao Zhang is a Ph.D. Candidate from Beijing University of Posts and Telecommunications in China. His major is Computer Science. His research interests include network support for cloud computing and network virtualization.

Kai Shuang received the Ph.D. Degree in Computer Science from Beijing University of Posts and Telecommunications, China, in 2006. He is currently a Associate Professor at the Beijing University of Posts and Telecommunications. His research interests include distributed computing systems and the architecture of Internet service.

Fangchun Yang received his Ph.D. Degree in Communication and Electronic System from the Beijing University of Posts and Telecommunications, China, in 1990. He is currently a Professor at the Beijing University of Posts and Telecommunications. He is a fellow of the IET. His research interests include network intelligence, services computing, communications software, soft switching technology, and network security.

Yan Luo obtained his Ph.D. in Computer Science from University of California Riverside, USA, in 2005. He is currently an Assistant Professor of the Department of Electrical and Computer Engineering at the University of Massachusetts Lowell since 2005. His research interests span the areas of computer architecture and network systems, with recent focus on network virtualization, data center networking, high performance networking with programmable architectures.

Jie Wang received his PhD in Computer Science from Boston University, USA, in 1991. He is currently Professor Chair of the Department of Computer Science at the University of Massachusetts, Lowell. He is also Director of the Center for Network Information Security in the university co-Director of Cyber Forensics Lab in the department. His research interests include computational complexity theory, combinatorial optimization algorithms, computational medicine, network security. He has worked as a security consultant in financial industry. His recent research focus is on network dynamics, knowledge discovery, wireless sensor networks.