Joint Delay-Cost Optimization for SFC Placement with Shareable VNF Instances in MEC

Abstract. With the rise of Network Function Virtualization (NFV) technology, Service Function Chain (SFC), which is composed of Virtual Network Function (VNF) has become a mainstream form of services in MEC network. The multi-instance nature and shareability of VNFs pose a challenge for SFC in selecting appropriate VNF instances. Additionally, the end-to-end delay and deployment cost need to be considered during the SFC placement. Besides, load balancing of MEC network is crucial to the reliability of service. However, previous works have not taken a comprehensive insight into these problems. In this paper, we study the NFV-enabled SFC placement, aiming to minimize service delay and deployment cost, while achieving load balancing in MEC network by leveraging the shareability of VNF instances. We formulate the Joint Delay-Cost Optimization problem for SFC Placement with Shareable VNF instances in MEC network (JDCOP-S) and design a Path Tree and Backtracking Pruning based Placement Algorithm, named as PTBP-PA. The proposed PTBP-PA transforms the MEC network topology into a path tree structure and utilizes backtracking pruning algorithm to search placement strategy with minimum system utility. Simulation results show that PTBP-PA outperforms baseline methods in terms of service delay, deployment cost, and load balancing.

Keywords: Mobile Edge Computing · Network Function Virtualization · Service Function Chain · Service Placement · Delay · Cost · Load Balancing.

1 Introduction

Mobile devices such as smartphones and pads have become very popular for daily work and communications. To meet the requirements of low latency and high quality during the service, computationally intensive mobile technologies and applications have emerged. However, mobile devices are constrained by limited computing capabilities and storage resources, which leads to a challenge to process data effectively. Mobile Edge Computing (MEC) as a variance of Edge Computing addresses this issue by bringing computing resources closer to the edge of the network and providing end users with desirable services [16].

Network Function Virtualization (NFV) technology transforms dedicated hardware middleboxes into Virtual Network Function (VNF) instances that can be run on general-purpose servers [4]. The technology decouples network functions from hardware devices, which can reduce deployment cost and improve flexibility of the network. NFV incorporates multi-tenancy technology to realize the sharing of VNF instances among multiple users. VNF sharing decreases the number of VNF instances that need to be initialized on servers, thereby saving a significant number of resources [7].

Applying NFV to MEC can not only reduce service delay but also decrease deployment cost for end users. Service in NFV-enabled network consists of a sequence of VNFs that need to be placed on edge servers, also known as Service Function Chain (SFC) [4]. Considering the limited resources of edge servers compared to cloud servers, load balancing is also crucial to ensure high-quality and reliable services [3]. Take the augmented reality game Pokémon Go[1] as an example, depicted in Fig.1. In the MEC network, there are two candidate service placement strategies, marked in red-line and green-line respectively. The red-line strategy selects servers with high resource utilization, increasing the risk of server overload and degrading service quality. However, the green-line strategy chooses servers with lower resource utilization, leading to improved reliability and service quality. We therefore prefer the green-line strategy rather than the red one.

For each SFC request, VNFs need to be placed on edge servers in MEC network, this is SFC Placement. Several studies [3, 4, 12] focus on optimizing SFC placement to minimize delay or deployment cost but neglects shareability of VNF instances or joint optimization of delay and cost. There are research efforts [5, 13, 15] that effectively leverage the shareability of VNF instances, while the objective

[1] https://www.pokemongo.com

of other works [8, 9] focuses on minimizing delay and cost. Nonetheless, the issue of load balancing of MEC is not considered in these works.

This paper focuses on SFC placement in MEC by utilizing shareable VNF instances. It aims to jointly optimize user experience and service cost while considering the load balancing in the MEC network. Main contributions of this paper are as follows:

- We firstly model service delay and deployment cost based on the characteristics of edge computing and introduce the Gini coefficient to assess the load balancing of servers and links.
- We define the Joint Delay-Cost Optimization problem for SFC Placement with Shareable VNF instances in MEC, also referred to as JDCOP-S, and show its NP-hardness.
- To solve JDCOP-S, we design a Path Tree and Backtracking Pruning based Placement Algorithm (PTBP-PA). The proposed PTBP-PA transforms the network topology into a path tree structure and searches efficient placement strategies with backtracking pruning algorithm. It can provide with a high-performance placement strategy after a short period of time.
- We finally evaluate the performance of the proposed PTBP-PA through experimental simulations. Simulation results demonstrate that our algorithm outperforms the baseline placement algorithms in terms of service delay, deployment cost and load balancing.

The rest of this paper is organized as follows. Sec.2 presents related works. Sec.3 formulates the JDCOP-S and shows the problem is NP-hard. In Sec.4, we propose a heuristic algorithm PTBP-PA to solve JDCOP-S. Simulation results and performance analysis are presented in Sec.5. Sec.6 concludes this paper.

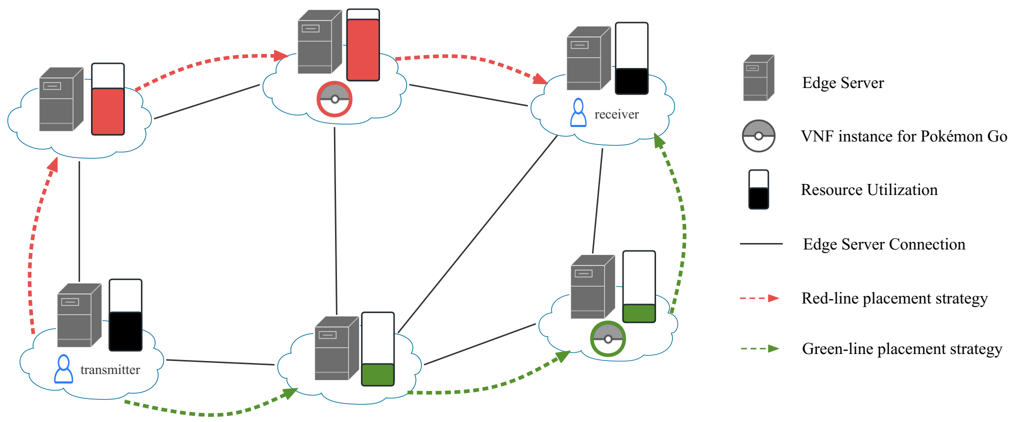

receiver
transmitter
Edge Server
VNF instance for Pokémon Go
Resource Utilization
Edge Server Connection
Red-line placement strategy
Green-line placement strategy

Fig.1. Two candidate service placement strategies for Pokémon Go users

2 Related Works

This section will discuss works in the literature that address service placement in MEC network.

In NFV-enabled network, services can be generally classified into specific VNFs, unicast SFCs and multicast SFCs. We first review studies focused on specific VNFs and multicast SFCs. Local search algorithm is used in [6] to deploy specific VNFs on edge servers to minimize total deployment cost. Similarly, [1] designs centralized and distributed deployment algorithms using preference lists to deploy user requests with specific VNFs. However, these works cannot effectively be applied for SFC services. Multicast SFCs require chaining of a sequence of network functions from a source to a set of destinations. [2] studies the joint multislot design of user scheduling to minimize the time needed to serve all the users for multicast transmission in multiuser MISO systems. Steiner tree algorithm is used to obtain the placement strategy with low deployment cost in [13].

Unicast SFC has unique source server and destination server. There are several studies which focus on unicast SFC placement, also known as SFC placement with different objectives. [11] transforms SFC placement into a graph model and uses the shortest path algorithm to obtain the placement strategy with high reliability. Similarly, [12] uses the shortest path algorithm to get the placement strategy with minimum system cost based on auxiliary graphs. Tabu algorithm is ued in [7] to get placement strategies with low configuretion cost. [9] incorporates the instability of SFC data into the model and uses Markov estimation technique to compute a placement strategy with low system cost while satisfying the delay requirement of users. However, load balancing of MEC network, which is a key metric to service placement, is not considered in these works. [10] incorporates the utilization ratio of servers into resource usage cost model and devices an online algorithm to find the placement strategy with low service cost. Colony optimization algorithm is used in [3] to find placement strategies that minimize the standard deviation of remaining CPU and RAM resources on edge servers.

Service delay, deployment cost, and load balancing are important metrics that need to be considered in NFV-enabled MEC network. We will take a comprehensive insight into SFC placement and endeavor to propose a model to address these issues.

3 System Model and Problem Formulation

3.1 System Model

System model is composed of the following three parts: user requests, physical network with VNF instances, and utility model. Key notations in the model are listed in Table 1.

Table 1. Key Notations

Notation	Description
$\boldsymbol{U}$	The set of users
$\boldsymbol{K}$	The set of VNF types
$\boldsymbol{V}$	The set of servers in the network
$\boldsymbol{E}$	The set of links in the network
D_u	The total service delay of $u \in \boldsymbol{U}$
C_u	The total deployment cost of $u \in \boldsymbol{U}$
W_u	The utility of $u \in \boldsymbol{U}$
$x_{u,i}^v$	Whether the ith VNF on SFC of $u \in \boldsymbol{U}$ is placed on $v \in \boldsymbol{V}$ (=1) or not (=0)
y_u^e	Whether $u \in \boldsymbol{U}$ uses $e \in \boldsymbol{E}$ to transmit data (=1) or not (=0)
$z_u^{v,k}$	Whether $u \in \boldsymbol{U}$ needs to initialize a VNF instance of type $k \in \boldsymbol{K}$ on $v \in \boldsymbol{V}$ (=1) or not (=0)

User with SFC request

In an NFV-enabled MEC network, user requests are formed as Service Function Chain (SFC). In this paper, we assume that all SFCs are totally ordered, which means each VNF on the SFC has unique preceding node or succeeding node.

The user requests submitted to the MEC network can be represented as $r_u = (s_u,\ d_u,\ SFC_u,\ b_u)$. s_u and d_u represent the source server where user data is originated from and the destination server where user data is received respectively. We use $SFC_u = \{f_{u,1},\ f_{u,2},\ \ldots,\ f_{u,I_u}\}$ to denote the set of VNFs on SFC of u, where $f_{u,i}$ is the ith VNF of SFC_u and I_u is the length of the SFC. Each packet of data flow must pass through these VNFs in the specified order before reaching the destination. b_u represents the size of data flow.

Physical Network with VNF instances

An MEC network can be modeled as an undirected graph $\boldsymbol{G} = (\boldsymbol{V},\ \boldsymbol{E})$, where $\boldsymbol{V}$ is the set of edge servers and $\boldsymbol{E}$ is the set of links between servers in MEC network. Each server $v \in \boldsymbol{V}$ can be represented as $v = (C_v,\ C_v^{remain},\ cost_v,\ delay_v,\ Ins_v)$, where C_v is the total computing resources of v for implementing various VNF instances, C_v^{remain} is the remaining computing resources of v. Since the storage resources are relatively cheap compared to computing resources, we focus solely on computing resources. The cost and delay of processing each unit of data flow by v are denoted as $cost_v$ and $delay_v$ respectively, while Ins_v is the set of existing VNF instances of all types on v. Each link $e \in \boldsymbol{E}$ can be denoted as e=$(B_e,\ B_e^{remain},\ cost_e,\ delay_e)$, where B_e is the total bandwidth resources of e, B_e^{remain}

represents the remaining bandwidth resources. The cost and delay of transmitting each unit of data flow along e are denoted as $cost_e$ and $delay_e$ respectively.

For simplicity, we assume that different types of VNFs among all SFCs can be classified into $|\boldsymbol{K}|$ types. For each type of VNF instance, it can be represented as $ins = (k,\ C_k,\ \mu_k,\ \mu_{remain},\ cost_k)$, where $k \in \boldsymbol{K}$ represents its type. C_k denotes the amount of computing resources required to initialize a VNF instance of type k on servers. Multi-tenancy technology enables the sharing of VNF instances between multiple users. Suppose that each VNF instance has a maximum processing capacity μ_k, while μ_{remain} represents the remaining processing capacity. If the remaining processing capacity is sufficient to process a user's data flow, the user can share the computing resources allocated to the instance with other users. $cost_k$ represents the cost of initializing a VNF instance of type k.

In the MEC network, servers and links have limited computing and bandwidth resources. Therefore, resource constraints must be considered during the placement of SFCs. Otherwise, it will be impossible for system to operate stably, potentially resulting in service interruption.

Utility Model

Based on service delay and deployment cost, we define a utility function to achieve the optimization of SFC placement. We also introduce Gini coefficient to balance the system load. The utility model will be deliberated as follows.

A. Service Delay

The end-to-end delay of a SFC has a significant impact on service quality, which consists of two parts: the processing delay and transmission delay, defined as follows:

Processing Delay: The processing delay experienced by user $u \in \mathbf{U}$ is generated from the data processing by VNF instances on server $v \in \boldsymbol{V}$. Without loss of generality, we assume that the processing delay on v is proportional to the size of data flow b_u, i.e.,

$$D_{u,v}^{process} = delay_v \cdot b_u, \forall u \in \boldsymbol{U}, v \in \boldsymbol{V} \tag{1}$$

where $delay_v$ is the delay of processing each unit of data flow by v.

Transmission Delay: When user $u \in \mathbf{U}$ needs to transmit data with link $e \in \boldsymbol{E}$, the transmission delay on e, which is also proportional to the size of data flow, can be calculated as:

$$D_{u,e}^{transmit} = delay_e \cdot b_u, \forall u \in \boldsymbol{U}, e \in \boldsymbol{E} \tag{2}$$

Where $delay_e$ is the delay of transmitting each unit of data flow by e.

We use a binary variable $x_{u,i}^v$ to denote whether the ith VNF on SFC of $u \in \boldsymbol{U}$ will be placed on $v \in \boldsymbol{V}$ (=1) or not (=0), and y_u^e to denote whether $u \in \boldsymbol{U}$ will use $e \in \boldsymbol{E}$ to transmit data (=1) or not (=0). For each user request r_u, the total service delay can be defined as the sum of the processing delay of each VNF and the transmission delay generated on each link traversed by data:

$$D_u = \sum_{i=1}^{I_u} \sum_{v \in \boldsymbol{V}} (x_{u,i}^v \cdot D_{u,v}^{process}) + \sum_{e \in \boldsymbol{E}} (y_u^e \cdot D_{u,e}^{transmit}), \forall u \in \boldsymbol{U} \tag{3}$$

B. Placement Cost

As the network service provider of MEC network charges user on a pay-as-you-go basis, the deployment cost will also affect users' service experience, which includes the processing cost of edge servers, the transmission cost of links and the initialization cost of VNF instances.

In MEC network, achieving load balancing is crucial to avoid server or link overloads and improve service reliability [3]. Common practices include incorporating resource utilization of servers or links into the cost model or considering the minimization of maximum resource utilization as an objective. However, these methods may not provide with a comprehensive evaluation of overall load balancing of the network, focusing on the current load of individual servers or links. To address this issue, we introduce the Gini coefficient as an evaluation metric to assess the load balancing in an MEC network.

The Gini coefficient is a statistical measure used to quantify the degree of inequality in the distribution of resources such as income or wealth. It ranges from 0 to 1, with a value closer to 1 indicating a more imbalanced distribution, and a value closer to 0 indicating a more balanced distribution. To measure the level of load balancing among servers in MEC network, we calculate the Gini coefficient of server resource utilization using the following formula:

$$G_{server} = \frac{\sum_{v1\in \boldsymbol{V}} \sum_{v2\in \boldsymbol{V}} |U_{v1} - U_{v2}|}{2|\boldsymbol{V}| \sum_{v\in \boldsymbol{V}} U_v} \tag{4}$$

where U_v is the resource utilization of server v, which is the ratio of the used computing resources of v to its total computing resources. Similarly, we user the following formula to calculate the Gini coefficient of link resource utilization:

$$G_{link} = \frac{\sum_{e1\in \boldsymbol{E}} \sum_{e2\in \boldsymbol{E}} |U_{e1} - U_{e2}|}{2|\boldsymbol{E}| \sum_{e\in \boldsymbol{E}} U_e} \tag{5}$$

where U_e is the resource utilization of link e, which is the ratio of the used bandwidth resources of e to its total bandwidth resources.

Processing Cost: The processing cost is generated by computing resource usage of server $v \in \boldsymbol{V}$, which is proportional to the size of data flow and may be influenced by the load balancing of servers:

$$C_{u,v}^{process} = cost_v \cdot b_u \cdot e^{G'_{server} - G_{server}}, \forall u \in \boldsymbol{U}, v \in \boldsymbol{V} \tag{6}$$

where $cost_v$ represents the cost of processing each unit of data flow by v. G_{server} and G'_{server} are Gini coefficients of server loads before and after the placement respectively.

Based on Multi-tenancy technology, users can share the computing resources allocated to a VNF instance with others. In this case, the value of $e^{G'_{server} - G_{server}}$ equals to 1 and the processing cost is simply determined by the product of $cost_v$ and b_u. However, when the user needs to initialize a new VNF instance on a server, if the initialization can achieve a more balanced load distribution in MEC network, the value of $e^{G'_{server} - G_{server}}$ will be smaller than 1, resulting in a decrease in the processing cost. Conversely, a more imbalanced load distribution will increase the processing cost significantly. To control deployment cost, it is necessary to design rational strategies for VNF instance selection.

Transmission Cost: The transmission cost is generated by bandwidth resource usage of link $e \in \boldsymbol{E}$, which is also determined by the size of data flow and the load balancing of links in MEC network:

$$C_{u,e}^{transmit} = cost_e \cdot b_u \cdot e^{G'_{edge} - G_{edge}}, \forall u \in \boldsymbol{U}, e \in \boldsymbol{E} \tag{7}$$

where $cost_e$ is the cost of transmitting each unit of data flow by e. $e^{G'_{edge} - G_{edge}}$ is used to balance the link loads in MEC network, which is similar to $e^{G'_{server} - G_{server}}$.

Initialization Cost: The initialization cost of creating a new VNF instance of type $k \in \boldsymbol{K}$ on server $v \in \boldsymbol{V}$ is defined as:

$$C_{v,k}^{initialize} = cost_k, \forall v \in \boldsymbol{V}, k \in \boldsymbol{K} \tag{8}$$

where $cost_k$ represents the cost of initializing a VNF instance of type k. For the sake of convenience, we assume that the initialization cost of creating a VNF instance of type k is consistent regardless of the edge server on which it is initialized.

We use a binary variable $z_u^{v,k}$ to denote whether $u \in \boldsymbol{U}$ needs to initialize a new VNF instance of type $k \in \boldsymbol{K}$ on $v \in \boldsymbol{V}$ (=1) or not (=0). For each user request r_u, the total deployment cost can be defined as the sum of the processing cost generated by computing resource usage of servers, the transmission cost generated by bandwidth resource usage of links, and the initialization cost of creating new VNF instances:

$$C_u = \sum_{i=1}^{I_u} \sum_{v\in \boldsymbol{V}} (x_{u,i}^v \cdot C_{u,v}^{process}) + \sum_{e\in \boldsymbol{E}} (y_u^e \cdot C_{u,e}^{transmit}) + \sum_{v\in \boldsymbol{V}} \sum_{k\in \boldsymbol{K}} (z_u^{v,k} \cdot C_{v,k}^{initialize}), \forall u \in \boldsymbol{U} \tag{9}$$

C. Utility Function

Service delay and deployment cost are key factors that have an impact on both user experience and service quality. To achieve comprehensive optimization, we introduce a weighted function that defines the utility for $u \in \boldsymbol{U}$:

$$W_u = \alpha \cdot D_u + (1 - \alpha) \cdot C_u, \forall u \in \boldsymbol{U} \tag{10}$$

where α and 1-α are the weights assigned to delay and cost respectively.

The weighted function considers user preferences for service delay and deployment cost, with α determining the priority. Higher α values prioritize service delay, whereas lower α values prioritize deployment cost. By adjusting the weights, we can strike a balance between the two factors based on specific scenarios and requirements.

3.2 Problem Formulation

Based on the model described above, we propose a Joint Delay-Cost Optimization problem for SFC Placement with Shareable VNF instances in MEC, i.e., JDCOP-S, which aims to jointly optimize service delay and deployment cost while ensuring load balancing of edge servers and links with shareable VNF instances. In the given MEC network represented by $\boldsymbol{G} =(\boldsymbol{V}, \boldsymbol{E})$ and a set of users $\boldsymbol{U}$ who submit service requests to the system, we define the JDCOP-S as follows:

$$Minimize \sum_{u \in \boldsymbol{U}} W_u \tag{11}$$

$$subject\ to: \sum_{v \in \boldsymbol{V}} x_{u,i}^{v} = 1, \forall u \in \boldsymbol{U}, i \in [1, I_u] \tag{12}$$

$$\sum_{ins \in ins_v} C_{ins.k} \leq C_v, \forall v \in \boldsymbol{V} \tag{13}$$

$$\sum_{u \in \boldsymbol{U}} (y_u^e \cdot b_u) \leq B_e, \forall u \in \boldsymbol{U} \tag{14}$$

$$x_{u,i}^{v}, y_u^e, z_u^{v,k} \in \{0,1\}, \forall u \in \boldsymbol{U}, i \in [1, I_u], v \in \boldsymbol{V}, k \in \boldsymbol{K}, e \in \boldsymbol{E} \tag{15}$$

The objective function is aimed to minimize the utility of all users. Each VNF of SFC request can only be placed on a single server (Eq. (12)). Eq. (13) and Eq. (14) define that the used resources cannot exceed the total resource capacity of servers and links.

3.3 NP-Hardness of the Problem

We prove the NP-hardness of JDCOP-S by transforming it into the Knapsack problem, a well-known NP-hard problem. In the Knapsack problem, given a fixed-capacity knapsack and a set of items with different weights and values, the goal is to select a subset of items that maximizes the total value while ensuring the total weight does not exceed the knapsack capacity.

We consider a special form of JDCOP-S in MEC, where there is a single edge server with a fixed computing resource capacity. Each SFC request consists of a single VNF, and a new VNF instance is created when a user offloads the requirement to the edge server. By transforming the JDCOP-S into the Knapsack problem, the edge server represents the knapsack, and each user's service requirement becomes an item in the Knapsack problem. The objective is to minimize the utility values of users while respecting the server's computing resource capacity, which is equivalent to selecting as many items as possible within the knapsack's limited capacity to maximize the total value. Since the Knapsack problem is NP-hard, we can conclude that the JDCOP-S problem is also NP-hard.

4 Algorithm Design

We propose a Path Tree and Backtracking Pruning based Placement Algorithm (PTBP-PA) for SFC placement in MEC network. The algorithm consists of two parts: a path tree creation algorithm that creates path trees for each pair of servers in MEC network and a backtracking pruning placement algorithm which is used to search the placement strategy with minimum utility.

4.1 Path Tree Creation Algorithm

In order to achieve an efficient SFC placement, we transform the MEC network topology into a path tree structure, which can represent hierarchical relationships between paths. The root node of the path tree represents the starting point, each node has multiple child nodes, and leaf nodes indicate the end point. Each branch from the root node to a leaf node on the path tree corresponds to a path from the starting point to the end point, as depicted in Fig.2.

We design Algorithm 1 for the creation of a path tree. This algorithm takes l connected paths as input and outputs the corresponding path tree, in which l is the number of top paths with minimum hops from the source server to the destination server. The algorithm iterates through the l connected paths and utilizes the function createTree() to create each branch of the path tree based on current connected path (lines 2-4).

Algorithm 1: Path Tree Creation Algorithm

```
Data: paths: l connected paths between source server s and destination server d
Result: the path tree between s and d
1  root=TreeNode(s)
2  for path ∈ paths do
3      createTree(path,1,root)
4  endfor
5  return root
```

The function createTree() is an auxiliary function for Algorithm 1 to create a branch of the path tree based on the given connected path. Since there may be overlaps between multiple connected paths, it is important to handle these overlaps efficiently. For example, in the path tree shown in Fig.2, there are two possible connected paths from server S1 to S3, i.e., S1→ S2→ S3 and S1→ S2→ S5→ S3. Both paths include the sub path S1→ S2. To simplify the search process for placement strategies, we merge these sub paths when creating the path tree. We check whether the current path branch already exists in the path tree. It the branch already exists, we proceed to the next level of recursion (lines 4-9). Otherwise, we need to create a new tree node before the recursion (lines 10-12). This ensures that the path tree is created properly and avoids unnecessary redundancy.

Function 1: createTree()

```
Data: path: a connected path from source server to destination server
      curIndex: index of server in path which has not been created in the path tree
      curTreeNode: current TreeNode in the path tree
Result: a branch of the path tree
1  if curIndex ≥ len(path) then
2      return
3  end
4  for nextTreeNode ∈ curTreeNode.nexts do
5      if nextTreeNode.serverIndex=path[curIndex] then
6          createTree(path, curIndex+1, nextTreeNode)
7          return
8      end
9  endfor
10 newNode=TreeNode(path[curIndex])
11 curTreeNode.nexts.append(newNode)
12 createTree(path, curIndex+1,newNode)
```

The positions of servers and the connectivity of links in MEC network remain unchanged. We can pre-create and store path trees for each pair of servers before SFC placement. This eliminates the need to recreate the path tree each time an SFC needs to be deployed and reduces the time overhead.

4.2 Backtracking Pruning Placement Algorithm

In this section, we introduce the backtracking pruning placement algorithm to obtain high-performance placement strategy for a single SFC request.

Once we have obtained the path tree based on the source and destination server, the backtracking pruning algorithm is used to explore various placement strategies along each branch, as shown in Algorithm 2. For each server on the current branch, we have two choices: either leave the server without any service placement, treating it as a transit node for the data flow (lines 7-16), or place the first undeployed VNF on the server using function placementOnServer() (lines 17-31). For example, the SFC placement strategy shown in Fig.2 utilizes VNF instances on servers S1, S5, and S3, while server S4 acts solely as a transit node without any service placement. For each choice, we evaluate the utility of current strategy and verify if it satisfies the resource constraints of servers and links. We recursively explore each branch of the path tree, repeating these steps until all VNFs have been placed and the data flow reaches the destination server (lines 4-6). This allows us to systematically explore various placement strategies while considering resource utilization, service delay, and deployment cost.

Algorithm 2: Backtracking Pruning Placement Algorithm for a single SFC request

```
   Data: $P_{cur}$: current placement strategy
         $P_{best}$: current optimal placement strategy
         $T_{cur}$: current path tree node
         $V_{cur}$: the first unplaced VNF on the SFC
   Result: $P_{best}$
 1 if $P_{cur}.utility \geq P_{best}.utility$ then
 2     return
 3 end
 4 if all VNFs has been placed and data has reached the destination then
 5     update $P_{best}$ and return
 6 end
 7 if data has not reached the destination then
 8     for $T_{next} \in T_{cur}$.nexts do
 9         if the link $e$ which connects $T_{cur}$ and $T_{next}$ does not have enough bandwidths then
10             continue
11         end
12         update $e$ and $P_{cur}$
13         Algorithm 2($P_{cur}$, $P_{best}$, $T_{next}$, $V_{cur}$)
14         restore $e$ and $P_{cur}$
15     endfor
16 end
17 if there are VNFs that have not been placed then
18     if the server $v$ of $T_{cur}$ does not have enough resources to place $V_{cur}$ then
19         return
20     end
21     place $V_{cur}$ on server $v \leftarrow$ placementOnServer($v$, $V_{cur}$.k, $b_u$)
22     for $T_{next} \in T_{cur}$ + $T_{cur}$.nexts do
23         if the link $e$ which connects $T_{cur}$ and $T_{next}$ does not have enough bandwidths then
24             continue
25         end
26         update $e$ and $P_{cur}$
27         Algorithm 2($P_{cur}$, $P_{best}$, $T_{next}$, $V_{cur}$.next)
28         restore $e$ and $P_{cur}$
29     endfor
30     restore server $v$
31 end
```

The function placementOnServer() is an auxiliary function that selects the existing VNF instance with the highest resource utilization or initializes a new VNF instance on the given server for the first unplaced VNF on the SFC. The function will sort all existing VNF instances of type k on the server based on the remaining processing capacity μ_{remain} before selection (line 1). A lower remaining processing capacity indicates higher resource utilization. Then it iterates through the sorted list to find a suitable instance that can accommodate the user's data flow (lines 2-7). If no suitable instance is found, a new VNF instance of type k is created (line 8).

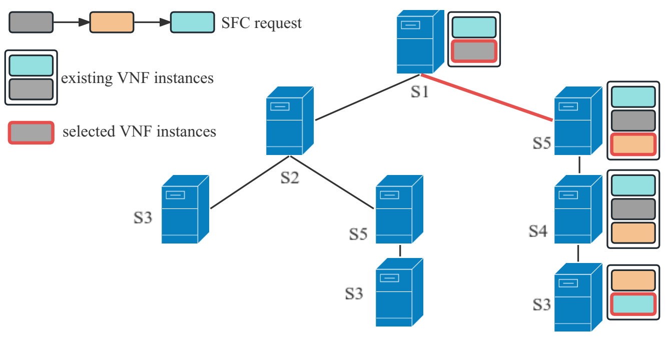


Fig.2. A possible SFC placement strategy on the path tree

By intelligently selecting existing VNF instances or creating new ones on the server, the function placementOnServer() aims to optimize the utilization of computing resources allocated to each VNF instance. This strategy helps improve the overall resource utilization, reduces the number of initialized VNF instances, and ultimately reduces the system cost and complexity.

Function 2: placementOnServer()

```
Data: $s$: the server which will be used to place services
      $k$: type of VNF
      $b$: size of data flow
Result: the used VNF instance
1  sort all VNF instances of type $k$ on $s$ in ascending order based on $\mu_{remain}$
2  for $ins \in ins_s$ of type $k$ do
3      if $ins.\mu_{remain} \geq b_u$ then
4          $ins.\mu_{remain}$ -= $b_u$
5          return $ins$
6      end
7  endfor
8  initial a new instance $ins$ of type $k$ on $s$
9  return $ins$
```

4.3 PTBP-PA for JDCOP-S

To solve the previously defined JDCOP-S, we design a Path Tree and Backtracking Pruning based Placement Algorithm, named as PTBP-PA, which demonstrates the overall process of SFC placement in MEC network. Detailed steps of the algorithm is shown in Algorithm 3.

The proposed PTBP-PA first finds the top l connected paths with minimum hops for each pair of servers in MEC network by Depth-First-Search algorithm and calls Algorithm 1 to create and store path trees (lines 1-6). Then, it sorts all users in reverse order based on their size of data flow and finally invokes Algorithm 2 to get the placement strategy for each user in order (lines 7-10).

Algorithm 3 enables us to find a high-performance SFC placement strategy for each user. Additionally, path trees are precreated and stored, which eliminates the need for recreation during the subsequent placement process.

Algorithm 3: Path Tree and Backtracking Pruning based Placement Algorithm

```
1  for $s \in \boldsymbol{V}$ do
2      for $d \in \boldsymbol{V}$ do
3          find the top $l$ shortest paths between $s$ and $d$ with DFS
4          get the path tree between $s$ and $d$ $\leftarrow$ Algorithm 1
5      endfor
6  endfor
7  sort all users in $\boldsymbol{U}$ in descending order based on $b_u$
8  for $u \in \boldsymbol{U}$ do
9      get placement strategy $\leftarrow$ Algorithm 2
10 endfor
```

5 Simulation and Numerical Analysis

This section describes the parameter settings for the simulation experiments and compares the performance differences between PTBP-PA and other placement algorithms, including SSD [12], greedy algorithm and random algorithm.

5.1 Simulation Settings

The MEC network simulated in the experiments is based on the topology structure of a US operator network[2] , which includes 16 edge servers and 25 links, as shown in Fig.3. Each edge server has a computing resource capacity of 8000 MIPS and the bandwidth resource capacity of each link is set to 1200 Mbps [14]. The coefficients associated with the cost and delay of servers and links, denoted as $cost_v$, $delay_v$, $cost_e$, and $delay_e$, are all set to 1 for simplicity.

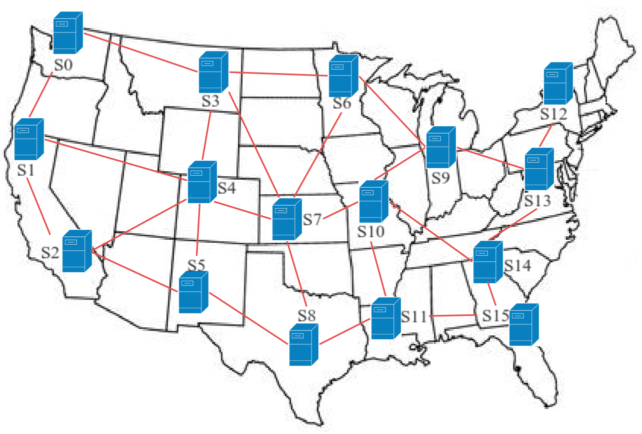


Fig.3. Simulated MEC network

There are a total of 15 types of VNF instances in MEC network. The required computing resources needed to initialize each type of VNF instances are randomly set within the range of (0, 100] MIPS and the maximum data processing capacity is randomly set within the range of (0, 100] Mbps. The initialization cost for creating each type of VNF instances is set to 100. For each user request, we set the length of their service function chain to 3, and the size of data flow is randomly set within the range of (0, 10] Mbps.

5.2 Baseline Algorithms

Here are brief introductions to the comparative algorithms used in the experiments:
- SSD: This algorithm creates a deployment auxiliary graph based on a connected path between the source and destination server, as well as the VNFs of SFC. It utilizes the shortest path algorithm to obtain the optimal placement strategy on that path and outputs the strategy with the lowest utility value among all connected paths [12].
- Greedy: The greedy placement algorithm starts from the user's source server and selects deployment servers and links with lowest utility until all required VNFs are placed and the data reaches the destination server.
- Random: The random placement algorithm starts from the user's source server and randomly selects deployment servers and data transmission links until all required VNFs are placed, and the data is transmitted to the destination server.
- PTBP-PA: Our algorithm finds the top 10 connected paths with minimum hops for each pair of servers in MEC and transforms these connected paths into path trees and utilizes backtracking pruning technique to search the placement strategy with lowest utility.

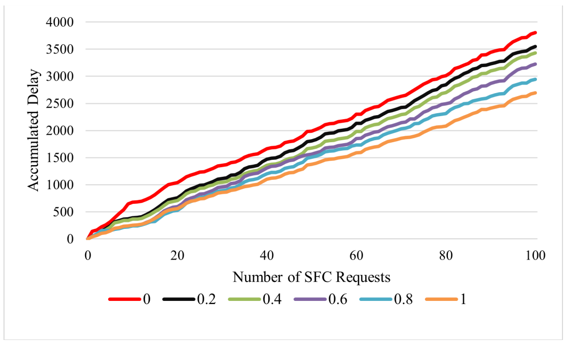
[2] Monarch Network Architects: www.monarchna.com/topology.html

5.3 Performance Analysis

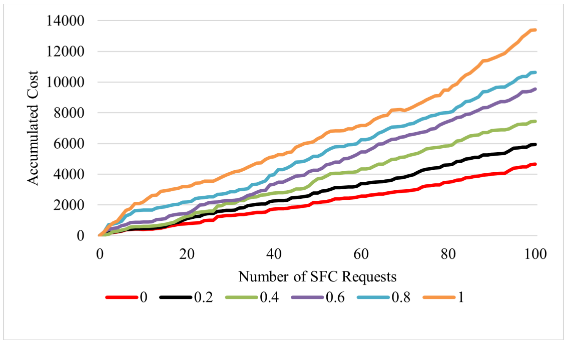
Parameter Impacts on the Algorithm Performance

We investigate the influence of parameter α in Eq. (10) on algorithm performance from two aspects: accumulated delay and cost. In the utility function, α represents the weight of service delay, while 1-α represents the weight of deployment cost. When α is set to 1, the focus is solely on service delay, ignoring deployment cost. Conversely, when α is set to 0, only deployment cost is considered, ignoring service delay.

Fig.4 shows the accumulated delay and cost generated by PTBP-PA when deploying multiple users under different values of α. From Fig.4a and Fig.4b, we can see that as the value of α increases, the accumulated delay decreases gradually, while the accumulated cost increases. The results indicate that increasing α reduces service delay but simultaneously increases deployment cost. On the other hand, decreasing α reduces deployment cost but leads to higher service delay. By adjusting the value of α, the algorithm can be customized for different scenarios.



(a) Accumulated delay



(b) Accumulated cost

Fig.4. Comparison of Algorithm Performance when α changes

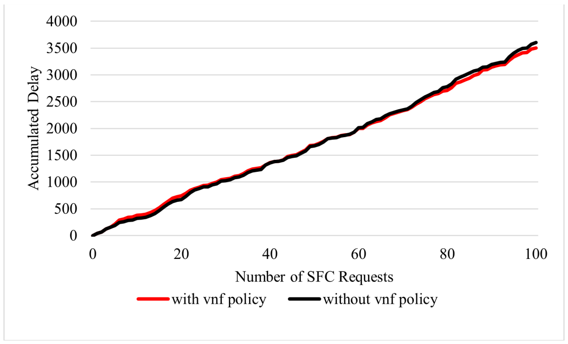
Comparison of Algorithm Performance with Different Factors

In this part, we first investigate the impact of VNF instance selection policy, specifically the function placementOnServer(), on the performance of PTBP-PA. We compare the PTBP-PA with placementOnServer() for VNF instance selection (with-vnf-policy) and PTBP-PA with random selection of VNF instances (without-vnf-policy). In next experiments, α which represents the weight of service delay in utility function is set to 0.5.

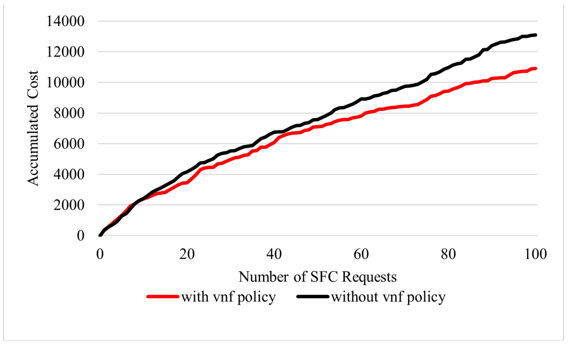
Fig.5 presents the accumulated delay and cost of PTBP-PA under different VNF instance selection strategies. From Fig.5a, it can be observed that the accumulated delay of both VNF selection strategies in PTBP-PA are similar and negligible. However, from Fig.5b, the accumulated cost of without-vnf-policy is higher than that of with-vnf-policy, and the difference increases with the number of user requests. When processing 100 consecutive user requests, without-vnf-policy has approximately 17% higher accumulated cost than with-vnf-policy. Fig.6 illustrates the maximum resource utilization of servers and links under different VNF instance selection policies. With-vnf-policy achieves approximately 50% lower resource utilization compared to without-vnf-policy. These results demonstrate that with-vnf-policy, which prioritizes selecting existing VNF instances with high resource utilization, can reduce deployment cost and utilize system resources effectively.

We also study the impact of Gini coefficient on the performance of load balancing. We compare the PTBP-PA with Gini coefficient (with-Gini) and PTBP-PA without Gini coefficient (without-Gini).

Fig.7 illustrates the maximum resource utilization of servers and links in MEC network for with-Gini and without-Gini under different numbers of user requests. From Fig.7a, it can be observed that the maximum server resource utilization of with-Gini is lower than that of without-Gini, and the difference increases with the number of user requests. When processing 100 consecutive user requests, without-Gini has approximately 30% higher maximum resource utilization compared to with-Gini. Fig.7b demonstrates that the maximum link resource utilization of with-Gini remains below 0.15, on average approximately 58% lower than that of without-Gini. The simulation results indicate that the introduction of Gini coefficient helps effectively utilize the resources of servers and links while achieving a balanced system load in MEC network.

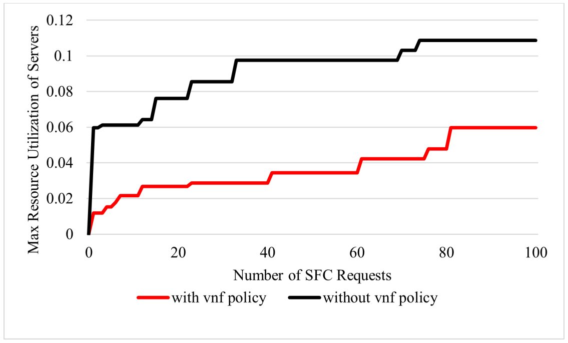


(a) Accumulated delay

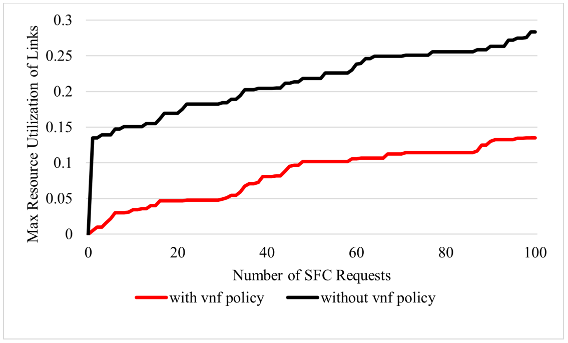


(b) Accumulated cost

Fig.5. Comparison of Accumulated Delay and Cost with and without VNF selection policy

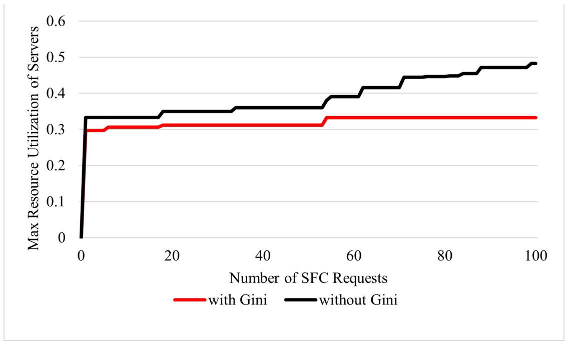


(a) Max Resource Utilization of Servers

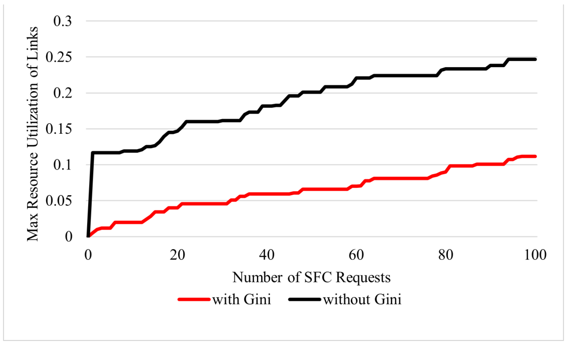


(b) Max Resource Utilization of Links

Fig.6. Comparison of Load Balancing with and without VNF selection policy



(a) Max Resource Utilization of Servers



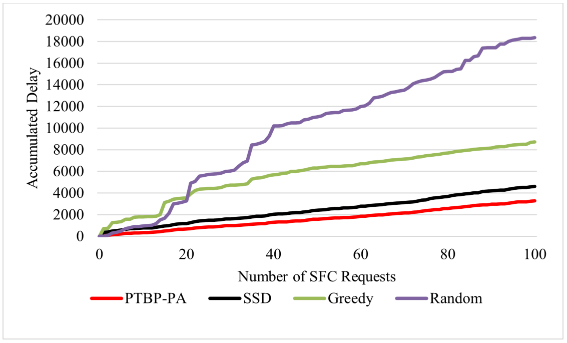
(b) Max Resource Utilization of Links

Fig.7. Comparison of Load Balancing with and without Gini

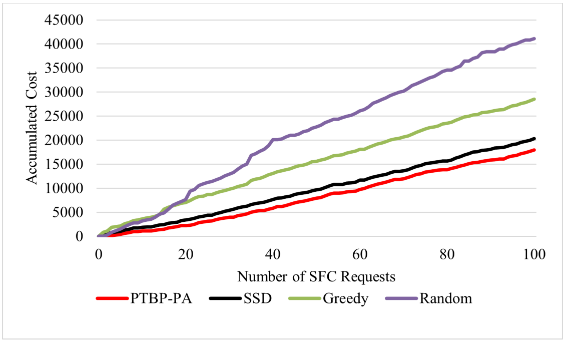
Comparison with baseline Algorithms

In this section, we compare the performance of PTBP-PA with SSD algorithm, greedy algorithm, and random algorithm in terms of accumulated delay, accumulated cost, maximum resource utilization of servers, maximum resource utilization of links, and running time. For simplicity, we will not differentiate the impact of service delay and deployment cost on SFC placement and accordingly α is set to 0.5.

PTBP-PA achieves the lowest accumulated delay, approximately 30% lower than SSD algorithm in Fig.8a. It also achieves the lowest accumulated cost, approximately 12% lower than SSD algorithm in Fig.8b. This is due to PTBP-PA's dynamic adjustment of placement strategies in the changing network, while SSD algorithm lacks this capability. The random algorithm performs poorly, and although the greedy algorithm selects strategies with lowest utility at each step, it lacks global optimization and long-term performance. The simulation results show that PTBP-PA is more effective in reducing service delay and deployment cost during SFC placement.



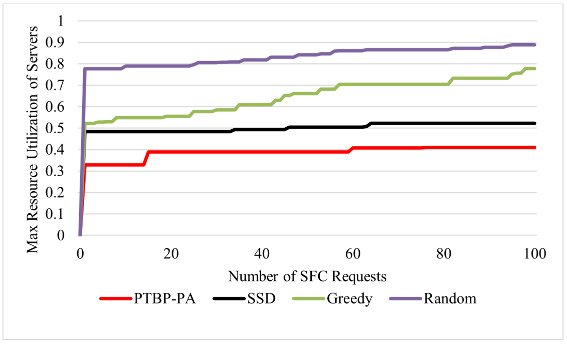
(a) Accumulated delay



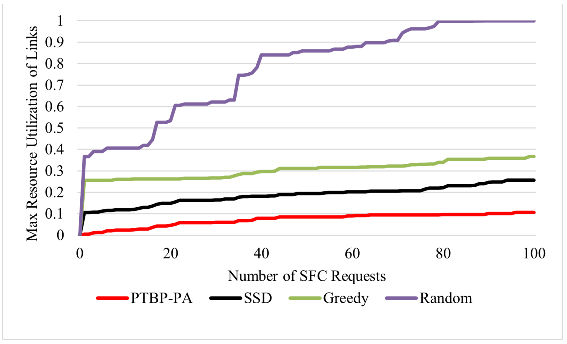
(b) Accumulated cost

Fig.8. Comparison of Accumulated delay and Cost

Fig.9 compares the maximum server and link resource utilization of four algorithms as the number of user requests increases. Fig.9a shows that PTBP-PA achieves the lowest server resource utilization, consistently below 0.5. The maximum server resource utilization of SSD fluctuates within a small range of 0.45 to 0.55, averaging approximately 0.1 higher than PTBP-PA. The greedy algorithm and random algorithm perform poorly due to their lack of global optimization. From Fig.9b, PTBP-PA maintains the lowest values below 0.15, which are approximately 0.11 lower than that of SSD, while the maximum link resource utilization of greedy algorithm is approximately 0.23 higher than that of PTBP-PA. The random algorithm has the highest maximum link resource utilization among the four algorithms, reaching 1 when deploying 80 user requests. The simulation results demonstrate that PTBP-PA can effectively balance the system load in MEC network.



(a) Max Resource Utilization of Servers



(b) Max Resource Utilization of Links

Fig.9. Comparison of Max Resource Utilization of Servers and Links

Fig.10 compares the running time of four algorithms when deploying different numbers of user requests. From Fig.10, we can see that the greedy algorithm and random algorithm have relatively low running time due to their simple algorithm steps. In contrast, the running time of PTBP-PA and SSD algorithm increases with the number of user requests. However, PTBP-PA has a lower running time compared to SSD algorithm, averaging about 12% lower. This is because PTBP-PA avoids duplicate exploration of placement strategies on the same sub paths by using the path tree structure, which can reduce the overall running time of the algorithm.

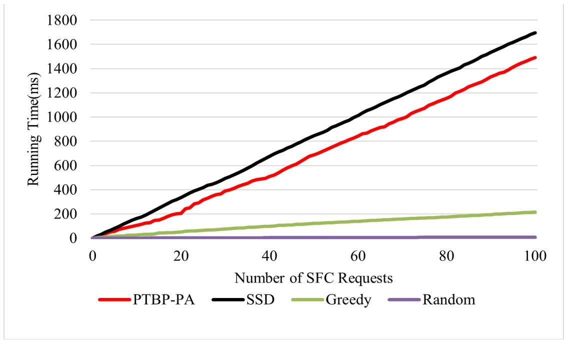


Fig.10. Comparison of Running Time

6 Conclusion

In this paper, we have targeted optimization of SFC placement in NFV-enabled MEC network. We model the service delay and deployment cost and introduce the Gini coefficient to measure the load balancing in MEC network. Our model aims to comprehensively reduce user service delay and deployment cost while ensuring load balancing of MEC network. Based on this objective, we formulate the Joint Delay-Cost Optimization problem for SFC Placement with Shareable VNF instances in MEC and name it as JDCOP-S. To address this problem, we design a Path Tree and Backtracking Pruning based Placement Algorithm (PTBP-PA), which transforms the MEC network topology into a path tree structure and utilizes backtracking pruning technique to search the placement strategy with lowest utility. Simulation results demonstrate that our proposed PTBP-PA performs well in terms of service delay, deployment cost, and system load balancing. Future research work will include the SFC placement in heterogeneous scenarios when user requests arrive randomly in the system, and the exploration of service reliability in MEC network.

Reference

[1] Arisdakessian, S., Wahab, O.A., Mourad, A., Otrok, H., Kara, N.: Fogmatch: An intelligent multi-criteria iot-fog scheduling approach using game theory. IEEE/ACM Transactions on Networking **28**(4), 1779–1789 (2020). https://doi.org/10.1109/TNET.2020.2994015

[2] Bandi, A., Shankar, M.R.B., Chatzinotas, S., Ottersten, B.: Joint multislot scheduling and precoding for unicast and multicast scenarios in multiuser miso systems. IEEE Transactions on Wireless Communications **21**(7), 5004–5018 (2022). https://doi.org/10.1109/TWC.2021.3135701

[3] Cabrera, C., Svorobej, S., Palade, A., Kazmi, A., Clarke, S.: Maaco: A dynamic service placement model for smart cities. IEEE Transactions on Services Computing **16**(1), 424–437 (2023). https://doi.org/10.1109/TSC.2022.3143029

[4] Castanho, M.S., Dominicini, C.K., Martinello, M., Vieira, M.A.M.: Chaining-box: A transparent service function chaining architecture leveraging bpf. IEEE Transactions on Network and Service Management **19**(1), 497–509 (2022). https://doi.org/10.1109/TNSM.2021.3122135

[5] Chen, L., Shen, C., Zhou, P., Xu, J.: Collaborative service placement for edge computing in dense small cell networks. IEEE Transactions on Mobile Computing **20**(2), 377–390 (2021). https://doi.org/10.1109/TMC.2019.2945956

[6] Chen, Y., Zhang, S., Jin, Y., Qian, Z., Xiao, M., Ge, J., Lu, S.: Locus: User-perceived delay-aware service placement and user allocation in mec environment. IEEE Transactions on Parallel and Distributed Systems **33**(7), 1581–1592 (2022). https://doi.org/10.1109/TPDS.2021.3119948

[7] Doan, T.V., Nguyen, G.T., Reisslein, M., Fitzek, F.H.P.: Sap: Subchain-aware nfv service placement in mobile edge cloud. IEEE Transactions on Network and Service Management **20**(1), 319–341 (2023). https://doi.org/10.1109/TNSM.2022.3201388

[8] Gao, B., Zhou, Z., Liu, F., Xu, F., Li, B.: An online framework for joint network selection and service placement in mobile edge computing. IEEE Transactions on Mobile Computing **21**(11), 3836–3851 (2022). https://doi.org/10.1109/TMC.2021.3064847

[9] Li, J., Liang, W., Ma, Y.: Robust service provisioning with service function chain requirements in mobile edge computing. IEEE Transactions on Network and Service Management **18**(2), 2138–2153 (2021). https://doi.org/10.1109/TNSM.2021.3062650

[10] Li, J., Liang, W., Xu, W., Xu, Z., Jia, X., Zhou, W., Zhao, J.: Maximizing user service satisfaction for delay-sensitive iot applications in edge computing. IEEE Transactions on Parallel and Distributed Systems **33**(5), 1199–1212 (2022). https://doi.org/10.1109/TPDS.2021.3107137

[11] Liang, W., Ma, Y., Xu, W., Xu, Z., Jia, X., Zhou, W.: Request reliability augmentation with service function chain requirements in mobile edge computing. IEEE Transactions on Mobile Computing **21**(12), 4541–4554 (2022). https://doi.org/10.1109/TMC.2021.3081681

[12] Liu, Y., Shang, X., Yang, Y.: Joint sfc deployment and resource management in heterogeneous edge for latency minimization. IEEE Transactions on Parallel and Distributed Systems **32**(8), 2131–2143 (2021). https://doi.org/10.1109/TPDS.2021.3062341

[13] Ma, Y., Liang, W., Wu, J., Xu, Z.: Throughput maximization of nfv-enabled multicasting in mobile edge cloud networks. IEEE Transactions on Parallel and Distributed Systems **31**(2), 393–407 (2020). https://doi.org/10.1109/TPDS.2019.2937524

[14] Pei, J., Hong, P., Xue, K., Li, D.: Resource aware routing for service function chains in sdn and nfv-enabled network. IEEE Transactions on Services Computing **14**(4), 985–997 (2021). https://doi.org/10.1109/TSC.2018.2849712

[15] Poularakis, K., Llorca, J., Tulino, A.M., Taylor, I., Tassiulas, L.: Service placement and request routing in mec networks with storage, computation, and communication constraints. IEEE/ACM Transactions on Networking **28**(3), 1047–1060 (2020). https://doi.org/10.1109/TNET.2020.2980175

[16] Zaw, C.W., Tran, N.H., Han, Z., Hong, C.S.: Radio and computing resource allocation in co-located edge computing: A generalized nash equilibrium model. IEEE Transactions on Mobile Computing **22**(4), 2340–2352 (2023). https://doi.org/10.1109/TMC.2021.3120520