Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 8416592, 10 pages
https://doi.org/10.1155/2019/8416592

Research Article

EPVNE: An Efficient Parallelizable Virtual Network Embedding Algorithm

Yuanzhen Li and Yingyu Zhang

School of Computer Science, Liaocheng University, Liaocheng, Shandong 252059, China

Correspondence should be addressed to Yuanzhen Li; liyuanzhen@163.com

Received 6 August 2019; Revised 22 October 2019; Accepted 31 October 2019; Published 22 November 2019

Academic Editor: Hui Cheng

Virtual network embedding (VNE) problem is a key issue in network virtualization technology, and much attention has been paid to the virtual network embedding. However, very little research work focuses on parallelized virtual network embedding problems which assumes that the substrate infrastructure supports parallel computing and allows one virtual node to be mapped to multiple substrate nodes. Based on the work of Liang and Zhang, we extend the well-known VNE to parallelizable virtual network embedding (PVNE) in this paper. Furthermore, to the best of our knowledge, we give the first formulation of the PVNE problem. A new heuristic algorithm named efficient parallelizable virtual network embedding (EPVNE) is proposed to reduce the cost of embedding the VN request and increase the VN request acceptance ratio. EPVNE is a two-stage mapping algorithm, which first performs node mapping and then performs link mapping. In the node mapping phase, we present a simple and efficient virtual node and physical node sorting formula and perform the virtual node mapping in order. When mapping virtual nodes, we map virtual nodes to physical nodes that just meet the CPU requirements. Substrate nodes with more CPU resources will be retained for subsequent virtual network mapping requests. In the link mapping phase, Dijkstra's algorithm is used to find a substrate path for each virtual link. Finally, simulations are carried out and simulation results show that our algorithm performs better than the existing heuristic algorithms.

1. Introduction

As we all know, the Internet has become one of the infrastructures of today's social communications, information exchange, economic and commercial operations, and multimedia services [1]. The existing Internet architecture has played an important role in promoting the rapid development of the Internet. Various new applications emerge in an endless stream, and different applications have different requirements for the network environment in terms of quality of service, security, and scalability [2]. However, the existing network architectures and protocols are difficult to meet the needs of new application development, and there is a certain degree of ossification. As a new technology for building a new Internet architecture, network virtualization can effectively solve this bottleneck of the current Internet.

In recent years, network virtualization technology has been widespread concern in industry and academia [3–5]. Network virtualization technology allows multiple virtual heterogeneous networks to coexist on top of a shared underlying physical network infrastructure. Each virtual network is a piece of resources of the underlying physical network, which consists of virtual nodes (for example, virtual routers and cloud computing center) and virtual links [6]. Additionally, with the programmability of the underlying physical network, virtual networks can run IP or non-IP protocols in the selected network. As a result, network virtualization technologies can deploy new network architectures, protocols, and applications without affecting existing networks, effectively supporting innovation in network technologies. It not only provides a feasible way to evolve from the current Internet to the future network but also provides one of the key features that the Internet should have in the future [7].

In the virtualization network environments, the virtualized network model divides the role of Internet service

provider (ISP) into two separate entities: infrastructure provider (InP) and service provider (SP) [8, 9]. In such a network environment, an infrastructure provider manages a substrate network (SN), which is composed of substrate nodes and physical communication links. Infrastructure providers use virtualization technology to abstract the underlying substrate resources into functional entities, provide a unified programming interface, shield the underlying physical network resources from the upper layers, and split the substrate resources into multiple virtual slices for use by different service providers. In addition, infrastructure providers can communicate and collaborate with each other to create a complete interdomain infrastructure for service providers. Service provider rents slices of substrate resources (e.g., nodes and links) and then creates his own customized virtual network (VN) to provide value-added services (e.g., content distribution) to end users. Service provider deploys customized protocols in virtual networks and is responsible for running, managing, and maintaining virtual networks. Service provider generates a virtual network request according to the needs of the service (e.g., CPU resources, memory resources, bandwidth resources, latency requirements, and geographic location) and forwards the request to the infrastructure provider to establish a virtual network. Regardless of whether the virtual network spans multiple infrastructure providers' domains, the service provider has a unified view of the entire virtual network.

The problem of allocating infrastructure network resources according to virtual network requests with node and link resource constraints is called virtual network embedding (VNE) problem [10]. The virtual network embedding is an important challenge in the realization process of network virtualization technology and has become a hotspot for many scholars. The VNE problem is an NP-hard problem [11]. Even after all virtual nodes have been mapped, mapping virtual links with bandwidth resource constraints is still NP-hard. The virtual network embedding problem is one of the main challenges faced by network virtualization technology. It has become a hot issue in this research field and has received extensive attention from researchers. Many researches on VNE have been reported [5]. However, there is very few focus on parallelizable virtual networks embedding [12]. In the parallelizable virtual network embedding (PVNE) environment, a virtual node could be mapped into multiple substrate nodes. With parallelization, several substrate nodes can parallely accomplish the computation to which the virtual node is dedicated. With parallelization, a substrate network can carry more virtual networks. For example, in cloud computing, MapReduce processes large-scale data in parallel [13, 14]. Another important advantage of parallelization is its robustness that the computation can quickly migrate to other substrate nodes in the event of a substrate node crash [15]. With the development of cyber-physical system and cloud computing [16], the need for parallelization will increase. Parallelization also provides new ideas for the implementation of cyber-physical system and cloud computing [17].

Consider the example in Figure 1, where Figure 1(a) shows a virtual network request and Figure 1(b) shows a substrate network. The corresponding number near each vertex (or edge) is the CPU (or bandwidth) capacity/constraint. This virtual network request contains four virtual machines (VMs), linked by four links. If parallelization is not supported, this request would be rejected, as there is no substrate node that has more than 50 units of available CPU resources which the virtual node d requires. In PVNE, a feasible mapping scheme is shown in Figure 1(c). The master mapping is $\{a \longrightarrow B,\ b \longrightarrow H,\ c \longrightarrow C,\ d \longrightarrow D\}$, and the slave mapping is $\{a \longrightarrow \Phi,\ b \longrightarrow \Phi,\ c \longrightarrow \Phi,\ d \longrightarrow F\}$. The link mapping is $\{\{ab\} \longrightarrow \{BH\},\ \{Ac\} \longrightarrow \{BC\},\ \{bed\} \longrightarrow \{HD\},\ \{CD\} \longrightarrow \{CD\}\}$. In the mapping scheme of Figure 1(c), it can be seen that the virtual nodes a, b, and c are mapped to one substrate node, respectively. However, no physical node can satisfy the resource requirements of the virtual node d. In this case, a parallelized mapping scheme is required; that is, a virtual node is mapped to multiple substrate nodes. Finally, the virtual node d is mapped to the physical nodes D and F.

In this paper, we study parallelizable virtual network embedding (PVNE) and propose an efficient parallelizable virtual network embedding (EPVNE) algorithm. We summarize the main contributions here as follows:

(i) Based on optimization theory and resource integration technology, an optimized mathematical model for parallel virtual network embedding is proposed.

(ii) A new heuristic algorithm named EPVNE is proposed to reduce the cost of embedding the VN request and increase the VN request acceptance ratio. Firstly, the substrate nodes and virtual nodes need to be ordered by a metric proposed in this paper which is simple and can be calculated efficiently. Secondly, when mapping each virtual node n_v, our algorithm will search for a substrate node whose CPU value is just larger than n_v. Substrate nodes with more CPU resources will be retained for subsequent virtual network mapping requests.

(iii) Extensive simulations have been performed to evaluate the performance of our new algorithm. The results demonstrate that our algorithm performs better than the existing heuristic algorithms.

This paper is organized as follows: the related work is introduced in Section 2. In Section 3, we describe system model and problem formulation. We then present the proposed algorithm in Section 4. Section 5 introduces the experimental conditions and related parameter settings and focuses on the experimental results. Finally, Section 6 concludes the paper.

2. Related Work

Researchers have conducted in-depth research on virtual network embedding algorithms from different perspectives, and a series of research results have emerged. Several papers [3–5] summarize the virtual network mapping algorithm. Fischer A [5] summarized the research content of VNE,

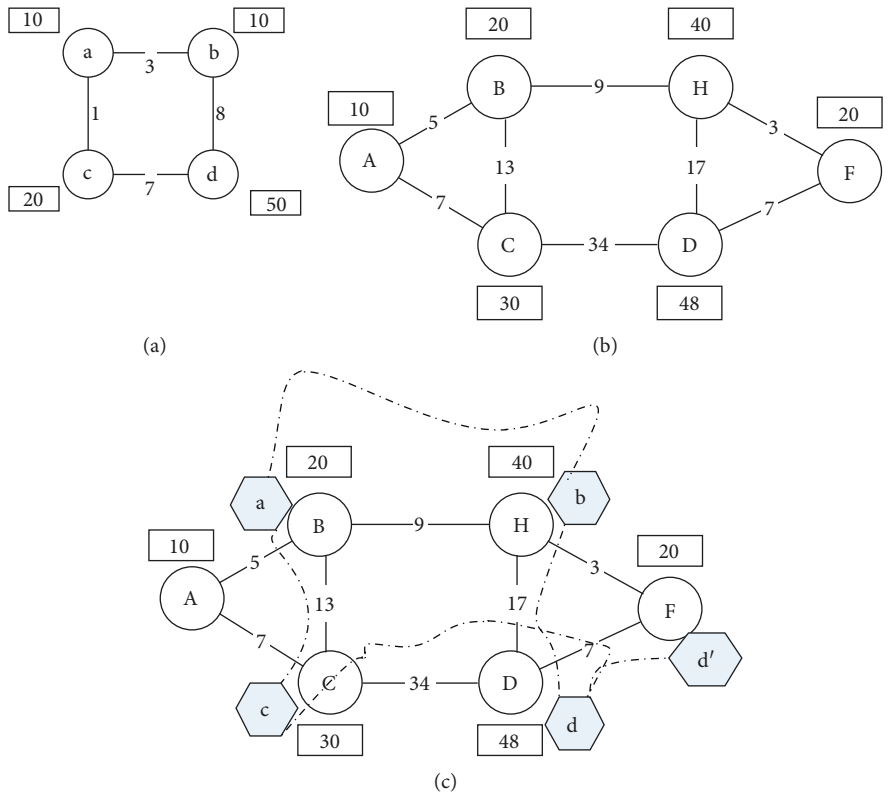

Figure 1: Parallelizable virtual network embedding. (a) Virtual network request. (b) Substrate network. (c) Substrate network with parallelization.

provided the classification scheme of the VNE algorithm, and discussed further research works. Haotong Cao [4] summarized the existing typical VNE exact solutions and proposed future research directions. Since the VNE problem is NP-hard in terms of computational complexity, many VNE algorithms proposed in the literature are heuristic. Haotong Cao investigated, analyzed, and summarized some representative heuristic solutions in the paper [3]. In general, the virtual network mapping algorithm can be classified as follows.

2.1. Consider Both Node and Link Resource Constraints vs. Ignore Node or Link Resource Constraints. Depending on whether the node and link resource constraints of the underlying network or virtual network are fully considered, the virtual network mapping algorithm is divided into a mapping algorithm that considers both node and link resource constraints and a mapping algorithm that ignores node or link resource constraints. For example, in the virtual network mapping process, the literature [11, 18] considers the resource constraints of nodes and links at the same time; while the literature [19, 20] ignores the resource constraints of nodes or links. The path splitting, path migration, and customized embedding algorithms can be used to obtain better mapping performance [11].

2.2. Static vs. Dynamic. Depending on the underlying network resource allocation method, the virtual network embedding algorithm can be divided into static embedding algorithms [11, 21] and dynamic embedding algorithms [19]. The static embedding algorithm refers to statically assigning fixed underlying network resources to the virtual network, and the dynamic mapping algorithm dynamically allocates the underlying network resources according to the virtual network's own resource requirements. The dynamic reconfiguration mapping algorithm refers to dynamically adjusting the mapping scheme without changing the virtual network resource requirements [19].

2.3. Online vs. Offline. According to the different processing methods for virtual network requests, the virtual network embedding algorithm can be divided into offline embedding algorithms [22, 23] and online embedding algorithms [11, 18, 21, 24]. Offline embedding means that all virtual network request information is known before the embedding algorithm is implemented while the online embedding algorithm does not make any assumptions about the arrival time, duration, and topology information of the virtual network request. The offline embedding algorithm is computationally intensive, and it is difficult to find the solution to the problem in polynomial time.

2.4. Centralized vs. Distributed. Depending on the calculation method of virtual network mapping, the virtual network mapping algorithm can be divided into a centralized mapping algorithm and a distributed mapping algorithm.

The centralized virtual network mapping algorithm allocates corresponding resources for virtual network requests according to the underlying network resource status by the central decision-making organization [11, 21, 24]. Centralized mapping algorithms can generally find the optimal solution for VNE problems in small-scale networks. However, this type of method has a common problem with a single point of failure. If the central node is attacked or fails, the entire map will fail. In addition, in large-scale networks, the scalability of the algorithm is a problem that needs to be solved. The distributed virtual network mapping algorithms generally do not have a computing center, and the virtual network mapping process is completed through the underlying nodes [23]. The advantage of the distributed mapping algorithm over centralized algorithms is better scalability. However, there are some disadvantages. The amount of calculation will be very large. There is also an increase in overhead. There is a tradeoff between feasibility and optimality.

2.5. One Stage vs. Two Stage. According to the mapping order of virtual nodes and virtual links, the virtual network mapping algorithm can be divided into one-stage mapping algorithm and two-stage mapping algorithm. The node mapping and link mapping of the one-stage mapping algorithm are completed in the same stage. In other words, the virtual link in the one-stage mapping algorithm maps when mapping virtual nodes [20, 25]. The node mapping phase of the two-stage mapping algorithm is separate from the link mapping phase. This type of algorithm generally maps all virtual nodes first and then performs virtual link mapping [11, 24].

2.6. Concise vs. Redundant. In the real substrate network, various failures often occur. In a network virtualization environment, failure of a single physical network entity will affect all virtual networks mapped to that physical network entity. In order to solve the problem, a very realistic idea is to provide redundancy or backup. The redundant virtual network mapping refers to the provision of redundant resources in the virtual network mapping, including node resources and link resources, to deal with node failures. In contrast, the concise virtual network mapping algorithm does not provide redundant resources. The concise solution uses only as few substrate resources as possible to meet the needs of the proposed virtual network without leaving additional resources for any failures. This also means that virtual networks cannot be guaranteed to recover from accidental failures, although more substrate resources can be reserved to embed more virtual networks.

The redundant VNE methods [26, 27] reserve additional resources for virtual networks to prevent certain substrate components from failing during operation. This type of approach can improve the reliability of mapping virtual networks. However, a tradeoff must be made between the reliability of the solution and its cost of embedding. The higher the degree of reliability, the more the substrate resources are consumed and the fewer the number of embedded virtual networks. In general, redundant VNE algorithms are more reliable and more popular than concise algorithms.

2.7. Heuristic Solutions vs. Exact Solutions vs. Metaheuristic Solutions. Recently, Cao et al. divided the virtual network mapping algorithm into exact solution [4] and heuristic solution [3]. The exact solution solves the virtual network mapping problem based on the optimization theory [28]. The VNE problem can generally be abstracted into an integer linear programming problem or a mixed integer programming problem. This problem can be solved using branch and bound, branch and cut, and branch and price. Of course, there are also some software tools available for solving this problem, such as GLPK, ALEVIN, CPLEX, and Matlab.

Virtual network mapping problems can be solved using the heuristic algorithm [11, 21, 24]. In the study of virtual network mapping problems, in order to obtain acceptable execution time, system performance can be appropriately reduced. That is, the heuristic algorithm obtains a suboptimal solution in an acceptable time. These heuristic algorithms include the greedy method, the k-shortest path method, the multicommodity solution, and the subgraph isomorphism detection method.

Some metaheuristic algorithms perform well in solving some problems, such as flowshop scheduling problems [29–32]. Therefore, some researchers use the metaheuristic algorithm [33–35] to solve the virtual network mapping algorithm. These researchers have made useful attempts and achieved some results.

The above is a description of the classification of virtual network mapping algorithms. It should be noted that the different classifications are independent. That is, any selected VNE algorithm can be, for example, static, centralized, and concise. Let us discuss the results of recent research. As the research progressed, people came up with some novel ideas and were not able to classify them according to the previous classification. Ni et al. [36] proposed a multidomain virtual network embedding algorithm based on particle swarm optimization (PSO). As Internet traffic has grown exponentially, network energy consumption has increased dramatically worldwide. Some researchers have considered energy consumption when studying virtual network mapping problems [37, 38]. Energy efficiency, concurrency, and topology awareness are all considered when designing virtual network mapping algorithms [39]. With the rapid development of artificial intelligence in the world, people consider using artificial intelligence to solve the problem of virtual network mapping. Learning actively and making online decisions based on previous experiences are used in paper [40]. Haeri and Trajkovic [8] formalized the virtual node mapping problem by using the Markov decision process (MDP) framework. The Monte Carlo tree search algorithm is used to design an action strategy (node mapping) for the proposed MDP. Yao et al. [41] used reinforcement learning to study virtual network mapping problems. They designed and implemented a strategy

network based on reinforcement learning to develop node mapping decisions. In order to cope with the dynamic resource requirements, a fitness-based dynamic virtual network embedding (DYVINE) algorithm is proposed with the goal to maximize the resource utilization by maximizing the acceptance rate [42]. A self-adaptive VNE algorithm is proposed in paper [43].

The paper mentioned above does not discuss the case where virtual nodes can be mapped to multiple physical nodes. The literature [12] first proposed to map a virtual node to multiple physical nodes. In this article, we will further study the problem and propose our solution.

3. System Model and Problem Formulation

In this section, we will first model the substrate network of InPs and VN of SPs and give the VN embedding problem description, followed by the definition of objectives.

Similar to the papers of other scholars, the two parameters of CPU and bandwidth are mainly studied in this paper. The subscript s indicates the substrate network, and the subscript v indicates the virtual network. In the following, we will model the substrate network and the virtual network as undirected weighted graphs, where the vertices represent nodes and the edges represent links. Each vertex is associated with CPU capacity/constraint, and each edge is associated with bandwidth capacity/constraint.

3.1. Network Model. We represent the substrate network as a weighted undirected graph $G_s = (N_s, L_s, A_s^n, A_s^l)$, where N_s represents the set of the substrate nodes and L_s is the set of the substrate links. The notation A_s^n represents the attribute of the substrate nodes, and the notation A_s^l represents the attribute of the physical links. The attributes of a node could be CPU processing power, storage capacity, and geographic location of the node. The attributes of a link could be bandwidth, delay, and delay jitter. In this paper, we only consider the available CPU processing power of the node and the available bandwidth properties of the link.

Similarly, the topology of the virtual network can also be represented by a weighted undirected graph $G_v = (N_v, L_v, A_v^n, A_v^l)$, where N_v is the set of virtual nodes and L_v is the set of virtual links. In addition, A_v^n and A_v^l represent the CPU requirement constraints on the virtual node and the bandwidth requirements on the virtual link, respectively. For the last two symbols in the substrate network and virtual network representation, A_s^n represents the amount of resources that the substrate node can provide, and A_v^n represents the resource demand of the virtual network node. Similarly, A_s^l represents the amount of resources that the substrate link can provide, and A_v^l represents the resource requirements of the virtual network link.

Each VN request may be represented by $\mathrm{VNR}^{(i)}(G_v, t_a, t_d)$, where the variables t_a and t_d denote the arrival time of the VN request and the duration of the VN staying in the substrate network, respectively. When the ith VN request arrives, the infrastructure provider should allocate the substrate network resources to the service provider while satisfying the resource requirements of the virtual nodes and links. If there are no enough substrate resources available to satisfy the requirements of the virtual network request, the VN request should be rejected or postponed. When the virtual network has completed its task and no longer uses the substrate network resources, the allocated substrate resources are released.

Figure 1 shows an example of these notations. In this article, letters i and j represent substrate nodes and u and v denote virtual nodes. The CPU value of node m is denoted as $\mathrm{CPU}(m)$, and the bandwidth value of link l is denoted as $\mathrm{BW}(l)$. Figure 1(b) shows a substrate network. There are 6 nodes and 8 links in the substrate network. $N_s = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$ and $L_s = \{\mathrm{AB}, \mathrm{BH}, \mathrm{HF}, \mathrm{FD}, \mathrm{CD}, \mathrm{AC}, \mathrm{BC}, \mathrm{DH}\}$. The number inside the rectangle indicates the available CPU resources of the node it is close to, for example, $\mathrm{CPU}(\mathrm{A}) = 10$ and $\mathrm{CPU}(\mathrm{B}) = 20$. The number on the link indicates the bandwidth of the link, for example, $\mathrm{BW}(\mathrm{AB}) = 5$ and $\mathrm{BW}(\mathrm{AC}) = 7$. A virtual network request is shown in Figure 1(a). There are four virtual nodes and four virtual links in the virtual network request. $N_v = \{a, b, c, d\}$, and $L_v = \{ab, ac, bd, cd\}$. The number in the rectangular box indicates the resource requirements of the virtual network node, and the number on the link indicates the link bandwidth requirement. As mentioned earlier, if parallelization is not supported, the virtual network request in Figure 1(a) will be rejected or delayed. If parallelization is supported, Figure 1(c) shows one of the mapping schemes.

3.2. Parallelizable Virtual Network Embedding. Yu Liang and Sheng Zhang divided the parallelizable virtual network embedding into three components: master mapping, salve mapping, and link mapping. The master mapping M_{ms} maps a virtual node n_v to a substrate node, denoted by $M_{ms}(n_v)$, and the slave mapping M_{sl} maps a virtual node n_v to a subset of the neighbors of $M_{ms}(n_v)$, denoted by $M_{sl}(n_v)$.

As shown in Figure 1, in a parallel virtual network mapping, a virtual node can be mapped to multiple physical networks, which can improve the mapping successful ratio. However, please note that parallelization may result in additional CPU and bandwidth costs. To reflect these additional computation and bandwidth consumptions caused by parallelization, Liang and Zhang [12] proposed the penalty factor pf and the additional bandwidth consumption Z. In the aforementioned example in Figure 1, if $\mathrm{pf} = 1.2$, then 60 ($50 * 1.2 = 60$) units of CPU should be allocated to d in D and F. For additional bandwidth consumption, an additional Z unit which is a constant should be allocated between each pair of master and slave nodes. Mapping a virtual node to too many substrate nodes cannot significantly improve computational performance. Liu Yang and Zhang Sheng proposed a parameter named $speedup$ which means that a virtual node can only map to $speedup$ substrate nodes at most.

3.3. Problem Formulation and Objectives. Based on the introduction of Section 3.2, mapping the virtual node to multiple substrate nodes will lead to extra CPU and bandwidth consumption. The main idea of this paper is that

the virtual node should not be mapped to multiple substrate nodes as much as possible. Please note that in the following formulas (1), (3), and (4), $\sum_{i\in N_s}\lceil x_i^u/\mathrm{CPU}(u)\rceil = 1$ represents that the virtual node u is mapped to a unique substrate node, and $\sum_{i\in N_s}\lceil x_i^u/\mathrm{CPU}(u)\rceil > 1$ indicates that the virtual node u is mapped to multiple substrate nodes.

Variables:

(i) f_{ij}^{uv}: a binary variable, where it is 1 if virtual link l_{uv} is routed on physical link l_{ij} and 0 otherwise.

(ii) x_i^u: a real variable, such that $x_i^u > 0$ if virtual node u is mapped to the substrate node i, and x_i^u unit CPU resource of the substrate node i is allocated to virtual node u; $x_i^u = 0$ if virtual node u is not mapped to the substrate node i.

Objective:

$$\begin{aligned}\mathrm{Min}: &\sum_{l_{uv}\in L_v}\sum_{l_{ij}\in L_s} f_{ij}^{uv}\times \mathrm{BW}(l_{uv})\\ &+\sum_{u\in N_v}\left(\left\lceil\frac{\sum_{i\in N_s}\lceil(x_i^u)/(\mathrm{CPU}(u))\rceil-1}{speedup}\right\rceil\times(\mathrm{pf}-1)+1\right)\\ &\times\mathrm{CPU}(u)+\sum_{u\in N_v}\left(Z\times\left(\sum_{i\in N_s}\left\lceil\frac{x_i^u}{\mathrm{CPU}(u)}\right\rceil-1\right)\right),\end{aligned} \tag{1}$$

where $\lceil x\rceil$ is the least integer that is larger than x.

Constraints:

(i) Capacity constraints:

$$\forall l_{ij}\in L_s, \forall l_{uv}\in L_v: f_{ij}^{uv}\times \mathrm{BW}(l_{uv})\le \mathrm{BW}\left(l_{ij}\right), \tag{2}$$

$$\forall i\in N_s, \sum_{u\in N_s}\left\lceil\frac{x_i^u}{\mathrm{CPU}(u)}\right\rceil\le 1, \tag{3}$$

$$\forall u\in N_v, \begin{cases}\sum_{i\in N_s} x_i^u = \mathrm{CPU}(u), & \text{if } \sum_{i\in N_s}\left\lceil\frac{x_i^u}{\mathrm{CPU}(u)}\right\rceil = 1,\\ \sum_{i\in N_s} x_i^u = \mathrm{pf}\times\mathrm{CPU}(u), & \text{if } \sum_{i\in N_s}\left\lceil\frac{x_i^u}{\mathrm{CPU}(u)}\right\rceil > 1.\end{cases} \tag{4}$$

(ii) *Speedup* constraints:

$$\forall u\in N_v, \sum_{i\in N_s}\left\lceil\frac{x_i^u}{\mathrm{CPU}(u)}\right\rceil\le speedup. \tag{5}$$

(iii) Domain constraints:

$$\forall i,j\in N_s, u,v\in N_v, f_{ij}^{uv}\in\{0,1\}, \tag{6}$$

$$\forall u\in N_v, i\in N_s, \mathrm{CPU}(u)\ge x_i^u\ge 0. \tag{7}$$

Remarks:

(i) The objective function tries to minimize the cost of embedding the VN request. The objective function consists of three parts. The first part is the cost of mapping the virtual links on the substrate links. The second part is the cost when the virtual node is mapping. The third is the cost of the links between the slave nodes and the master node.

(ii) Constraint set (2) refers to virtual link constraints. The substrate link (i, j) must meet the bandwidth requirement for the virtual link (u, v). In this paper, path splitting is unsupported, and supporting path splitting will be our future work. Constraint set (3) refers to that no more than one virtual node is placed on a substrate node. Constraint set (4) ensures that the CPU assigned to the virtual node u by the substrate nodes is equal to its requirement.

(iii) Constraint set (5) enforces the number of substrate nodes which the virtual node u is mapped to is less than or equal to *speedup*.

(iv) Finally, constrain sets (6) and (7) denote the binary and real-domain constrains on the variables f_{ij}^{uv} and $x_i^u x_i^u$, respectively.

4. Our Proposed EPNVE Scheme

The proposed heuristic EPVNE employs a greedy approach to deal with node mapping, and the link mapping utilizes Dijkstra's algorithm to find the shortest path that meets the resource demands. EPVNE is shown in Algorithm 1; it consists of three phases.

In the initialization phase (lines (1)–(3) in Algorithm 1), all virtual nodes are sorted and placed in a queue in the decreasing order of $\mathrm{CPU}(n_v)$. Similarly, all substrate nodes are sorted in the decreasing order of $\mathrm{ACPU}(n_v)$ which can be derived from the following equation:

$$\forall n_s\in N_s, \mathrm{ACPU}(n_s) = \mathrm{CPU}(n_s) + \sum_{m\in V_{\mathrm{nei}}^{unused}(n_s)}\mathrm{CPU}(m), \tag{8}$$

where $V_{\mathrm{nei}}(n_s)$ denotes the set of direct neighbors of node n_s and $V_{\mathrm{nei}}^{unused}(n_s)$ denotes the set of direct neighbors of node n_s which has not been used. There is a similar calculation formula (2) in paper [12] to calculate $\mathrm{RC}_{\mathrm{nei}}(n_s)$ whose function is similar to $\mathrm{ACPU}(n_s)$ in this paper. The calculation formula (2) in paper [12] is complicated. The value it calculates is of little significance for a mapping guide. The result of its calculation is of little significance for one mapping. Therefore, a simpler calculation method is used in our paper.

In the node mapping phase (lines (4)–(9) in Algorithm 1), EPVNE maps each virtual node to one substrate node (using Mapping-one-to-one shown in Algorithm 2) as much as possible. If no substrate node CPU resource can meet the resource requirements of the virtual node, the

```
(1) //initialization phase
(2) Q ⟵ Sorted virtual nodes in nonincreasing order according to their CPU requirement;
(3) P ⟵ Sorted substrate nodes in nonincreasing order according to their ACPU;
(4) //node mapping phase
(5) Mapping the virtual nodes in queue Q in turn,
      i ⟵ 0 to Q·length-1;
(6) If CPU(Q·[i]) ≤ CPU(P[0]) then
(7)     Mapping Q[i] to one substrate nodes using Algorithm 2;
(8) Else
(9)     Mapping Q[i] to multiple substrate nodes using Algorithm 3;
(10) //link mapping phase
(11) Dijkstra's algorithm is used to find a substrate path for each virtual link.
```

Algorithm 1: EPVNE.

virtual node is mapped to multiple substrate nodes (using Mapping-one-to-multi described in Algorithm 3).

In the link mapping phase (lines (10)-(11) in Algorithm 1), Dijkstra's algorithm is used to find a substrate path for each virtual link.

Mapping-one-to-one shown in Algorithm 2 will search for a substrate node whose CPU value is just larger than $Q[i]$ for mapping $Q[i]$. Substrate nodes with more CPU resources will be retained for subsequent virtual network mapping requests.

Mapping-one-to-multi described in Algorithm 3 finds up to *speedup-1* nodes from the one-hop neighbors of $P[0]$ to map $Q[i]$. When searching, it proceeds in the ascending order, first looking for the smallest one-hop neighbor, then the second smallest one, and so on. $|Neigh|$ in line (5) in Algorithm 3 represent the cardinality of *Neigh*.

In the following, we analyze the computational complexity of Algorithms 1–3. The complexities of Algorithms 2 and 3 are $O(|N_s|)$ and $O(|N_s| * \text{speed})$, respectively. The complexity of Algorithm 3 is larger than that of Algorithm 2. $|N_s|$ denotes the number of physical nodes, and $|N_v|$ is the number of virtual nodes.

In the initialization phase of Algorithm 1, the complexity of virtual nodes sorting is $O(|N_v|^2)$ and the complexity of substrate nodes sorting is $O(|N_s|^2)$. In the node mapping phase of Algorithm 1, the complexity is $|N_v|$ multiplied by the complexity of Algorithm 3, that is, $|N_v| * O(|N_s| * \text{speed}) = O(|N_v| * |N_s| * \text{speed})$. The complexity of the Dijkstra algorithm is $O(|N_s|^2)$. In the link mapping phase of Algorithm 1, the complex is $|L_v| * O(|N_s|^2)$, and $|L_v|$ is the number of virtual links. Finally, the complexity of Algorithm 1 is the sum of the complexity of the above three phases, that is $O(|N_v|^2) + O(|N_s|^2) + O(|N_v| * |N_s| * \text{speed}) + |L_v| * O(|N_s|^2)$. The complexity of *EPVNE* is acceptable.

5. Performance Analysis

In this section, we first introduce our simulation setup and then present performance comparison results. The substrate network topology is configured to have 20 nodes, and each pair of nodes is connected with probability 0.4. The CPU and bandwidth resources of the substrate nodes and links are real numbers uniformly distributed between 50 and 100. The number of virtual nodes in each virtual network follows a uniform distribution between 2 and 10. Each pair of virtual nodes is connected with probability 0.5. The bandwidth requirements at virtual links are generated randomly from the range [50, 70]. The CPU requirements of the virtual nodes are also randomly generated, and the ranges are [50, 90], [70, 110], [90, 130], [110, 150], [130, 170], [150, 190], and [170, 210]. The three parameters described in Section 3.2 are set as follows: $\text{pf} = 1.2$, $Z = 2.0$, and $\textit{speedup} = 5$. In this section, we concentrate on comparing *EPVNE*, *ProactiveP* [12], *LazyP* [12], and *RW-MaxMatch* [21]. *ProactiveP* always maps a virtual machine to *speedup* nodes, and *LazyP* does not map a virtual machine to multiple nodes unless there are not enough substrate resources. *RW-MaxMatch* is a virtual node-first mapping algorithm based on a novel topology-aware node ranking measure. Since the *RW-MaxMatch* algorithm is widely quoted, we also compare our algorithm with it. 2500 experiments were performed for each scene, and the average was taken as the result.

The following metrics are used for performance comparison: (i) Acceptance ratio, which is the ratio of the number of accepted virtual network requests to all requests; (ii) CPU ratio, which is the ratio of the amount of occupied CPU resources in the substrate network to overall CPU resources in the virtual network; and (iii) Z ratio, which is the ratio of the additional bandwidth consumption caused by parallelization to the constant Z. The latter two metrics are indicators that measure parallelism. *RW-MaxMatch* does not support parallelization. In fact, the CPU ratio of *RW-MaxMatch* is equal to 100%, and the Z ratio of *RW-MaxMatch* is zero.

Figure 2 shows the comparison of the acceptance ratio. In general, *EPVNE* achieves a high acceptance ratio than *LazyP*, *ProactiveP*, and *RW-MaxMatch*. As the CPU requirements of the virtual nodes increase, the acceptance rate shows a downward trend. The reason is very obvious. The CPU requirements of the virtual network nodes are small, the substrate nodes' resources are relatively abundant, and the acceptance rate is higher. Because the *RW-MaxMatch* algorithm does not support parallelization, that is, it cannot map a virtual node to multiple physical nodes, its performance is the worst. The *ProactiveP* algorithm always maps a virtual node to multiple physical nodes, resulting in more

```
(1) For j = 0 to P·length-1 do
(2)   If CPU(P·[j]) ≥ CPU(Q[i]) and CPU(P·[j + 1]) < CPU(Q[i]) then
(3)     Break
(4)   End if
(5) End for
(6) Map Q[i] to P[j]
(7) Remove P[j] from P
```

Algorithm 2: Mapping-one-to-one.

```
(1) Neigh ⟵ {P[0]}
(2) Sum ⟵ CPU(P[0])
(3) For j = P·length-1 to 1 do
(4)   If P[j] ∈ V_nei^unused(P[0]) do
(5)     If |Neigh| == speedup do
(6)       Remove the node with the lowest CPU value from Neigh
(7)       Sum = Sum-CPU (the node with the lowest CPU value)
(8)     End if
(9)     add P[j] to Neigh
(10)    Sum = Sum + CPU(P[j])
(11)  End if
(12)  If Sum ≥ Q[i] do
(13)    Break;
(14)  End if
(15) End for
(16) Map Q[i] to the elements in Neigh
(17) Remove the elements in Neigh from P
```

Algorithm 3: Mapping-one-to-multi.

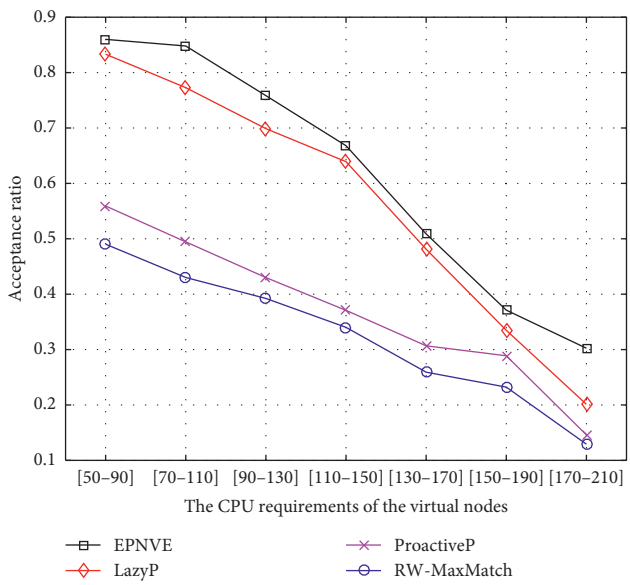

Figure 2: Acceptance ratio.

overhead. Therefore, its performance is second to last. *LazyP* and *EPVNE* map a virtual node to multiple nodes only when one physical node cannot meet the virtual node requirements, so the two algorithms perform better. *LazyP* uses a more complicated calculation formula which does not truly reflect the situation of the remaining physical resources, resulting in the mapping failure. In the experiments, we found that *ProactiveP* maps each virtual node to multiple substrate nodes. As a result, in the late mapping, too many substrate nodes are used and the available substrate nodes are lacking, resulting in mapping failure. This is the reason why the *ProactiveP* acceptance ratio curve in Figure 2 is significantly lower than the other two.

The comparison of the CPU ratio is depicted in Figure 3. Because *ProactiveP* always maps a virtual node to multiple physical nodes, its CPU ratio is highest and is close to 1.2. In addition, because the mapping success rate of *EPVNE* is relatively high, that is, *EPVNE* can successfully map the case which *LazyP* cannot map, the CPU ratio of *EPVNE* is also relatively high. When the CPU requirement virtual node is relatively small, the node can be mapped to a single substrate node without causing additional CPU consumption. With the increase of CPU requirement, a single substrate node cannot meet the CPU resource requirements of the virtual nodes, and multiple substrate nodes must complete the mapping together. In the end, the CPU requirement increases and the CPU ratio tends to be 1.2, which is the value of pf.

Figure 4 which shows the comparison of the Z ratio is similar to the situation in Figure 3. In fact, the Z ratio represents the average number of additional links or number of slave nodes per virtual network. The curves of *ProactiveP*

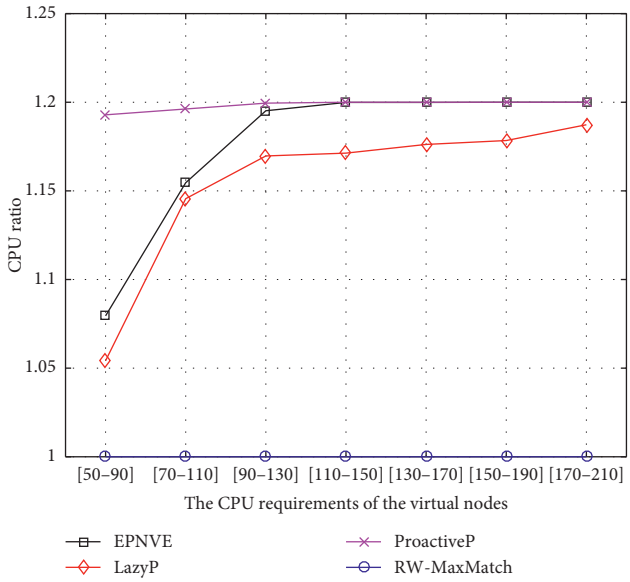

Figure 3: CPU ratio.

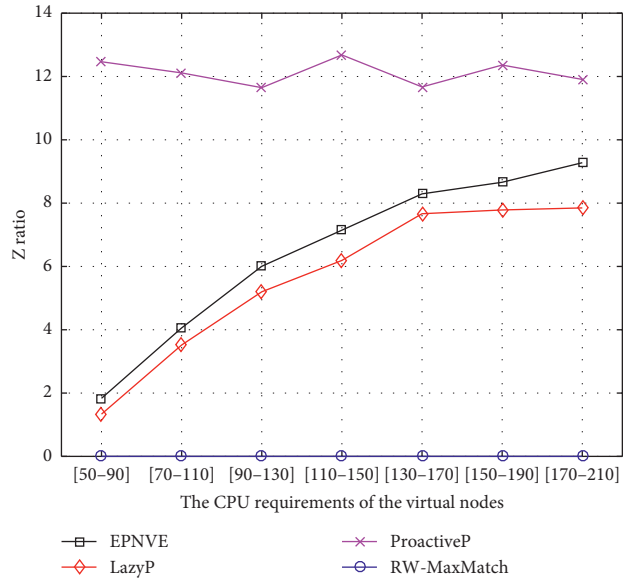

Figure 4: Z ratio.

in Figures 3 and 4 are smoother and higher than the other two because the algorithm maps each virtual node to multiple substrate nodes regardless of the actual situation of network resources. The reason why the two curves of *EPVNE* in Figures 3 and 4 are higher than that of *LazyP* is that the acceptance rate of *EPVNE* is high. *EPVNE* successfully maps some of the virtual network requests that *LazyP* failed to map, and these virtual network requests require more extra links and additional CPU resources.

6. Conclusion

Network virtualization technology allows multiple heterogeneous virtual networks to be built on top of a shared underlying physical network. Network virtualization technology enables network operators to deploy new network architectures or protocols without affecting the existing Internet, providing a viable path for evolution from the current network to the future network. As one of the main challenges facing network virtualization, the virtual network mapping problem is NP-hard. It has become a research hotspot in the field of network virtualization.

In this paper, we studied parallelizable virtual network embedding. Firstly, the formulation of *PVNE* is presented. Secondly, an efficient parallelizable virtual network embedding (*EPVNE*) algorithm is proposed. Finally, extensive simulations are done to evaluate the performance of *EPVNE*. The results demonstrate that our algorithm performs better than the existing heuristic algorithms. Our future work is to take path splitting into account.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "VMPlanner: optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks*, vol. 57, no. 1, pp. 179–196, 2013.

[2] Y. Zhou, Y. Zhang, H. Liu, N. Xiong, and A. V. Vasilakos, "A bare-metal and asymmetric partitioning approach to client virtualization," *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 40–53, 2014.

[3] H. Cao, H. Hu, Z. Qu, and L. Yang, "Heuristic solutions of virtual network embedding: a survey," *China Communications*, vol. 15, no. 3, pp. 186–219, 2018.

[4] H. Cao, L. Yang, Z. Liu, and M. Wu, "Exact solutions of VNE: a survey," *China Communications*, vol. 13, no. 6, pp. 48–62, 2016.

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

[6] X. Liu, Z. Zhang, X. Li, and S. Su, "Optimal virtual network embedding based on artificial bee colony," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, p. 273, 2016.

[7] P. Zhang, H. Yao, C. Fang, and Y. Liu, "Multi-objective enhanced particle swarm optimization in virtual network embedding," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, p. 167, 2016.

[8] S. Haeri and L. Trajkovic, "Virtual network embedding via Monte Carlo tree search," *IEEE Transactions on Cybernetics*, vol. 48, no. 2, pp. 510–521, 2018.

[9] H. Cao, Y. Zhu, G. Zheng, and L. Yang, "A novel optimal mapping algorithm with less computational complexity for virtual network embedding," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 356–371, 2018.

[10] X. Chen, C. Li, and Y. Jiang, "Optimization model and algorithm for energy efficient virtual node embedding," *IEEE Communications Letters*, vol. 19, no. 8, pp. 1327–1330, 2015.

[11] M. L. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 19–29, 2008.

[12] Y. Liang and S. Zhang, "Embedding parallelizable virtual networks," *Computer Communications*, vol. 102, no. 4, pp. 47–57, 2017.

[13] K. Z. Gao, P. N. Suganthan, Q. K. Pan, M. F. Tasgetiren, and A. Sadollah, "Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion," *Knowledge-Based Systems*, vol. 109, no. 6, pp. 1–16, 2016.

[14] Q. Qi, J. Wang, Z. Ma et al., "Knowledge-driven service offloading decision for vehicular edge computing: a deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4192–4203, 2019.

[15] H.-Y. Sang, J.-Q. Duan, and J.-Q. Li, "An effective invasive weed optimization algorithm for scheduling semiconductor final testing problem," *Swarm and Evolutionary Computation*, vol. 38, no. 1, pp. 42–53, 2018.

[16] J.-Q. Li and Q.-K. Pan, "Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm," *Information Sciences*, vol. 316, pp. 487–502, 2015.

[17] J. Xia, G. Chen, and W. Sun, "Extended dissipative analysis of generalized Markovian switching neural networks with two delay components," *Neurocomputing*, vol. 260, pp. 275–283, 2017.

[18] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proceedings of the 28th Conference on Computer Communications*, pp. 783–791, IEEE, Rio de Janeiro, Brazil, April 2009.

[19] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proceedings of the 25th IEEE International Conference on Computer Communications*, Barcelona, Spain, April 2006.

[20] J. Shamsi and M. Brockmeyer, "QoSMap: QoS aware mapping of virtual networks for resiliency and efficiency," in *Proceedings of the IEEE Globecom Workshops 2007*, Washington, DC, USA, November 2007.

[21] X. Cheng, S. Su, Z. Zhang et al., "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 39–47, 2011.

[22] J. Fan and M. H. Ammar, "Dynamic topology configuration in service overlay networks: a study of reconfiguration policies," in *Proceedings of the 25th IEEE International Conference on Computer Communications*, pp. 1–12, Barcelona, Spain, April 2006.

[23] I. Houidi, W. Louati, and D. Zeghlache, "A distributed virtual network mapping algorithm," in *Proceedings of the IEEE International Conference on Communications*, pp. 5634–5640, Beijing, China, April 2008.

[24] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, 2012.

[25] H. Cao, Y. Guo, Y. Hu, S. Wu, H. Zhu, and L. Yang, "Location aware and node ranking value-assisted embedding algorithm for one-stage embedding in multiple distributed virtual network embedding," *IEEE Access*, vol. 6, pp. 78425–78436, 2018.

[26] J. Ai, H. Chen, Z. Guo, and G. Cheng, "Defending against link failure in virtual network embedding using a hybrid scheme," *China Communications*, vol. 16, no. 1, pp. 129–138, 2019.

[27] X. Zheng, J. Tian, X. Xiao, X. Cui, and X. Yu, "A heuristic survivable virtual network mapping algorithm," *Soft Computing*, vol. 23, no. 5, pp. 1453–1463, 2019.

[28] L. F. S. Moura, L. P. Gaspary, and L. S. Buriol, "A branch-and-price algorithm for the single-path virtual network embedding problem," *Networks*, vol. 71, no. 3, pp. 188–208, 2017.

[29] Y.-Y. Han, J. J. Liang, Q.-K. Pan, J.-Q. Li, H.-Y. Sang, and N. N. Cao, "Effective hybrid discrete artificial bee colony algorithms for the total flow time minimization in the blocking flow shop problem," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 1–4, pp. 397–414, 2013.

[30] H.-Y. Sang, Q.-K. Pan, P.-Y. Duan, and J.-Q. Li, "An effective discrete invasive weed optimization algorithm for lot-streaming flow shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 29, no. 6, pp. 1337–1349, 2018.

[31] J. Li, Q. Pan, and S. Xie, "An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems," *Applied Mathematics and Computation*, vol. 218, no. 18, pp. 9353–9371, 2012.

[32] J.-Q. Li, J.-D. Wang, Q.-K. Pan et al., "A hybrid artificial bee colony for optimizing a reverse logistics network system," *Soft Computing*, vol. 21, no. 20, pp. 6001–6018, 2017.

[33] I. Pathak, A. Tripathi, and D. P. Vidyarthi, "A model for virtual network embedding using artificial bee colony," *International Journal of Communication Systems*, vol. 31, no. 10, p. e3573, 2018.

[34] X. Cheng, S. Su, Z. Zhang et al., "Virtual network embedding through topology awareness and optimization," *Computer Networks*, vol. 56, no. 6, pp. 1797–1813, 2012.

[35] P. Zhang, H. Yao, M. Li, and Y. Liu, "Virtual network embedding based on modified genetic algorithm," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 481–492, 2019.

[36] Y. Ni, G. Huang, S. Wu, C. Li, P. Zhang, and H. Yao, "A PSO based multi-domain virtual network embedding approach," *China Communications*, vol. 16, no. 4, pp. 105–119, 2019.

[37] M. Zhu, Q. Sun, S. Zhang, P. Gao, B. Chen, and J. Gu, "Energy-aware virtual optical network embedding in sliceable-transponder-enabled elastic optical networks," *IEEE Access*, vol. 7, pp. 41897–41912, 2019.

[38] P. Zhang, "Incorporating energy and load balance into virtual network embedding process," *Computer Communications*, vol. 129, pp. 80–88, 2018.

[39] A. Jahani, L. M. Khanli, M. T. Hagh, and M. A. Badamchizadeh, "EE-CTA: energy efficient, concurrent and topology-aware virtual network embedding as a multi-objective optimization problem," *Computer Standards & Interfaces*, vol. 66, Article ID 103351, 2019.

[40] S. Wang, J. Bi, J. Wu, A. V. Vasilakos, and Q. Fan, "VNE-TD: a virtual network embedding algorithm based on temporal-difference learning," *Computer Networks*, vol. 161, pp. 251–263, 2019.

[41] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, 2018.

[42] C. K. Dehury and P. K. Sahoo, "DYVINE: fitness-based dynamic virtual network embedding in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1029–1045, 2019.

[43] Z. Li, Z. Lu, S. Deng, and X. Gao, "A self-adaptive virtual network embedding algorithm based on software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 362–373, 2019.