Computer Networks 211 (2022) 109010

Contents lists available at ScienceDirect

Computer Networks

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

Efficient instance reuse approach for service function chain placement in mobile edge computing

Songli Zhang [a,*], Weijia Jia [b,c,**], Zhiqing Tang [b,a], Jiong Lou [a,b], Wei Zhao [d]

[a] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, PR China
[b] BNU-UIC Institute of Artificial Intelligence and Future Networks Beijing Normal University (BNU Zhuhai), Guangdong, 519087, PR China
[c] Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai, Guangdong, 519087, PR China
[d] Shenzhen Institute of Advanced Technology, Chinese Academy of Science, Shenzhen, Guangdong, 518000, PR China

ARTICLE INFO

Keywords:
Mobile edge computing
Service function chain
Instance reuse

ABSTRACT

The combination of mobile edge computing and network function virtualization has led to the emergence of Virtualized Network Function (VNF) in a broader range of application scenarios. These latency-sensitive and highly dynamic services can be provided by combining multiple VNFs into Service Function Chains (SFCs). However, existing work has conspicuously neglected that online placing SFC with instance reuse can significantly improve resource utilization and save initialization time, which requires considering both the dynamic distribution of required VNFs over time and resource constraints on the edge network. In this paper, we initiate the study of Online SFC placement combined with Instance Reuse. An OSIR algorithm is proposed to gain a tradeoff between service costs and users' quality of experience. The OSIR is designed based on deep reinforcement learning, which improves the system performance by maximizing the long-term cumulative reward. In OSIR, an SFC queue network is designed to extract the dynamic distribution of required VNFs over time, composed of memory space and the long short-term memory learning approach. The experimental results with real-world data traces show that OSIR can efficiently and effectively improve system performance and outperform the best result of all existing algorithms ranging from 17% to 26%.

1. Introduction

In recent years, Mobile Edge Computing (MEC) [1] is introduced to meet the growing demand of computation-intensive and latency-sensitive services from mobile users [2]. Compared with cloud computing, MEC can significantly reduce the end-to-end latency for mobile users by deploying edge nodes close to mobile users at the network edge [3]. However, the heavy reliance on customized hardware severely hinders the development of MEC [4,5]. In order to provide flexible services in MEC, Network Function Virtualization (NFV) has been widely advocated by Service Providers (SPs) [6]. NFV transforms heavy hardware middleboxes (e.g., firewall, encryption, and load balancer) into a set of light software-based Virtual Network Functions (VNFs). And VNF instances can be hosted in virtual environment, e.g., container [7]. SPs can provide and update their service flexibly by deploying VNF instances in edge networks while decreasing OPerating EXpenditures (OPEX) and CAPital EXpenditures (CAPEX) [8]. Multiple VNFs can be composed as Service Function Chains (SFCs) to provide complex services [9].

However, facing the limited resources in edge nodes and the urgent demand to gain a tradeoff between service costs and users' Quality of Experience (QoE), SPs eagerly call for an intelligent SFC placement approach for these dynamic services to obtain better network performance and resource utilization [10,11]. Traditional SFC placement approaches usually adopt heuristic solutions, assuming that they can obtain abundant prior knowledge or predict the network environment accurately [12]. Multiple time slots are divided and scheduling postponed in the general scheduling research to avoid this shortage [13,14]. However, these assumptions and approaches sacrifice the flexibility of NFV. Moreover, most of the existing studies on SFC placement ignore that VNF instance can be reused. Some recent frameworks in the industry make it possible to share the initialized VNF instances among different services [15,16], which can effectively save the initialization time and improve the resource utilization of edge nodes. Existing preliminary instance reuse algorithms ignore that the instance distribution significantly affects the routing path of future SFC placement [17,18].

* Corresponding author.
** Corresponding author at: Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai, Guangdong, 519087, PR China.
E-mail addresses: zhang_sl@sjtu.edu.cn (S. Zhang), jiawj@uic.edu.cn (W. Jia), domain@sjtu.edu.cn (Z. Tang), lj1994@sjtu.edu.cn (J. Lou), wzhao@aus.edu (W. Zhao).

https://doi.org/10.1016/j.comnet.2022.109010
Received 10 June 2021; Received in revised form 13 January 2022; Accepted 24 April 2022
Available online 6 May 2022

To the best of our knowledge, our work is the first attempt to solve the problem of online SFC placement with instance reuse. There are three major challenges lying in this problem. The first is how to place the entire SFC successfully in the resource-constrained edge network [19]. Resource capacities of both node and link need to be considered simultaneously in SFC placement. Moreover, the SFC request is usually an ordered list of multiple VNFs. The SFC request fails to be executed if one or more required VNFs are not allocated due to insufficient computation resources or communication resources. Thus, VNF instance reuse is raised to improve resource utilization effectively [16]. However, instance reuse brings more complex constraints, which making the SFC placement more complex.

The second challenge is how to reuse the initialized instance among SFC requests intelligently. VNF instance reuse may prevent SFC requests from being deployed on the shortest path, incurring high forwarding costs and low users' QoE. Besides, the long routing path can consume more link resources, making the edge network unable to accept more requests. Therefore, in order to reuse VNF instances among SFC requests efficiently, the instance deployment may not be optimal for the current SFC request. Proper reusing needs to consider two problems: (1) Whether to reuse the initialized instance; (2) Which initialized instance to be chosen to reuse. Moreover, the dynamic network environment and user requests make these problems more challenging.

Last but not least, the third challenge lies in how to make the online decision of SFC placement based on the dynamic network environment and user requests. In the problem of online SFC placement with instance reuse, every online decision changes the distribution of VNF instances, which can affect the routing path of future SFC requests and further result in different placement costs. It is difficult to predict the future required VNFs directly to place the instance on the node to make more SFC requests reusing efficiently, which sets up obstacles for obtaining the long-term cumulative reward.

In this paper, by jointly considering the above three challenges, the OSIR algorithm is proposed to solve the problem of Online SFC placement with Instance Reuse, aiming to (1) minimize the resource consumption of nodes and links for SPs and (2) maximize the QoE of mobile users. QoE is described by the throughput of accepted requests and the transmission delay of users. Moreover, the problem is reformulated as a Markov Decision Process (MDP). The state includes not only the edge network information and the request characters at the current time, but also the distribution of required VNFs of multiple past SFC requests. Then, the SFC Queue Network (SQN) composed of an SQN memory and the Long Short-Term Memory (LSTM) is designed to extract the distribution of required VNFs over time. A customized reward function is designed to teach the agent to place the full SFC through immediate rewards, which is calculated based on the partial throughput and the current cost after placing each VNF successfully. If some VNFs of the SFC are not placed, a large negative reward is given, and a retrace mechanism is designed to backtrack the environment. Finally, OSIR is designed based on Asynchronous Advantage Actor–Critic (A3C) to obtain the long-term cumulative reward.

We summarize our key contributions as follows:

- This work is the first to model and solve the problem of online SFC placement with instance reuse, aiming to minimize the resource consumption and maximize the QoE. The problem is modeled by a set of constraints and is further formulated as an MDP.
- To solve this problem, an A3C-based algorithm named OSIR is proposed, where an SQN is devised to extract the distribution of required VNFs over time. The reward function of OSIR is designed by immediate rewards based on the model and our optimization objective. A retrace mechanism is applied to deal with the failed placement.

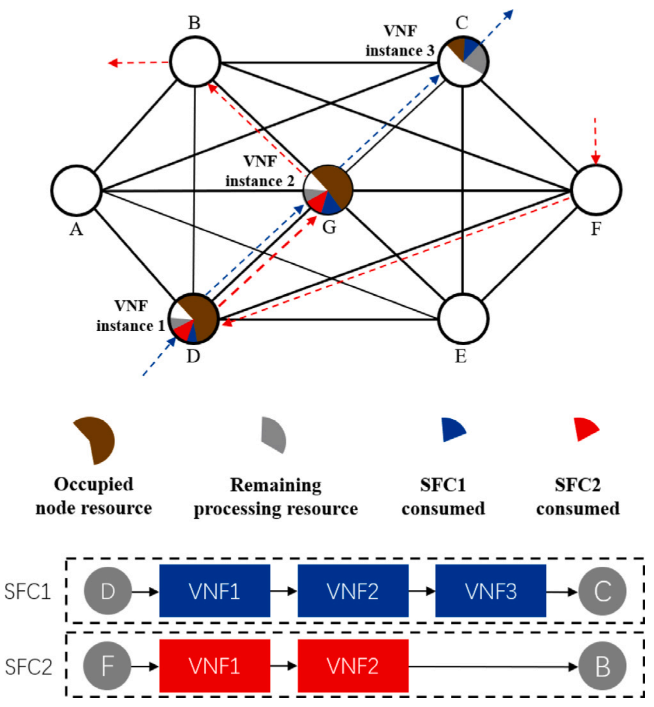


Fig. 1. A case of VNF instance reuse.

- The experimental results with real-world data traces show that OSIR can efficiently and effectively obtain a solution by scheduling SFC requests and reusing VNF instances intelligently. Specifically, OSIR outperforms the best result of existing algorithms ranging from 17% to 26%.

The remainder of this paper is organized as follows. In Section 2, we give a brief background introduction of related work. Section 3 describes the system model and our problem formulation. Section 4 presents our algorithms and solutions to this type of problem. Numerical results and evaluation are presented in Section 5. Finally, Section 6 presents our discussions and Section 7 concludes the paper.

2. Background and related work

In this section, a case of VNF instance reuse is first illustrated. Then, existing research about the SFC placement problem is summarized. Finally, the application of deep reinforcement learning (DRL) in MEC is introduced.

2.1. Instance reuse

In practice, since SFC is composed of multiple VNFs, the same VNFs may exist in multiple SFC requests. These same VNFs usually perform the same or similar processing steps on the same packet [16]. When a VNF instance still has remaining processing capacity, it can be shared by several SFC requests to avoid initializing a new VNF instance and save the node resource. Reusing these instances between different SFCs efficiently can significantly improve system performance, which is regarded as a typical way of computation acceleration [18].

Fig. 1 shows a case of VNF instance reuse. When two SFC requests (SFC1 and SFC2) come in order, SFC1 is placed in the edge network along with Path1 = D → G → C. SFC1 consumes the memory and computing resources of nodes to initialize three instances (VNF instances 1, 2, and 3) and the outbound bandwidth of the nodes D and G. The delay of SFC1 can be calculated along Path1. For SFC2, although there is a direct Path2 = F → B from source F to destination B, there are

Table 1
Notations.

Symbol	Meaning
$\mathbf{G_e} = (\mathbf{N_e}, \mathbf{L_e})$	Edge network
$\mathbf{U}$	Set of SFC requests
$\mathbf{F}$	Set of VNF
$\mathbf{N_u}$	Ordered list of VNFs in SFC request u
$\mathbf{O_u} = \{o_f^u \mid f \in \mathbf{F}\}$	Required processing resource for each type VNF instance of SFC u
c_i^{mem}	Memory capacity of node n_i
c_i^{com}	Computing capacity of node n_i
c_i^{bw}	Outbound bandwidth capacity of node n_i
c_f	Processing capacity of type-f VNF instance
m_f^{mem}	Memory cost of type-f VNF instance
m_f^{com}	Computing cost of type-f VNF instance
$d_{i,i'}$	Transmission delay of link $l_{i,i'}$
p_i^f	Number of type-f instances placed on n_i
y_u	Traffic of SFC request u
ϕ_u	Source of SFC request u
ψ_u	Destination of SFC request u
Δ_u	Time to live of SFC request u
k_t^u	1 if SFC request u running at time t, 0 otherwise
$b_{u,j}^i$	1 if f_j^u placed on the node n_i, 0 otherwise
x_f^u	1 if initializing a new type-f VNF instance when placing the SFC request u, 0 otherwise
μ_u	Throughput of SFC request u
W_{TP}	Total throughput
W_{DC}	Total delay cost
W_{RC}	Total resources cost
$\overline{c_i^{mem}}$	Remaining memory resources of n_i
$\overline{c_i^{com}}$	Remaining computing resources of n_i
$\overline{c_i^{bw}}$	Remaining outbound bandwidth of n_i
$\overline{c_i^{f}}$	Remaining processing resource of type-f VNF instance on node n_i
$\xi_{bw}, \xi_{mem}, \xi_{com}$	Weights of bandwidth, memory and computing resource
S_τ, A_τ, R_τ	State, action, reward in DRL

no already initialized VNF instances 1 and 2 on Path2. If SPs choose this shortest path, they need to initialize new VNF instances 1 and 2 on Path2.

In fact, there is still remaining processing capacity in VNF instances 1 and 2 on nodes D and G, respectively. If SFC2 is placed along Path3 (Path3 = F → D → G → B) and reuses the instances initialized by SFC1, the node resource consumption caused by initializing new instances can be avoided. However, Path3 passes through more nodes than Path2, which generally results in higher transmission delay and more consumption of outbound bandwidth. When the SFC requests are placed online, the impact between the SFC request placement is more complex. How to deploy VNF instances on critical nodes or paths to make more SFC request reusing efficiently is challenging and a critical problem need to be solved.

2.2. SFC placement

In NFV, effectively allocating network resources to improve Quality of Service (QoS) is a significant problem, which has been proved NP-hard [20] and can be classified into four typical categories: (1) the SFC placement problem; (2) the traffic routing problem in NFV; (3) joint the SFC placement and the traffic routing problem; (4) the VNF redeployment and consolidation problem [21]. In this paper, our research is based on the SFC placement problem.

In the literature, some studies construct the special model for their problem and then propose corresponding heuristic algorithms to solve the SFC placement problem according to different QoS parameters (e.g., service cost, delay, stability) [22–25]. These problems and algorithms suppose SPs to have a good prior knowledge of the global environment. Chen et al. [26] and Zhang et al. [27] model the SFC request according to M/M/1 queue and design heuristic algorithms to solve the latency-aware SFC placement problem. However, since SFC requests usually contain different VNFs, the model based on queuing theory is difficult to describe the fine-grained differences in SFC requests. More importantly, instance reuse is an effective method to save initialization time and improve node resource utilization, which has been proved and achieved in the industry [16,28].

Although Guo et al. [18] reduce service costs by sharing the same VNF, they neglect the high delay and QoE decline caused by path extension. Reuse is proposed to improve the resource utilization of edge servers and physical links [17], and a heuristic algorithm is designed to solve this problem while providing latency guarantees. Although they model the SFC request with queuing theory, the difference of different SFC requests is not only reflected in time. The characters of past SFC requests (e.g., the type of SFC, the type of VNF required, and the traffic) have a serious impact on deployment policy. More importantly, Jin et al. [17] search all the possible paths in their first step. For dense graphs, this dramatically increases the execution time and is difficult to adapt to the actual large-scale network. More importantly, they all focus on instance reuse when placing a single SFC each, ignoring that multiple SFC requests make the problem more complex.

2.3. Reinforcement learning in MEC

DRL [29] is regarded as a novel method to solve various online problems and challenges, which has been applied in the Internet of Things, video caching, and Unmanned Aerial Vehicle, etc. [30,31]. By training an agent to interact with the environment, the DRL agent learns a strategy to maximize the long-term cumulative reward. In related research about resource scheduling, DRL has proved its strong ability to make continuous online decision [32–34].

Moreover, recent SFC placement solutions have been combined with reinforcement learning. Gu et al. [35] propose a model-assisted approach based on DRL to place SFC requests to the edge network. Sun et al. [36] learn the current state of the network through Q-learning and propose a dynamic SFC placement algorithm. Their state setting in DRL focuses on modeling the edge network while ignoring the improvement brought by adequate SFC requests modeling. Zheng et al. [37] design a heuristic algorithm and a DRL algorithm to solve the SFC placement problem offline and online, respectively. However, the distribution of

SFC requests over time is ignored. Solozabal et al. [11] propose an encoder–decoder architecture with a Bahdanau attentional mechanism to capture the distribution of SFC requests over time, but they make strong assumptions about the network topology, which cannot adapt to a wide range of edge systems. In addition, all of them neglect the potential of instance reuse in SFC placement.

3. System model

In this section, the NFV system and SFC request are first described. Then, the problem is formulated with a set of linear constraints and optimization objectives. To focus on the problem itself, the term instance is used to refer to any virtual environment where VNF can be hosted. The important notations are summarized in Table 1.

3.1. System overview

In this paper, an online service scenario is considered, where the mobile users' SFC requests are handled by VNF instances deployed in an edge network. Further, we focus on solving instance reuse in a typical SFC placement problem without considering traffic routing [21], which is both NP-hard and needs an online decision solution.

The edge network consists of a set of edge nodes. There exists an undirected link between each node pair. $\mathbf{U}$ is the set of all users' requests, and $u \in \mathbf{U}$ is a SFC request. Each SFC contains multiple VNFs and is an ordered list. For each u, SPs need to decide how to place it in the current edge network.

3.1.1. VNF

$\mathbf{F}$ is the set of different types of VNFs. $|\cdot|$ is used to represent the number of elements in the set, e.g., $|\mathbf{F}|$ is the number of VNF type. Each $f \in \mathbf{F}$ is a type of VNF, and there are memory and computing resource cost for each type-f VNF instance, denoted by m_f^{mem} and m_f^{com}, respectively.

Initializing a new type-f VNF instance will cause a start-up delay d_f. Then, a type-f VNF instance can provide a fixed processing capacity c_f, which is determined by factors such as the actual deployment method and the VNF type. An instance can be reused only when it has the remaining processing capacity. Maintaining an empty instance can only waste the resources of edge nodes and lead to low resource utilization. Thus, the service providers always choose to withdraw an instance after all requests in it are completed. Since instances are deployed in the edge node by virtual environments such as VMs or containers, withdrawing an instance only needs to terminate the corresponding software process. Thus, when there is no VNF running in the instance, the instance can be released immediately, which is almost at no cost.

3.1.2. Edge network

The edge network is modeled as an undirected graph $\mathbf{G_e} = (\mathbf{N_e}, \mathbf{L_e})$, where $\mathbf{N_e}$ is the set of nodes and $\mathbf{L_e}$ is the set of links that connect every two nodes. Each $n_i \in \mathbf{N_e}$ represents the edge node and $l_{i,i'} \in \mathbf{L_e}$ represents the link from node n_i to node $n_{i'}$. In the edge network, each node n_i has a memory capacity c_i^{mem}, a computing capacity c_i^{com} and a outbound bandwidth c_i^{bw}. p_i^f represents the number of type-f instances that is placed on node n_i. Finally, from node n_i to node $n_{i'}$, the transmission delay is defined as $d_{i,i'}$.

3.1.3. SFC request

In general, various requests are generated from different users over time. For each request u, its features can be denoted as a tuple ($\mathbf{N_u}$, $\mathbf{O_u}$, ϕ_u, ψ_u, y_u, Δ_u). $\mathbf{N_u} = [f_1^u, f_2^u, \dots, f_j^u, \dots, f_{|\mathbf{N_u}|}^u]$ represents the ordered list of VNFs in SFC request u, where f_j^u is the jth VNF in SFC request u and $f_j^u \in \mathbf{F}$. Thus, the chain length of u is $|\mathbf{N_u}|$. Besides, $\mathbf{O_u} = \{o_f^u | f \in \mathbf{F}\}$ is introduced to represent the required processing resources of various type VNFs of SFC request u, where o_f^u is the required processing resource of VNF f and:

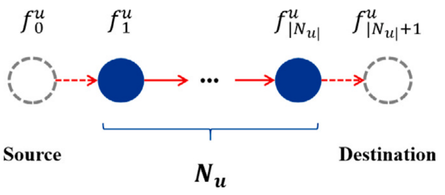


Fig. 2. Adding virtual nodes.

$$o_f^u \begin{cases} > 0, & f \in \mathbf{N_u} \\ = 0, & f \notin \mathbf{N_u} \end{cases}. \tag{1}$$

Besides, $\phi_u \in \mathbf{N_e}$, $\psi_u \in \mathbf{N_e}$ and y_u in the tuple are source, destination and traffic of request u, respectively. Considering the geographical distribution of mobile users and edge nodes, and the limited coverage of edge nodes, each user request can only be accessed from a specific source node and exited from a specific destination node [38,39]. In this way, the model of the geographical distribution of mobile users and nodes can be simplified [40,41]. The time to live (TTL) of u is denoted by Δ_u, which is calculated based on start time and end time.

Finally, a variable $b_{u,j}^i \in \{0,1\}$ is introduced. When the function f_j^u is placed on the node n_i in the edge network, $b_{u,j}^i = 1$, 0 otherwise. k_t^u is introduced to check whether request u exists at the time point t. If the SFC request u is running, $k_t^u = 1$, 0 otherwise.

To simplify the model, two virtual nodes are added to the head and tail of the SFC request u. As shown in Fig. 2, f_0^u and $f_{|\mathbf{N_u}|+1}^u$ are assumed to be placed on nodes ϕ_u and ψ_u, respectively:

$$b_{u,0}^i = \begin{cases} 1, & n_i = \phi_u \\ 0, & n_i \neq \phi_u \end{cases}, \tag{2}$$

$$b_{u,|\mathbf{N_u}|+1}^i = \begin{cases} 1, & n_i = \psi_u \\ 0, & n_i \neq \psi_u \end{cases}. \tag{3}$$

The virtual nodes occupy no resources.

3.2. Problem formulation

The problem of online SFC placement with instance reuse is described by a set of linear constraints.

Firstly, in order to place an entire SFC request, each VNF in the SFC needs to be placed. z_u is used to check whether the SFC request u is accepted:

$$z_u = \mathbf{1}_{\{\sum_{f_j^u \in \mathbf{N_u}} \sum_{n_i \in \mathbf{N_e}} b_{u,j}^i\}}, \tag{4}$$

where $\mathbf{1}_{\{\lambda\}}$ is the indicator function such that $\mathbf{1}_{\{\lambda\}} = 1$ if $\lambda > 0$ and 0 otherwise. If SFC request u is accepted, $z_u = 1$, 0 otherwise. When a VNF in the SFC is not placed on the edge network, other VNFs should be rejected. To satisfy this requirement, the first constraint is:

$$|\mathbf{N_u}| \times z_u = \sum_{f_j^u \in \mathbf{N_u}} \sum_{n_i \in \mathbf{N_e}} b_{u,j}^i, \ \forall u \in \mathbf{U}. \tag{5}$$

When the SFC request u is accepted, each VNF in u should only be served on one edge node, denoted by:

$$\sum_{n_i \in \mathbf{N_e}} b_{u,j}^i = \mathbf{1}_{\{\sum_{n_i \in \mathbf{N_e}} b_{u,j}^i\}}, \ \forall u \in \mathbf{U}, \forall f_j^u \in \mathbf{N_u}. \tag{6}$$

Then, it need to be ensured that the memory and computing resource at edge node n_i cannot be overbooked. Thus, the memory and computing resource constraints can be described as:

$$\sum_{f \in F} p_i^f \times m_f^{mem} \le c_i^{mem}, \quad \forall n_i \in \mathbf{N_e}, \tag{7}$$

$$\sum_{f\in F} p_i^f \times m_f^{com} \leq c_i^{com}, \quad \forall n_i \in \mathbf{N_e}. \tag{8}$$

Similarly, for each node n_i in the edge network, the total traffic of all SFCs passing through it must not exceed the outbound bandwidth capacity. The bandwidth is consumed only when the next following VNF is placed on the different edge node. Thus, the bandwidth constraint can be described as:

$$\sum_{u\in\mathbf{U}}\sum_{j=0}^{|\mathbf{N_u}|} b_{u,j}^i \times (b_{u,j}^i - b_{u,j+1}^i) \times y_u \times k_t^u \times z_u \leq c_i^{bw}, \quad \forall n_i \in \mathbf{N_e}. \tag{9}$$

Finally, all initialized instances can be shared by different SFC requests to make full use of their remaining processing capacities. The total processing capacity of VNF f on node n_i is determined by the number of type-f VNF instances placed on it. Therefore, to guarantee the service demand of each type VNF, processing capacity constraints can be expressed as:

$$\sum_{u\in\mathbf{U}} o_f^u \times k_t^u \times z_u \leq c_f \times p_i^f, \quad \forall n_i \in \mathbf{N_e}, \forall f \in \mathbf{F}. \tag{10}$$

Our goal is to efficiently utilize the edge network's resources and improve QoE by intelligently placing SFC requests over time. Generally, efficiently utilizing edge resources is transformed to minimize resource cost [17]. The QoE is determined by the throughput and total transmission delay of accepted requests. The throughput value of the SFC request u is calculated based on the traffic, SFC chain length and TTL, denoted by:

$$\mu_u = \xi_v \times y_u \times |\mathbf{N_u}| \times \Delta_u + v_{lb}, \tag{11}$$

where ξ_v is the constant weight and v_{lb} is the lower bound of throughput value. Also, the total throughput W_{TP} can be calculated as:

$$W_{TP} = \sum_{u\in\mathbf{U}} \mu_u \times z_u \tag{12}$$

Besides, if a new type-f instance is initialized when placing the SFC request u, $x_f^u = 1$, otherwise $x_f^u = 0$. To effectively utilize both the node and link resources of the edge network, the total edge network resources cost W_{RC} is defined as:

$$\begin{aligned} W_{RC} = &\xi_{bw} \sum_{u\in\mathbf{U}}\sum_{j=0}^{|\mathbf{N_u}|}\sum_{n_i\in\mathbf{N_e}} b_{u,j}^i \times (b_{u,j}^i - b_{u,j+1}^i) \times y_u \times z_u \\ &+ \xi_{mem}\sum_{u\in\mathbf{U}}\sum_{f\in F} x_f^u \times m_f^{mem} + \xi_{com}\sum_{u\in\mathbf{U}}\sum_{f\in F} x_f^u \times m_f^{com}, \end{aligned} \tag{13}$$

where ξ_{bw}, ξ_{mem} and ξ_{com} are the weights of bandwidth, memory and computing resource, respectively.

Finally, the delay cost W_{DC} of all requests u flowing through the edge network can be calculated as:

$$W_{DC} = \sum_{u\in\mathbf{U}}\sum_{j=0}^{|\mathbf{N_u}|}\sum_{n_i\in\mathbf{N_e}}\sum_{n_{i'}\in\mathbf{N_e}} b_{u,j}^i \times b_{u,j+1}^{i'} \times d_{i,i'} \times z_u + \sum_{u\in\mathbf{U}}\sum_{f\in F} x_f^u \times d_f. \tag{14}$$

Summing up all the issues, online SFC placement with instance reuse can be formulated as follows:

Problem 1.

$$\begin{aligned} \max \quad & \xi_{TP} \times W_{TP} - \xi_{RC} \times W_{RC} - \xi_{DC} \times W_{DC}, \\ s.t. \quad & (1) - (14) \quad . \end{aligned} \tag{15}$$

where ξ_{TP}, ξ_{RC} and ξ_{DC} are the weights of the throughput, resource and delay, respectively.

Analysis: Problem 1 is classified into a typical constrained offloading decision problem. Because several non-linear constraints exist in Problem 1, it is difficult to directly solve Problem 1 even for each single SFC. Traditional optimization algorithms constantly adjust the offloading strategies by repeatedly iterating, trying to find the optimal solution. However, due to the increase of the node number in edge network, these optimization algorithms based on iteration consume lengthy execution time. At the same time, due to the system dynamics and the difficulty of predicting the future environment accurately, the traditional heuristic algorithm cannot maximize the long-term reward.

In our problem, the arrival of requests and the update of the edge network have no memory and satisfy the first-order Markov property [30]. Therefore, it can be modeled as an MDP. In order to solve the MDP, in the process of designing the state space, a group of LSTM is added to extract the timing sequence relationship of requests. At the same time, an online offloading algorithm is proposed based on DRL, in which the state space combines the current network and request state with the LSTM time sequence extraction result. With DRL, all linear and non-linear constraints are reflected in the reward setting with almost no distinction [35,42]. And Problem 1 can be handled with interactions among states, actions, and rewards. Furthermore, the DRL agent aims to maximize the long-term cumulative rewards, which is also the optimization objective.

4. Algorithm

In this section, an efficient online learning approach named OSIR is proposed to solve Problem 1. OSIR utilizes the LSTM-based A3C model to schedule SFC requests and reuse VNF instances intelligently. The algorithm settings are first described. Then, the SQN is introduced to extract the distribution of the required VNF over time. Finally, the training process of the OSIR agent is summarized.

4.1. Algorithm settings

The agent and the environment are two primary components in DRL. In each step, the agent observes the environment state S_τ and gives an action A_τ according to a policy. Then the environment returns a reward $R_\tau(S_\tau, A_\tau)$ and updates to the next state $S_{\tau+1}$. Finally, the agent updates the policy to obtain a higher long-term cumulative reward by ceaselessly interacting with the environment.

The state, action, reward, and policy of reinforcement learning are defined as follows.

State: The state includes all remaining resources in the edge network and the current SFC request u. The remaining memory resource $\overline{c_i^{mem}}$, computing resource $\overline{c_i^{com}}$ and outbound bandwidth $\overline{c_i^{bw}}$ of each node n_i are obtained as follows:

$$\overline{c_i^{mem}} = c_i^{mem} - \sum_{f\in F} p_i^f \times m_f^{mem}, \tag{16}$$

$$\overline{c_i^{com}} = c_i^{com} - \sum_{f\in F} p_i^f \times m_f^{com}, \tag{17}$$

$$\overline{c_i^{bw}} = c_i^{bw} - \sum_{u\in\mathbf{U}}\sum_{j=0}^{|\mathbf{N_u}|} b_{u,j}^i \times (b_{u,j}^i - b_{u,j+1}^i) \times y_u \times k_t^u. \tag{18}$$

Similarly, for each edge node n_i, its remaining processing capacity $\overline{c_i^f}$ of type-f VNF instance is denoted by:

$$\overline{c_i^f} = c_f \times p_i^f - \sum_{u\in\mathbf{U}} o_f^u \times k_t^u \times z_u. \tag{19}$$

The SFC request u has an ordered list of VNFs, i.e., $\mathbf{N_u} = [f_1^u, f_2^u, \ldots, f_j^u, \ldots, f_{|\mathbf{N_u}|}^u]$, and each $f_j^u \in \mathbf{N_u}$ is placed in order. In each step, the ordered VNF list $\mathbf{N_u}$, required processing resources $\mathbf{O_u}$, TTL Δ_u, traffic y_u, last placed node $b_{u,j-1}^i$, destination ψ_u and throughput μ_u are all in the state. $\widehat{f_j^u} = [0,1]^{|\mathbf{F}|}$ is defined as the one-hot vector of f_j^u. Thus, when deploying VNF $f_j^u \in \mathbf{N_u}$, state S_τ can be represented by:

$$\begin{aligned} S_\tau = [&\overline{c_1^{mem}}, \overline{c_2^{mem}}, \dots, \overline{c_{|N_e|}^{mem}},\\ &\overline{c_1^{com}}, \overline{c_2^{com}}, \dots, \overline{c_{|N_e|}^{com}},\\ &\overline{c_1^{bw}}, \overline{c_2^{bw}}, \dots, \overline{c_{|N_e|}^{bw}},\\ &\overline{c_1^{f_1}}, \overline{c_2^{f_1}}, \dots, \overline{c_{|N_e|}^{f_1}},\\ &\overline{c_1^{f_2}}, \overline{c_2^{f_2}}, \dots, \overline{c_{|N_e|}^{f_2}},\\ &\dots,\\ &\overline{c_1^{f_{|F|}}}, \overline{c_2^{f_{|F|}}}, \dots, \overline{c_{|N_e|}^{f_{|F|}}},\\ &o_1^u, o_2^u, \dots, o_{|F|}^u\\ &b_{u,j-1}^1, b_{u,j-1}^2, \dots, b_{u,j-1}^{|N_e|},\\ &b_{u,|\mathbf{N_u}|+1}^1, b_{u,|\mathbf{N_u}|+1}^2, \dots, b_{u,|\mathbf{N_u}|+1}^{|N_e|},\\ &\Delta_u, y_u, \mu_u, \widehat{f_j^u}] \end{aligned} \tag{20}$$

Action: For each state S_τ, the agent needs to schedule the VNF $f_j^u \in \mathbf{N_u}$ to an edge node. So the action space A_τ is defined as the set of all edge nodes:

$$A_\tau \in N_e. \tag{21}$$

Reward: When executing an action $A_\tau = n_i$, if the node n_i can provide enough resources, the VNF $f_j^u \in \mathbf{N_u}$ is placed successfully. Then, the agent gets an immediate reward R_τ. A reward memory $\mathbf{R}_u^{\mathbf{M}}$ is designed to store each immediate reward of the current SFC request u. The successful placement needs to follow these constraints below.

When executing action $A_\tau = n_i$, the remaining processing resource of type-f_j^u instance on node n_i needs to meet the processing resource demand of SFC request u, denoted by:

$$\overline{c_i^{f_j^u}} \geq o_{f_j^u}^u. \tag{22}$$

Otherwise, the agent needs to initialize a new type-f_j^u instance on node n_i to apply for more processing resources. Only when the node n_i has sufficient memory resources and computing resources, a new type-f_j^u VNF instance can be initialized on node n_i, denoted by:

$$\overline{c_i^{mem}} \geq m_{f_j^u}^{mem}, \tag{23}$$

$$\overline{c_i^{com}} \geq m_{f_j^u}^{com}, \tag{24}$$

When Eq. (23) or (24) is not satisfied, it indicates that a new type-f_j^u instance cannot be initialized on the selected node. Then, it is regarded as a placement failure.

In addition to checking whether the action A_τ follows processing capacity constraints, the bandwidth from the last placed node $A_{\tau-1}$ or source ϕ_u to A_τ needs to be checked. If $A_\tau = n_i$ and $b_{u,j-1}^i = 0$, the VNF f_j^u and f_{j-1}^u are placed on different nodes. Then the outbound bandwidth resource of the last placed node is consumed, requiring to satisfy the bandwidth constraint:

$$\overline{c_i^{bw}} \geq b_{u,j-1}^i \times (b_{u,j-1}^i - b_{u,j}^i) \times y_u, \ \forall n_i \in \mathbf{N_e}. \tag{25}$$

Otherwise, the placement is also regarded a placement failure.

When $j = \mathbf{N_u}$, all included VNF in SFC request u are placed, and the outbound bandwidth from A_τ to ψ_u needs to meet the following requirement:

$$\overline{c_i^{bw}} \geq (b_{u,j}^i - b_{u,j+1}^i) \times y_u, \ \forall n_i = A_\tau. \tag{26}$$

If the VNF f_j^u is placed successfully, the immediate reward R_τ is designed based on the optimization objective in Problem 1, which contains all three components in Problem 1:

$$R_\tau = \xi_{TP} \times R_\tau^{TP} - \xi_{RC} \times R_\tau^{RC} - \xi_{DC} \times R_\tau^{DC}. \tag{27}$$

where the throughput R_τ^{TP} is calculated as:

$$R_\tau^{TP} = \mu_u / |\mathbf{N_u}|. \tag{28}$$

Then, the resource cost R_τ^{RC} and the delay cost R_τ^{DC} are calculated as:

$$R_\tau^{RC} = \xi_{bw} \times \sum_{n_i \in \mathbf{N_e}} b_{u,j-1}^i \times (b_{u,j-1}^i - b_{u,j}^{i'}) \times y_u \tag{29}$$
$$+ \xi_{mem} \times x_{f_j^u}^u \times m_f^{mem} + \xi_{com} \times x_{f_j^u}^u \times m_f^{com},$$

$$R_\tau^{DC} = \sum_{n_i \in \mathbf{N_e}} b_{u,j-1}^i \times b_{u,j}^{i'} \times d_{i,i'} + x_{f_j^u}^u \times d_f, \tag{30}$$

where $n_{i'} = A_\tau$.

When the action $A_\tau = n_i$ is selected by the agent and the node n_i cannot satisfy the constraints of processing capacity or outbound bandwidth, the action A_τ is regarded as the terrible action. Then, a large negative value is given as the reward R_τ, which is calculated based on the previous multiple immediate rewards. By this punishment, the agent can be aware that placing an incomplete chain is useless. The reward R_τ for the terrible A_τ is calculated as follows:

$$R_\tau = -\xi_{TP} \times \mu_u - \sum_{R_j \in \mathbf{R}_u^{\mathbf{M}}} R_j. \tag{31}$$

In OSIR, all the linear and non-linear constraints in Problem 1 are reflected in the reward function. Furthermore, OSIR uses punishment to avoid choosing the action which violates constraints. For example, in the reward function, the constraints on node resources in Eq. (7) and (8) are transformed into Eq. (23) and (24), respectively.

Algorithm 1 Reward Calculation

```
Input: $S_\tau, A_\tau$
Output: $R_\tau$
1: /* SFC request $u$ */
2: if $j = 1$ then
3:     $\mathbf{R_u^M} = \emptyset$
4: end if
5: if (Eq. (22) or (Eq. (23) and Eq. (24))) and (Eq. (25)) and ($j \neq |\mathbf{N_u}|$
   or Eq. (26)) then
6:     Calculate $R_\tau$ based on Eq. (27)–(30)
7:     $\mathbf{R_u^M} = \mathbf{R_u^M} \cup \{R_\tau\}$
8:     if $j = |\mathbf{N_u}|$ then
9:         $j = j + 1$
10:        Recalculate $R_\tau^{RC'}$ and $R_\tau^{DC'}$ based on Eq. (29)–(30)
11:        $R_\tau = R_\tau + R_\tau^{RC'} + R_\tau^{DC'}$
12:    end if
13: else
14:    Calculate $R_\tau$ based on Eq. (31)
15: end if
16: end
```

Alg. 1 shows the process of reward calculation. At first, in line 3, a reward memory $\mathbf{R}_u^{\mathbf{M}}$ is initialized as an empty set. In line 5, when the action A_τ selected by the agent satisfies the resource constraints, VNF f_j^u is placed successfully. Then, the reward R_τ is calculated and added into $\mathbf{R}_u^{\mathbf{M}}$. In lines 8–12, after all $f \in \mathbf{N_u}$ placed in the edge network, the last VNF $f_{|\mathbf{N_u}|}^u$ needs to be connected to the virtual node $f_{|\mathbf{N_u}|+1}^u$ on the destination ψ_u, and the corresponding delay and resource cost are calculated. In line 14, when placement fails due to insufficient processing capacity or outbound bandwidth, the reward R_τ is recalculated.

Policy: The policy is the probability distribution of taking action A_τ in the current state S_τ, denoted by $\pi_\theta(S_\tau, A_\tau)$:

$$\pi_\theta(S_\tau, A_\tau) \in (0, 1], \tag{32}$$

$$\sum_{A'_\tau \in |\mathbf{N_e}|} \pi_\theta(S_\tau, A'_\tau) = 1, \tag{33}$$

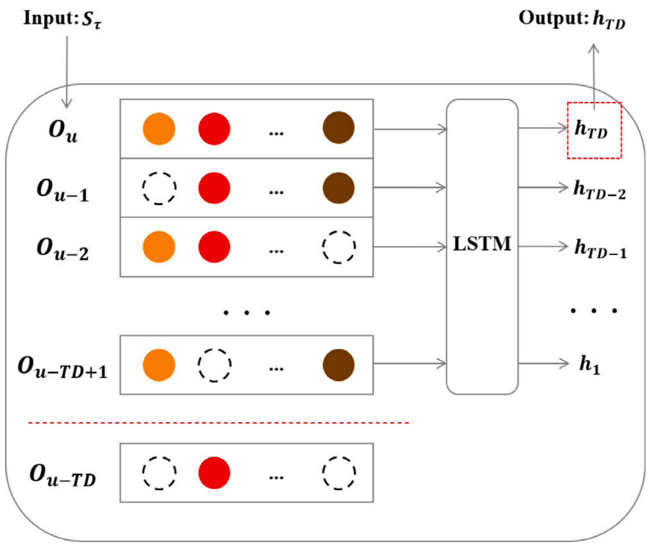


Fig. 3. The structure of SQN.

where all A'_τ are possible actions for the current S_τ. DRL agent updates the policy with all the historical information for maximizing the long-term cumulative reward.

The goal of OSIR is to maximize the cumulative discount reward, i.e., $\mathbb{E}_{\pi_\theta}\left[\sum_{\tau=0}^{\infty}\gamma^\tau R_\tau\left(S_\tau, A_\tau\right)\right]$, where γ is the discount factor for future reward. Thus, Problem 1 can be transformed into:

$$\pi_\theta^* = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{\tau=0}^{\infty}\gamma^\tau R_\tau\left(S_\tau, A_\tau\right)\right]. \tag{34}$$

4.2. SFC queue network

To extract the distribution of required VNF over time, the SQN is designed based on LSTM and added into OSIR. LSTM is widely used in time series prediction problems because it can consider the relationship between the relative position relation between layers of input [43,44].

Fig. 3 shows the structure of SQN in detail, which mainly contains two components: SQN memory and the LSTM. The SQN memory is a layered queue to store the past required VNF, and the length is defined as Time Depth (TD). Each layer in SQN memory is the required processing resources of the SFC request, e.g., $\mathbf{O_u}$ in Fig. 3. The LSTM contains two layers of the LSTM neural network. The input of SQN is the state S_τ.

As shown in Alg. 2, the working process of SQN includes update checking and relationship extraction. In lines 1–6, when the agent receives a state S_τ, SQN checks whether the environment gets a new SFC request u. If not, SQN memory remains unchanged. Otherwise, the SQN captures $\mathbf{O_u}$ from S_τ. Then, the SQN discards the last layer of the SQN memory (i.e., $\mathbf{O_{u-TD}}$), and $\mathbf{O_u}$ is added to SQN memory.

When the agent needs to make a decision for S_τ, SQN learns the distribution of past required VNF from the current SQN memory through the LSTM neural network. As shown in lines 7–9 of Alg. 2, required processing resources of several past requests compose a queue, i.e., $[\mathbf{O_{u-TD+1}}, \mathbf{O_{u-TD+2}}, \dots, \mathbf{O_{u-1}}, \mathbf{O_u}]$, which is the input of the LSTM and fed into the neural network in order. In this way, $\mathbf{O_{u-TD+1}}$ to $\mathbf{O_u}$ constitute the time series of required processing resources of multiple consecutive SFC requests, and the output h_{TD} of the last layer contains the distribution of required VNF, which helps the agent learn the trend of required VNF at the moment.

Algorithm 2 SQN

```
Input: $S_\tau$
Output: $h_{TD}$
1: /* Update Checking */
2: if $j = 1$ then
3:     Delete $\mathbf{O_{u-TD}}$ from SQN Memory
4:     Capture $\mathbf{O_u}$ from $S_\tau$
5:     Push $\mathbf{O_u}$ into SQN Memory
6: end if
7: /* Relationship Extraction */
8: LSTM Input = $[\mathbf{O_{u-TD+1}}, \mathbf{O_{u-TD+2}}, ..., \mathbf{O_{u-1}}, \mathbf{O_u}]$
9: Get LSTM Output = $h_{TD}$
10: end
```

4.3. OSIR training

The OSIR is designed based on A3C [45]. In addition to the actor and critic neural network, OSIR contains an SQN, which is designed to extract the distribution of required VNF over time. After placing VNF f_j^u, the agent sets the required processing resource of type-f_j^u instance to 0, i.e., $o^u_{f_j^u} = 0$. Then, the agent tries to place f_{j+1}^u until all VNFs in SFC request u are placed. Moreover, at any time, $\sum_{f\in\mathbf{F}} \mathbf{1}_{\{o_f^u\}}$ represents the number of VNFs in SFC request u that have not placed, which is obtained as:

$$\sum_{f\in\mathbf{F}} \mathbf{1}_{\{o_f^u\}} = |\mathbf{N_u}| - (j-1). \tag{35}$$

Alg. 3 summarizes the workflow of OSIR. OSIR gives action to each VNF $f_j^u \in \mathbf{N_u}$ in order from $j = 1$ to $j = |\mathbf{N_u}|$, and gets an immediate reward. When failing to placement, a retrace mechanism is used to release the resources occupied by the SFC request u to refresh the network state.

Algorithm 3 OSIR Workflow

```
Input: SFC request $u$
Output: $(S_\tau, A_\tau, R_\tau), ...$
1: Receive SFC request $u$
2: Generate $S_\tau$ and cache $S_\tau$ locally
3: for $j = 1$ to $|\mathbf{N_u}|$ do
4:     The agent observes the state $S_\tau$
5:     if $j = 1$ then
6:         Update SQN according to Alg. 2
7:     end if
8:     The agent selects $A_\tau$ according to the policy
9:     The environment returns the reward $R_\tau(S_\tau, A_\tau)$ according to Alg. 1
10:    if (Eq. (22) or (Eq. (23) and Eq. (24))) and (Eq. (25)) and ($j \neq |\mathbf{N_u}|$ or Eq. (26)) then
11:        $o^u_{f_j^u} = 0$
12:    else
13:        /* Placement Failure */
14:        Reset the state with the cached $S_\tau$ in line 2
15:        Break
16:    end if
17:    Observe $S_{\tau+1}$
18: end for
19: Waiting for the SFC request $u + 1$
20: end
```

In line 2, when receiving a new SFC request u, the environment combines SFC request u with the information of edge network to form state S_τ, and caches S_τ locally. As shown in lines 3 - 20, OSIR places each $f_j^u \in \mathbf{N_u}$ into the edge network one by one. $(S_\tau, A_\tau, R_\tau, S_{\tau+1})$

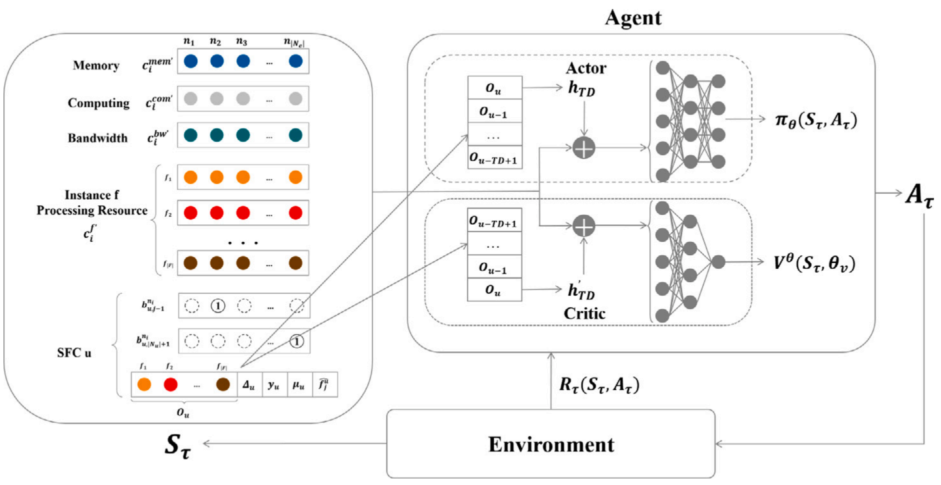


Fig. 4. OSIR framework.

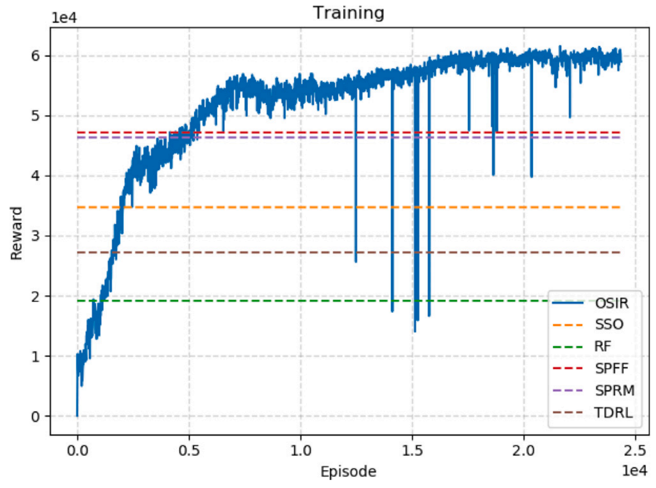


Fig. 5. Training process on $|\mathbf{N_e}| = 10$.

is obtained through the interaction between the environment and the agent. In lines 5–7, when the SQN in the agent obtains a new SFC u, it extracts $\mathbf{O_u}$ from S_τ and updates the SQN. In line 8 and 9, the agent selects an action A_τ according to the policy output by the actor-network. Meanwhile, the environment returns $R_\tau(S_\tau, A_\tau)$ to the agent by Alg. 1.

In line 11, while placing f_j^u successfully, the environment sets $o^u_{f^u_j} = 0$. Then, the agent needs to place the SFC $\mathbf{N_{u'}} = [f^u_{j+1}, \dots, f^u_{|\mathbf{N_u}|}]$, and observes the new state. The agent continues to interact with the environment and get new states, actions and rewards until $j = \mathbf{N_u}$ or placement failure.

In lines 12–15, when placement fails due to insufficient processing resources or communication resources, the environment rejects this request u and resets the current state with the previously cached S_τ. The edge network backtracks to the time before placing request u.

After placing SFC request u, in line 19, the environment waits a new SFC $u+1$ placement, and all above steps are repeated.

The OSIR framework is shown in Fig. 4. At each time step τ, the agent observes the current state S_τ from the environment. Then the agent checks whether the current request u is a new request. If so, SQN captures $\mathbf{O_u}$ from S_τ and adds it to SQN memory. After that, the actor-network and the critic-network get the output h_{TD} and h'_{TD} from the SQN network, respectively. The input of the actor and critic neural network contain the state S_τ and the output of SQN. π_θ and V^θ are calculated through the neural network. According to π_θ, the agent chooses the best action A_τ, and the environment returns $R_\tau(S_\tau, A_\tau)$ based on Alg. 1. The output V^θ of the critic-network assists in training the actor-network.

5. Performance evaluation

The OSIR performance is evaluated through extensive numerical experiments. In this section, experiment settings are first introduced. Then the experimental results are presented and analyzed.

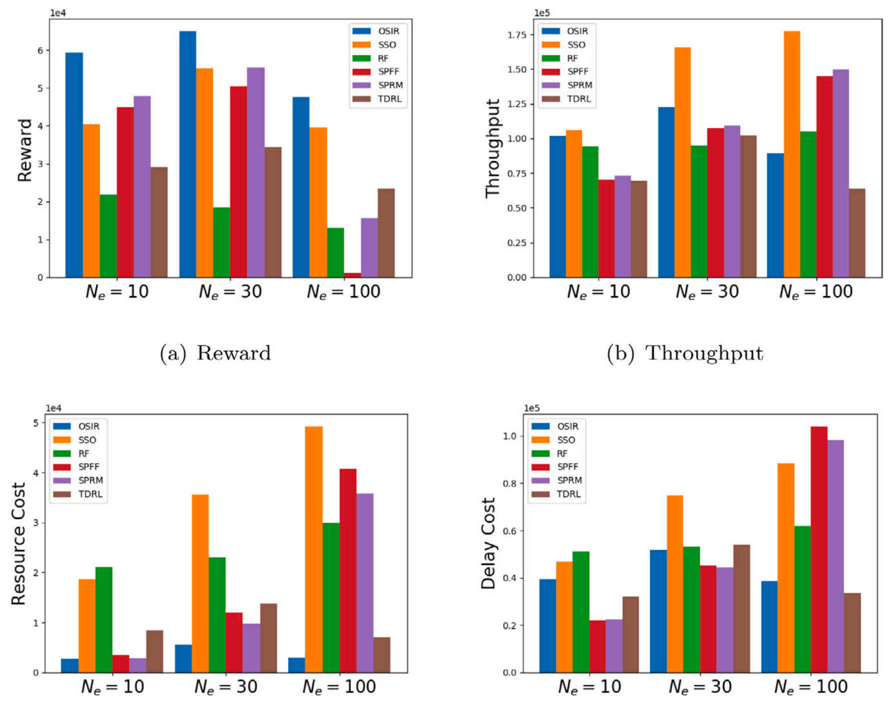
5.1. Experiment settings

5.1.1. Parameter settings

Edge Network: The edge network is set as a fully connected network with identical communication links. The number of nodes is from 10 to 100, each with $[1, 1000]$ units of memory and $[1, 32]$ units of computing resource. The outbound bandwidth of each node ranges from 1200 Mbps to 1800 Mbps. Transmission delay includes intra-pod delay in one node and inter-pod delay between different nodes [46]. The intra-pod delay is set ranging from 20 to 40 μ s and the inter-pod delay ranging from 80 to 150 μ s.

SFC Request: The data comes from real-world traces [47], which is a time-stamped cluster task data set. After preprocessing (e.g., filtering out some requests with low TTL Δ_u), 3997 SFC requests are extracted from 18 h. Each SFC request's start time and end time are set the same as the real-world traces. The first and last requests of the preprocessed data are the earliest start and the latest start, respectively. 20 different commonly-deployed VNFs are simulated for composing SFCs [20], and each request requires an SFC consisted of 1 to 7 VNFs. The source and destination of each SFC are randomly generated. There are 30 kinds of SFC. Each type of VNF instance takes up $[50, 200]$ units of memory and $[1, 4]$ units of computing resources, which can provide $[5, 10]$ units of processing capacity. The required processing resource for each VNF is set to 1, i.e., $o^u_{f^u_j} = 1, \forall u \in \mathbf{U}$. The traffic for each SFC ranges from 30 Mbps to 60 Mbps. The start-up delay d_f of type-f instance is set from 100 ms to 400 ms [48]. Finally, the training and test sets are generated in a 1:1 ratio according to the above rules.

Others: In all experiments, the setting of throughput, resource cost, and delay cost data from [35] is used. Moreover, when calculating the total reward, factors ξ_{TP}, ξ_{DC}, and ξ_{RC} are 0.5, 0.4, and 0.1, respectively. For resource cost, the weights of memory, computing and bandwidth resource are set as $\xi_{bw} = 0.6$, $\xi_{mem} = 0.2$, $\xi_{com} = 0.2$, respectively. The learning rate and gamma of reinforcement learning are 0.0001 and 0.99, respectively. All the experiments are conducted in Python 3.7 on a desktop with a 3.7 GHz 8-Core Intel Core CPU I9-10900K processor.



(a) Reward (b) Throughput

(c) Resource Cost (d) Delay Cost

Fig. 6. Overall performance.

The neural networks in OSIR and baselines are implemented using the Pytorch 1.8.1. Each simulation result has been repeated ten times to avoid the randomness.

5.1.2. *Algorithm benchmark*

The experiment results of OSIR are measured after the training of DRL converges. Fig. 5 shows the OSIR training process on $|\mathbf{N_e}| = 10$. The episode is the metric to measure the time series in the reinforcement learning training process, which is one complete interaction between the agent and environment [49]. In our experiment, all training data are scheduled one by one in each episode according to their start time. The training time varies according to the number of edge nodes, and it may take a few minutes in our experiment for each episode. Since all training processes are offline, even the training time is long on a large number of nodes, and it has little impact on online decision-making. After 5,000 episodes of training, OSIR performs better than all other baselines when $|\mathbf{N_e}| = 10$. After training, the performance of OSIR gradually converges to stability. The OSIR is compared with five existing baseline algorithms, the details are as follows.

- **Reuse First (RF):** it is a greedy algorithm, which tries to reuse instances as much as possible to place SFC requests.
- **Shortest Path + First Fit assignment (SPFF):** it is a two-stage heuristic algorithm [11,17]. In the first stage, SPFF finds the shortest path (SP). In the second stage, it assigns the VNFs along the shortest path by choosing the first fit node.
- **Shortest Path + Reuse Max (SPRM):** it is also a two-stage heuristic algorithm [17]. In the first stage, SPRM finds the shortest path. In the second stage, each VNF in the SFC is assigned along the shortest path by reusing as many available VNF instances as possible.
- **Single-Step Optimal (SSO):** A heuristic algorithm is proposed to solve the problem of instance reuse by searching all feasible paths and traversing the optimal solution [17]. However, there are many feasible paths in the dense network, resulting in a highly long execution time (related experiments are shown in Section 5.2.4). By modifying this algorithm to adapt our problem, the SSO is designed to find each SFC request's solutions one by one. When receiving a new SFC, SSO finds the current optimal placement strategy with the highest reward by directly solving the linear equations of Problem 1, which is implemented through Cplex[1].
- **Traditional DRL (TDRL)** [51]: it is a traditional DRL without considering instance reuse especially.

5.2. *Experimental results and analysis*

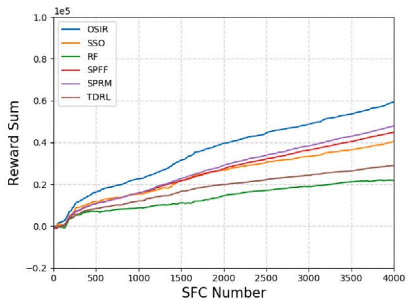
The OSIR is first evaluated by comparing with all baselines. Then, the cumulative reward is analyzed over time, and the Cumulative Distribution Function (CDF) of resource cost and delay cost is studied. Finally, the execution time of different algorithms is compared.

5.2.1. *Performance of OSIR*

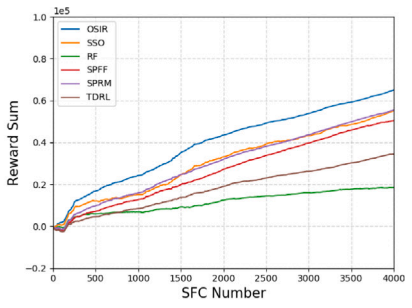
Firstly, a set of experiments is conducted to verify the performance of OSIR with different scale of edge network in Fig. 6. In addition to total reward, how OSIR improves the system performance is analyzed by decomposing the reward, including: throughput, resource cost, and delay cost.

In Fig. 6, OSIR improves the reward of serving SPs by better accepting SFC comparing with the baseline algorithms. In Fig. 6(b), Figs. 6(c) and 6(d), when the number of nodes is small, OSIR significantly reduces resource cost and delay cost while maintaining throughput. It means that OSIR can improve the reward mainly by optimizing placement strategy, when facing computation-intensive or resource-limited. With the increasing node number, although OSIR may reject some SFC request and reduce the throughput, it significantly reduces resource cost. Thus, when the edge network provides sufficient resources, OSIR

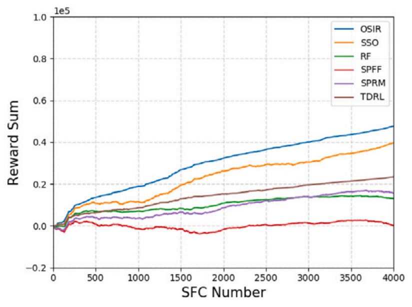
[1] According to the guidelines [50], the Cplex-Optimizer implements an algorithm that is based on the classical Branch & Bound paradigm and heuristic algorithms.



(a) $|\mathbf{N_e}| = 10$



(b) $|\mathbf{N_e}| = 30$



(c) $|\mathbf{N_e}| = 100$

Fig. 7. Total cumulative reward.

can intelligently select requests by the DRL agent. By comparing the reward in Fig. 6(a), the OSIR can be adapted at different scale of edge network and exhibit superiority. It is better than the best baseline of each experiment group by 17% to 26%.

The experimental results of the RF algorithm are the worst in most cases. Since RF tries to reuse blindly to reduce node resource consumption. However, it also results in excessive forwarding, which causes a non-negligible increase in link resource consumption and leads to higher delay cost and resource cost. Moreover, the throughput is often inferior to other algorithms due to superfluous resource consumption. For the SSO algorithm, although the optimal solution for the current SFC request is selected at each step, its performance is inferior to OSIR due to the inability to consider long-term cumulative rewards. For the SPFF algorithm, it is worth noticing that its overall performance is similar to SSO in small edge clusters. However, the node resources are abundant with more edge nodes. SPFF tends to choose the shortest path, which results in initializing a large number of redundant instances. As shown in Fig. 6(c), when $|\mathbf{N_e}| = 100$, SPFF increases the resource costs compared with other algorithms, so the final reward performance is abysmal. The SPRM also chooses the shortest path for each SFC request as much as possible. When the number of edge nodes is small, its performance is similar to SPFF. However, with the increase in the number of edge nodes, although SPRM tries to reuse instances to avoid initializing redundant instances, its result is also far less than the OSIR. For the TDRL, its performance is always inferior to the OSIR. Because the SQN in OSIR helps learn the complex distribution of the required VNF better.

5.2.2. *Cumulative reward*

Fig. 7 shows the total cumulative rewards of different algorithms. Then, every 100 consecutive SFCs are grouped, and their cumulative rewards are counted, as shown in Fig. 8. By grouping every 100 consecutive SFCs and calculating their cumulative rewards, the placement of SFCs in different time slots can be evaluated.

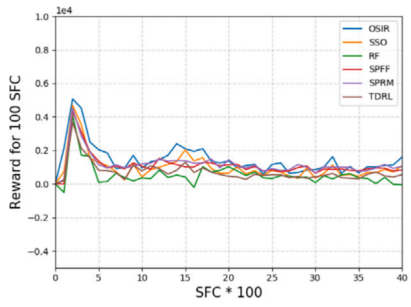
Firstly, through the observation of results in Figs. 7 and 8, there is little difference in cumulative rewards between all algorithms when the environment receives few SFC requests. In particular, the results of SSO are almost the same with OSIR before placing 300 SFC requests in Fig. 7. However, as more and more SFC requests arrive over time, all edge nodes become crowded, and resources are consumed rapidly. In Fig. 8, before placing 300 SFC requests, the result of OSIR is close to the SSO. Then, in most of the time, OSIR has a higher cumulative reward than others over 100 consecutive SFCs requests. It means that OSIR can reuse instances more intelligently by observing the environment when some VNF instances have been placed in the environment.

In Fig. 7, the cumulative reward of RF is always lower than SSO and OSIR. Moreover, in Fig. 8, almost all the time, OSIR has a higher cumulative reward than RF over 100 consecutive SFC requests. Thus, with more SFC requests, the reward gap between RF and OSIR is getting larger. The SPFF and SPRM algorithm both perform better than other baselines when the number of edge nodes is small in Fig. 7. The reason is that the first step of the SPFF and SPRM algorithm is to find the shortest path from the source to the destination for each SFC request. In Fig. 8, when the environment receives few SFC requests, the reward of SPFF is always negative most of the time, which means that the sum of resource consumption and delay cost is always higher than the throughput. In Fig. 8(a) and Fig. 8(b), by avoiding deploying redundant instances on more nodes, the small-scale edge network prevents the resource cost from increasing. The performance of SPFF gradually becomes similar to other algorithms. However, in Fig. 8(c), the node resources are almost sufficient at all times. SPFF is forced to create new instances repeatedly, leading to poor performance. The TDRL is very mediocre in any situation.

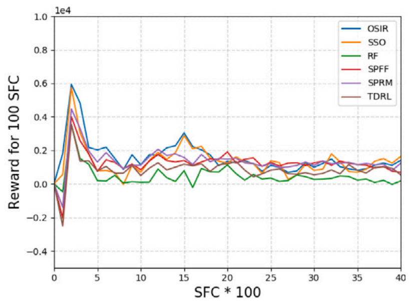
5.2.3. *Cost CDF*

In this part, the resource cost and delay cost of different algorithms are evaluated by CDFs, which are shown in Figs. 9 and 10, respectively.

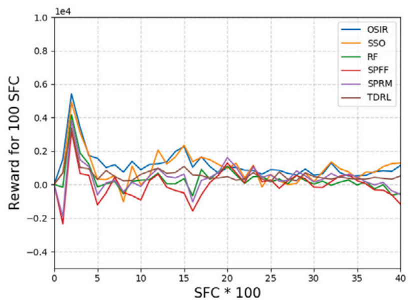
Although SPRM and SPFF perform similarly with OSIR in Fig. 9(a), their results decrease significantly with the increasing node number. In Fig. 9(c), the results of SPFF are worse than all other algorithms. By analyzing the CDF of the delay cost in Fig. 10, both SPFF and SPRM show their advantages of significantly reducing delay costs when



(a) $|\mathbf{N_e}| = 10$

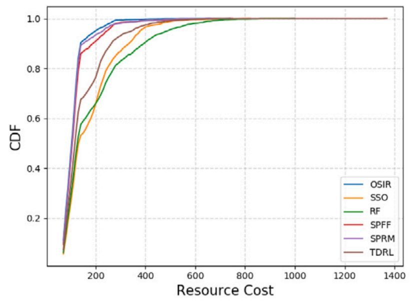


(b) $|\mathbf{N_e}| = 30$

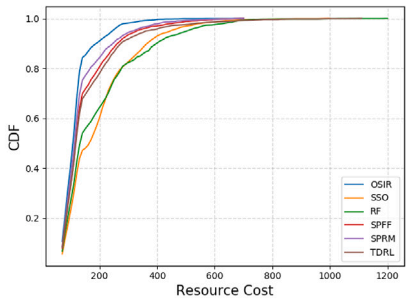


(c) $|\mathbf{N_e}| = 100$

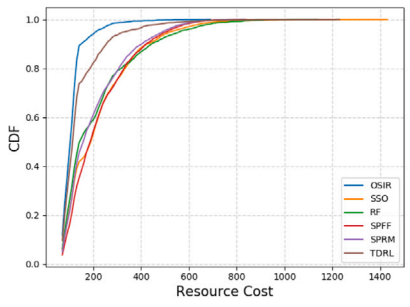
Fig. 8. Cumulative reward over 100 consecutive SFC requests.



(a) $|\mathbf{N_e}| = 10$

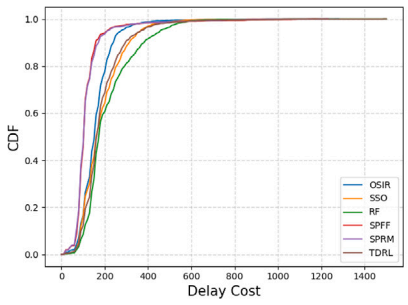


(b) $|\mathbf{N_e}| = 30$

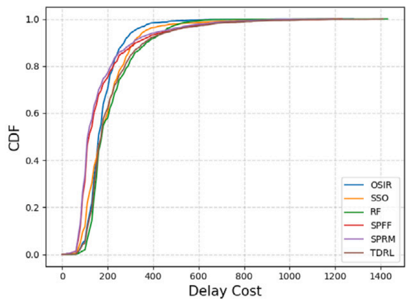


(c) $|\mathbf{N_e}| = 100$

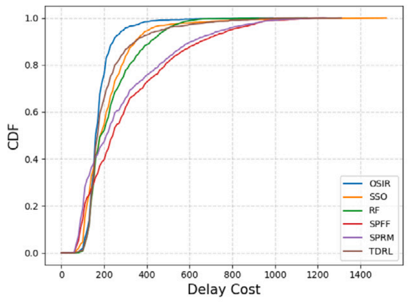
Fig. 9. Cumulative distribution function of resource cost.



(a) $|\mathbf{N_e}| = 10$



(b) $|\mathbf{N_e}| = 30$



(c) $|\mathbf{N_e}| = 100$

Fig. 10. Cumulative distribution function of delay cost.

the number of edge nodes is small. However, although the first step of SPFF and SPRM is to choose the shortest path, as the number of edge nodes increases, their delay costs are even worse than all other algorithms in Fig. 6(d). Combining the resource costs of SPFF and SPRM in Fig. 6(c), the reason for the rapid increase in delay costs is that with the increase in the number of nodes, there may be no reusable instances in the shortest path. Furthermore, it leads to initializing redundant VNF instances, which incurs high delay costs caused by start-up delay and resource cost.

In Fig. 9, SSO performs almost as same as RF on resource costs, which may be lacking the consideration of the future impact. Thus, SSO cannot place the instances of widely used VNF type on critical nodes, resulting in some requests consuming more resources. RF is not outstanding at saving edge resources. Because reusing blindly results in longer routing paths and more bandwidth resource consumption. Further, although the results of SSO and RF are similar in terms of resource cost in each experiment, RF always has a higher delay cost due to the longer routing path.

As shown in Figs. 9 and 10, OSIR consumes less resource costs and delay costs for most SFC requests under different experiments. The OSIR algorithm uses LSTM to extract the distribution of required VNF, which can learn the changing of users' service demands at the current time. Then, when placing SFC requests and VNF instances, OSIR considers the future reward through DRL and place VNF instances on nodes, which can reduce the resource cost and the delay cost of more SFC requests. However, TDRL, which is also DRL-based without other special designs, cannot significantly reduce resource and delay costs. Only in Fig. 10(a), the OSIR generates more delay cost than SPFF and SPRM since SPFF and SPRM always place SFC requests on the shortest path. However, it clearly shows that SPFF and SPRM increase resource cost in Fig. 9. Further, in Fig. 6, the performances of both SPFF and SPRM in total reward are worse than OSIR in all experiments. By analyzing the CDF of both resource cost and delay cost, the OSIR can optimize total reward by effectively reducing resource cost and delay cost.

Table 2
Execution time.

$\lvert N_e \rvert$	10	30	100
OSIR	23.59 s	24.89 s	26.32 s
SSO	627.02 s	10408.66 s	127194.40 s
RF	1.52 s	1.56 s	1.87 s
SPFF	4.47 s	79.98 s	2902.08 s
SPRM	131.51 s	356.34 s	3530.15 s
TDRL	2.22 s	2.71 s	3.01 s

5.2.4. *Execution time*

The execution time of different algorithms in each experiment is shown in Table 2.

The execution time of OSIR is mainly caused by the forward propagation in the neural network. When the node number is small ($|\mathbf{N_e}| = 10$), it takes about half a minute to find the placement approach of 3997 SFC requests one by one. However, with the increasing node number, the execution time of OSIR does not increase significantly.

Since RF is a greedy algorithm, it can place SFC requests quickly under different experiments. Although it is the fastest among all algorithms, its performance is far worse than OSIR and SSO on both throughput and total reward, as shown in Fig. 6. SSO can reduce the execution time of the algorithm [17] by directly solving the linear constraints. However, its execution time is still the largest due to a large amount of data. Moreover, with the increasing node number, its execution time even grows exponentially. Therefore, when faced with large-scale edge networks or dense graphs, the execution time of SSO cannot be tolerated. SPFF can also figure out solutions quickly on a small-scale edge network. However, with the increasing node number, the execution time also grows exponentially. When $|\mathbf{N_e}| = 30$, its execution time has far exceeded OSIR. The execution time of the SPRM also increases significantly as the number of nodes grows. Moreover, the execution time of SPRM is longer than SPFF since the greedy strategy in SPRM is more complicated than SPFF. Due to the simple structure without other special designs in the TDRL, TDRL has a short execution time during testing, but it performs very poorly in all experimental results.

6. Discussion

From the experimental results, we can see the effectiveness of OSIR. The following issues deserved further investigations.

Instance Reuse in the Joint SFC Placement and Traffic Routing Problems: We have divided the resource allocation in NFV into four typical problems in Section 2.2. Our problem focuses on online instance reuse in a typical SFC placement problem. Therefore, in order to avoid routing problems, the edge network is set up as fully connected. However, when the edge network is partially connected, solving the problem of instance reuse in joint SFC placement and routing problems will become an important research topic.

Instance Management Cost: The management of the instance includes the start-up, maintenance, and release. We have discussed the costs of start-up and release of instances in Section 3.1.1. Further, the maintenance of the instance also requires maintenance costs. Since the instance is supposed to be released immediately when there is no running SFC request in it, the time to live of instances is usually short, and the maintaining cost is usually very low. However, if the instance is not released immediately, the maintenance cost of the instance may also be worth discussing.

Reducing Training Time: As shown in Fig. 5, on 10 edge nodes, OSIR performs better than all baselines after training of 5000 episodes. Although the model of OSIR is trained offline and used for online decision-making, how to reduce training time is still a question worth discussing. A model-assisted method to accelerate training in DRL is proposed [35], but it is hard to apply in an asynchronous DRL algorithm, which may lead to non-converge. Transfer learning or meta reinforcement learning may be a possible approach to reduce our training time, but applying them is very complicated.

7. Conclusion

In this paper, to serve computation-intensive and latency-sensitive services in MEC, the problem of online SFC placement with instance reuse is proposed, and an OSIR algorithm is proposed to solve it. OSIR can gain a tradeoff between operating costs and users' QoE, which is designed based on A3C to find the solution for SFC placement to obtain the long-term cumulative reward. Further, a special SQN is designed to extract the distribution of required VNF over time. The setting of rewards in OSIR is based on our problem's model, aiming to optimize throughput, resource cost, and delay cost. Extensive experiments on different edge networks with real-world data traces prove that the OSIR can efficiently and effectively improve the long-term cumulative reward by scheduling SFC requests and reusing VNF instances intelligently. Our future work will mainly focus on implementing and evaluating our algorithms on practical NFV platforms, e.g., OPNFV Brahmaputra.

CRediT authorship contribution statement

Songli Zhang: Conceptualization, Methodology, Data curation, Software, Writing – original draft, Writing – review & editing. **Weijia Jia:** Writing – review & editing, Supervision, Funding acquisition. **Zhiqing Tang:** Conceptualization, Writing – review & editing. **Jiong Lou:** Methodology, Writing – review & editing. **Wei Zhao:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by Guangdong Key Lab of AI and Multi-modal Data Processing, BNU-HKBU United International College (UIC), Zhuhai, No. 2020KSYS007; Chinese National Research Fund (NSFC), No. 61872239; the Institute of Artificial Intelligence and Future Networks funded by Beijing Normal University (Zhuhai) Guangdong, China.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646, http://dx.doi.org/10.1109/JIOT.2016.2579198.

[2] G. Forman, J. Zahorjan, The challenges of mobile computing, Computer 27 (4) (1994) 38–47, http://dx.doi.org/10.1109/2.274999.

[3] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: A survey, IEEE Internet Things J. 5 (1) (2018) 450–465, http://dx.doi.org/10.1109/JIOT.2017.2750180.

[4] J. Matias, J. Garay, N. Toledo, J. Unzilla, E. Jacob, Toward an SDN-enabled NFV architecture, IEEE Commun. Mag. 53 (4) (2015) 187–193, http://dx.doi.org/10.1109/MCOM.2015.7081093.

[5] H. Hawilo, A. Shami, M. Mirahmadi, R. Asal, NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC), IEEE Netw. 28 (6) (2014) 18–26, http://dx.doi.org/10.1109/MNET.2014.6963800.

[6] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J.J. Ramos-Munoz, J. Lorca, J. Folgueira, Network slicing for 5G with SDN/NFV: concepts, architectures, and challenges, IEEE Commun. Mag. 55 (5) (2017) 80–87, http://dx.doi.org/10.1109/MCOM.2017.1600935.

[7] R. Cziva, D.P. Pezaros, Container network functions: Bringing NFV to the network edge, IEEE Commun. Mag. 55 (6) (2017) 24–31, http://dx.doi.org/10.1109/MCOM.2017.1601039.

[8] F. Schardong, I. Nunes, A. Schaeffer-Filho, NFV resource allocation: A systematic review and taxonomy of vnf forwarding graph embedding, Comput. Netw. (2020) 107726.

[9] X. Shang, Y. Huang, Z. Liu, Y. Yang, Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme, IEEE Trans. Mob. Comput. (2021) 1, http://dx.doi.org/10.1109/TMC.2020.3048885.

[10] M. Golkarifard, C.F. Chiasserini, F. Malandrino, A. Movaghar, Dynamic VNF placement, resource allocation and traffic routing in 5G, Comput. Netw. 188 (2021) 107830.

[11] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, F. Liberal, Virtual network function placement optimization with deep reinforcement learning, IEEE J. Sel. Areas Commun. 38 (2) (2020) 292–303, http://dx.doi.org/10.1109/JSAC.2019.2959183.

[12] Z.M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, K. Mizutani, State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2432–2455, http://dx.doi.org/10.1109/COMST.2017.2707140.

[13] V. Farhadi, F. Mehmeti, T. He, T.L. Porta, H. Khamfroush, S. Wang, K.S. Chan, Service placement and request scheduling for data-intensive applications in edge clouds, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 1279–1287, http://dx.doi.org/10.1109/INFOCOM.2019.8737368.

[14] S. Pasteris, S. Wang, M. Herbster, T. He, Service placement with provable guarantees in heterogeneous edge computing systems, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 514–522, http://dx.doi.org/10.1109/INFOCOM.2019.8737449.

[15] B. Yi, X. Wang, M. Huang, A generalized VNF sharing approach for service scheduling, IEEE Commun. Lett. 22 (1) (2018) 73–76, http://dx.doi.org/10.1109/LCOMM.2017.2761874.

[16] X. Fei, F. Liu, Q. Zhang, H. Jin, H. Hu, Paving the way for NFV acceleration: A taxonomy, survey and future directions, ACM Comput. Surv. 53 (4) (2020) 1–42.

[17] P. Jin, X. Fei, Q. Zhang, F. Liu, B. Li, Latency-aware VNF chain deployment with efficient resource reuse at network edge, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 267–276, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155345.

[18] H. Guo, Y. Wang, Z. Li, X. Qiu, H. An, P. yu, N. Yuan, Cost-aware placement and chaining of service function chain with vnf instance sharing, in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1–8, http://dx.doi.org/10.1109/NOMS47738.2020.9110360.

[19] J. Gil Herrera, J.F. Botero, Resource allocation in NFV: A comprehensive survey, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 518–532, http://dx.doi.org/10.1109/TNSM.2016.2598420.

[20] B. Addis, D. Belabed, M. Bouet, S. Secci, Virtual network functions placement and routing optimization, in: 2015 IEEE 4th International Conference on Cloud Networking, CloudNet, 2015, pp. 171–177, http://dx.doi.org/10.1109/CloudNet.2015.7335301.

[21] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, X. Fu, Recent advances of resource allocation in network function virtualization, IEEE Trans. Parallel Distrib. Syst. 32 (2) (2021) 295–314, http://dx.doi.org/10.1109/TPDS.2020.3017001.

[22] A. Tomassilli, F. Giroire, N. Huin, S. Pérennes, Provably efficient algorithms for placement of service function chains with ordering constraints, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 774–782, http://dx.doi.org/10.1109/INFOCOM.2018.8486275.

[23] C. You, L.M. Li, Efficient load balancing for the VNF deployment with placement constraints, in: ICC 2019 - 2019 IEEE International Conference on Communications, ICC, 2019, pp. 1–6, http://dx.doi.org/10.1109/ICC.2019.8761564.

[24] W. Ma, O. Sandoval, J. Beltran, D. Pan, N. Pissinou, Traffic aware placement of interdependent NFV middleboxes, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9, http://dx.doi.org/10.1109/INFOCOM.2017.8056993.

[25] D. Li, P. Hong, K. Xue, J. Pei, Virtual network function placement and resource optimization in NFV and edge computing enabled networks, Comput. Netw. 152 (2019) 12–24.

[26] H. Chen, X. Wang, Y. Zhao, T. Song, Y. Wang, S. Xu, L. Li, MOSC: A method to assign the outsourcing of service function chain across multiple clouds, Comput. Netw. 133 (2018) 166–182.

[27] Q. Zhang, Y. Xiao, F. Liu, J.C. Lui, J. Guo, T. Wang, Joint optimization of chain placement and request scheduling for network function virtualization, in: 2017 IEEE 37th International Conference on Distributed Computing Systems, ICDCS, 2017, pp. 731–741, http://dx.doi.org/10.1109/ICDCS.2017.232.

[28] M. Li, Q. Zhang, F. Liu, Finedge: A dynamic cost-efficient edge resource management platform for NFV network, in: 2020 IEEE/ACM 28th International Symposium on Quality of Service, IWQoS, 2020, pp. 1–10, http://dx.doi.org/10.1109/IWQoS49365.2020.9212908.

[29] Y. Li, Deep reinforcement learning: An overview, 2017, arXiv preprint arXiv:1701.07274.

[30] N.C. Luong, D.T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, D.I. Kim, Applications of deep reinforcement learning in communications and networking: A survey, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3133–3174, http://dx.doi.org/10.1109/COMST.2019.2916583.

[31] K. Wang, C.-M. Chen, M.S. Hossain, G. Muhammad, S. Kumar, S. Kumari, Transfer reinforcement learning-based road object detection in next generation IoT domain, Comput. Netw. 193 (2021) 108078.

[32] P. Sun, Z. Guo, S. Liu, J. Lan, J. Wang, Y. Hu, SmartFCT: Improving power-efficiency for data center networks with deep reinforcement learning, Comput. Netw. 179 (2020) 107255.

[33] F. Wang, C. Zhang, F. wang, J. Liu, Y. Zhu, H. Pang, L. Sun, Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized QoE, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 910–918, http://dx.doi.org/10.1109/INFOCOM.2019.8737456.

[34] F. Restuccia, T. Melodia, Deepwierl: Bringing deep reinforcement learning to the internet of self-adaptive things, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 844–853, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155461.

[35] L. Gu, D. Zeng, W. Li, S. Guo, A.Y. Zomaya, H. Jin, Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning, IEEE J. Sel. Areas Commun. 38 (2) (2020) 279–291, http://dx.doi.org/10.1109/JSAC.2019.2959182.

[36] J. Sun, G. Huang, G. Sun, H. Yu, A.K. Sangaiah, V. Chang, A Q-learning-based approach for deploying dynamic service function chains, Symmetry 10 (11) (2018) 646.

[37] J. Zheng, C. Tian, H. Dai, Q. Ma, W. Zhang, G. Chen, G. Zhang, Optimizing NFV chain deployment in software-defined cellular core, IEEE J. Sel. Areas Commun. 38 (2) (2020) 248–262, http://dx.doi.org/10.1109/JSAC.2019.2959180.

[38] H. Yao, C. Bai, M. Xiong, D. Zeng, Z. Fu, Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing, Concurr. Comput.: Pract. Exper. 29 (16) (2017) e3975.

[39] R. Hou, S. Zhou, Y. Zheng, M. Dong, K. Ota, D. Zeng, J. Luo, M. Ma, Cluster routing-based data packet backhaul prediction method in vehicular named data networking, IEEE Trans. Netw. Sci. Eng. 8 (3) (2021) 2639–2650, http://dx.doi.org/10.1109/TNSE.2021.3102969.

[40] L. Gu, J. Hu, D. Zeng, S. Guo, H. Jin, Service function chain deployment and network flow scheduling in geo-distributed data centers, IEEE Trans. Netw. Sci. Eng. 7 (4) (2020) 2587–2597, http://dx.doi.org/10.1109/TNSE.2020.2997376.

[41] H. Hou, H. Jin, X. Liao, D. Zeng, Stochastic analysis on fog computing empowered mobile crowdsensing with D2D communications, in: 2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI, 2018, pp. 656–663, http://dx.doi.org/10.1109/SmartWorld.2018.00131.

[42] Y. Li, L. Chen, D. Zeng, L. Gu, A customized reinforcement learning based binary offloading in edge cloud, in: 2020 IEEE 26th International Conference on Parallel and Distributed Systems, ICPADS, 2020, pp. 356–362, http://dx.doi.org/10.1109/ICPADS51040.2020.00055.

[43] A. Sagheer, M. Kotb, Time series forecasting of petroleum production using deep LSTM recurrent networks, Neurocomputing 323 (2019) 203–213.

[44] F.A. Gers, D. Eck, J. Schmidhuber, Applying LSTM to time series predictable through time-window approaches, in: Neural Nets WIRN Vietri-01, Springer, 2002, pp. 193–200.

[45] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, J. Kautz, Reinforcement learning through asynchronous advantage actor-critic on a gpu, 2016, arXiv preprint arXiv:1611.06256.

[46] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, J. Zhang, NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning, in: Proceedings of the International Symposium on Quality of Service, 2019, pp. 1–10.

[47] Alibaba, Alibaba production cluster data v2018, 2021, https://github.com/alibaba/clusterdata, (Accessed 6 June 2021).

[48] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, F. Huici, My VM is lighter (and safer) than your container, in: Proceedings of the 26th Symposium on Operating Systems Principles, in: SOSP '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 218–233, http://dx.doi.org/10.1145/3132747.3132763.

[49] M.A. Wiering, M. Van Otterlo, Reinforcement learning, Adapt., Learn. Optim. 12 (3) (2012).

[50] Cplex, IBM cplex-optimizer, 2021, https://www.ibm.com/analytics/cplex-optimizer. (Accessed 6 June 2021).

[51] N.C. Luong, D.T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, D.I. Kim, Applications of deep reinforcement learning in communications and networking: A survey, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3133–3174, http://dx.doi.org/10.1109/COMST.2019.2916583.

Songli Zhang received the B.S. degree from Department of Electrical Engineering, Shandong University, China, in 2019 and is currently a Ph.D. candidate in Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research interests includenetworkfunctionvirtualization, mobile edge computing, resource allocation, and reinforcement learning.

Weijia Jia is currently a Chair Professor and Director of BNU-UIC Institute of Artificial Intelligence and future Networks, Beijing Normal University at Zhuhai; VP for Research of BNU-HKBU United International College. His contributions have been recognized as optimal network routing and deployment, anycast and QoS routing, sensors networking, AI (knowledge relation extractions; NLP etc.) and edge computing. He has over 600 publications in the prestige international journals/conferences and research books and book chapters. He is the Fellow of IEEE and the Distinguished Member of CCF.

Zhiqing Tang received the B.S. degree from School of Communication and Information Engineering, University of Electronic Science and Technology of China, China, in 2015 and is currently a Ph.D. candidate in Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research interests include mobile edge computing, resource allocation, and reinforcement learning.

Jiong Lou received the B.S. degree from Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2016 and is currently a Ph.D. candidate in Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research interests include mobile edge computing, resource allocation, and reinforcement learning.

Wei Zhao is currently the chairman of the academic committee and the chair professor of the Shenzhen Institutes of Advanced Technology (SIAT) of the Chinese Academy of Science (CAS). He is the Fellow of IEEE. He has made significant contributions in the cyber–physical system, distributed computing, real-time systems, computer networks, and cyberspace security. He led the effort to define research agenda and to create the very first research funding program for the cyber–physical system, when he served as the NSF CNS Division Director in 2006. His research group has received numerous awards.