A Sub-Action Aided Deep Reinforcement Learning Framework for Latency-Sensitive Network Slicing

Da Xiao[a,*], Shuo Chen[b], Wei Ni[c], Jie Zhang[b], Andrew Zhang[a], Renping Liu[a]

[a]*School of Electrical and Data Engineering, University of Technology Sydney, Global Big Data Technologies Centre, CB11 81-113, Broadway, Ultimo, Sydney, 2007, NSW, Australia*
[b]*School of Computer Science and Engineering, Nanyang Technological University, Computational Intelligence Laboratory, 50 Nanyang Avenue, Singapore, 639798, Singapore*
[c]*Data61, Commonwealth Scientific and Industrial Research Organisation, Corner Vimiera and Pembroke Rd, Marsfield, Sydney, 2122, NSW, Australia*

Abstract

Network slicing is a core technique of fifth-generation (5G) systems and beyond. To maximize the number of accepted network slices with limited hardware resources, service providers must avoid over-provisioning of quality-of-service (QoS), which could prevent them from lowering capital expenditures (CAPEX)/operating expenses (OPEX) for 5G infrastructure. In this paper, we propose a sub-action aided double deep Q-network (SADDQN)-based network slicing algorithm for latency-aware services. Specifically, we model network slicing as a Markov decision process (MDP), where we consider virtual network function (VNF) placements to be the actions of the MDP, and define a reward function based on cost and service priority. Furthermore, we adopt the Dijkstra algorithm to determine the forwarding graph (FG) embedding for a given VNF placement and design a resource allocation algorithm — binary search assisted gradient descent (BSAGD) — to allocate resources to VNFs given the VNF-FG placement. For every service request, we first use the DDQN to choose an MDP action to determine the VNF placement (main action). Next, we employ the Dijkstra algorithm (first-phase sub-action) to find the shortest path for each pair of adjacent VNFs in the given VNF chain. Finally, we implement the BSAGD (second-phase sub-action) to realize this service with the minimum cost. The joint action results in an MDP reward that can be utilized to train the DDQN. Numerical evaluations show that, compared to state-of-the-art algorithms, the proposed algorithm can improve the cost-efficiency while giving priority to higher-priority services and maximizing the acceptance ratio.

Keywords: Network slicing, QoS over-provisioning, VNF-FG placement

1. Introduction

To meet the diverse industrial and market demands, the International Telecommunication Union (ITU) has classified the current-generation mobile networks (5G) into three main categories: ultra-reliable low latency communications (URLLC), enhanced mobile broadband (eMBB), and massive machine-type communications (mMTC). Packets belonging to the same category are aggregated and then travel through the corresponding network slice, which is composed of an ordered set of virtual network functions (VNFs) and virtual links (VLs) connecting them. A network slice can be symbolized by a service function chain (SFC) or a VNF-forwarding graph (VNF-FG). 5G imposes more stringent latency requirements on payload traffic than its predecessor, 4G systems, in support of latency-sensitive applications, such as remote surgery and self-driving vehicles [1]. Therefore, efficient and automatic placement of VNFs becomes one of the most critical components for meeting such requirements. Furthermore, within the same category, traffic from different users may have different quality-of-service (QoS) requirements; hence, we need to create several sub-slices [2, 3] within a category to meet the diverse requirements.

Several works have aimed to address the VNF-FG placement problem. Some are heuristic-based methods that are good for stationary systems but could have degraded performance in dynamic systems. Others are deep reinforcement learning (DRL)-based approaches, which are efficient when properly designed but inefficient when their action space becomes enormous. In addition, two main open issues remain in all the existing works. First, only a few focus on the sub-slices issue, let alone latency-sensitive sub-slices. Second, when deploying network slices, most works adopt the classical virtual network embedding (VNE) model, which assumes that the required resources of VNFs are already specified in SFC requests [4] and thus do not need to consider the resource allocation issue of VNFs. However, in some real cases, customers might not be knowledgeable enough to configure VNFs; hence, service providers must help to allocate resources to VNFs.

In this paper, to address these issues, as in [5], we first assume that customers do not know the required resources of VNFs and thus specify their SFC requests in terms of (i) a sequence chain of VNFs, (ii) requested traffic, and (iii) latency

*Corresponding Author
Email addresses: Da.Xiao@student.uts.edu.au (Da Xiao), schen@ntu.edu.sg (Shuo Chen), Wei.Ni@data61.csiro.au (Wei Ni), ZhangJ@ntu.edu.sg (Jie Zhang), Andrew.Zhang@uts.edu.au (Andrew Zhang), RenPing.Liu@uts.edu.au (Renping Liu)

requirement. Then, we propose a sub-action aided double deep Q-network (SADDQN)-based network slicing algorithm for deploying latency-sensitive sub-slices with different latency requirements.

Specifically, the proposed algorithm handles SFC requests one after another. For a request, the algorithm sequentially decides (i) the placement of the VNFs in this SFC, (ii) the forwarding graph (FG) embedding given the VNF placement, and (iii) the assignment of CPU and bandwidth resources to the VNFs, given the placement of the VNF-FG.

We assume there are L latency-sensitive SFC requests, also known as service requests. Accordingly, we model the network slicing for all requests as a Markov decision process (MDP) and represent the MDP action space as the possible VNF placements of a single request. We prioritize requests based on their latency requirements and define the reward function of the MDP based on priority and resource cost. For every incoming request, the DDQN first chooses an MDP action to determine the VNF placement. Given the VNF placement, the Dijkstra algorithm is then employed to embed the VLs. Finally, based on the VNF-FG placement, the latency requirement, and requested traffic of this service, we propose a resource allocation algorithm — binary search assisted gradient descent (BSAGD) — to provide the request with the minimum resource such that its latency requirement is satisfied. In response to the joint action (VNF-FG placement and resource allocation), an MDP reward is returned to train our DDQN. Once trained, the SADDQN model approximates the optimal solution of ensuring priority allocation for higher-priority services and maximizing the acceptance ratio while minimizing the total cost.

The main contributions of this paper can be summarized as follows:

1. We formulate the SFC embedding problem as an MDP with appropriate state, by including the requested traffic into the state, rather than by using the required VNF capacity (CPU, memory, ...) to replace traffic, as in [6]. This change is essential because, in some cases, VNF capacity does not accurately reflect real network traffic, which is critical to latency-sensitive services.
2. Given a VNF placement (main action) and the optimal path traversing the VNF chain (the first-phase sub-action), we propose a resource allocation algorithm (the second-phase sub-action) to realize the network slice with the minimum cost. In other words, instead of assuming that the required resources of VNFs are known in advance, we optimally allocate resources to all VNFs in a given VNF chain based on the requested traffic and latency requirement of this service. In this way, for any given VNF placement, we only need to take the main action into the action space because we find the optimal first-phase and second-phase sub-action for the main action. Otherwise, we would have to enumerate chaining and resource allocation options for each VNF placement; thus, the action space would be too large for the DDQN algorithm to converge.
3. Most QoS-related works fail to slice a network slice into sub-slices. We prioritize services (the lower latency threshold a service requests, the higher priority it receives), define a reward function based on priority and cost, and propose a SADDQN framework to ensure priority allocation for higher-priority services and maximize the acceptance ratio while minimizing the total cost.

The rest of this paper is organized as follows. In Section II, we discuss related work. Then, we present the system model of VNF-FG placement for latency-sensitive services and formulate the problem in Section III. In Section IV, we present our algorithm, and in Section V, we numerically evaluate the performance of our algorithm. Finally, we conclude the paper in Section VI.

2. Related Work

Many studies have been devoted to the placement of network services techniques [7, 8, 9]. Since our objective is to deploy as many latency-sensitive sub-slices as possible with limited hardware resources, we pay close attention to those works studying latency-sensitive or traffic-aware services. In general, they fall into the following two categories.

2.1. Heuristic-Based Approaches

Most techniques were proposed to formulate and solve optimization problems [10, 11, 12, 13, 14, 4, 15, 16, 17, 18, 5]. Some works, [12] and [13], are based on strong assumptions. For instance, the authors in [12] assumed that each physical link has a delay of 30 ms. In [13], the authors used the packet loss ratio to reflect network congestion in simulations, but they assumed that the packet loss ratio is a fixed value, such as 0.02 or 0.03.

Some works first make predictions about network parameters and then determine solutions for VNF. The authors in [14] first proposed a traffic forecasting method. Then, they devised two VNF placement algorithms based on the forecast traffic to guide online VNF scaling. While the forecast traffic curve is very close to the real-time traffic curve in most cases, its variation trend is sometimes one time-step later than that of the real-time traffic curve. In our work, we aim to accommodate as many latency-sensitive services as possible with the minimum cost. For a deployed service, if the predicted traffic rate is lower than the real-time traffic rate, then the latency requirement of this service might not be satisfied. Hence, we cannot tolerate prediction error and thus design our algorithm in a stable environment where the traffic characteristics that we are interested in can be learned from traffic history.

Some works, [4, 15, 16] and [17], focus on the convergence rate of their algorithms. In [4], the authors formulated the VNF placement as a binary integer programming model and proposed the service function chains embedding approach (SFC-MAP) algorithm, in which the shortest path algorithm is iterated based on a multi-layer cost graph to acquire the optimal placement solution. The authors of [15] designed an eigendecomposition-based approach to address VNF forwarding graph (VNF-FG) placement. This matrix-based method reduces the complexity and accelerates the convergence. How-

ever, it does not explore the immense space of possible actions. A novel approach that combines Markov approximation with matching theory, sampling-based Markov approximation (SAMA), was proposed in [16] to minimize the joint operational and traffic cost. In [17], the authors formulated the VNF placement and flow routing problems as integer linear programming (ILP) optimization problems and designed a set of heuristics to find near-optimal solutions. In our work, we accelerate the convergence rate by compressing the action space using our resource allocation algorithm (BSAGD). Although the convergence rate is important, cost-effectiveness is more important regarding our objective. Therefore, we focus on the optimal acceptance ratio and cost-utility rather than on the convergence rate.

Other studies, [5, 11] and [18], are based on delay models. In [11], the authors proposed a fine-grained delay model and extended the VNF placement optimization in [10] with delay constraints. The authors in [18] formulated the problem of finding the optimal number of VNFs and their locations as an ILP. Then, they proposed a cost-efficient proactive VNF placement and chaining algorithm to resolve it. The algorithm aims to accept each request with the minimum cost while meeting its latency requirement. However, as in [11], it does not prioritize value-added services based on their latency requirements. Thus, infrastructure with insufficient resources might not ensure priority allocation of high-priority services. Recently, the authors in[5] first formulated the power-aware and delay-constrained joint VNF placement and routing problem as an ILP. Then, they proposed a fast heuristic algorithm — Holu — to resolve it. The authors calculated the end-to-end delay of a VNF chain by summing the propagation delay of each link and the processing delay of each VNF. However, queuing delay and propagation delay might also be considered in some real cases. Therefore, creating or choosing a delay model that achieves an accurate estimation is critical for the VNF placement of latency-sensitive services. In our work, to ensure priority allocation of services with stricter latency requirements, we prioritize requests based on their latency requirements. To ensure an accurate estimation of the end-to-end delay, we design our SADDQN framework based on a mature 5G end-to-end delay model, whose accuracy has been verified in [19].

In summary, heuristics are efficient in stable systems. Nevertheless, in scenarios where environmental characteristics, such as network topology, service requests, and incoming traffic, are prone to change, these approaches must be frequently retriggered to obtain the optimal solution for the new environment. However, our proposed SADDQN algorithm, whose agent interacts with the environment during the training phase, can learn the environment changes before it converges.

2.2. Reinforcement Learning-Based Approaches

Various techniques based on reinforcement learning (RL) have also been developed [6, 20, 21, 22, 23]. The valuable ability to learn from past experience makes these approaches noteworthy. The authors in [21] proposed a distributed reinforcement learning algorithm for VNF-FG allocation, which accelerates the convergence. However, they did not consider whether they should design a dedicated network for communication between agents. In [22], the authors made VNF placement decisions based on end-to-end performance predictions. Although the prediction algorithm performs very well in dynamic conditions, as in [14], it would need to reserve resources for VNFs in case of prediction error. Therefore, developing prediction-based VNF placement solutions is not cost-efficient. Instead of learning from scratch, the authors in [23] trained a DRL agent to learn how to reduce the optimality gap of heuristic-based solutions. For the latency issue, [6] proposed to extract latency in real time, making it very attractive. Adopting the real-time model can greatly improve the accuracy of the model. However, there exist several limits in [6]. First, they used VNF-FGs requested by clients to symbolize traffic in the deep reinforcement learning state. It is important to take VNF-FGs into the state, but it is equally important to include the incoming traffic of each service, an indispensable feature of the environment. This is because, for a given VNF-FG configuration, different incoming traffic may result in different real latency. Second, the authors assumed that the requested resources of VNFs are normalized and distributed uniformly, which could be impractical in some real cases. For instance, in an OpenStack (an open-source cloud computing infrastructure software project) supported cloud environment, there are several available 'flavors'[24] for resources such as VCPU, RAM, and Disk. Therefore, these required resources are discrete random variables rather than continuous random variables.

Most works [6, 12, 13, 4, 15, 16, 18, 5, 21, 22, 23] followed the classical VNE model and directly treated the assumed values of required resources as the input of their approaches. However, in some real cases, customers are familiar with the QoS parameters but not the VNF size. Therefore, in our work, we design a VNF resource allocation algorithm for a service based on its incoming traffic and latency requirement.

3. System Model and Problem Formulation

3.1. Network Infrastructure and Service Request

According to the 3rd Generation Partnership Project (3GPP) view [25, 26], the network slice management function (NSMF) receives requests for allocation of network slices with certain characteristics. Accordingly, it interfaces with management and orchestration (MANO) to plan VNFs allocation using the network function virtualization (NFV) infrastructure. In this paper, the DDQN-agent, which takes the role of MANO, is responsible for choosing the location for VNFs of an incoming request. Once the VNF placement is decided, the decision is delivered to the SDN-controller, which accordingly derives the shortest path in terms of hop count for any two adjacent VNFs in this VNF chain and replies the hop count results back to the DDQN-agent. With the VNF location and the hop count information, the agent assigns resources to each VNF in such a way that the service is realized with the minimum cost.

3.1.1. NFV Infrastructure

As for the infrastructure, we consider a non-blocking architecture whereby every tier is connected to the next tier with

Symbol	Definition
L	Number of service requests
K_i	Number of VNFs in service request i
$\mathbb{S}$	Set of servers in the system infrastructure
N_i	Number of newly activated servers for accommodating service i
λ_i	Packet arrival rate of the i^{th} service
M_{tot}	Total MIPS quantity of a server
W_{tot}	Total bandwidth of a server
$C_{i,j}^{\text{pro}}$	The maximum number of packets that can be processed by the j^{th} VNF of the i^{th} service when all the CPU resources of the host server are allocated to it.
$C_{i,j}^{\text{tran}}$	The maximum number of packets that can be transmitted by the j^{th} VNF of the i^{th} service when all the bandwidth resources of the host server are allocated to it.
m_n	Remaining MIPS quantity of the n^{th} server
w_n	Remaining bandwidth of the n^{th} server
$\mu_{i,j}$	Processing and transmitting rates of the j^{th} VNF of the i^{th} service (in the unit of packets per second)
$x_{i,j}^n$	Whether the j^{th} VNF of the i^{th} request nests on the n^{th} server
$h(s_n, s_{n'})$	Number of hops between server n and server n'
$h(v_j, v_{j+1})$	Number of hops between two adjacent VNFs
D_i^{R}	Latency requirement of the i^{th} service
D_i	Real latency of the i^{th} service
ρ_i	Priority of the i^{th} service
β	Price that converts the traffic cost to a monetary cost
γ	Price that converts the CPU cost to a monetary cost
α	Price that converts the operational cost to a monetary cost

Table 1: Main parameters and symbols

equal aggregate bandwidth [27]. It is facilitated by a topology known as fat-tree topology. We adopt a widely used fat-tree topology, depicted in Fig.1. Let $\mathbb{S}$ denote the set of servers, as given by

$$\mathbb{S} = \{s_n \mid n \in \{1, 2, \ldots, |\mathbb{S}|\}\},$$

where $|\mathbb{S}|$ is the number of servers and s_n indicates the n^{th} server in the infrastructure.

3.1.2. Service Request

Let L denote the number of requests, each of which is composed of an ordered SFC and has a distinctive delay requirement. We assume that, within the URLLC category, there are T types of SFC. Hence, we can prioritize the T types of services and give each of them a priority value. And we define user request i as follows:

$$r_i = (D_i^{\text{R}}, \rho_i, F_i, \lambda_i),$$

where D_i^{R} is the end-to-end delay requirement, λ_i is the requested data rate, ρ_i is the priority value, and $F_i = \{f_{i1}, f_{i2}, \cdots, f_{iK_i}\}$ is the requested SFC, an ordered set of VNFs through which the traffic should be routed.

Without loss of generality, we consider the following example shown in Fig.1. When an SFC (a VNF-FG) request (ingress→VNF1→VNF2→VNF3→VNF4→VNF5→egress) arrives, we suppose that the DDQN-agent chooses server1 to place VNF1 and VNF2, server2 to place VNF3 and VNF4,

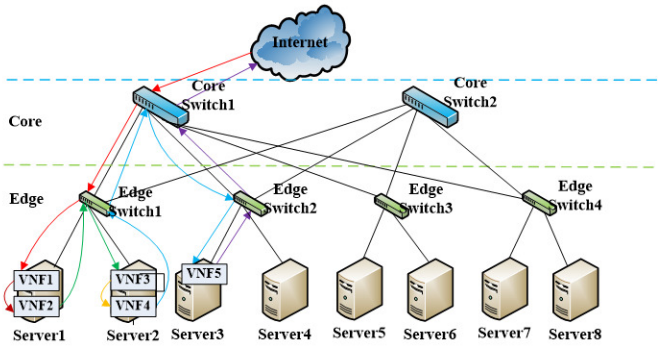

Figure 1: Network infrastructure considered in this paper.

and server3 to place VNF5. Once the incoming traffic stream finishes its journey from the source (ingress) to the destination (egress), we obtain the real delay of this stream to see whether it meets the corresponding latency requirement.

3.2. Traffic Model

We model the packet arrivals of flow i at the first VNF (VNF1) as a Poisson process with arrival rate λ_i. As in [28], we define the time profile for flow i traveling through VNF1 as a two-dimensional time vector $\left[\tau_{i,1}^{\text{pro}}, \tau_{i,1}^{\text{tran}}\right]$. $\tau_{i,1}^{\text{pro}}$ denotes the CPU processing time for a packet in flow i to move through VNF1, when all CPU resources of the host server are allocated to this flow. $\tau_{i,1}^{\text{tran}}$ denotes the transmission time for a packet in flow i to

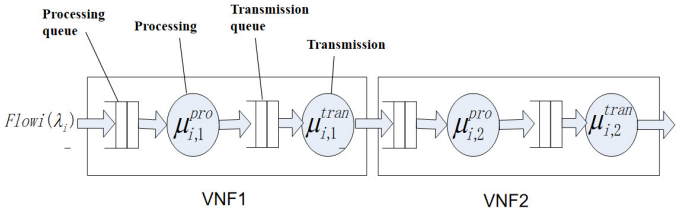

Figure 2: M/D/1 queuing model.

go through the outgoing link of VNF1, when the whole bandwidth resources on the outgoing link of the host are allocated to flow i. Correspondingly, the rate vector $\left[C^{\text{pro}}_{i,1}, C^{\text{tran}}_{i,1}\right]$ (in the unit of packets per second) for flow i traversing VNF1 is the reciprocal of the time profile, $C^{\text{pro}}_{i,1} = 1/\tau^{\text{pro}}_{i,1}$ and $C^{\text{tran}}_{i,1} = 1/\tau^{\text{tran}}_{i,1}$. Let us consider flow i in the M/D/1 model in Fig.2. If $\mu^{\text{tran}}_{i,1} > \mu^{\text{pro}}_{i,1}$, it may lead to resource waste on link transmission; if $\mu^{\text{tran}}_{i,1} < \mu^{\text{pro}}_{i,1}$, packets will accumulate in the transmission queue, causing an increase in the queuing delay [19]. Therefore, to avoid the queuing delay of the transmission queue and bandwidth waste, we need to have

$$\frac{1}{\mu^{\text{pro}}_{i,1}} = \frac{1}{\mu^{\text{tran}}_{i,1}} = \frac{1}{\mu_{i,1}}. \tag{1}$$

To achieve (1), we have to allocate CPU and bandwidth resources to flow i passing through VNF1 according to the percentage $C^{\text{tran}}_{i,1}/C^{\text{pro}}_{i,1}$ or $C^{\text{pro}}_{i,1}/C^{\text{tran}}_{i,1}$. For simplicity, we define a resources package C_{tot}, as given by

$$C_{\text{tot}} = \begin{cases} \left\{M_{\text{tot}}, \frac{C^{\text{pro}}_{i,1}}{C^{\text{tran}}_{i,1}} W_{\text{tot}}\right\}, & C^{\text{tran}}_{i,1} > C^{\text{pro}}_{i,1} \\ \left\{\frac{C^{\text{tran}}_{i,1}}{C^{\text{pro}}_{i,1}} M_{\text{tot}}, W_{\text{tot}}\right\}, & C^{\text{tran}}_{i,1} \le C^{\text{pro}}_{i,1}, \end{cases} \tag{2}$$

where M_{tot} and W_{tot} indicate the total MIPS and bandwidth of a server, respectively. When we need to allocate MIPS and bandwidth resources to a VNF, we fetch a percentage of C_{tot}.

3.3. Delay Model

When a traffic stream goes through an embedded VNF chain, the end-to-end packet delay should be calculated by summing the packet queuing delay and packet processing delay on all intermediate VNFs and the packet transmission delay on all links [19].

In [19], the authors decoupled the packet processing of different flows traveling through an NFV node and thus regarded the average packet processing rate of each flow as an approximated service rate. They also applied the theory to the transmission rate. Accordingly, they developed an M/D/1 queuing model to calculate packet delay at the first NFV node for each flow. Based on the analysis of packet inter-arrival time at the subsequent NFV node, they further adopted an M/D/1 queuing model to evaluate the average packet delay for each flow at the subsequent NFV node.

Due to the slice isolation requirements, in our work, we consider unshared VNFs [16, 29], which means two or more chains cannot share a VNF. This solution logically separates the traffic of all network sub-slices and decouples all flows. Therefore, it is reasonable for us to apply the delay model in [19] to our system.

For the first VNF, the average packet delay of flow i is determined by

$$D_{i,1} = \frac{1}{\mu^{\text{pro}}_{i,1}} + \frac{\lambda_i}{2\left(\mu^{\text{pro}}_{i,1}\right)^2 (1-\rho_{i,1})} + \frac{1}{\mu^{\text{tran}}_{i,1}}, \tag{3}$$

where the first term is the average processing delay, the second term is the average queuing delay of the processing queue [19], and the third term is the average transmission delay. The utilization $\rho_{i,1}$ is given by

$$\rho_{i,1} = \frac{\lambda_i}{\mu^{\text{pro}}_{i,1}}, \tag{4}$$

where λ_i is the packet arrival rate and $\mu^{\text{pro}}_{i,1}$ is the processing rate.

For the j^{th} $(j > 1)$ VNF that flow i travels through, the average packet delay is expressed as

$$D_{i,j} = \begin{cases} \frac{1}{\mu^{\text{pro}}_{i,j}} + \frac{\lambda_i}{2\left(\mu^{\text{pro}}_{i,j}\right)^2 (1-\rho_{i,j})} + \frac{1}{\mu^{\text{tran}}_{i,j}}, & \mu^{\text{tran}}_{i,j-1} > \mu^{\text{pro}}_{i,j} \\ \frac{1}{\mu^{\text{pro}}_{i,j}} + \frac{1}{\mu^{\text{tran}}_{i,j}}, & \mu^{\text{tran}}_{i,j-1} \le \mu^{\text{pro}}_{i,j}, \end{cases} \tag{5}$$

where $1/\mu^{\text{pro}}_{i,j}$ is the average processing delay, $\lambda_i/2\left(\mu^{\text{pro}}_{i,j}\right)^2 (1-\rho_{i,j})$ is the average queuing delay of the processing queue, and $1/\mu^{\text{tran}}_{i,j}$ is the transmission delay.

In (5), if the transmission rate of the $(j-1)^{\text{th}}$ VNF is lower than or equal to the processing rate of the j^{th} VNF, then there will be no queuing delay of the processing queue on the j^{th} VNF, as given in the second segment. Otherwise, in the first segment, we have the average queuing delay of the processing queue based on M/D/1 queue theory [19].

As in (1), we let the processing rate equal the transmission rate to eliminate the queuing delay of the transmission queue:

$$\frac{1}{\mu^{\text{pro}}_{i,j}} = \frac{1}{\mu^{\text{tran}}_{i,j}} = \frac{1}{\mu_{i,j}}, \qquad j > 1. \tag{6}$$

If the j^{th} and the $(j+1)^{\text{th}}$ VNFs are located on the same server, traveling from the j^{th} VNF to the $(j+1)^{\text{th}}$ VNF, the flow will suffer no forwarding delay caused by switches. Otherwise, the flow may go through a sequence of network switches and physical links. In our work, as in [19] and [30], we assume that the processing and transmission rates allocated to flow i from these switches are the same as the transmission rate of the j^{th} VNF to maximize resource utilization. Therefore, the queuing delay, either of the transmission queue or of the processing queue, does not need to be considered on these switches. Consequently, to go from the j^{th} VNF to the $(j+1)^{\text{th}}$ VNF, the total packet delay for flow i traversing n_j switches is given by

$$D^{\text{f}}_{i,j} = \frac{2n_j}{\mu^{\text{tran}}_{i,j}}, \qquad 1 \le j \le K_i. \tag{7}$$

In general, the average end-to-end delay of flow i traveling through an embedded VNF chain, consisting of K_i intermediate VNFs, is the total of the average delay for packets to move through all K_i VNFs and the average delay on the switches and links along the path [19], as given by

$$D_i = \sum_{j=1}^{K_i} D_{i,j} + \sum_{j=1}^{K_i} D_{i,j}^{\mathrm{f}}. \tag{8}$$

3.4. Cost Model

3.4.1. Operational Cost

We define the operational cost as the total power consumption of all active nodes [16], expressed by $\alpha \sum_{i=1}^{|S|} Q_i$, where α is the price that converts the power to a monetary cost and Q_i is the active power of server i. The power of active node Q_i is always set to 250 Watts uniformly [16]. Therefore, we simplify the operational cost in the system as follows:

$$c_{\mathrm{ope}} = \sum_{i=1}^{L} c_{\mathrm{ope},i} = \sum_{i=1}^{L} \alpha \times 250 \times N_i, \tag{9}$$

where N_i is the number of newly activated servers for accommodating service i.

3.4.2. Traffic Cost

In a service chain, a VNF needs to forward packets to the next VNF through the virtual link connecting them. These virtual connections are embedded on the active physical links of nodes. Therefore, the hop count between two adjacent VNFs depends on the network topology [31]. We use $h\left(v_j, v_{j+1}\right)$ to indicate the hop count between two adjacent VNFs.

We define the network traffic cost in the same way as in [21] and [31], calculating the traffic cost of two adjacent VNFs based on the hop count between them and the allocated bandwidth of the virtual link. Hence, the total network traffic cost can be written as

$$\begin{aligned} c_{\mathrm{tran}} &= \sum_{i=1}^{L} c_{\mathrm{tran},i} \\ &= \beta \sum_{i=1}^{L} \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\mathrm{tran}}} \times W_{\mathrm{tot}} \times h\left(v_j, v_{j+1}\right), \end{aligned} \tag{10}$$

where

$$h\left(v_j, v_{j+1}\right) = \sum_{n=1}^{|\mathbb{S}|} \sum_{n'=1}^{|\mathbb{S}|} x_{i,j}^{n} \times x_{i,j+1}^{n'} \times h\left(s_n, s_{n'}\right). \tag{11}$$

In (10), β is the price that converts the traffic cost to a monetary cost. In (11), the binary variable $x_{i,j}^{n}$ indicates whether the j^{th} VNF of the i^{th} request nests on the n^{th} server, $x_{i,j+1}^{n'}$ denotes whether the $(j+1)^{\mathrm{th}}$ VNF of the i^{th} request nests on the n'^{th} server, and $h\left(s_n, s_{n'}\right)$ is the hop count between server n and server n'.

3.4.3. Server Cost

Compared with memory and capacity, CPU's computational power is much more important for latency-sensitive services. Therefore, as for server cost, we consider only CPU resources in million instructions per second (MIPS), and the total server cost can be written as

$$c_{\mathrm{ser}} = \sum_{i=1}^{L} c_{\mathrm{ser},i} = \gamma \sum_{i=1}^{L} \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\mathrm{pro}}} \times M_{\mathrm{tot}}, \tag{12}$$

where γ is the price that converts the server cost to a monetary cost.

3.5. Problem Formulation

In this part, we formulate the objective as an optimization problem. The purpose is to maximize the number of accepted higher-priority service requests while minimizing the total cost, under infrastructure resource constraints.

The weighted sum of the accepted requests can be defined as:

$$u = \sum_{i=1}^{L} u_i = \sum_{i=1}^{L} \rho_i \times f(D_i^R - D_i). \tag{13}$$

In the function above,

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0, \end{cases} \tag{14}$$

D_i^{R} and D_i, respectively, denote the latency requirement and the real latency of the i^{th} service and ρ_i indicates the priority of service i.

The total cost can be calculated as:

$$c = \sum_{i=1}^{L} c_i = \sum_{i=1}^{L} (c_{\mathrm{ser},i} + c_{\mathrm{tran},i} + c_{\mathrm{ope},i}). \tag{15}$$

Thus, the optimization problem can be written as

$$(P1): \max_{x_{i,j}^{n}, \mu_{i,j}} \eta_1 u - \eta_2 c$$

$$\text{s.t.} \begin{cases} \displaystyle\sum_{i=1}^{L} \sum_{j=1}^{K_i} x_{i,j}^{n} \frac{\mu_{i,j}}{C_{i,j}^{\mathrm{pro}}} \leq 1, \ n = 1, 2, \ldots, |\mathbb{S}| & \text{(16a)} \\ \displaystyle\sum_{i=1}^{L} \sum_{j=1}^{K_i} x_{i,j}^{n} \frac{\mu_{i,j}}{C_{i,j}^{\mathrm{tran}}} \leq 1, \ n = 1, 2, \ldots, |\mathbb{S}| & \text{(16b)} \\ \displaystyle\sum_{n=1}^{|\mathbb{S}|} x_{i,j}^{n} = 1, \ i = 1, 2, \ldots, L, j = 1, 2, \ldots, K_i & \text{(16c)} \\ x_{i,j}^{n} \in \{0, 1\}, \\ i = 1, 2, \ldots, L, j = 1, 2, \ldots, K_i, n = 1, 2, \ldots, |\mathbb{S}| \\ \mu_{i,j} \in \mathbb{N}_+, i = 1, 2, \ldots, L, j = 1, 2, \ldots, K_i. \end{cases}$$

In the objective function above, η_1 and η_2 are the weights of two objectives. Since the objective of maximizing the number of accepted higher-priority services is more important, we assign a higher weight to it.

In the constraints above, the binary variable $x_{i,j}^n$ indicates whether the j^{th} VNF of the i^{th} request nests on the n^{th} server. For any server in the infrastructure, the total MIPS allocated to the embedded VNFs cannot exceed the CPU capacity of the server, so we have (16a), and the total bandwidth allocated to the embedded VNFs cannot exceed the maximum bandwidth of the server's outgoing link, so we have (16b). Each VNF can be deployed at only one server, so we have (16c).

It can be difficult to solve this optimization problem using heuristic approaches because heuristics can hardly trace the dynamics of the network. Alternatively, learning-based techniques can be utilized to learn the network dynamics, such as the arrival pattern of various services and the changing topology of the infrastructure, and solve such a problem. The goal of the learning technique is to learn a policy that determines what action to take in each environment state. In the following, we introduce a model based on DDQN [32] for the joint VNF-FG placement and resource allocation problem regarding the latency requirement.

4. Proposed VNF-FG Placement and Resource Allocation Scheme

4.1. Overview of Our Proposed SADDQN Algorithm

For our VNF-FG embedding problem, an agent handles the service requests one after another. To achieve its objective, the agent learns how to behave in the environment, the state of which consists of the available resources provided by the infrastructure and the characteristics of the service request to be processed, by performing actions and observing the results. Specifically, for a single request (VNF-FG), the agent sequentially fulfills three tasks. First, it chooses a VNF placement (main action), the locations of all the VNFs in this VNF-FG. Second, it employs the Dijkstra algorithm to derive the optimal path traversing the VNFs placed by the first task (the first-phase sub-action). Third, it utilizes our proposed resource allocation algorithm — BSAGD (the second-phase sub-action) — to allocate resources to all VNFs in such a way that the service request is realized with the minimum cost.

By performing the joint action (VNF placement, its corresponding optimal path and resource allocation solution) for the current request to the environment, the agent will receive a reward, which will be utilized to improve its action (VNF placement) for subsequent requests. Meanwhile, the state of the environment will be updated, and the action for the next request will be determined. Hence, we can model the VNF-FG embedding for all requests as a Markov decision process (MDP). Because one VNF placement has only one corresponding optimal path and resource allocation solution, we represent the MDP action space as the possible VNF placements of a single request. Since our aim is to maximize the number of accepted higher-priority services while minimizing the total cost, we prioritize those requests based on their latency requirements and define the reward function of the MDP based on service priority and resource cost. The details of the state, action, and reward function in the context of our problem are provided in Sections IV-C, IV-D, and IV-E.

We propose a sub-action aided DDQN (SADDQN) algorithm to resolve our VNF-FG embedding problem. A general overview of the SADDQN used in our scheme is shown in Fig. 3. The state, indicated by a vector, is fed into the evaluation neural network, which outputs a vector of Q-values, with each indicating the expected discounted cumulative reward of a corresponding action (VNF placement). At time step t, the Q-value of performing action a_t under state s_t based on policy π is given by

$$Q^{\pi}(s_t, a_t) = E(\sum_{i=t}^{L} \gamma^{(i-t)} R(s_i, a_i) | s_t, a_t). \tag{17}$$

The objective of the agent is to learn a policy that maximizes the expected return $Q^{\pi}(s_t, a_t)$. Once the optimal policy is achieved, given a state, the agent can find the best action by taking the largest Q-value from the output vector. At the beginning of training, the weights of the evaluation neural network are random; thus, the policy is poor. For each given state, the maximum Q-value may not account for the best VNF placement. Hence, we continuously feed the framework requests, and the agent iteratively optimizes the neural networks. Specifically, at time step t, by performing the joint action (VNF placement and its corresponding optimal sub-action) for a service request to the environment, the agent receives a reward that is used to conduct the back-propagation process and update the weights of the evaluation network. When the next request arrives, this training process is iterated in the next time step. The loop ends when the weights of the neural networks converge. Then, given a state, the agent can choose the optimal VNF placement according to the output Q-values of the evaluation network. Additionally, the corresponding optimal routing and resource allocation solution can be obtained from the routing and BSAGD modules, given the VNF placement. The iterative training process is summarized in Algorithm 1.

Algorithm 1 Sub-action Aided DDQN Algorithm

Input: $\lambda_1, \lambda_2, \ldots, \lambda_L, D_1^R, D_2^R, \ldots, D_L^R$

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function Q̂ with weights θ⁻
θ⁻ = θ
for episode = 1, 2, ..., I do
    Initialize state of the environment
    for t = 1, 2, ..., L do
        Workflow of each learning step
    end for
end for
```

The workflow for each learning step can be summarized as follows:

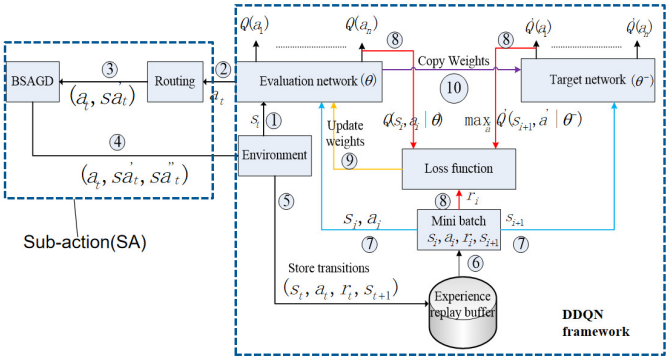

Figure 3: Flowchart of the SADDQN algorithm. Upon receiving a service request, the SADDQN maps the requested VNF-FG to the infrastructure after sequentially deciding the VNF placement for all VNFs, optimal path traversing the VNF chain, and resource allocation for all VNFs. First, the agent chooses the locations for all VNFs using the DDQN framework. Next, based on the VNF sequence in the VNF-FG, the routing module derives the shortest path traversing all the VNFs placed by the previous step. Finally, with the given VNF placement and the shortest path, the BSAGD module allocates resources to all VNFs in such a way that the end-to-end delay requirement of this service is satisfied with the minimum cost. The agent will then perform the joint action (the VNF placement, routing, and resource allocation) to the environment. In response to the joint action, a reward is returned to train the DDQN framework. The training process is iterated when a new request arrives. Once trained, given an environment, the agent knows how to perform the best action, and the SADDQN model approximates the optimal solution of our objective.

a) Step 1: The agent observes the state (s_t) of the environment.

b) Step 2: The agent chooses a main action (a_t), i.e., the VNF placement, randomly with probability ϵ or according to the evaluation network with probability $1-\epsilon$. Next, it delivers a_t (the chosen VNF placement) to the routing module, which derives an optimal path ($sa_t^{'}$) traversing all the VNFs based on the VNF sequence.

c) Step 3: a_t and $sa_t^{'}$ (the chosen VNF placement and its corresponding optimal path) are fed into the BSAGD module to obtain a resource allocation solution ($sa_t^{''}$) that realizes the service with the minimum cost.

d) Step 4: a_t, $sa_t^{'}$ and $sa_t^{''}$ (the chosen VNF placement, its corresponding optimal routing and resource allocation solution) are jointly applied to the environment.

e) Step 5: The agent receives a reward (r_t) from the environment, and the experience tuple (s_t, a_t, r_t, s_{t+1}) is stored in an experience replay buffer. Here, we need to add only the main action (a_t) to the experience tuple because we have only one corresponding optimal sub-action for any main action.

f) Step 6: To train the DDQN framework, a mini-batch of N tuples is uniformly sampled from the experience replay buffer.

g) Step 7: For tuple i, s_i and a_i are fed into the evaluation network (θ), while s_{i+1} is fed into the target network (θ^-).

h) Steps 8, 9: A loss function is created, and the weights of the evaluation network are updated by minimizing the loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\theta))^2, \tag{18}$$

where

$$y_i = r_i + \gamma max_{a'} Q'(s_{i+1}, a'|\theta^-). \tag{19}$$

i) Step 10: The weights of the target network are updated by copying the weights of the evaluation network every Z time steps.

4.2. State

We define the state as a vector, including the latency requirement, VNF chain, and requested traffic of a service request and the remaining resources of each server. In our environment, the number of intermediate VNFs in service i is K_i. Then, the characteristic vector of service i can be given by

$$\boldsymbol{r}_i = \begin{bmatrix} \lambda_i & D_i^R & I_1 & I_2 & \cdots & I_{K_i} \end{bmatrix},$$

where λ_i and D_i^{R} denote the requested traffic and the delay requirement of the i^{th} service and $I_1, I_2, \ldots, I_{K_i}$ indicate the ID of the first, second, …, K_i^{th} VNF in the VNF chain. For the network infrastructure, the characteristic vector of the n^{th} server can be expressed as

$$\boldsymbol{s}_n = \begin{bmatrix} m_n & w_n \end{bmatrix},$$

where m_n and w_n denote the remaining MIPS and bandwidth of the n^{th} server.

As a result, the state vector of the environment is written as

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{r}_i & \boldsymbol{s}_1 & \cdots & \boldsymbol{s}_{|\mathbb{S}|} \end{bmatrix}.$$

4.3. Action

4.3.1. Main Action (VNF Placement)

We use a $1 \times K_i$ row vector $\boldsymbol{a}_i$ to symbolize a VNF placement for the i^{th} request. Element $a_{i,j}$ symbolizes on which server the j^{th} VNF of the i^{th} request nests. For the request in Fig. 1, we use a 1×5 row vector, $\boldsymbol{a}_1 = \begin{bmatrix} 1 & 1 & 2 & 2 & 3 \end{bmatrix}$, to indicate its main action. This vector indicates that the first and the second VNFs are located on server1, the third and fourth VNFs are located on server2, and the fifth VNF is located on server3.

4.3.2. First-Phase Sub-Action

With a given VNF placement, to minimize the resource cost, we must find the shortest path traversing the VNF chain. Therefore, the Dijkstra algorithm, which finds the shortest path in terms of hop count, is adopted to connect adjacent VNFs. Instead of running the Dijkstra algorithm from the ingress node to the egress node, we run the Dijkstra algorithm from the ingress node to the first VNF, from the first VNF to the second VNF, and so on, until the egress node.

4.3.3. Second-Phase Sub-Action (Binary Search Assisted Gradient Descent (BSAGD))

For a VNF-FG, after determining the locations for all VNFs and the corresponding optimal path traversing the VNF chain, we must allocate resources to all VNFs such that this service's

latency requirement is satisfied with the minimum cost. In general, to find the minimum cost for realizing service i, there are two steps to take. In the first step, we prove that the minimum delay $D_i^{\min}(c_i)$ is a monotone decreasing function of variable c_i. In the second step, we use our BSAGD algorithm, a binary search algorithm in which gradient descent is iterated, to search for the minimum cost and the corresponding resources of each VNF ($\mu_{i,j}$) to satisfy the delay requirement of service i. Because $\mu_{i,j}$ is an integer, we adopt the relax-and-round mechanism to obtain the optimal integer solution $(\widetilde{\mu}_{i,j}, j = 1, \ldots, K_i)$ for realizing service i.

Based on (1), (3), (4), (5), (6), (7), and (8), the problem of minimizing the average delay with limited cost c_i can be formulated as follows:

$$
\begin{aligned}
(P2): \min_{\boldsymbol{\mu}_i} D_i(\boldsymbol{\mu}_i) = {} & \frac{\lambda_i}{2\left(\mu_{i,1}\right)^2\left(1-\frac{\lambda_i}{\mu_{i,1}}\right)} + \sum_{j=1}^{K_i}\frac{2}{\mu_{i,j}} \\
& + \sum_{j=1}^{K_i}\frac{2(h(v_j, v_{j+1})-1)}{\mu_{i,j}} g(h(v_j, v_{j+1})) \\
& + \sum_{j=2}^{K_i}\frac{\lambda_i}{2\left(\mu_{i,j}\right)^2\left(1-\frac{\lambda_i}{\mu_{i,j}}\right)} g(\mu_{i,j-1}-\mu_{i,j})
\end{aligned}
$$

$$
\text{s.t.}\begin{cases}
\mu_{i,j} \le \mu_{i,j+1} \text{ or } \mu_{i,j+1} \le \mu_{i,j},\ j = 1, \ldots, K_i - 1 & \text{(20a)} \\
\gamma \sum_{j=1}^{K_i}\frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} + \beta \sum_{j=1}^{K_i}\frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} & \\
\times h\left(v_j, v_{j+1}\right) + \alpha \times 250 \times N_i = c_i & \text{(20b)} \\
\sum_{j=1}^{K_i} x_{i,j}^n \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} \le m_n, & \\
n = 1, \ldots, |\mathbb{S}| & \text{(20c)} \\
\sum_{j=1}^{K_i} x_{i,j}^n \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} \le w_n, & \\
n = 1, \ldots, |\mathbb{S}|. & \text{(20d)}
\end{cases}
$$

In the objective function above,

$$
g(x) = \begin{cases} 0, & x \le 0 \\ 1, & x > 0. \end{cases} \tag{21}
$$

In the constraints above, (20a) is a possible condition of $\mu_{i,j}$ and $\mu_{i,j+1}$, $j = 1, \ldots, K_i - 1$. We provide VNF chain i with limited cost c_i, so we have (20b). In (20b), the first term is the total CPU cost of all VNFs. The second term is the traffic cost, which depends on the hop count of each pair of adjacent VNFs. The third term is the operational cost, a linear function of newly activated servers. For any server in the infrastructure, the total MIPS allocated to the embedded VNFs cannot exceed the remaining CPU MIPS of the server, so we have (20c), and the total bandwidth allocated to the embedded VNFs cannot exceed the remaining bandwidth of the server's outgoing link, so we have (20d).

Proposition 1. *The minimum end-to-end delay expression in terms of variable c_i is a monotone decreasing function of c_i.*

Proof. The proof is provided in Appendix A.

Proposition 2. *P2 is a convex optimization problem.*

Proof. The proof is provided in Appendix B.

Since P2 is a convex optimization problem, given a constant cost C_i, we can use gradient descent to find the minimum average delay $D_i^{\min}(C_i)$. Furthermore, $D_i^{\min}(c_i)$ is a monotone decreasing function of variable c_i. Therefore, for realizing service i, we can use a binary search algorithm in which gradient descent is iterated to find the minimum cost and the corresponding optimal resource allocation for all VNFs.

Our BSAGD algorithm is summarized in Algorithm 2.

Algorithm 2 BSAGD Algorithm

Input: $C_i^{\min}, C_i^{\max}, C_{\text{unit}}$
Output: $v_i^{\min}, \boldsymbol{\mu}_i$

$t_i^{\min} = C_i^{\min}, v_i^{\min} = 0, t_i^{\max} = C_i^{\max}$
$i = 1, \boldsymbol{\mu}_i = \mathbf{0}$
while $v_i^{\min} - t_i^{\max} > C_{\text{unit}}$ or $i = 1$ **do**
  $i = i + 1$
  Substituting $t_i^{\max}$ into P2 and using gradient descent to find the minimum delay and the corresponding $\boldsymbol{\mu}_i$
  if $D_i^{\min}(t_i^{\max}) < D_i^{\text{R}}$ **then**
    $v_i^{\min} = t_i^{\max}$
    $t_i^{\max} = \frac{t_i^{\max} + t_i^{\min}}{2}$
    update $\boldsymbol{\mu}_i$
  else if $D_i^{\min}(t_i^{\max}) > D_i^{R}$ **then**
    $t_i^{\min} = t_i^{\max}$
    $t_i^{\max} = \frac{t_i^{\min} + v_i^{\min}}{2}$
  else
    $v_i^{\min} = t_i^{\max}$
    update $\boldsymbol{\mu}_i$
  end if
end while

return $v_i^{\min}, \boldsymbol{\mu}_i$

Given a VNF placement and a condition of $\mu_{i,j}$ and $\mu_{i,j+1}$, $j = 1, \ldots, K_i - 1$, we can obtain the unit cost C_{unit}, the maximum cost ($C_i^{\max}$), and the minimum cost ($C_i^{\min}$). $C_i^{\max}$ and $C_i^{\min}$ are used as the initial two ends of the binary search algorithm. Let tentative maximum cost $t_i^{\max} = C_i^{\max}$ and tentative minimum cost $t_i^{\min} = C_i^{\min}$. If $C_i^{\max}$ cannot obtain a delay shorter than or equal to requirement D_i^{R}, there is no solution for this placement under this condition. Otherwise, we store $C_i^{\max}$ in verified minimum cost $v_i^{\min}$ and update the rate vector $\boldsymbol{\mu}_i$. Then, we go to the middle point if $C_i^{\max}$ has a delay shorter than D_i^{R}. If the middle point cannot obtain a delay shorter than or equal to D_i^{R}, we update $t_i^{\min}$ and $t_i^{\max}$ accordingly and continue searching in the upper half. Otherwise, we store this middle point in $v_i^{\min}$ and update $\boldsymbol{\mu}_i$. If the delay of this middle point is shorter than D_i^{R}, we update $t_i^{\max}$ accordingly and continue searching in the lower

half. When the difference between $v_i^{\min}$ and $t_i^{\max}$ is less than the unit cost C_{unit} or the current $v_i^{\min}$ has a delay equal to D_i^{R}, there is no need to continue looping, and we return $v_i^{\min}$ and $\boldsymbol{\mu}_i$ for realizing service i. For any possible condition of $\mu_{i,j}$ and $\mu_{i,j+1}$, $j = 1, \ldots, K_i - 1$, we can obtain a $v_i^{\min}$ and the corresponding $\boldsymbol{\mu}_i$. The minimum of these verified minimum cost values is the minimum cost for realizing service i, given a VNF placement.

4.4. Reward

A performance model must be created to assess whether the latency requirement has been achieved with the given resources (MIPS, bandwidth) and the current workload. In our work, we first leverage the simulation tool CloudSimSDN-NFV [33] to obtain the latency of each service. Then, we issue a penalty for a service whose latency requirement is not satisfied and a reward for a successful accommodation. For the i^{th} service, the reward function regarding delay is defined as

$$R_{\text{delay},i} = \begin{cases} \rho_i, & D_i \le D_i^R \\ 0, & D_i > D_i^R, \end{cases} \tag{22}$$

where D_i is the average delay of the i^{th} service we obtain from CloudSimSDN-NFV.

Regarding the cost, we define reward functions for operational cost, server cost, and traffic cost of service i as follows:

$$R_{\text{ope},i} = \alpha \times 250 \times N_i, \tag{23}$$

$$R_{\text{ser},i} = \gamma \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}}, \tag{24}$$

$$R_{\text{tran},i} = \beta \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} \times h\left(v_j, v_{j+1}\right). \tag{25}$$

As in [16], α, β, and γ can be adjusted to change the impact factor of each category.

Since our objective is to maximize the number of accepted higher-priority service requests while minimizing the total cost, the reward function of service i is expressed as a weighted sum of the delay reward and cost reward functions

$$R_i = \eta_1 R_{\text{delay},i} - \eta_2 (R_{\text{ope},i} + R_{\text{ser},i} + R_{\text{tran},i}). \tag{26}$$

Maximizing the acceptance ratio is our top priority; therefore, we assign a greater weight to delay rewards.

5. Simulation Results

5.1. Simulation Setup

In this section, we evaluate the performance of the proposed SADDQN algorithm regarding the admission ratio and the cost efficiency under different network sizes. We use CloudSimSDN-NFV, an NFV environment simulation tool extended from CloudSimSDN [34] and CloudSim [35], as our simulation framework. To merge the DDQN algorithm into CloudSimSDN-NFV, we import deeplearning4j and nd4j written in JAVA. For the infrastructure, we consider fat-tree, a widely used network topology for data centers (Fig. 1). Two scenarios, i.e., 8-Node Fat-Tree and 16-Node Fat-Tree, are utilized to assess our algorithms. The bandwidth of each link is 1 Gbps, and the total MIPS of each server is 3000. For the cost model, we set $\alpha = 1$ \$/W, $\gamma = 0.1$ \$/MIPS, $\beta = 0.1$ \$/Mbps.

In our experiment, we assume there are three types of latency-sensitive services, the characteristics of which are summarized in Table II. Additionally, the rate vectors of all VNFs for three different packet types are provided in Table III.

The neural network is set up with the following parameters. Adam [36] is adopted to learn the neural network parameters. The learning rate is 0.005, and the discount factor is 0.99. The parameters of the target network are updated every 100 episodes, and the batch size is 64. We use a fully connected deep neural network (DNN), which involves hyperbolic tangent and rectified linear unit (ReLu) [37] as activation function in the middle layer, and the output layer is connected to a linear activation function [38].

Regarding the traffic, we assume that, for any service, the packet arrival process can be modeled as a Poisson process, with parameter λ indicating the average arrival rate. According to traffic history, λ of a request service is uniformly distributed in the data rate interval of this service type in Table II.

Our proposed SADDQN algorithm consists of an offline training phase and an online testing phase. In the training phase, we randomly choose a service type and its requested data rate from Table II to create a service request. We continuously feed our SADDQN framework with the service requests until the weights of neural networks converge.

During the online testing phase, we can run the trained SADDQN model online to optimize the VNF placement, as well as its corresponding route and resource allocation, for any given state.

5.2. Algorithms to Compare

5.2.1. Standard DDQN

For comparison purposes, we have designed a standard DDQN algorithm based on a popular assumption: the required resources of each VNF have been specified in service requests. The only difference between the standard DDQN and our proposed SADDQN is that the standard one assumes that the capacity of each VNF has been specified by customers, but our proposed SADDQN allocates resources to each VNF for customers. For the sake of fairness, for any algorithm, we assume that VNFs are embedded in containers, for which CPU request can be as small as 1/1000 of the total CPU resources. For the standard DDQN, according to the number of vCPUs of those 'flavors', we have $\frac{1}{64}$, $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, and the whole of the CPU resources of a server for any container/VNF to choose in our simulation.

5.2.2. Holu

The Holu algorithm [5] addresses the VNF placement and routing problem with the objective of minimizing the number

Table 2: Overview of three service types.

Service type	Delay requirement	SFC	Packet size	Data rate
SFC1	20 ms	VNF1→VNF2→VNF3	4000 bits	[100-900] packets/s
SFC2	30 ms	VNF1→VNF2→VNF4→VNF6	16000 bits	[100-200] packets/s
SFC3	40 ms	VNF1→VNF2→VNF5→VNF2→VNF1	20000 bits	[1000-2000] packets/s

Table 3: Rate vectors of all VNFs for three packet types.

Packet size / VNF type	4000 bits	16000 bits	20000 bits
VNF1	[125000,250000] packets/s	[125000,62500] packets/s	[80000,50000] packets/s
VNF2	[125000,250000] packets/s	[125000,62500] packets/s	[80000,50000] packets/s
VNF3	[125000,250000] packets/s	–	–
VNF4	–	[125000,62500] packets/s	–
VNF5	–	–	[80000,50000] packets/s
VNF6	–	[125000,62500] packets/s	–

of online physical machines (PMs) and network switches under end-to-end delay and resource constraints. This fast heuristic framework efficiently solves the power-aware and delay-constrained joint VNF placement and routing (PD-VPR) problem in an online manner. Specifically, Holu decomposes the PD-VPR into two sub-problems and solves them sequentially: i) a VNF placement problem that maps VNFs to PMs using a centrality-based PM ranking strategy and ii) a routing problem that efficiently splits the delay budget between consecutive VNFs in the SFC and finds a delay-constrained least-cost (DCLC) shortest-path through the selected PMs using the Lagrange relaxation-based aggregated cost (LARAC) algorithm. Although the power consumption model of Holu is different from the cost model of our proposed algorithm, the objectives of the two algorithms (reducing costs while satisfying the delay constraint of each SFC) are the same. Therefore, it is reasonable to run Holu in our system model and compare it with our proposed SADDQN. In our simulation, as in [5], we fix the CPU capacity for each VNF for the Holu algorithm. Based on the composition of three SFCs, we set the CPU capacity of VNF1 and VNF2 to $\frac{1}{4}$ of a server's CPU capacity and that of the other VNFs to $\frac{1}{64}$.

5.3. Simulation Results

We compare our proposed SADDQN, the standard DDQN and Holu with respect to the acceptance ratio, cost-utility and average end-to-end delay. To see how granularity in resource allocation impacts the performance of the standard DDQN, we set the CPU parameter of the minimum 'flavor' to $\frac{1}{32}$ (DDQN1) and $\frac{1}{64}$ (DDQN2). In other words, on a server, the minimum percentage of CPU resource that DDQN1 can allocate to a VNF is $\frac{1}{32}$, while it is $\frac{1}{64}$ for DDQN2.

From Fig. 4, we can see that as the number of requests increases, SADDQN always obtains the highest acceptance ratio among all algorithms in both topologies. That is because our proposed SADDQN algorithm uses the resource optimization algorithm (BSAGD) to allocate resources to VNFs, achieving finer granularity than that of any other algorithm. The standard

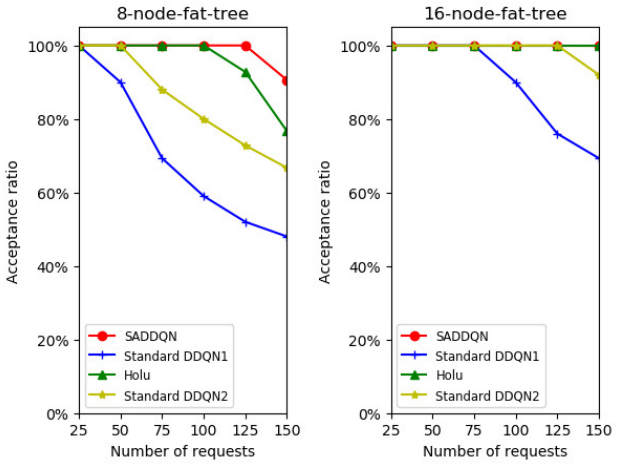

Figure 4: Acceptance ratio comparison.

DDQN, which can only choose the CPU capacity for VNFs based on 'flavors', excessively provisions higher-priority services and thus does not have sufficient resources to accommodate lower-priority services when we increase the number of service requests. DDQN2 outperforms DDQN1 because of the CPU parameter of the minimum 'flavor': the smaller it is, the more requests the standard DDQN accepts. The Holu algorithm fixes the CPU capacity for each type of VNF and allows different services to share VNFs. To some extent, the sharing policy alleviates the over-provisioning issue. However, the problem remains, and the severity depends on the sharing percentage of each VNF. For instance, if we initiate a new instance of a VNF for a service request and no subsequent services share this VNF instance, its resource utility will degrade.

Fig. 5 indicates the accommodation results of different services. It depicts the situation of the 8-node fat-tree topology when the number of requests is 150. The DDQN-based algorithms ensure priority allocation for higher-priority services when the resources are insufficient to accept all requests. Of

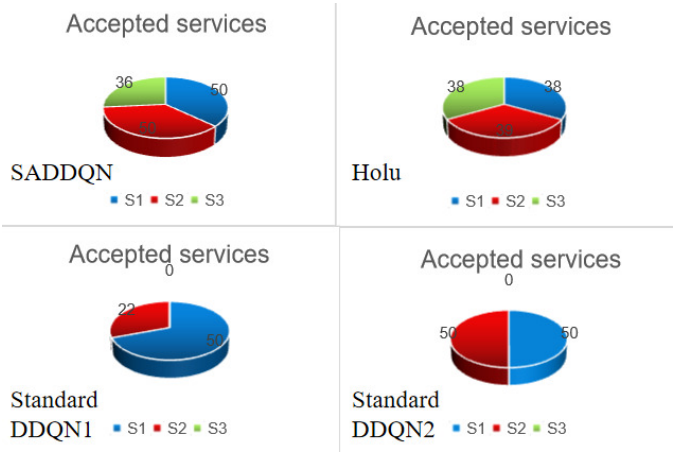

Figure 5: Acceptance of different services.

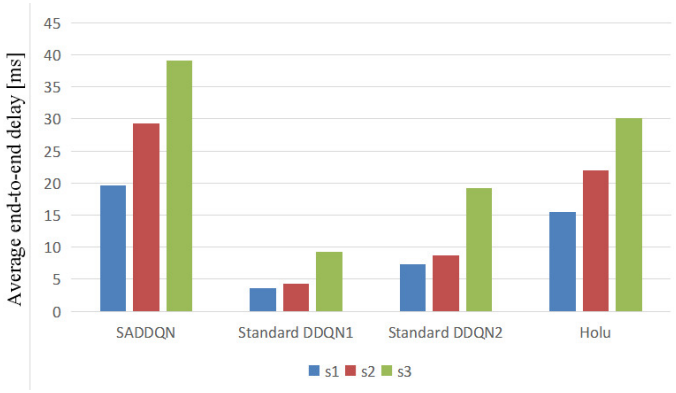

Figure 6: Average end-to-end delay comparison.

all the requests accepted by SADDQN, 50 are of the highest priority, 50 are of the second-highest priority, and only 36 have the lowest priority. That is, 14 lowest-priority requests are discarded because of a lack of resources. DDQN1 accepts 50 highest-priority requests and 22 second-highest priority requests, while DDQN2 accepts 50 for both priorities. Meanwhile, these two standard DDQN algorithms drop all the lowest-priority requests. There are two reasons for the above results. First, we assign greater reward values to the successful placement of higher-priority service requests. Second, the learning-based algorithms can learn the arrival distribution of different services. With the object of maximizing the total discounted cumulative reward, the DDQN-based models discard the lower-priority services when there are insufficient resources to accommodate all. However, the Holu algorithm, which does not consider service priority, attempts to accommodate services in sequence and thus cannot ensure priority allocation for higher-priority services. Hence, it treats every request equally and accommodates nearly the same number of requests for the three categories. In conclusion, our proposed SADDQN outperforms the other algorithms regarding the acceptance ratio. Meanwhile, it ensures priority accommodation for higher-priority services.

We further compare our proposed SADDQN with the standard DDQN and the Holu algorithm in terms of the average end-to-end delay. In Fig. 6, we can see that compared to the other algorithms, our proposed SADDQN obtains an average end-to-end delay closer to the requirement for any type of service. The reason is that we optimize the resource allocation for VNFs of each service. Specifically, we approach the latency requirement of a service using the proposed BSAGD algorithm, which iteratively searches the minimum cost for realizing the service. By contrast, without the proposed sub-action concept or the resource optimization algorithm, the standard DDQN always over-provisions these flows. The Holu algorithm first proposes a PM ranking mechanism based on power consumption to resolve the VNF placement. Then, it employs a delay-constrained least-cost (DCLC) shortest-path algorithm to find the path between the selected VNFs. Although the Holu algorithm aims to satisfy the end-to-end delay of each service with the minimum cost, it does not take the over-provisioning of resources into consideration. Therefore, from the bar chart for the standard DDQN and Holu, we can see that each flow suffers a delay shorter than the required delay by at least a few milliseconds. The results indicate that the standard DDQN and Holu consume extra resources to accommodate services, as observed in Figs. 7 and 8.

We can see from Figs. 8 and 7 that, of all the algorithms, our proposed SADDQN achieves the highest cost utility because our BSAGD algorithm flexibly allocates resources to VNFs and thus provides a finer granularity in resource allocation. In contrast, the others use fixed-capacity VNFs, inevitably resulting in over-provisioning. Furthermore, DDQN-based algorithms minimize the cost from the global perspective, while Holu minimizes the cost instantaneously rather than farsightedly. Nevertheless, one of Holu's advantages over the standard DDQN is its VNF sharing policy, which mitigates the overconsumption of resources. For this reason, in Figs. 7 and 8, as the number of service requests increases, the Holu algorithm is second only to SADDQN in terms of cost utility. Moreover, in Fig. 7, the average cost per request does not change substantially for the SADDQN algorithm, while the cost for the Holu algorithm fluctuates. The reason is that because of the VNF sharing policy, the curve of Holu may ascend when new VNF instances must be initiated and descend when the current running VNFs can be utilized by new services. The standard DDQN algorithm does not accept VNF sharing for security reasons. Hence, the average cost per request of DDQN1 and DDQN2 remains stable when all requests can be accepted. However, as the number of requests increases, the two curves start to descend because the lowest-priority services, which have a higher cost than services from the other two categories, are gradually discarded. In conclusion, our proposed SADDQN is superior to all the other approaches in terms of cost utility.

6. Conclusion

In this paper, we proposed a sub-action aided DDQN (SADDQN) algorithm to maximize the acceptance ratio and ensure priority allocation for higher-priority requests while minimiz-

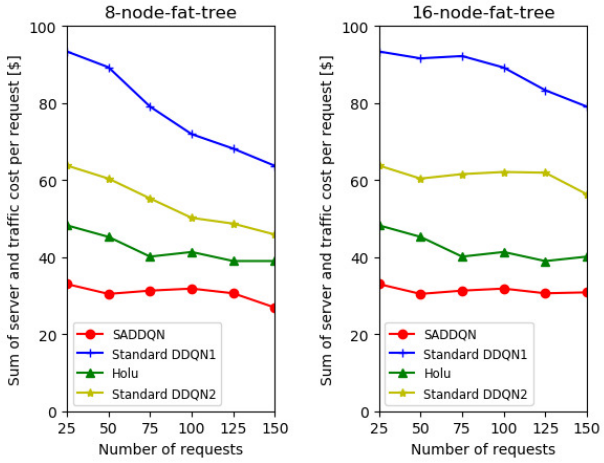

Figure 7: Joint traffic and CPU cost comparison.

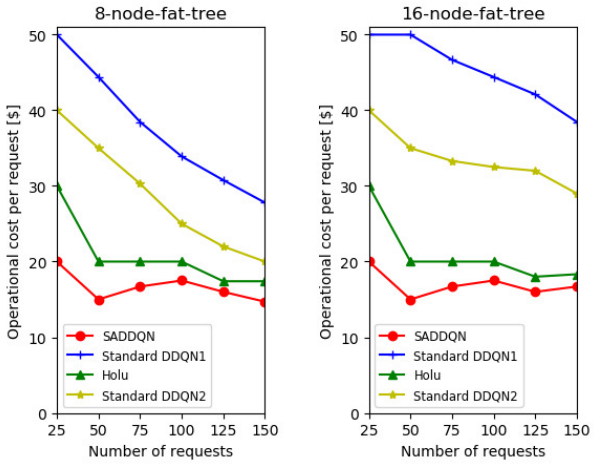

Figure 8: Operational cost comparison.

ing the total cost for latency-aware network sub-slices. We considered a multi-edge cloud scenario in which service requests with different priorities are fed into the DDQN-agent, and developed an intelligent policy to place the VNF-FGs and allocate resources. Considering the objective, we defined our new state, main action and sub-action, and reward function for the DDQN framework and proposed an algorithm (sub-action) for allocating resources. Simulations showed that our proposed SADDQN can maximize the acceptance ratio and ensure priority allocation for higher-priority services while minimizing the total cost for latency-aware services with different latency requirements. Numerical results showed that, compared with the other two popular algorithms, our proposed scheme performs better in terms of minimizing the cost and maximizing the acceptance ratio; thus, it resolves the QoS over-provisioning issue, as expected.

Acknowledgment

This work was supported in part by Ericsson Australia Pty Ltd, and in part by the China Scholarship Council for four-year study at the University of Technology Sydney. We would like to take this opportunity to thank three people who have provided us with invaluable assistance. We gratefully acknowledge the contribution of Doctor Xu Wang, who shared his experience in writing a paper and choosing a journal. We also thank Doctor Shenghong Li and Yiwen Qu for giving valuable suggestions on how to design a reinforcement learning based algorithm.

APPENDIX A. PROOF OF PROPOSITION 1

Based on (20b), we have

$$\begin{cases} \sum_{j=1}^{K_i} a_j \times \mu_{i,j} = c_i - \alpha \times 250 \times N_i & \text{(27a)} \\ a_j = \gamma \dfrac{M_{\text{tot}}}{C_{i,j}^{\text{pro}}} + \beta \dfrac{W_{\text{tot}} \times h\left(v_j, v_{j+1}\right)}{C_{i,j}^{\text{tran}}}, \\ j = 1, 2, \ldots, K_i. & \text{(27b)} \end{cases}$$

According to (27a), no matter what the optimal solution for P2 is, we can conclude that the optimal solution $\widetilde{\boldsymbol{\mu}}_{\boldsymbol{i}}$ satisfies

$$\begin{cases} a_j \times \widetilde{\mu}_{i,j} = q_j \times (c_i - \alpha \times 250 \times N_i), \\ j = 1, 2, \ldots, K_i & \text{(28a)} \\ \sum_{j=1}^{K_i} q_j = 1 & \text{(28b)} \\ 0 < q_j < 1, \ \ j = 1, 2, \ldots, K_i. & \text{(28c)} \end{cases}$$

Substituting the optimal solution $\widetilde{\boldsymbol{\mu}}_{\boldsymbol{i}}$ into P2, we have

$$\begin{aligned} D_i^{min}(\widetilde{\boldsymbol{\mu}}_{\boldsymbol{i}}) &= \frac{\lambda_i}{2\left(\widetilde{\mu}_{i,1}\right)^2 \left(1 - \frac{\lambda_i}{\widetilde{\mu}_{i,1}}\right)} + \sum_{j=1}^{K_i} \frac{2}{\widetilde{\mu}_{i,j}} \\ &+ \sum_{j=1}^{K_i} \frac{2(h(v_j, v_{j+1}) - 1)}{\widetilde{\mu}_{i,j}} g(h(v_j, v_{j+1})) \\ &+ \sum_{j=2}^{K_i} \frac{\lambda_i}{2\left(\widetilde{\mu}_{i,j}\right)^2 \left(1 - \frac{\lambda_i}{\widetilde{\mu}_{i,j}}\right)} g(\widetilde{\mu}_{i,j-1} - \widetilde{\mu}_{i,j}) \\ &= \frac{1}{2}\left(\frac{1}{\widetilde{\mu}_{i,1} - \lambda_i} - \frac{1}{\widetilde{\mu}_{i,1}}\right) + \sum_{j=1}^{K_i} \frac{2}{\widetilde{\mu}_{i,j}} \\ &+ \sum_{j=1}^{K_i} \frac{2(h(v_j, v_{j+1}) - 1)}{\widetilde{\mu}_{i,j}} g(h(v_j, v_{j+1})) \\ &+ \sum_{j=2}^{K_i} \frac{1}{2}\left(\frac{1}{\widetilde{\mu}_{i,j} - \lambda_i} - \frac{1}{\widetilde{\mu}_{i,j}}\right) g(\widetilde{\mu}_{i,j-1} - \widetilde{\mu}_{i,j}). \end{aligned} \quad \text{(29)}$$

Let

$$D_{i1}^{min}(\widetilde{\mu}_{i,j}) = \frac{1}{\widetilde{\mu}_{i,j} - \lambda_i} - \frac{1}{\widetilde{\mu}_{i,j}}, \ \ j = 1, 2, \ldots, K_i, \quad \text{(30)}$$

$$D_{i2}^{min}(\widetilde{\mu}_{i,j}) = \frac{2}{\widetilde{\mu}_{i,j}},\ j = 1, 2, \ldots, K_i. \tag{31}$$

Then, we have

$$\begin{aligned} D_i^{min}(\widetilde{\boldsymbol{\mu}_i}) = & \frac{D_{i1}^{min}(\widetilde{\mu}_{i,j})}{2} + \sum_{j=2}^{K_i} \frac{D_{i1}^{min}(\widetilde{\mu}_{i,j})}{2} g(\widetilde{\mu}_{i,j-1} - \widetilde{\mu}_{i,j}) \\ & + \sum_{j=1}^{K_i} D_{i2}^{min}(\widetilde{\mu}_{i,j})(h(v_j, v_{j+1}) - 1) g(h(v_j, v_{j+1})) \\ & + \sum_{j=1}^{K_i} D_{i2}^{min}(\widetilde{\mu}_{i,j}),\ j = 1, 2, \ldots, K_i. \end{aligned} \tag{32}$$

Substituting (27b) and (28a) into (30) and (31) and then differentiating D_{i1}^{min} and D_{i2}^{min} with respect to c_i, we have

$$\begin{aligned} \frac{\mathrm{d}D_{i1}^{min}(c_i)}{\mathrm{d}c_i} = & \frac{a_j}{q_j}\Big(\frac{1}{(c_i - 250 \times \alpha \times N_i)^2} \\ & - \frac{1}{(c_i - 250 \times \alpha \times N_i - \frac{a_j}{q_j}\lambda_i)^2}\Big), \end{aligned} \tag{33}$$

$$\frac{\mathrm{d}D_{i2}^{min}(c_i)}{\mathrm{d}c_i} = \frac{a_j}{q_j} \frac{-1}{(c_i - 250 \times \alpha \times N_i)^2}. \tag{34}$$

Based on (27b) and (28c), we know $a_j > 0$ and $q_j > 0$. Therefore, we can conclude that $\frac{\mathrm{d}D_{i1}^{min}(c_i)}{\mathrm{d}c_i} < 0$ and $\frac{\mathrm{d}D_{i2}^{min}(c_i)}{\mathrm{d}c_i} < 0$. Furthermore, based on (21), we know $g(\mu_{i,j-1} - \mu_{i,j}) = 1$ or 0, and $g(h(v_j, v_{j+1})) = 1$ or 0. At this stage, we can conclude that $\frac{\mathrm{d}D_i^{min}(c_i)}{\mathrm{d}c_i} < 0$; thus, $D_i^{\min}(c_i)$ is a monotone decreasing function of c_i.

APPENDIX B. PROOF OF PROPOSITION 2

The standard form of an optimization problem can be expressed as [39]:

$$\min f_0(x)$$

$$\text{s.t.} \begin{cases} f_i(x) \le 0, & i = 1, 2, \ldots, m \\ h_i(x) = 0, & i = 1, 2, \ldots, p. \end{cases} \tag{35}$$

The problem is to find an $\mathbf{x}$ that minimizes $f_0(\mathbf{x})$ among all $\mathbf{x}$ that satisfy the conditions $f_i(\mathbf{x}) \le 0, i = 1, \ldots, m$, and $h_i(\mathbf{x}) = 0, i = 1, \ldots, p$. To be a convex optimization problem, it needs to satisfy three additional requirements [39]:

1. the objective function must be convex,
2. the inequality constraint functions must be convex,
3. the equality constraint functions must be affine.

Now let's prove that our problem satisfies these three requirements and thus is a convex optimization problem.

6.1. Objective function is convex

In our problem, the objective function shown below is twice differentiable; that is, its Hessian or second derivative exists at each point in dom D_i, which is open. Then, D_i is convex if and only if dom D_i is convex and its Hessian is positive semidefinite [39].

$$\begin{aligned} D_i(\boldsymbol{\mu_i}) = & \frac{\lambda_i}{2\left(\mu_{i,1}\right)^2\left(1 - \frac{\lambda_i}{\mu_{i,1}}\right)} + \sum_{j=1}^{K_i} \frac{2}{\mu_{i,j}} \\ & + \sum_{j=1}^{K_i} \frac{2(h(v_j, v_{j+1}) - 1)}{\mu_{i,j}} g(h(v_j, v_{j+1})) \\ & + \sum_{j=2}^{K_i} \frac{\lambda_i}{2\left(\mu_{i,j}\right)^2\left(1 - \frac{\lambda_i}{\mu_{i,j}}\right)} g(\mu_{i,j-1} - \mu_{i,j}). \end{aligned}$$

The hessian matrix of D_i can be expressed as

$$H(D_i) = \begin{bmatrix} \frac{\partial^2 D_i}{\partial \mu_{i,1}^2} & \frac{\partial^2 D_i}{\partial \mu_{i,1}\partial \mu_{i,2}} & \cdots & \frac{\partial^2 D_i}{\partial \mu_{i,1}\partial \mu_{i,K_i}} \\ \frac{\partial^2 D_i}{\partial \mu_{i,2}\partial \mu_{i,1}} & \frac{\partial^2 D_i}{\partial \mu_{i,2}^2} & \cdots & \frac{\partial^2 D_i}{\partial \mu_{i,2}\partial \mu_{i,K_i}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 D_i}{\partial \mu_{i,K_i}\partial \mu_{i,1}} & \frac{\partial^2 D_i}{\partial \mu_{i,K_i}\partial \mu_{i,2}} & \cdots & \frac{\partial^2 D_i}{\partial \mu_{i,K_i}^2} \end{bmatrix},$$

where

$$\begin{aligned} \frac{\partial^2 D_i}{\partial \mu_{i,1}^2} = & \frac{3 + 4(h(v_1, v_2) - 1) \times g(h(v_1, v_2))}{\mu_{i,1}^3} \\ & + \frac{1}{\left(\mu_{i,1} - \lambda_i\right)^3}, \end{aligned}$$

when $2 \le j \le K_i$,

$$\frac{\partial^2 D_i}{\partial \mu_{i,j}^2} = \begin{cases} \frac{1}{\left(\mu_{i,j} - \lambda_i\right)^3} + \frac{3+4(h(v_j, v_{j+1}) - 1) \times g(h(v_j, v_{j+1}))}{\mu_{i,j}^3}, \\ \mu_{i,j} < \mu_{i,j-1} \\ \frac{4+4(h(v_j, v_{j+1}) - 1) \times g(h(v_j, v_{j+1}))}{\mu_{i,j}^3},\ \mu_{i,j} \ge \mu_{i,j-1}, \end{cases}$$

and

$$\frac{\partial^2 D_i}{\partial \mu_{i,j}\partial \mu_{i,j'}} = 0,\ j \ne j'.$$

In the M/D/1 model, obviously, $\mu_{i,j} > \lambda_i$ is satisfied for any j. Therefore, dom $D_i = \{\boldsymbol{\mu_i} \in R^{K_i} | \mu_{i,j} > \lambda_i, j = 1, \ldots, K_i\}$ is convex. Based on (21), we know $g(h(v_j, v_{j+1})) = 1$ or 0. So we have $\frac{\partial^2 D_i}{\partial \mu_{i,j}^2} > 0, j = 1, \ldots, K_i,$; thus, $H(D_i)$ is a positive-definite matrix. At this stage, we can conclude that the objective function is convex.

6.2. Inequality constraint functions are convex

The standard form of inequality constraints in our problem can be expressed as

$$f_j(\boldsymbol{\mu_i}) = \mu_{i,j} - \mu_{i,j+1} \le 0,\ j = 1, 2, \ldots, K_i - 1, \text{ or}$$

$$f_j(\boldsymbol{\mu_i}) = \mu_{i,j+1} - \mu_{i,j} \le 0,\ j = 1, 2, \ldots, K_i - 1,$$

$$f_{K_i-1+n}(\boldsymbol{\mu_i}) = \sum_{j=1}^{K_i} x_{i,j}^n \times \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} - m_n \leq 0,$$
$$n = 1, 2, \ldots, |\mathbb{S}|,$$

$$f_{K_i+|\mathbb{S}|-1+n}(\boldsymbol{\mu_i}) = \sum_{j=1}^{K_i} x_{i,j}^n \times \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} - w_n \leq 0,$$
$$n = 1, 2, \ldots, |\mathbb{S}|.$$

Evidently, for any constraint function, dom $f_j = \{\boldsymbol{\mu_i} \in R^2 | \mu_{i,j} > \lambda_i, \mu_{i,j+1} > \lambda_i\}$, $j = 1, 2, \ldots, K_i - 1$ or dom $f_n = \{\boldsymbol{\mu_i} \in R^L | \mu_{i,1} > \lambda_i, \ldots, \mu_{i,L} > \lambda_i\}$, $n \geq K_i$ (suppose that L VNFs nest on the n^{th} server) is convex. Furthermore, its Hessian is zero matrix and thus positive semidefinite. Hence, inequality constraints are all convex.

6.3. Equality constraint function is affine

When service i is provided with a constant cost C_i, the standard form of the equality constraint in our problem can be expressed as

$$h_1(\boldsymbol{\mu_i}) = \gamma \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{pro}}} \times M_{\text{tot}} + \beta \sum_{j=1}^{K_i} \frac{\mu_{i,j}}{C_{i,j}^{\text{tran}}} \times W_{\text{tot}} \times h\left(v_j, v_{j+1}\right) + \alpha \times 250 \times N_i - C_i = 0.$$

According to the definition of affine: A set C is affine if the line through any two distinct points in C lies in C[39], we know that any line or line segment is affine. Therefore, the equality constraint in our problem is affine.

References

[1] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, G. Fettweis, The 5g-enabled tactile internet: Applications, requirements, and architecture, in: 2016 IEEE Wireless Communication and Networking Conference Workshops (WCNCW), Doha, Qatar, 2016, pp. 1–6.

[2] S. Ravindran, S. Chaudhuri, J. Bapat, D. Das, Required delay-based network sub-slices resource optimization for 5g radio access network, in: 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Goa, India, 2019, pp. 1–6.

[3] I. Afolabi, T. Taleb, P. A. Frangoudis, M. Bagaa, A. Ksentini, Network slicing-based customization of 5g mobile services, IEEE Network 33 (5) (2019) 134–141.

[4] J. Pei, P. Hong, K. Xue, D. Li, Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system, IEEE Transactions on Parallel and Distributed Systems 30 (10) (2019) 2179–2192.

[5] A. Varasteh, B. Madiwalar, A. V. Bemten, W. Kellerer, C. Mas-Machuca, Holu: Power-aware and delay-constrained vnf placement and chaining, IEEE Transactions on Network and Service Management 18 (2) (2021) 1524–1539.

[6] P. T. A. Quang, Y. Hadjadj-Aoul, A. Outtagarts, A deep reinforcement learning approach for vnf forwarding graph embedding, IEEE Transactions on Network and Service Management 16 (4) (2019) 1318–1331.

[7] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, H. Flinck, Network slicing and softwarization: A survey on principles, enabling technologies, and solutions, IEEE Communications Surveys & Tutorials 20 (3) (2018) 2429–2453.

[8] J. Gil Herrera, J. F. Botero, Resource allocation in nfv: A comprehensive survey, IEEE Transactions on Network and Service Management 13 (3) (2016) 518–532.

[9] S. Mostafavi, V. Hakami, M. Sanaei, Quality of service provisioning in network function virtualization: A survey, Computing 103 (2021) 917–991.

[10] A. Baumgartner, V. S. Reddy, T. Bauschert, Combined virtual mobile core network function placement and topology optimization with latency bounds, in: 2015 Fourth European Workshop on Software Defined Networks, Bilbao, Spain, 2015, pp. 97–102.

[11] D. B. Oljira, K. Grinnemo, J. Taheri, A. Brunstrom, A model for qos-aware vnf placement and provisioning, in: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, Germany, 2017, pp. 1–7.

[12] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, L. P. Gaspary, Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 2015, pp. 98–106.

[13] Q. Zhang, Y. Xiao, F. Liu, J. C. S. 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), Atlanta, GA, USA, 2017, pp. 731–741.

[14] H. Tang, D. Zhou, D. Chen, Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers, IEEE Transactions on Parallel and Distributed Systems 30 (3) (2019) 530–543.

[15] M. Mechtri, C. Ghribi, D. Zeghlache, A scalable algorithm for the placement of service function chains, IEEE Transactions on Network and Service Management 13 (3) (2016) 533–546.

[16] C. Pham, N. H. Tran, S. Ren, W. Saad, C. S. Hong, Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach, IEEE Transactions on Services Computing 13 (1) (2020) 172–185.

[17] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, B. Akbari, Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining, IEEE Transactions on Network and Service Management 16 (1) (2019) 374–388.

[18] M. Dieye, et al., Cpvnf: Cost-efficient proactive vnf placement and chaining for value-added services in content delivery networks, IEEE Transactions on Network and Service Management 15 (2) (2018) 774–786.

[19] Q. Ye, W. Zhuang, X. Li, J. Rao, End-to-end delay modeling for embedded vnf chains in 5g core networks, IEEE Internet of Things Journal 6 (1) (2019) 692–704.

[20] P. T. A. Quang, Y. Hadjadj-Aoul, A. Outtagarts, Deep reinforcement learning based qos-aware routing in knowledge-defined networking, in: Qshine 2018 - 14th EAI International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Ho Chi Minh City, Vietnam, 2018, pp. 1–13.

[21] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, F. De Turck, S. Latré, Design and evaluation of learning algorithms for dynamic resource management in virtual networks, in: 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 2014, pp. 1–9.

[22] M. Bunyakitanon, X. Vasilakos, R. Nejabati, D. Simeonidou, End-to-end performance-based autonomous vnf placement with adopted reinforcement learning, IEEE Transactions on Cognitive Communications and Networking 6 (2) (2020) 534–547.

[23] A. Rkhami, Y. Hadjadj-Aoul, A. Outtagarts, Learn to improve: A novel deep reinforcement learning approach for beyond 5g network slicing, in: IEEE Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2021, pp. 1–9.

[24] Open Science Data Cloud, Virtual machines (vms) (2014).
URL `https://www.opensciencedatacloud.org/support/instances.html`

[25] T. Tovinger, Management, orchestration and charging for 5g networks (Mar. 2018).
URL `https://www.3gpp.org/news-events/1951-sa5_5g`

[26] 3GPP, 3gpp tr 28.801 v15.1.0 (Oct. 2018).
URL `https://www.3gpp.org/ftp/TSG_SA/WG5_TM/TSGS5_116/SA_78/28801-f10.doc`

[27] P. Goransson, C. Black, T. Culver, Software Defined Networks:A Com-

prehensive Approach, Elsevier, 2016, Ch. 7, p. 159.
[28] W. Wang, B. Liang, B. Li, Multi-resource generalized processor sharing for packet processing, in: 2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS), Montreal, QC, Canada, 2013, pp. 1–10.
[29] S. Gu, Z. Li, C. Wu, C. Huang, An efficient auction mechanism for service chains in the nfv market, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 2016, pp. 1–9.
[30] S. Larsen, P. Sarangam, R. Huggahalli, Architectural breakdown of end-to-end latency in a tcp/ip network, in: 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07), Gramado, Brazil, 2007, pp. 195–202.
[31] A. Greenberg, J. R. Hamilton, N. Jain, et al., Vl2: a scalable and flexible data center network, ACM SIGCOMM Computer Communication Review 39 (4) (2009) 51–62.
[32] H. V. Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning (Sep. 2015).
URL https://arxiv.org/abs/1509.06461
[33] J. SON, T. He, R. Buyya, CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments, Software: Practice and Experience 49 (12) (2019) 1748–1764.
[34] J. Son, A. V. Dastjerdi, R. N. Calheiros, Y. Y. X. Ji, R. Buyya, Cloudsimsdn: Modeling and simulation of software-defined cloud data centers, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 2015, pp. 475–484.
[35] V. Mnih, K. Kavukcuoglu, D. Silver, et al., Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.
[36] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (Jan. 2017).
URL https://arxiv.org/abs/1412.6980v9
[37] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Vol. 15, Fort Lauderdale, FL, USA, 2011, pp. 315–323.
[38] A. L. Maas, Rectifier nonlinearities improve neural network acoustic models, in: in ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.
[39] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004, Ch. 4, pp. 129–138.