Joint Correlation-aware VNF Selection and Placement in Cloud Data Center Networks

Biyi Li, Bo Cheng, Meng Wang, Xuan Liu, Yi Yue, and Junliang Chen
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications, Beijing, China
Email: lbyi0402@bupt.edu.cn chengbo@bupt.edu.cn

***Abstract*—Network Function Virtualization (NFV) brings great flexibility and scalability to the deployment of network services by decoupling network functions from dedicated devices, which has attracted more attention from both academia and industry. Network services in NFV are deployed in the form of Service Function Chain (SFC), which consists of multiple ordered Virtual Network Functions (VNFs). However, how to effectively place VNFs remains a problem to be solved. In this paper, we investigate joint correlation-aware VNF selection and placement problem. We first formulate the problem as an Integer Linear Programming (ILP) problem and propose a method based on self-learning matrix to partition VNF correlation. Then, we design a Joint Correlation-aware VNF Placement (JCVP) algorithm based on Dynamic Programming to transform the problem into several VNF mapping subproblems. Extensive simulation results show that compared with the previous algorithms our approach has better performance in link occupancy, SFC acceptance, and VNF utilization rate.**

***Index Terms*—Service Function Chain, Virtual Network Function, Placement, Correlation-aware, Dynamic Programming**

I. Introduction

In order to meet the increasing demand of network service, operators provide massive network functions (such as firewall, load balance or intrusion detection) in Data Center Networks (DCN) to manage traffic distribution and improve network security and performance in the traditional network environment. However, most of the network functions are heavily dependent on dedicated hardware. While providing a high quality of service, these devices also need to follow strict protocols, resulting in complex management and maintenance issues [1].

Network Function Virtualization (NFV), as a novel concept, initiated by European Telecommunication Standards Institute (ETSI) [2], addresses these problems by implementing network functions on commodity hardware in the form of virtual machine or containers through virtualization and cloud technologies. Network functions deployed as software applications are called Virtual Network Functions (VNF), and multiple VNFs can be orderly connected and chained to constitute a Service Function Chain (SFC) [4], which realize complex and diverse business services.

In the past, developing a new service chain took a great deal of time and cost. Each network function on chain requires dedicated devices and these devices are reconfigured according to a proprietary protocol. Software-based VNF means operators can get rid of the dependence on special devices and the complexity of service development. In addition, building network functions with virtual machines can improve the utilization of network resources significantly and decrease the Capital Expenditure (CAPEX) and Operating Expense (OPEX) of operators [3]. Therefore, the research on VNF has developed rapidly and attracted more attention from both academia and industry.

TABLE I
VNF Classification Examples

Function Types	Function Examples
Packet inspection	Firewall, DDoS, NIDS, Gateway, VPN
Traffic optimization	Traffic shapping, TCP optimization, DPI Monitor, Load Balance
Protocol proxies	NAT, HTTP proxy, TCP proxy, DNS cache

However, there are still many issues in NFV to be solved. One of the key issues is the optimal VNF selection and placement. In large-scale DCNs, VNFs with the same type are often deployed at various locations, resulting in a serious waste of resources. From the perspective of operators, this is no doubt undesirable. Furthermore, unreasonable VNFs placement schemes will not only increase the difficulty of traffic guidance and the end-to-end delay but also violate Quality of Service (QoS) guarantees for users.

In general, when new SFC requirements arrive, the link occupancy can be decreased efficiently if we launch all VNFs on the same server or rack. However, a single server is limited to resource and cannot accommodate too many VNFs. According to the investigation [5], we find that the connection of VNFs in SFC is correlated. Hence, it is necessary to consider joint correlation-aware VNFs to support optimal mapping of SFCs. To facilitate understanding the correlation between VNFs, we list several examples of VNF by service type as shown in Table I. Intuitively, the correlation between VNFs of the same service type is significantly higher than the correlation of different service types. For instance, in the process of traffic detection, traffic usually needs to pass multiple VNFs such as VPN, Firewall and DDoS in an orderly manner. In addition, traffic shaping, TCP optimization and Load Balancing functions usually work together in the process of traffic optimization. Therefore, placing these correlated

978-1-7281-2583-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPADS47876.2019.00034

VNFs closely can make more efficient use of network resources.

In this paper, we study the joint VNF selection and placement problem. Our objective is to minimize the link occupancy in VNF placement problem while ensuring an effective SFC acceptance, a high VNF utilization rate and a polynomial execution time. To this end, the main contributions of this paper can be summarized as follows:

- We formulate the Joint Correlation-aware VNF Selection and Placement problem as an ILP problem and propose a method based on self-learning matrix to partition VNF correlation.
- We design a heuristic algorithm named JCVP (Joint Correlation-aware VNF Placement) to divide the problem into VNF mapping subproblems having a recurrence relationship.
- We conduct extensive simulations in DCNs compared with the previous algorithms. The results show that our approach has better performance in resource utilization, SFC acceptance, and VNF utilization rate.

The paper is organized as follows. In Section II, we review the previous work related to VNF placement. The Joint VNF Selection and Placement problem is formulated in Section III. In Section IV, we propose the method to define VNF correlation and describe the heuristic algorithm. Section V reports the results of simulation. At last, we summarize our paper in Section VI.

II. Related Work

In this paper, we focus on the placement of VNF. Currently, there have been some of the literature on solving the NP-Hard VNF Selection and Placement problem by proposing various exact and heuristic algorithms. We review the previous work and highlight the optimization brought by our algorithm.

Many existing works [6] - [10] studied the joint placement problem of VNF and formulated it as an Integer Line Programming (ILP) model. In [6], authors formulated the problem as an ILP and implemented using CPLEX in minor scale networks. Also, they proposed a Multi-Stage heuristic (MSH) to solve the problem. The authors of [10] proposed a Greedy algorithm and a simulated annealing algorithm to study how to minimize two different objectives separately (the end-end delay and bandwidth consumption). A heuristic algorithm was designed to find a solution in a shorter time but simplifies the problem with only one type of VNF in [7]. However, the work overlooked various non-functional constraints, such as the dependencies between VNFs. Most of the aforementioned methods are efficient in finding solutions but not consider the correlation between VNFs thus they are not suitable for large-scale DCNs.

Authors in [11] proposed an exact mathematical model using decomposition methods to solve the Service Chain provisioning problem. Related work [13] proposed a min-K-cut method for clustering the VNFs. Their goal was to put each VNF of the same cluster on a pod to minimum communication overhead but they did not offer an effective mapping algorithm for VNF from different partitions. In our system model, the link constraints from no-correlation VNFs are naturally embedded.

In our study, we conduct a more detailed division of the correlation-aware VNF using self-learning matrix method. We emphasize that the type and order of VNFs in SFC are relevant and rational use of it can help operators select and map VNFs efficiently.

TABLE II
Notations Used In This Paper

Notation	Definition
$G(V, L)$	the optical network, where V denotes the set of nodes and L denotes the set of links
F	the set of VNFs, with function $f \in F$
M	the set of VNF instances, with $m \in M$
R	the set of SFC requests, with request $r \in R$
rf_i	the i^{th} function in SFC request r
C_v^{cpu}, C_v^{mem}	the computing capacity (in terms of CPU cores and memory) of physical node v
C_m^{cpu}, C_m^{mem}	the computing capacity (in terms of CPU cores and memory) of VNF instance m
B_l	the bandwidth of link l
b_r	the required bandwidth of SFC request r
$c_{rf_i}^{IPS}, c_{rf_i}^{mem}$	the resource required to create rf_i (IPS indicates the number of instructions processed per second and mem indicates the consumed memory of VNF)
$x_m^{rf_i}$	a binary variable indicating whether function f is used in VNF instance m
x_v^m	a binary variable indicating whether m is created on node v
y_r^l	a binary variable indicating whether SFC request r through link l, where $l \in L$
α	a parameter denoting the relationship between IPS and CPU cores
D_{n*n}	the VNF correlation matrix, in which n denote the number of VNF type

III. Problem Formulation

In this section, we formulate the VNF placement problem as an ILP problem and describe our system model. All the notations used in this paper are listed in Table II.

A. System Model

We consider a DCN that is represented by an undirected graph G=(V, L), where V denotes the set of nodes and L denotes the set of links. Let $C_v^{cpu}, C_v^{mem}, C_m^{cpu}, C_m^{mem}$ denote the computing capacity (in terms of CPU cores and memory) of physical node v and VNF instance m respectively. Also, we assume that each link $l \in L$ is associated with a maximum available bandwidth B_l.

Let R denotes the set of SFC requests. Then we define rf_i as the i^{th} function in request $r \in R$ and b_r as the required bandwidth of request r. In particular, we assume that the computing capacity of VNF is limited by C_m^{cpu} and measured by the number of instructions processed per second (IPS). Let $c_{rf_i}^{IPS}, c_{rf_i}^{mem}$ be the IPS and memory required to create

rf_i respectively. A parameter α represents the relationship between IPS and CPU core. It means each CPU core can support α IPS. To simplify the model, we make m only be used by one type of VNF. D_{n*n} is defined as the VNF correlation matrix, in which n denote the number of VNF type.

B. Mathematic Formulation

Binary Decision Variables. We first define a decision variable $x_m^{rf_i}$ to indicate whether function rf_i is used in m:

$$x_m^{rf_i} = \begin{cases} 1 & \text{if } f \text{ on request } r \text{ is used in } m \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The second decision variable x_v^m denotes whether VNF instance m is created on physical node v:

$$x_v^m = \begin{cases} 1 & \text{if VNF instance } m \text{ is created on } v \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The last binary decision variable y_r^l is defined to denote whether SFC request r through link l, where $l \in L$:

$$y_r^l = \begin{cases} 1 & \text{if request } r \text{ traverses through link } l \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

System Constrains. Firstly, we take CPU and memory of physical nodes into consideration. The resources occupied by all VNF instances deployed on the same node cannot exceed the resource capacity of physical nodes. The constraints are described as:

$$\sum_m x_v^m C_m^{cpu} \le C_v^{cpu} \qquad \forall v \in V \tag{4}$$

$$\sum_m x_v^m C_m^{mem} \le C_v^{mem} \qquad \forall v \in V \tag{5}$$

Besides, we have to guarantee the capacity of VNF instances. In detail, the IPS and memory consumption of all request r through a VNF instance m cannot exceed the supported capacity of that VNF instance. We present them as follows and α denotes the number of IPS supported by each CPU core:

$$\sum_r \sum_i x_m^{rf_i} c_{rf_i}^{IPS} \le \alpha C_m^{cpu} \qquad \forall m \in M \tag{6}$$

$$\sum_r \sum_i x_m^{rf_i} c_{rf_i}^{mem} \le C_m^{mem} \qquad \forall m \in M \tag{7}$$

After that, the bandwidth is a major consideration for physical links. Obviously, the bandwidth of link l must be larger than the total bandwidth consumption of all SFC requests through it:

$$\sum_r y_r^l b_r \le B_l \qquad \forall l \in L \tag{8}$$

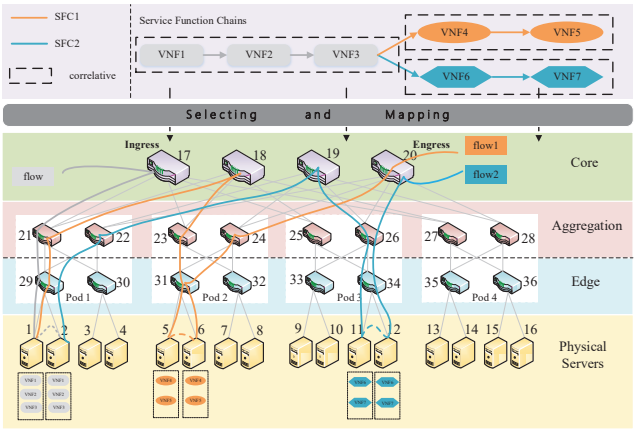

Fig. 1. Example of VNF mapping scheme

To ensure that the SFC request r only passes through VNF instance once, we define the following constraint:

$$\sum_m x_m^{rf_i} = 1 \qquad \forall r \in R \tag{9}$$

C. Objective

Our main objective in this paper is to study the joint correlation-aware VNF selection and placement problem to minimize link occupancy while ensuring an effective SFC acceptance, and a high VNF utilization rate. Therefore, the target function is expressed as follows, in which N represents the number of SFC requests accepted:

$$Minimize: \quad \frac{1}{N} \sum_r^R \sum_l^L y_r^l b_r \tag{10}$$

$$subject\ to: \quad (1)\text{-}(9)$$

This problem of VNF selection and placement can be reduced to VNF location problem and has been proved as a NP-Hard problem in [12]. Hence, we propose a self-learning matrix approach to partition VNF correlation and design a correlation-aware heuristic algorithm that can scale with problem size for large-scale DCNs in Section IV.

IV. Heuristic Algorithm Design

In this section, we propose a heuristic algorithm named JCVP (Joint Correlation-aware VNF Placement) to minimize link occupancy. As shown in Fig.1, when SFC requests in the network arrive, JCVP aims to place the high correlation-aware VNFs in the same server or pod. Meanwhile, to ensure SFC acceptance and VNF utilization rate, we consider maximizing the reuse of VNFs by taking advantage of VNF correlation aggregation. Hence, we first introduce the method to partition the correlation of VNFs. Then, we describe how the heuristic algorithm (JCVP) maps VNFs based on correlation results.

Algorithm 1 JCVP Algorithm

Input: $G(V, L)$, SFC requests R ($r \in R$)
Output: NodesList: substrate nodes hosting VNFs

```
Initialize Correlation matrix D_{n*n}
Search for all requests R^{pre} that were previously accepted
if R^{pre} ≠ null then
    for all each r ∈ R^{pre} do
        Update Correlation matrix D_{n*n}
    end for
end if
for rf_i ∈ R do
    if (num == Threshold N) then
        Update Correlation matrix D_{n*n}
        num ← 0
    end if
    if StronglyCorrelated(rf_i, rf_{i+1}) then
        S^k_{i+1}(rf_{i+1}) ← EdgePlacement(rf_{i+1})
    else if WeaklyCorrelated(rf_i, rf_{i+1}) then
        S^k_{i+1}(rf_{i+1}) ← PodPlacement(rf_{i+1}, podid)
    else
        S^k_{i+1}(rf_{i+1}) ← RackPlacement(rf_{i+1})
    end if
    num++
end for
return NodesList ← r
```

A. VNF Correlation Definition

D_{n*n} in Table II is defined as the VNF correlation matrix, in which n denotes the number of VNF types in F. Due to the dynamic and random feature of the SFC requests, it is difficult to predict the relation between VNFs in advance. Therefore, based on the self-learning method, we analyze each accepted SFC demand to constantly update the correlation matrix. The method is roughly divided into three steps. The specific content of each step is as follows:

- The first step is to initialize the correlation matrix D_{n*n} parameters. Since the same type of VNF in SFC is mostly not reconnected, we set each diagonal element of the matrix to -1 as infinity and other parameters of the matrix to 0.
- Then, the correlation matrix D_{n*n} is updated based on the all accepted SFC requests. In detail, whenever f_j is detected after f_i, matrix element D_{ij} is incremented by one. The result element D_{ij} represents the number of connection times of f_i and f_j in all accepted requests. For example, we assume that there is an accepted SFC request, in which the order of VNFs is $\{f_4 \rightarrow f_5 \rightarrow f_9 \rightarrow f_1 \rightarrow f_7\}$. Then matrix elements D_{45}, D_{59}, D_{91}, D_{17} will be incremented by one. Of course, if there is no SFC requests in advance, the self-learning method can still accurately partition VNF correlation.
- When new SFC requests are detected for access, the matrix D_{n*n} will be updated according to the self-learning process in the previous step.

We define the value of D_{ij} + D_{ji} as the basis for the correlation between f_i and f_j. A higher value indicates a stronger correlation between VNFs (two types of VNFs often appear together). On the contrary, a lower value indicates a weaker correlation between VNFs. Also, to ease the complexity and frequency of the correlation partitioning process, we update the correlation matrix D_{ij} after accepting a certain number of SFC requests.

Algorithm 2 RackPlacement Function

```
function RackPlacement(rf_{i+1})
for pod ∈ Rack do
    Find the most correlated VNF instance M* to rf_{i+1}
    Num[] ← getNumber(M*)
end for
PodId ← getMaxium(Num[])
S^k_{i+1}(rf_{i+1}) ← PodPlacement(rf_{i+1}, podid)
return
```

B. Correlation-aware Dynamic Programming Algorithm

Dynamic programming approach constructs the solution recursively by decomposing the problem and defining the state relationship between subproblems. The goal of our algorithm is to start from a single VNF placement and get the optimal solution for each subproblem (mapping a rf_i in request r to a suitable location) under the constraints of correlation.

We consider the SFC mapping process as a state model S. We assume a SFC request r, where the order of VNFs is $\{rf_1, rf_2, ..., rf_L\}$, then the state model S can be defined as $\{S_1, S_2, ..., S_L\}$ and the subproblem of VNF rf_i placement is solved in stage S_i. Under correlation constraints, the recursive relationship from rf_i to rf_{i+1} is described as follows:

$$S^k_{i+1}(rf_{i+1}) = min\{Cost(S^h_i, S^k_{i+1}) + S^h_i(rf_i)\}; h, k \in V \tag{11}$$

Function $Cost(S_i, S_{i+1})$ denotes the resource cost of VNF placement from rf_i to rf_{i+1}:

$$Cost(S_i, S_{i+1}) = \begin{cases} \frac{C_k^{cpu(u)}}{C_k^{cpu}} + \frac{C_k^{mem(u)}}{C_k^{mem}} & \text{if strongly correlated;} \\ \frac{C_k^{cpu(u)}}{C_k^{cpu}} + \frac{C_k^{mem(u)}}{C_k^{mem}} + \frac{B_l^u(h,k)}{B_l(h,k)} & \text{otherwise;} \end{cases} \tag{12}$$

in which, $C_k^{(u)}$ denotes the used resources of physical node k and $B_l^u(h, k)$ denotes the used bandwidth between physical nodes k and h. As we can see, the nodes with more available resources are more likely to be selected.

Algorithm 1 shows the pseudo code of the JCVP algorithm. We divide the VNF correlation into three levels, and different mapping functions will be performed according to the level of VNF. As mentioned earlier in this paper, we judge the correlation of VNF based on the matrix D_{n*n}. If they are

strongly correlated, the *EdgePlacement* function will be invoked to deploy them on the same edge. At this point, the link occupancy can be negligible. Otherwise, if they are weakly correlated or uncorrelated, *PodPlacement* and *RackPlacement* will be called respectively taking resource utilization and link bandwidth ratio into consideration comprehensively. Also, when nodes in the same edge are detected to have no resources available, we can only call the *PodPlacement* function, where VNF will be deployed on the same pod.

The *RackPlacement* function handles the VNF requests which are uncorrelated. As shown in Algorithm 2, the *RackPlacement* function identifies a PodId first by comparing the number of VNF instance most correlated to rf_{i+1} on each pod. Then, the function *PodPlacement* is invoked to acquire the least-loaded node.

V. Evaluation

In this section, we evaluate the performance of our proposed JCVP algorithm based on Java Alevin [14], a wider simulation environment for VNF resource allocation. To assess the quality of our solution, we compare our algorithm with a greedy algorithm and a Multi-Stage heuristic (MSH) [11].

A. Simulation Setting

In our study, we test a typical DCN topology (k-ary Fat-Tree) and set k to 10. It means that the number of Pods contained in this topology is 10, each Pod contains 5 edge switches and 5 aggregation switches. The total number of servers supported by the network is up to 250, meeting the basic requirements of the large scale DCN. Also, we set the capacity of each link as 500Mbps, the capacity of each link between pod and core layer as 5Gbps. We set the computing capacity of each server as 10 CPU cores and 32GB memory respectively. And we assume that each CPU core can support a million IPS.

In order to generate SFC requests with correlated VNFs, we divide five VNF sets $\{F_1, F_2, F_3, F_4, F_5\}$. Each set contains 10 types of VNF. In this simulation, each SFC request consists of two VNF sets randomly and traverses at most eight VNFs. The length of VNF set in SFC is uniformly distributed in the range of [1, 4]. Also, we define a probability parameter β to reflect the correlation between VNFs for each set. To be specific, when the current VNF rf_i is generated with $f_k \in F_1$, then rf_{i+1} will choose f_{k+1} with the probability of β. Last, we set the traffic rate of a SFC request is is uniformly distributed in the range of [50Mbps, 80Mbps].

B. Simulation Results

We evaluate the performance of our algorithm from the following four aspects which change with the number of SFC requests.

1) Link Occupancy: Fig.2 shows the average link occupation (Eq. 10) statistic for the three algorithms extracted from the simulation results. Compared with the greedy algorithm and the MSH algorithm, JCVP algorithm is the most effective in link resource utilization. This is because the traffic between

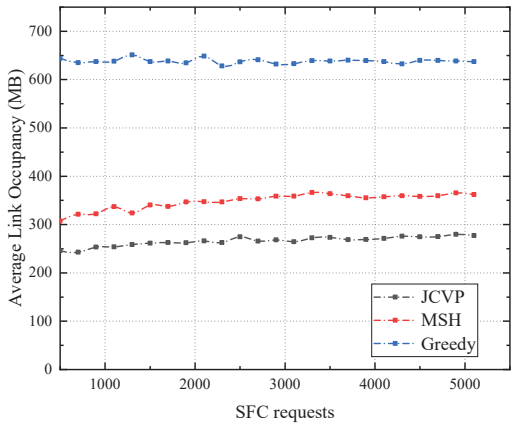

Fig. 2. Link Occupancy

the correlated VNFs occupies a large part of link resources. JCVP algorithm places the relevant VNFs closely to reduce this consumption and maps the unrelated VNFs with the constraint of minimizing link resources. As shown in this figure, the Greedy algorithm uses around 400MB more than the other two algorithms. The MSH algorithm consumes also around 50MB more than JCVP algorithm. Meanwhile, note that the JCVP algorithm can accept significantly more SFC requests than MSH algorithm. This will be analyzed in the next subsection.

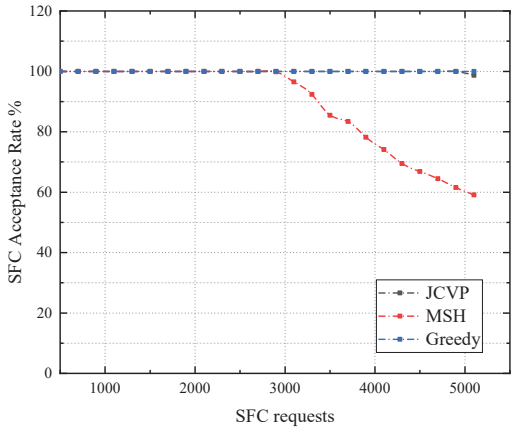

Fig. 3. SFC Acceptance Rate

2) SFC Acceptance Rate: We define the SFC acceptance rate as the ratio of the number of accepted requests to the total number of SFC requests. Fig.3 shows the SFC acceptance rate of the three algorithms. We can see that the MSH algorithm and greedy algorithm can accept almost all of SFC requests, while the MSH algorithm will reject partial requests when the number of requests reaches approximately 3000 even if there still are enough resources to map VNFs. And in the end, the acceptance rate was nearly 40% lower than the other two algorithms.

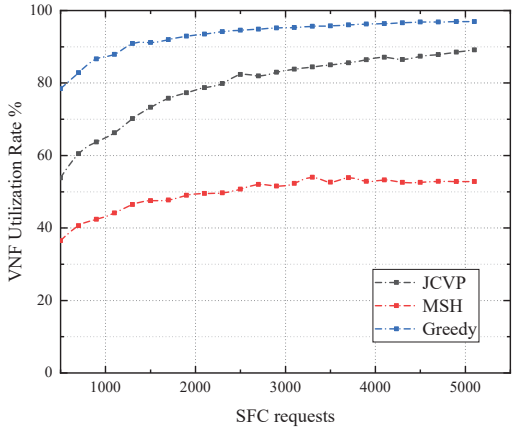

Fig. 4. VNF Utilization Rate

3) VNF Utilization Rate: As depicted in Fig.4, the Greedy algorithm performs the best in term of VNF Utilization Rate, while the MSH algorithm has the worst VNF Utilization Rate. However, as the number of SFC requests increases, the VNF utilization of the JCVP algorithm continues to approach the Greedy algorithm. This is due to the fact that our algorithm ensures these correlated VNFs are deployed closely in DCNs and can thus find more opportunities to reuse VNF instances in the future.

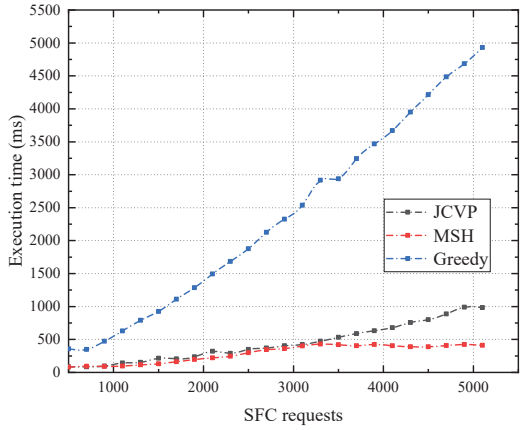

Fig. 5. Execution Time

4) Execution Time: The execution time of all the three algorithms depends on the change of the number of SFC requests greatly. As shown in Fig.5, the JCVP algorithm has better execution time than the Greedy algorithm. And the execution time of the greedy algorithm increases exponentially with the size of SFC, which is not suitable for large-scale DCNs. Execution times of the JCVP algorithm and MSH algorithm are close, while the JCVP algorithm can accept more SFC requests than MSH algorithm as previously illustrated in Fig.3.

VI. Conclusions

In this paper, we focus on Joint Correlation-aware VNF Selection and Placement problem in DCNs. We first formulate this problem as an ILP problem and then propose a self-learning matrix method to partition VNFs. We emphasize that the type and order of VNFs in SFC are correlated and a rational division of them can solve the contradiction between link occupancy and VNF utilization rate. With partition results as a guide, we propose a heuristic algorithm named JCVP to map VNFs into physical nodes. Extensive simulation results show that our approach has better performance in link occupancy, SFC acceptance, and VNF utilization rate.

Acknowledgment

This work was supported in part by the National Key Research and Development Program of China No.2018YFB1003804.

References

[1] Cisco Systems, "Cisco visual networking index: Forecast and methodology, 2016-2021," White Paper, 2016.
[2] ETSI GS NFV 001: "Network Functions Virtualisation (NFV); Use Cases".
[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, pp. 236-262, 2016.
[4] H. Moens and F. D. Turck, "Customizable function chains: Managing service chain variability in hybrid nfv networks," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 711-724, 2016.
[5] J. Gil Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Globecom Workshops*, vol. 18, pp. 236-262, 2016.
[6] M. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba et al. "On orchestrating virtual network functions in NFV," *CoRR, arXiv:1503.06377*, 2015.
[7] X. Li and C. Qian. "The virtual network function placement problem," *IEEE Conference on Computer Communications Workshops, INFOCOM Workshops*, pp. 69-70, 2015.
[8] X. Song, X. Zhang, S. Yu, S. Jiao and Z. Xu, "Resource-Efficient Virtual Network Function Placement in Operator Networks," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Singapore, pp. 1-7, 2017.
[9] V. Eramo, E. Miucci, M. Ammar and F. G. Lavacca, "An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures," in *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008-2025, Aug. 2017.
[10] J. Liu, Y. Li, Y. Zhang, L. Su and D. Jin, "Improve Service Chaining Performance with Optimized Middlebox Placement," in *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 560-573, 1 July-Aug. 2017.
[11] N. Huin, B. Jaumard and F. Giroire, "Optimal Network Service Chain Provisioning," in *IEEE/ACM Transactions on Networking,* vol. 26, no. 3, pp. 1320-1333, June 2018.
[12] R. Cohen, L. Lewin-Eytan, J. S. Naor and D. Raz, "Near optimal placement of virtual network functions," *2015 IEEE Conference on Computer Communications (INFOCOM)*, Kowloon, 2015, pp. 1346-1354.
[13] Zenan Wang, Jiao Zhang, Tao Huang and Yunjie Liu, "A clustering-based approach for Virtual Network Function Mapping and Assigning," *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, Vilanova i la Geltru, 2017, pp. 1-5.
[14] Fischer A, Botero J F, Duelli M, et al, "ALEVIN: a framework to develop, compare, and analyze virtual network embedding algorithms," in *Electronic Communications of the EASST*, pp. 7-13, 2011.