An Efficient Service Function Chain Placement Algorithm in a MEC-NFV Environment

Meng Wang†, Bo Cheng†*, Wendi Feng†§, Junliang Chen†
†State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications, Beijing, P.R. China
§Department of Computer Science and Engineering, University of Minnesota, Minneapolis, USA
{mengwang, chengbo, logan, chjl}@bupt.edu.cn

***Abstract*—Mobile Edge Computing (MEC) is a promising network architecture that pushes network control and mobile computing to the network edge. Recent studies propose to deploy MEC applications in the Network Function Virtualization (NFV) environment. The mobile network service in NFV is deployed as a Service Function Chains (SFC). In this paper, we solve the SFC placement problem in a MEC-NFV environment. We formulate the SFC placement problem as a weighted graph matching problem, including two sub-problems: a graph matching problem and a SFC mapping problem. To efficiently solve the graph matching problem, we propose a linear programming-based approach to calculate the similarity between physical nodes and VNFs. Based on the similarity, we design a Hungarian-based placement algorithm to solve the SFC mapping problem. Evaluation results show that our proposed solutions outperform the greedy algorithm in terms of execution time and resource utilization.**

***Index Terms*—Service Function Chain, Placement, Mobile Edge Computing, Network Function Virtualization, Linear Programming**

I. Introduction

With the advent of 5G and IoT, mobile applications with extreme requirements are increasing. Mobile Edge Computing (MEC) [1] and Network Function Virtualization (NFV) [2] are two emerging technologies to satisfy the mobile network service requests. Fig. 1 shows a MEC-NFV environment. In this figure, the network is considered as a three-layer architecture consisting of core level, edge level, and user level. The main idea of MEC is to push computing, network and storage functions to network edges (e.g., base stations, access points, and edge servers) to enable a good performance of latency intensive and computation critical network applications at the resource-limited edge devices [3].

NFV is a new network architecture that decouples network functions from the hardware. Due to the convenient and flexible management of Virtual Network Functions (VNFs), NFV significantly reduces the Capital Expenditure (CAPEX) and Operating Expense (OPEX) [4] and plays an important role in communication networks, such as mobile networks, enterprise networks, data center networks, and mobile edge networks. ETSI defines the network service request graphs as VNF Forwarding Graphs (VNF-FGs) [5]. Based on the network service requests, the NFV chain is described as a Service Function Chain (SFC) consisting of an ordered set of VNFs. The traffic in SFC is forwarded and controlled by the SDN controller [6]. Different from the SFC placement problem in traditional networks, mobile network services run at the resource-limited edge devices. Therefore, the SFC placement problem, which aims at optimizing resource utilization in a MEC-NFV environment, becomes a critical research issue.

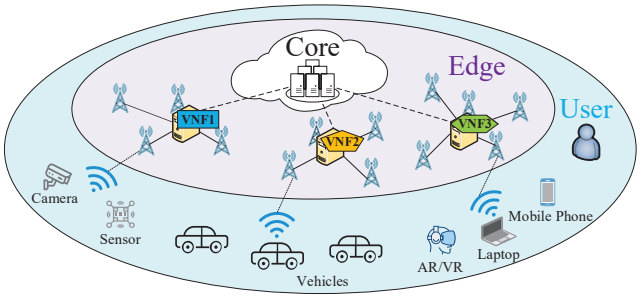


Fig. 1. SFC Placement in a MEC-NFV Environment.

Fig. 1 shows the SFC placement problem in a MEC-NFV environment. The SFC placement problem has received increasing attention [7]–[13]. Most of the existing works propose heuristic algorithms to solve the problem. However, these heuristic algorithms require long execution time and can only find a suboptimal result.

Given these facts, in this paper, we formulate the SFC request and the physical network as two weighted graphs. Thus, the SFC placement problem can be reformulated as an optimal matching problem that aims to minimize the distance between the adjacency matrices of two weighted graphs. Then we extend the Weighted Graph Matching Problem (WGMP) [14] for solving the optimal matching problem. The problem is divided into two sub-problems: a graph matching problem and a SFC mapping problem. To efficiently solve the two sub-problems, we propose a Linear Programming (LP)-based approach and a Hungarian-based algorithm. To the best of our knowledge, our work presents the first study that uses LP to solve the SFC placement problem aiming at minimizing the resource utilization while reducing the execution time.

In summary, the main contributions of this paper are summarized as follows:

*Bo Cheng is the corresponding author.

- We formulate the physical network and SFC request as two weighted graphs and formulate the SFC placement problem in the MEC-NFV environment as the WGMP consisting of graph matching and SFC mapping.
- We propose an LP-based approach and a Hungarian-based algorithm to solve the graph matching and SFC mapping in the WGMP. Our proposed solutions can run in polynomial time and optimize the resource utilization.
- We evaluate the performance of our proposed solutions. Evaluation results show our proposed solutions outperform the greedy algorithm in terms of execution time and resource utilization.

The rest of this paper is organized as follows. Section II discusses the related works. Section III formulates the SFC placement problem in the MEC-NFV environment. Section IV details the proposed solutions. Section V evaluates the performance. Finally, Section VI concludes this paper.

II. Related Works

The SFC placement problem has been widely studied in MEC. Jemaa *et. al.* [7] introduce VNF placement and optimization strategies to optimize resource utilization and prevent cloudlet overload. Laghrissi *et. al.* [8] propose an enhanced predictive placement algorithm for edge slicing in the edge cloud environment. Yala *et. al.* [9] propose a formulation of the VNF placement problem as an optimization problem of two objectives, namely maximizing service availability and minimizing access latency. And they propose a genetic algorithm to solve this problem. Chen *et. al.* [10] propose a metric that can better measure the condition of the physical resources. Based on this metric, they design an algorithm that can optimize resource utilization, running time and acceptance rate. Li *et. al.* [11] design a particle swarm optimization based edge server placement algorithm to reduce the energy consumption in mobile edge computing. Cziva *et. al.* [12] formulate the edge VNF placement problem and propose a dynamic placement scheduler to minimize VNF migrations. Song *et. al.* [13] propose a VNF resource allocation scheme based on context-aware grouping technology to minimize the delay of network services.

In summary, most of the existing works use heuristic algorithms to solve the SFC placement problem in MEC. Our work is different from the mentioned works since it divides the placement problem into LP-based graph matching and Hungarian-based SFC mapping. Our proposed solutions run in polynomial time and optimize resource utilization.

III. Problem Formulation

In this section, we formulate the SFC request and the physical network as two weighted graphs. Then we reformulate the SFC placement problem as an optimal graph matching problem. A summary of used notations is found in Table I.

TABLE I
Basic Notations Used throughout This Paper.

Symbol	Definition
PNG	
$N = \{n_1, n_2 \cdots\}$	the set of physical nodes.
$L = \{l_{ij} \cdots\}$	the set of physical links.
n_i, n_j	two physical nodes in physical network.
$l_{ij} = (n_i, n_j)$	the physical link between n_i and n_j.
VNF-FG	
$V = \{v_1, v_2 \cdots\}$	the set of VNFs.
$E = \{e_{pq} \cdots\}$	the set of logical links.
v_p, v_q	two VNFs in forwarding graph.
$e_{pq} = (v_p, v_q)$	the logical link between v_p and v_q.
Resource	
$C_{n_i}^{cpu}$	the CPU capacity of n_i.
$C_{n_i}^{mem}$	the memory capacity of n_i.
$C_{l_{ij}}^{bw}$	the bandwidth capacity between n_i and n_j.
cpu_{v_p}	the CPU consumption of v_p.
mem_{v_p}	the memory consumption of v_p.
$bw_{e_{pq}}$	the bandwidth consumption between v_p and v_q.
Variables	
$x_{n_i}^{v_p}$	whether v_p is mapped in n_i.
$y_{l_{ij}}^{e_{pq}}$	whether e_{pq} is mapped in l_{ij}.
α, β, γ	weights of cpu, memory, and bandwidth.

A. Physical Network Graph (PNG)

We formulate the physical network as a graph $P = (N, L)$, that is, Physical Network Graph (PNG). N and L indicate the set of physical nodes and physical links, respectively. $C_{n_i}^{cpu}$ and $C_{n_i}^{mem}$ indicate the CPU and memory capacity of each physical node $n_i \in N$. $C_{l_{ij}}^{bw}$ indicates the bandwidth capacity of each physical link $l_{ij} \in L$.

B. VNF-FG

The SFC request can be described as a VNF-FG $F = (V, E)$. V and E indicate the set of VNFs and logical links, respectively. We use cpu_{v_p} and mem_{v_p} to indicate the CPU and memory consumption of VNF $v_p \in V$. The bandwidth consumption of each logical link is the bandwidths of VNF flows passing through it. Therefore, we use $bw_{e_{pq}}$ to indicate the requested bandwidth of logical link $e_{pq} \in E$.

C. Objectives

In this subsection, we use two binary variables $x_{n_i}^{v_p}$ and $y_{l_{ij}}^{e_{pq}}$ to indicate the mapping status:

$$x_{n_i}^{v_p} = \begin{cases} 1 & \text{if } v_p \text{ is mapped in } n_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$y_{l_{ij}}^{e_{pq}} = \begin{cases} 1 & \text{if } e_{pq} \text{ is mapped in } l_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

We take CPU, memory and bandwidth constraints into consideration. In detail, the resource requirements of all VNFs mapped in the same physical node cannot exceed the resource

capacity of this physical node. Therefore, the CPU, memory, and bandwidth constraints are:

$$U_{n_i}^{cpu} = \frac{\sum_{v_p} cpu_{v_p} \cdot x_{n_i}^{v_p}}{C_{n_i}^{cpu}} \leq 1 \qquad \forall n_i \in N \tag{3}$$

$$U_{n_i}^{mem} = \frac{\sum_{v_p} mem_{v_p} \cdot x_{n_i}^{v_p}}{C_{n_i}^{mem}} \leq 1 \qquad \forall n_i \in N \tag{4}$$

$$U_{l_{ij}}^{bw} = \frac{\sum_{e_{pq}} bw_{e_{pq}} \cdot y_{l_{ij}}^{e_{pq}}}{C_{l_{ij}^r}^{bw}} \leq 1 \qquad \forall l_{ij} \in L \tag{5}$$

Our goal is to optimize the resource utilization:

$$\max \left(\alpha \frac{\sum_{n_i} U_{n_i}^{cpu}}{\sum_{n_i} x_{n_i}^{v_p}} + \beta \frac{\sum_{n_i} U_{n_i}^{mem}}{\sum_{n_i} x_{n_i}^{v_p}} + \gamma \frac{\sum_{l_{ij}} U_{l_{ij}}^{bw}}{\sum_{l_{ij}} y_{l_{ij}}^{e_{pq}}} \right) \tag{6}$$
$$s.t. \quad Eq.3 \quad to \quad Eq.5.$$

We introduce the weights (α, β, and γ) to separate the importance of CPU, memory, and bandwidth.

In this paper, we formulate the SFC placement problem as a weighted graph matching problem aiming to minimize the distance between two graphs. Matching similar nodes means that resource utilization can be improved.

IV. Proposed Solutions

In this section, we divide the SFC placement problem into graph matching and SFC mapping. Then we propose the LP-based approach and a Hungarian-based algorithm to solve the problem. While LP in Weighted Graph Matching Problem (WGMP) [14] is an old idea, it has not been widely applied to the SFC placement problem in the MEC-NFV environment.

A. Linear Programming in WGMP

The main idea of LP in WGMP is to minimize the distance between two weighted graphs. However, it also presents some limitations in the SFC placement problem.

1) WGMP requires two matching graphs of the same size. However, PNG size is significantly larger than VNF-FG.
2) LP computes the similarity between node and VNF, not applicable to multiple VNFs mapped in the same node.
3) The weight of node and VNF is similar but may not meet the resource requirements.

Therefore, we extend VNF-FG to the same size as PNG, with the expanded element values being zero. Besides, we design a mapping algorithm based on the Hungarian method, which can support multiple VNFs placed on the same node. In this algorithm, we verify the resource requirements.

B. LP-based Graph Matching

1) Adjacency Matrix (Step 1): In Algorithm 1, we define the physical network and SFC request as input. The output is the similarity matrix. At Step 1, A_P and A_F indicate the adjacency matrices of PNG and VNF-FG, respectively. We use p_{ii} to indicate the CPU capacity of node n_i. Note that we set a certain ratio between CPU and memory so that the association between the adjacency matrix and memory can be established. And we use p_{ij} to indicate the bandwidth capacity between node n_i and node n_j (so is f_{ii} and f_{ij}). We use the Dijkstra method to compute the weight between nodes (or VNFs) that are not directly connected.

Algorithm 1: LP-based Graph Matching

Input: The physical network: $P = (N, L)$;
The SFC request: $F = (V, E)$;
Output: The similarity matrix: M;

Step1: Compute adjacency matrices A_P and A_F:

$$A_P = \begin{cases} p_{ii} = C_{n_i}^{cpu} \\ p_{ij} = C_{l_{ij}}^{bw} \end{cases} \quad A_F = \begin{cases} f_{pp} = cpu_{v_p} \\ f_{pq} = bw_{e_{pq}} \end{cases}$$

Step2: Compute matrix A_{PF} from A_P and A_F:

$$A_{PF} = \begin{bmatrix} \{A_P - f_{11}I_n\} & \{-f_{21}I_n\} & \cdots & \{-f_{n1}I_n\} \\ \{-f_{12}I_n\} & \{A_P - f_{22}I_n\} & \cdots & \{-f_{n2}I_n\} \\ \vdots & \vdots & \ddots & \vdots \\ \{-f_{1n}I_n\} & \{-f_{2n}I_n\} & \cdots & \{A_P - h_{nn}I_n\} \end{bmatrix}$$

Where I_n is the identity matrix.

Step3: Use the Simplex method to solve the LP problem in *Eq.* 14. The matrix B is define by:

$$b_{ij} = \begin{cases} 1 & \text{for } i = 1, 2, 3, \cdots, n \text{ and } j = i, i+n, i+2n, \cdots, i+(n-1)n \\ 0 & \text{otherwise} \end{cases}$$

Step4: Get the similarity matrix M from vector m;

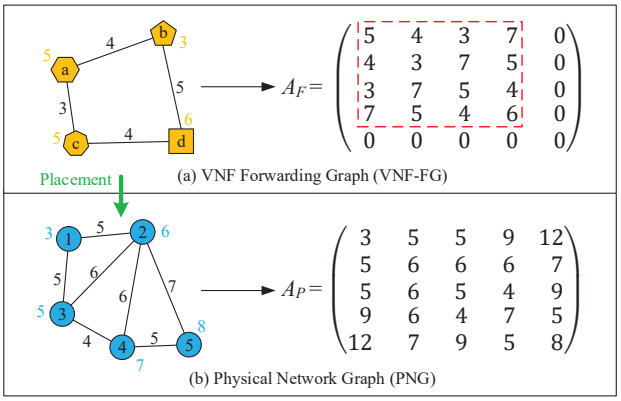


Fig. 2. The Adjacency Matrices of a) VNF-FG and b) PNG.

As Fig. 2(a) shows, the adjacency matrix ($4*4$) of VNF-FG is in the red dotted line box. As Fig. 2(b) shows, the adjacency matrix of PNG is a $5*5$ matrix A_P. Thus, we extend VNF-FG to a $5*5$ matrix A_F (the newly added elements are all zeros).

2) Distance (Step 2): Based on [14], the distance between PNG and VNF-FG can be defined as:

$$J(\Phi) = \sum_{i=1}^{n} \sum_{j=1}^{n} (p(n_i, n_j) - f(\Phi(n_i), \Phi(n_j)))^2 \tag{7}$$

Where $p(n_i, n_j)$ indicates the weight between node n_i and n_j. Φ indicates the one-to-one correspondence between VNF and node. For example, we assume that $(\Phi(n_i), \Phi(n_j)) = (v_p, v_q)$. Therefore, $f(\Phi(n_i), \Phi(n_j))$ is the weight between VNF v_p and v_q.

The WGMP is aimed at minimizing the distance $J(\Phi)$. Therefore, the matching problem can be reformulated as:

$$\min_{M} \left\| A_P - M A_F M^{\mathrm{T}} \right\|_1 \tag{8}$$

Where A_P and A_F are the adjacency matrices of PNG and VNF-FG, respectively. The permutation matrix M indicates the mapping function Φ. Note that $||.||$ is the L_1 norm.

The WGMP in *Eq.* 8 is equivalent to *Eq.* 9:

$$\min_{M} \left\| A_P M^{\mathrm{T}} - M^{\mathrm{T}} A_F \right\|_1 \tag{9}$$

We define a $n * n$ matrix R:

$$R = A_P M^{\mathrm{T}} - M^{\mathrm{T}} A_F \tag{10}$$

The matrices $R = \{r_{ij}\}$ and $M = \{m_{ij}\}$ can be partitioned by columns:

$$\begin{aligned} VEC(R) &= \{r_{11}, r_{21}, \cdots, r_{1n}, r_{2n}, \cdots, r_{nn}\}^{\mathrm{T}} \\ VEC(M^{\mathrm{T}}) &= \{m_{11}, m_{21}, \cdots, m_{1n}, m_{2n}, \cdots, m_{nn}\}^{\mathrm{T}} \end{aligned} \tag{11}$$

Therefore, the WGMP in *Eq.* 8 can be reformulated as:

$$\min_{M} \|VEC(R)\|_1 = \min_{M} \left\| A_{PF} VEC(M^{\mathrm{T}}) \right\|_1 \tag{12}$$

Where A_{PF} is an $n^2 * n^2$ matrix derived from $A_P = \{p_{ij}\}$ and $A_F = \{f_{ij}\}$, which is described in Algorithm 1 (Step 2).

From *Eq.* 9 to *Eq.* 12, we can conclude that the matching problem in *Eq.* 8 is equivalent to *Eq.* 13:

$$\min_{m} \|A_{PF} m\|_1 \qquad m \geq 0 \tag{13}$$

Where $m = VEC(M^{\mathrm{T}})$ is an $n^2 * 1$ vector.

3) Linear Programming (Step 3): Finally, we reformulate the minimization problem in *Eq.* 13 as an LP problem:

$$\begin{aligned} \min_{m,S,T} & \sum_{i}^{n^2} S_i + T_i \\ s.t. \quad & A_{PF} m + S - T = 0 \\ & Bm = e \\ & m \geq 0, S \geq 0, T \geq 0 \end{aligned} \tag{14}$$

Where $\{S_i\}$ and $\{T_i\}$ indicate two sets of real positive decision variables. B is a $2n*n^2$ matrix defined in Algorithm 1 (Step 3). B indicates the constraints that a permutation matrix M needs to keep the sum of any rows or columns to be 1.

Therefore, we use the Simplex method (Step 3 in Algorithm 1) to solve the LP problem in *Eq.* 14.

4) Similarity Matrix (Step 4): From Step 3 in Algorithm 1, we compute the vector m. In this step, we can get the similarity matrix M from vector m. As Fig. 3 shows, the data in green dotted line box indicates the similarity between VNFs and physical nodes. The rows and columns of the similarity matrix M correspond to VNFs and physical nodes, respectively. For example, m_{12} indicates the similarity between VNF v_a and node n_2.

Based on the highest similarity in Fig. 3, the optimal match should be that v_a, v_b, and v_c are mapped to n_3, n_1, and n_4, respectively. And v_d should be also mapped to n_4. However, n_4 cannot satisfy the resource consumption of v_c and v_d ($C^{cpu}_{n_1}$ = 7 while cpu_{v_c} = 5 and cpu_{v_d} = 6). Therefore, v_d should be mapped to n_5 since the similarity between v_d and n_5 is the next highest. Besides, we need to consider other constraints such as memory and bandwidth.

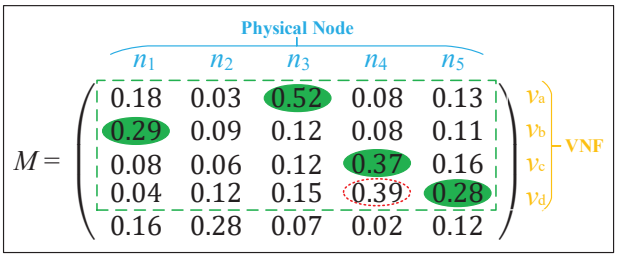


Fig. 3. Similarity Matrix *M*.

Based on these facts, we design a Hungarian-based SFC mapping algorithm that computes the mapping results to meet the resource requirements.

C. Hungarian-based SFC Mapping

In this subsection, we describe the Hungarian-based SFC mapping algorithm. We use the physical network, SFC request, and similarity matrix as input. The output of this algorithm is the mapping result matrix.

Algorithm 2: Hungarian-based SFC Mapping

Input: The physical network: $P = (N, L)$;
The SFC request: $F = (V, E)$;
The permutation matrix: M;
Output: The mapping result matrix: M_{res};

```
1  Initialize: find_node = false;
2  foreach v_p in V do
3      row_p = p^th row in permutation matrix M;
4      find_node = false;
5      while !find_node do
6          i = the index of highest element value in row_p;
7          if C^cpu_{n_i} ≥ cpu_{v_p} and C^mem_{n_i} ≥ mem_{v_p} then
8              if C^bw_l ≥ bw_e, ∀ l, e ∈ (n_{i-1}, n_i) then
9                  Update physical network status;
10                 m_pi = 1;
11                 find_node = true;
12             end
13             else
14                 row_p[i] = -1;
15             end
16         end
17         else
18             row_p[i] = -1;
19         end
20     end
21 end
22 return M_res;
```

At line 1 of Algorithm 2, we define *find_node* to indicate that whether we find a node that can hold VNF. Then we traverse all the VNFs to find nodes to hold them (line 2-21).

At line 3, we get the row in matrix M corresponding to this VNF. At line 4, we set *find_node* to false. Then, start find one node until *find_node* is true (line 5). At line 6, we get the index of the highest element (the highest similarity). If the largest value is at two or more places, we compare the node utilization of the corresponding nodes. And we choose the place with higher node utilization as the highest element. After getting the highest element, we determine whether the corresponding node can hold the VNF (line 7). At line 8, we check whether the links between the previous node and this node meet the link consumption. If all of the resources meet the requirements, the network state is updated (line 9-11). Otherwise, we set this highest value to -1 (line 14 and 18). Therefore, the VNF can find a node with the next highest similarity value (line 6).

D. Complexity Analysis

In Algorithm 1, the complexity of Step1, Step 2 and Step 4 is O($|N|^2$), O($|N|^4$), O($|N|^2$), respectively. Note that the Step 3 is implemented using a Simplex method but it has shown acceptable performance in most practical applications. Therefore, the time complexity of Algorithm 1 (except Step 3) is at the level of O($|N|^4$).

In Algorithm 2, the *foreach* (line 2) and *while* (line 5) run $|V|$ and $|N|$ times, respectively. Therefore, the time complexity of Algorithm 2 is at the level of O($|VN|$).

E. Baseline

In this subsection, we propose a bipartite matching-based greedy algorithm as a baseline to compare the performance of our proposed solutions.

Algorithm 3: Greedy Algorithm

Input: The physical network: $P = (N, L)$;
The SFC request: $F = (V, E)$;
Output: The mapping result matrix: M_{res};
1 **Step1:** Compute adjacency matrices A_P and A_F;
2 **Step2:** Initialize the potential value and compute bipartite matrix using A_P and A_F;
3 **Step3:** Compute max matching with min cost;
4 **Step4:** Check node mapping:
$\begin{cases} \textit{success,} & \textit{go to Step 5} \\ \textit{fail,} & \textit{update the potential value and go to Step 3} \end{cases}$
5 **Step5:** Compute link path and check link mapping:
$\begin{cases} \textit{success,} & m_{ij} = 1 \\ \textit{fail,} & \textit{update the potential value and go to Step 3} \end{cases}$

First, we compute adjacency matrices for PNG and VNF-FG (Step 1 of Algorithm 3). At Step 2, we initialize a potential value and create the bipartite graph. At Step 3, we compute the max matching with min cost. At Step 4, we check whether the node can hold VNF. If it succeeds, go to Step 5. Otherwise, update the potential value and return Step 3. Finally, we compute the link path and check link mapping (Step 5). If it succeeds, update the mapping result matrix. Otherwise, update the potential value and return Step 3.

V. Performance Evaluation

A. Simulation Setup

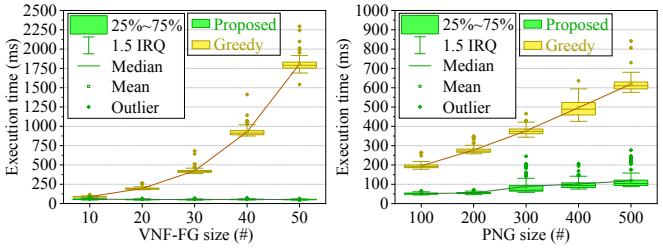
In this simulation, we evaluate the performance using a laptop of Windows 10 with 2.2 GHz Intel Core i5 processor and 8 GB memory. We implement LP-based approach and Greedy algorithm in Java based on Alevin [15], a widely used simulation environment for NFV resource allocation.

We run each scenario 100 times, so the statistics are almost unaffected by the accidental events. The GT-ITM [16] topology generator in NS-2 [17] can randomly generate graphs. The topology of VNF-FG is similar to the physical network since it is used to describe a network forwarding graph. Therefore, we use the GT-ITM tool to generate PNGs and VNF-FGs. 50% of the nodes (VNFs) are directly connected. We design the parameters of PNG and VNF-FG by reference to the simulation scenarios in the literature [18]. The parameters used can be found in Table II.

TABLE II
Parameters of PNG and VNF-FG.

Parameters	**PNG**	**VNF-FG**
Size	$N \in [100 - 500]$	$V \in [10 - 50]$
Node	$C_{n_i}^{cpu} \in [50 - 100]$	$cpu_{v_p} \in [0 - 20]$
Link	$C_{l_{ij}}^{bw} \in [50 - 100]$	$bw_{e_{pq}} \in [0 - 20]$
Connectivity	$\theta = 0.5$	$\theta = 0.5$

B. Execution Time



a) Execution Time with Different VNF-FG Size. b) Execution Time with Different PNG Size.

Fig. 4. Execution Time with Different a) VNF-FG Size and b) PNG Size.

As Fig. 4(a) shows, we evaluate the execution time of LP and Greedy with different VNF-FG size (PNG size is 100). We can conclude that our proposed LP-based approach runs faster than Greedy. Besides, the execution time of the LP-based approach is independent of the VNF-FG size. In contrast, the execution time of Greedy increases dramatically as the VNF-FG size increases. Moreover, we can observe that there are many outliers in Greedy, which means that sometimes Greedy iterates many times.

As Fig. 4(b) shows, we can conclude that LP runs faster than Greedy with different PNG size (VNF-FG size is 20). We can also observe that the execution time of both LP and Greedy increases as PNG size increases. However, the execution time of LP increases slowly while Greedy is fast.

In conclusion, our proposed LP-based approach runs faster than Greedy. And the execution time of LP is not affected by

the VNF-FG size since it is only affected by the size of the physical network adjacency matrix. The execution time of LP increases slowly as the PNG size increases.

C. Resource Utilization

For resource utilization, we use α, β, and γ to indicate the weights of CPU, memory, and bandwidth, respectively. We think that the three factors of CPU, memory, and bandwidth are equally important, so we set $\alpha = \beta = \gamma$.

In this paper, the node resource contains CPU and memory. We set a certain ratio between CPU and memory, so CPU and memory utilization are similar. Therefore, we use CPU utilization to indicate node utilization.

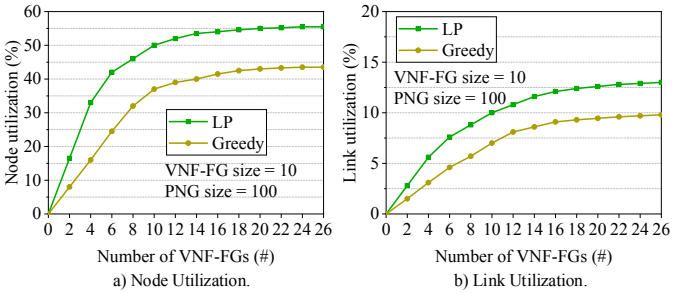


Fig. 5. a) Node Utilization and b) Link Utilization.

As Fig. 5(a) shows, we evaluate the node utilization with different numbers of VNF-FGs (VNF-FG size is 10 and PNG size is 100). We can observe that the node utilization of both LP and Greedy increases as the number of VNF-FGs increases. LP has a higher node utilization (56%) than Greedy (44%). It means that the LP-based graph matching can optimize the node utilization efficiently.

As Fig. 5(b) shows, the link utilization of LP-based approach (13%) is higher than Greedy (10%). We can observe that the link utilization is low. It is because the communication between two nodes may use several links, and the links used by different nodes are likely to be non-coincident.

In summary, the resource utilization of both LP-based approach and Greedy increases as the number of VNF-FGs increases. Our proposed LP-based approach outperforms the Greedy in resource utilization.

VI. Conclusion

This paper focuses on the SFC placement problem in a MEC-NFV environment. Different from the existing works, we formulate the problem as the WGMP. Then we divide the WGMP into two sub-problems: a graph matching problem and a SFC mapping problem. And we propose an LP-based approach and a Hungarian-based algorithm to solve two sub-problems. Evaluation results show that our proposed solutions can efficiently reduce the execution time and optimize resource utilization.

In our future work, we consider the trade-off between resource utilization and service performance (e.g. resilient [19] and end-to-end service delay). And we also study the relationship between resource utilization and the weight of different factors (i.e. CPU, memory, and bandwidth).

Acknowledgment

This work was supported in part by the National Key Research and Development Program of China (Grant No.2017YFB1400603), in part by the Natural Science Foundation of China (Grant No.61772479), and in part by the BUPT Excellent Ph.D. Students Foundation (Grant No.CX2019214).

References

[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing: A key technology towards 5G, https://www.etsi.org/images/files/etsiwhitepapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf," 2015. ETSI.

[2] M. Chiosi, S. Wright, J. Erfanian, and B. Smith, "Network functions virtualisation (NFV), http://portal.etsi.org/NFV/NFV_White_Paper.pdf," *SDN and OpenFlow World Congress*, 2012. ETSI, NFVGS.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, pp. 2322–2358, 2017.

[4] 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.

[5] E. G. N. 003, "Network functions virtualisation (NFV): Terminology for main concepts in NFV, https://www.etsi.org/deliver/etsi_gs/nfv/001_099/003/01.02.01_60/gs_nfv003v010201p.pdf," 2014. ETSI, NFVGS.

[6] Z. Guo, W. Chen, Y.-F. Liu, Y. Xu, and Z.-L. Zhang, "Joint switch upgrade and controller deployment in hybrid software-defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, pp. 1012–1028, 2019.

[7] F. B. Jemaa, G. Pujolle, and M. Pariente, "QoS-aware VNF placement optimization in edge-central carrier cloud architecture," *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, 2016.

[8] A. Laghrissi, T. Taleb, M. Bagaa, and H. Flinck, "Towards edge slicing: VNF placement algorithms for a dynamic and realistic edge cloud environment," *2017 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2017.

[9] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability drivenVNF placement in a MEC-NFV environment," *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, 2018.

[10] Z. Chen, S. Zhang, C. Wang, Z. Qian, M. Xiao, J. Wu, and I. Jawhar, "A novel algorithm for NFV chain placement in edge computing environments," *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2018.

[11] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 66–73, 2018.

[12] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vNF placement at the network edge," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 693–701, 2018.

[13] S. Song, C. Lee, H. Cho, G. Lim, and J. Chung, "Clustered virtualized network functions resource allocation based on context-aware grouping in 5G edge networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2019.

[14] H. A. Almohamad and S. O. Duffuaa, "A linear programming approach for the weighted graph matching problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, pp. 522–525, 1993.

[15] J. Gil-Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 518–532, 2016.

[16] USC, "GT-ITM topology generator, https://www.isi.edu/nsnam/ns/ns-topogen.html#gt-itm," 2001. University of Southern California.

[17] USC, "The network simulator - ns-2, https://www.isi.edu/nsnam/ns/," 2001. University of Southern California.

[18] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 725–739, 2016.

[19] Z. Guo, W. Feng, S. Liu, W. Jiang, Y. Xu, and Z.-L. Zhang, "Retroflow: maintaining control resiliency and flow programmability for software-defined wans," in *IWQoS*, 2019.