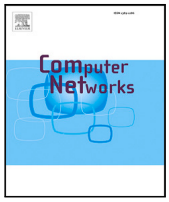
Computer Networks 242 (2024) 110254

Contents lists available at ScienceDirect

Computer Networks

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



A cost and demand sensitive adjustment algorithm for service function chain in data center network

Yuantao Wang, Zhaogang Shu *, Shuwu Chen, Jiaxiang Lin, Zhenchang Zhang

The Computer and Information College, Fujian Agriculture and Forestry, Fuzhou, 350002, China

ARTICLE INFO

Keywords:
Service function chain
Network function virtualization orchestrator
Resource management

ABSTRACT

The introduction of Network Function Virtualization (NFV) and Software-Defined Network (SDN) architectures has significantly reduced the Operational Expenditure (OPEX) and Capital Expenditure (CAPEX) of network system. However, NFV orchestration management also brings about challenges. After the initial deployment of VNFs, due to the volatility of network requests, the original deployment may not be able to meet user resource demands. The key issue is how to readjust resources dynamically to accommodate more network requests without violating Quality of Service (QoS) for users. Several existing techniques can be used to achieve this goal, such as horizontal scaling, vertical scaling, and virtual network function (VNF) migration. However, these techniques inevitably incur some overhead, such as the cost of instantiating VNF and link rerouting. Additionally, resource adjustment may also result in unbalanced distribution of network resources. In this paper, an Intelligent Service Function Chain Dynamic Adjustment Algorithm (ISFCDAA) is proposed to address the above challenges. Firstly, an Integer Linear Programming (ILP) model is established with the objective of minimizing the long-term adjustment cost and reducing the imbalance of resource distribution. Then we transform the optimization process into a Markov Decision Process (MDP). Secondly, to solve the problems that the state and action space is too large and the state transition probability is uncertain in MDP, a SFC dynamic adjustment algorithm based on deep reinforcement learning is proposed. This algorithm can obtain an approximate optimal adjustment strategy for ILP model. The simulation results show that ISFCDAA can reduce the adjustment overhead and maintain a balanced distribution of network resources while ensuring the QoS. Compared with the existing algorithms, the average standard deviation of resource distribution of ISFCDAA is reduced by up to 9.90%, the average acceptance rate of ISFCDAA is improved by up to 39.57%, and the average long-term profit is improved by up to 42.92%. The incorporation of cost and demand-sensitive considerations into ISFCDAA enhances its responsiveness to fluctuating network demands, solidifying its effectiveness in dynamic resource management scenarios.

1. Introduction

As the Internet continues to evolve, users have more and more diverse demands for the network services. In traditional networks, network service operators typically utilize specific network devices to provide different network services, such as firewall, load balancing and network address translation. However, the software and hardware in these specific network devices are tightly coupled, which reduces configuration flexibility and increases device maintenance and management costs. Especially for 5G/6G network, service operators need to spend a lot of money to purchase specific network devices, and it is also difficult to provide various network services in the traditional network architecture. For these reasons, the European Telecommunications Standards Institute (ETSI) proposed the Network Function Virtualization (NFV) technology [1]. The emergence of NFV technology decouples network functions from specific hardware by transforming traditional network function into virtual network function (VNF) through virtualization and placing the VNF on industry standard servers, as shown in Fig. 1. It does not require specific hardware devices, which improves the flexibility and scalability of VNF placement, and achieves low-cost and flexible network management. It enables VNF to be widely used in various network scenarios such as data center, 5G slice and edge computing.

However, NFV technology requires complex orchestration to allocate physical resources, so it is essential to combine programmable Software-Defined Network (SDN) technology [2]. Supported by SDN/NFV technology, multiple VNFs are connected in a specific order to form a Service Function Chain (SFC) based on user requirements and

* Corresponding author.
E-mail address: zgshu@fafu.edu.cn (Z. Shu).

https://doi.org/10.1016/j.comnet.2024.110254
Received 17 August 2023; Received in revised form 10 February 2024; Accepted 14 February 2024
Available online 16 February 2024

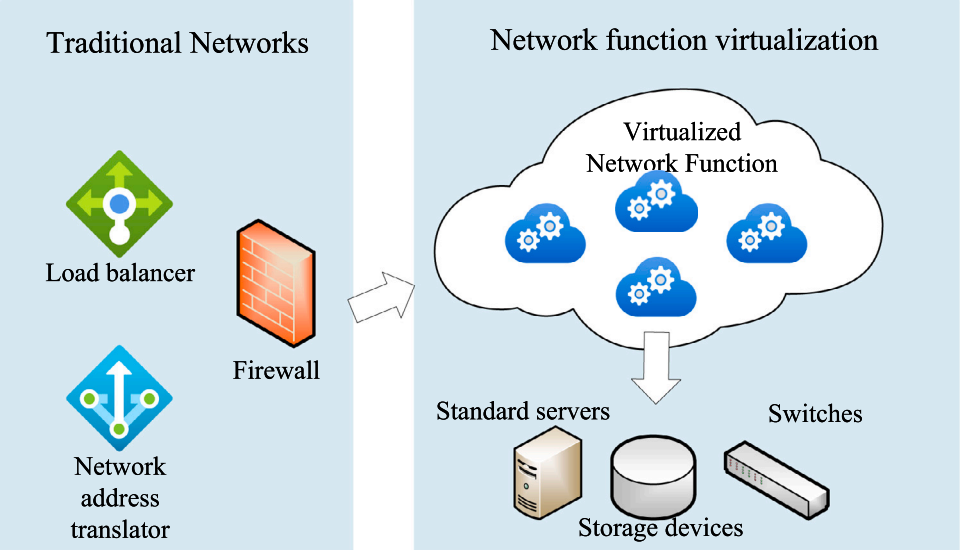


Fig. 1. Traditional network & network with NFV.

VNF attributes. The NFV orchestrator effectively places VNF instances on general-purpose servers based on the SFC requirements and physical device capacity constraints [3]. Subsequently, it determines the dynamic execution order of VNFs for SFC scheduling [4], aiming to reduce the overall service time. If the SFC requirements are time-varying [5], we should dynamically adjust physical resources for the requests that already present in the network to adapt to changing requirements. By adjusting resources, the load can be efficiently distributed among available resources, maximizing the efficiency of resource utilization, improving overall network throughput, and reducing operational costs. At the same time, more requests are being received, allowing network operators to gain significant long-term revenue. Nonetheless, achieving these benefits goes beyond simply resource adjustment as it involves the inevitable adjustment overhead. Therefore, a critical challenge lies in appropriately adjusting physical resources based on the current network state while maintaining a balanced distribution of resources and minimizing the adjustment overhead.

Existing work mainly focus on three resource adjustment strategies: (i) horizontal scaling, (ii) vertical scaling, and (iii) migration. In [6], horizontal scaling is performed by increasing or decreasing the number of VNF instances, which can quickly respond to fluctuations in requests and provide high elasticity and scalability. In [7], vertical scaling is applied to multi-core CPUs in order to control CAPEX. In [8], an algorithm is proposed to integrate VNF instances by migrating placed VNF instances. While each of these strategies has its own advantages, applying them individually may have limitations. For example, in vertical scaling, as the computational capacity of VNF instances increase, this strategy may not always be feasible if there is not enough residual resource in the underlying physical devices. Besides when there are a large number of requests, horizontal scaling cannot be satisfied with fewer physical devices. It is also noted that horizontal scaling tends to produce more resource fragmentation in the underlying physical devices. And in order to create a new instance, a longer routing path may be needed, which results in an inability to meet the latency. Similarly migration also involves additional resource consumption because of a longer path. And if VNF migrations are frequent or result in service disruptions, it can affect the Service Level Agreement (SLA). Overall, existing works rarely consider all three strategies simultaneously and pay little attention to the distribution of the resources. Therefore, we propose a dynamic adjustment algorithm called Intelligent SFC Dynamic Adjustment Algorithm (ISFCDAA) by combining horizontal, vertical scaling and migration. Notably, ISFCDAA is designed as a cost and demand sensitive adjustment algorithm, where 'cost' represents considerations related to adjustment expenses, while 'demand' reflects the algorithm's adjustment to changing resource demands. Through the integration of these factors, ISFCDAA flexibly adjusts the resources according to the requirements, achieves a balanced resource distribution, and minimizes the resource adjustment cost while ensuring QoS. The contributions of this paper are as follows:

1. We develop an Integer Linear Programming (ILP) model aimed at minimizing the standard deviation of resource distribution and the overhead of resource adjustment. Additionally, we provide a proof of the NP-hardness of the problem. Importantly, our work considers the balance of resource distribution, a dimension that has been largely overlooked in previous related works.
2. The optimization problem is transformed into Markov Decision Process (MDP), and then ISFCDAA, which considers the integration of cost and demand-sensitive aspects, is proposed to solve the problems with large state space and action space and unknown transition probabilities. Leveraging an advanced deep reinforcement learning method, ISFCDAA utilizes Double Deep Q-Network (DDQN) for decision-making, a novel approach in our specific context. This novel methodology enables ISFCDAA to intelligently adapt to the time-varying network environment, thereby facilitating effective resource adjustment.
3. The proposed algorithm and existing algorithms are evaluated on a real network in SDNlib [9]. The results show that ISFCDAA reduces the average standard deviation of the resource distribution by 2.41% compared to the Deep Q-Network (DQN) algorithm [10] and by 9.90% compared to the Reconfiguration Energy-Aware Placement (REAP) algorithm [11] in a single time slot. In the long term, the average acceptance rate of ISFCDAA is 15.11% higher than the DQN algorithm and 39.57% higher than the REAP algorithm. ISFCDAA improves the average long-term profit by 21.47% compared to the DQN algorithm and by 42.92% compared to the REAP algorithm. These results demonstrate the cost-effectiveness and sustainability of ISFCDAA in dynamic resource environments.

The remainder of this paper is organized as follows: Section 2 discusses related work on resource adjustment. Section 3 presents the

Table 1
Comparison of related works.

Literatures	Horizontal scaling	Vertical scaling	Migration	Consider scaling cost	Consider distribution of resources	Methodology
Hawilo et al. [12]	√	–	–	No	No	MILP formulation with heuristic solver
Buh et al. [13]	–	√	–	No	No	Heuristic solver
Eramo et al. [14]	–	–	√	Yes	No	A heuristic based on the Viterbi algorithm
Padhy et al. [11]	√	√	√	Yes	No	ILP formulation with heuristic solver
Ayoubi et al. [15]	–	–	√	Yes	No	MINLP formulation with heuristic solve
Zhang et al. [16]	–	–	√	Yes	Yes(partial)	ILP formulation with heuristic solver
Houidi et al. [17]	√	√	√	No	No	ILP formulation with the Greedy heuristic
Harutyunyan et al. [18]	√	√	√	Yes	No	ILP formulation with heuristic solver
Chen et al. [19]	√	√	√	No	No	INLP formulation with heuristic solver
Our proposed	√	√	√	Yes	Yes	MDP formulation with DDQN

In this table, "√" means that this work takes this element into account, while "–" means that it does not.

target problem. Section 4 introduces the proposed ISFCDAA. Section 5 the analyzes numerical results, and finally we conclude the paper in Section 6.

2. Related work

In this section, we will present the existing work on dynamic resource adjustment.

There are many adjustment solutions in the existing work. For example, a method was proposed in [12] that utilizes horizontal scaling to minimize VNF downtime and reduce SFC delay. In [13], the authors effectively utilized multi-core architecture to properly balance network load and optimize inter-core communication through vertical scaling. The research [14] reduced the energy consumption of the data center by migrating VNFs for aggregation. However, researchers only study individual strategies, and rarely consider the three strategies jointly. For this reason, the work [11] proposed a heuristic algorithm that considered three strategies to reduce overhead and energy consumption, but the authors did not consider the distribution of the remaining resources, which may lead to unbalanced resource utilization.

In the existing work, the goals of dynamic adjustment are also different for each researcher. The objective of [20] is to minimize energy consumption and reconfiguration costs, but this work was solved under fixed traffic and limited state–action spaces. The work [15] devised an availability-aware resource allocation and reconfiguration framework that focuses on providing the highest availability improvement with the lowest reconfiguration cost. In [16], a latency-aware migration strategy was proposed to optimize the physical resource distribution by migrating the last VNF of the SFC, which alleviated the burden of high-load nodes to some extent, but it did not consider the balanced resource distribution from a global perspective. In contrast to these works, we trade off the balanced resource distribution and the cost of resource adjustment, offering a fresh perspective. This approach aims to improve the performance of the network and enabling network operators to more effectively satisfy user demands.

Furthermore, the existing approaches to resource adjustment schemes also vary. For instance, in [17], a greedy heuristic algorithm was proposed to dynamically scale the resources, but the computational complexity was high when dealing with large-scale networks. Meanwhile, [18] devised a heuristic algorithm to tackle the scalability issue of the ILP-based approach that aimed at minimizing the service provisioning cost for the mobile network operators. Additionally, [19] implementing in a real-world environment, aimed to minimize the number of used servers in the network. Unlike existing heuristic methods, we employ deep reinforcement learning as the basis for the adjustment algorithm. Traditional methods often rely on empirical rules and heuristic strategies, while reinforcement learning can enable the intelligent agent to autonomously explore and learn the best strategy through the learning. To our knowledge, no prior research has applied DDQN specifically to address the challenges in dynamic resource adjustment. A comparison of related works is given in Table 1.

3. System model and problem formulation

In this section, we first describe the problem, then introduce the system model, notations, and concepts used in this paper, and finally define the problem formally. For ease of reference, the notations used in this paper are summarized in Table 2.

3.1. Problem description

After the SFC placement is completed, the requirements of the SFC continue to change, which may affect the resource consumption of the physical network and violate the SLA. Therefore, the NFV orchestrator needs to dynamically manage VNF resources to meet the changing demands. When the resource demands of SFC increase, it causes the CPU load of some physical servers to increase, which affects the SFC delay and reduces the QoS. When the resource demands of SFC decrease, it causes the resource utilization rate of servers to decrease, which in turn increases the energy consumption of the network. Because network energy consumption still remains high if servers have not effectively enter low-power states. Additionally, even when operating at low utilization, servers still consume 70% of their maximum power [21]. Consequently, if VNF on a server has low resource requirements for an extended period, migrating that VNF and subsequently putting the server to sleep could enhance overall energy efficiency. Therefore, we consider three types of strategies to dynamically adjust some instances on the server: (1) expanding/shrinking the resources of VNF instances, (2) instantiating new VNF instances, (3) migrating VNF instances. However, these strategies may incur overheads. When the resource demands increase, if the server still has enough remaining resources and the resource distribution is relatively balanced, it is appropriate to expand the resources of the VNF instance. Besides, the adjustment overhead is small. If the current server resources are not sufficient or the resource distribution is unbalanced, we can consider migrating the VNF instance or instantiating a new instance. However, creating a new VNF instance may incur a higher adjustment overhead. Similarly, migration may not only incur rerouting overhead, but also increase bandwidth consumption. Besides it is necessary to avoid that the migration affects end-to-end delay of currently executing request sharing the same VNF instance. Therefore, we propose an effective dynamic adjustment strategy that trade-off each adjustment strategy.

As shown in Fig. 2(a), there are three requests $SFC_1\{VNF_1, VNF_2, VNF_3\}$, $SFC_2\{VNF_1, VNF_3, VNF_4\}$, and $SFC_3\{VNF_2, VNF_5\}$ have been placed on the standard general-purpose servers, each with 10 units of CPU resources. The three network functions in SFC_1 are placed on Ser_1, Ser_2 and Ser_3 respectively. The network functions in SFC_2 are hosted on Ser_1, Ser_5 and Ser_4 respectively, where VNF_1 shares the instance on Ser_1 with VNF_1 in SFC_1. The network functions in SFC_3 share the instances on Ser_2 and Ser_5, respectively. The data in the brackets in the figure are the resources occupied by each instance.

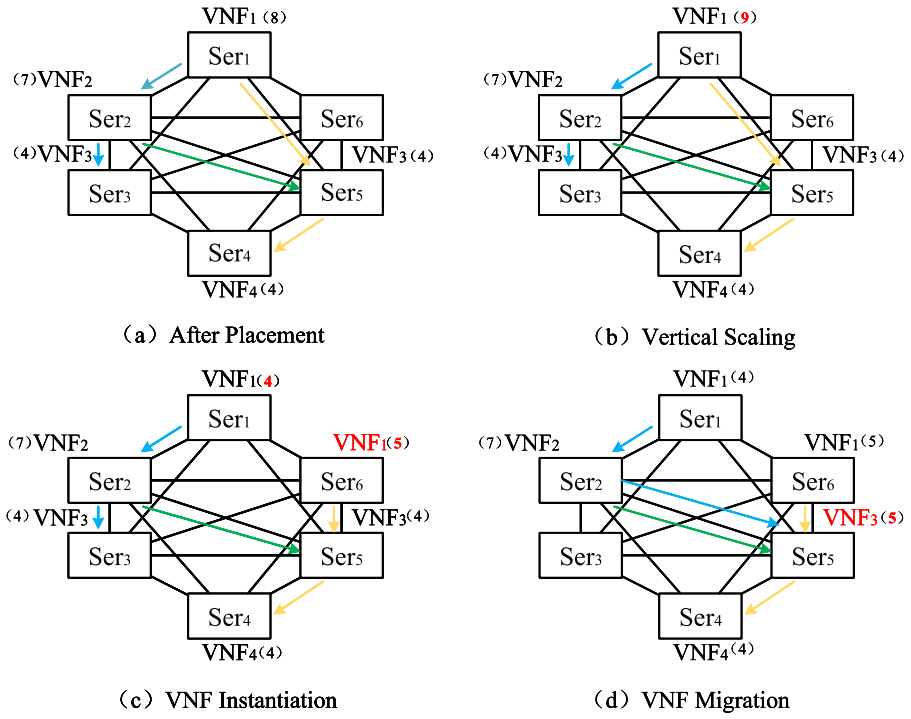


Fig. 2. Example of dynamic adjustment.

Table 2
Summary of notations.

Notation	Definition
N, L	The set of physical nodes and the set of physical links, respectively
C_u	Computational resources of a node u
B_{uv}, τ_{uv}	Bandwidth resources, transmission delay of link $l_{uv} = (u, v) \in L$, respectively
I	The set of SFC requests
s_i, t_i, D_i	The source node, the destination node, and the delay requirement of i, respectively
F_i	The set of VNFs of request i
r_j^i	Computational resources requirement of VNF_j of i
$f_{i,j}$	The jth forwarding path of i
b_i	Resources requirement of i
P_{m_i}	The unprocessed forwarding path after i migration
C_{ins}	Cost of instantiating a single VNF
$z_{iju}(t)$	Binary variable indicates whether or not VNF_j of i is placed onto physical node u at time slot t
$\pi_{(u,v)}^{f_{i,j}}(t)$	Binary variable indicates whether or not virtual link jth of i is mapped onto physical link $l_{uv} \in L$ at time t

As the demand of VNF_1 in SFC_2 increases over time, the instance in Ser_1 is vertical scaled to satisfy the change in demand since the server has enough resource capacity, as shown in Fig. 2(b). However, if the demand of VNF_1 persists for a long time, the node might be overloaded, leading to performance degradation. In this case, as shown in Fig. 2(c), a new instance of VNF_1 should be created on Ser_6, and SFC_2 must be rerouted. At this time, it is necessary to consider not only whether the resources on the server can be satisfied, but also whether the delay of the rerouted SFC_2 request is not be violated. Although this adjustment might generate overhead, the network resources distribution is more balanced, which generates positive feedback when the subsequent demands change, such as satisfying more requests. As shown in Fig. 2(d), Ser_5 has sufficient resources to satisfy the demand of VNF_3 in SFC_1, and the request delay could still be satisfied after migration. So when the demand of VNF_3 in SFC_1 decreases, the VNF_3 instance in Ser_3 can be migrated to Ser_5, and fully utilize the resources on Ser_5 to reduce the energy consumption. From this example, it can be observed that the operator can choose three different solutions (vertical scaling, horizontal scaling and migration) according to the changing requirements of the SFC. So our goals are to reduce the operator's operational cost and achieve a more balanced distribution of network resources by determining a suitable adjustment strategy that simultaneously guarantees the resource requirements and request delay.

3.2. System model

3.2.1. Physical network

The physical network is an undirected connected graph, denoted by $G = (N, L)$ where N denotes a set of physical machines which can host VNF instances and L is the set of physical links. The CPU computational resource of node $u \in N$ is denoted by C_u, and $B_{u,v}$ represents the available bandwidth of the link $l_{uv} \in L$ between node u and node v. τ_{uv} indicates the transmission delay of the link $l_{uv} \in L$. Each node has certain CPU resources, which limits the number of requests for shared VNF instances.

3.2.2. Service function chain

In the model, the set of VNFs is denoted by F and the set of requests is denoted by I. For each type of SFC request i, we generate the different resource requirement $r_j^i (j \subset F)$ of the same VNFj. Each request $i (i \in I)$ is composed of multiple ordered VNFs, and the VNF set formed by $F_i (F_i \subset F)$. The request $i (i \in I)$ is represented by a four-tuple(s_i, t_i, F_i, D_i), where s_i indicates the source, t_i indicates the destination, and D_i indicates the maximum delay allowed by the request. The jth virtual link of request i is denoted by $f_{i,j} (f_{i,j} \in i)$. b_i represents the bandwidth of i.

3.2.3. Cost structure

In this paper, the adjustment cost of SFC is divided into two main parts: migration cost and instantiation cost. Below are the definitions of these two costs.

Migration cost: When the VNF instance on the server is migrated to another server, it will spend some costs $c_{mig}(t)$. Similar to [16],

$c_{mig}(t) = C_b \cdot b_i \cdot \left|P_{m_i}(t)\right|$ is the additional cost of processing the total amount of unprocessed packets for the request i, where C_b represents the cost of using unit bandwidth on the physical link, b_i represents the bandwidth required to process the packets, and $\left|P_{m_i}(t)\right|$ denotes the quantity of new links containing unprocessed packets of i should be transmitted after migration. For example, there is a SFC which forwarding path is ($Ser_1 \rightarrow Ser_2 \rightarrow Ser_3$), and its forwarding path after migration is ($Ser_1 \rightarrow Ser_4 \rightarrow Ser_5 \rightarrow Ser_3$), so $\left|P_{m_i}(t)\right| = 3$. And it can be defined as Eq. (2), where $\pi_{u,v}^{f_{i,j}}(t)$ is a binary variable, indicating whether or not virtual link jth of i is mapped onto physical link $l_{uv} \in L$ at time t. Therefor the total migration cost is modeled in Eq. (1).

$$C_{mig}(t) = \sum_{i \in I} c_{mig}(t) = \sum_{i \in I} C_b \cdot b_i \cdot \left|P_{m_i}(t)\right| \tag{1}$$

$$\left|P_{m_i}(t)\right| = \sum_{f_{i,j} \in i} \sum_{(u,v) \in L} (\pi_{u,v}^{f_{i,j}}(t) - \pi_{u,v}^{f_{i,j}}(t-1)) \cdot \pi_{u,v}^{f_{i,j}}(t) \tag{2}$$

Instantiation cost: Same as [11], we define the instantiation cost $C_{delopy}(t)$ as the total cost of VNF instances created at time slot t. Therefore, we define $C_{delopy}(t)$ as:

$$C_{delopy}(t) = \sum_{i \in I} \sum_{j \in F_i} \sum_{u \in N} C_{ins} \cdot z_{iju}(t) \tag{3}$$

where C_{ins} is the cost of instantiating a single VNF. $z_{iju}(t)$ is a binary variable, and it recorded as 1 when VNF_j of i is instantiated on node u at time slot t.

Therefore, the total cost can be defined as:

$$C_{total}(t) = \alpha_1 \cdot C_{mig}(t) + \alpha_2 \cdot C_{delopy}(t) \tag{4}$$

where α_1 and α_2 are constant coefficients between costs.

3.3. Problem formulation

This section formally proposes a mathematical definition of the SFC dynamic adjustment problem. Firstly, relevant constraints are put on the problem. Secondly, the optimization goal of the problem is clarified.

3.3.1. Constraints

An effective adjustment strategy must ensure that the following conditions are satisfied.

Since CPU resources on each server are limited, each server can only support a limited number of VNFs. Therefore, we have

$$\sum_{i \in I} \sum_{j \in F_i} z_{iju}(t) \cdot r_j^i \le C_u, \forall u \in N \tag{5}$$

At the same time, the forwarding path bandwidth allocated to the corresponding SFC cannot exceed the bandwidth resource of the actual physical link. Hence, we have the constraint below.

$$\sum_{i \in I} \sum_{f_{i,j} \in i} b_i \cdot \pi_{u,v}^{f_{i,j}}(t) \le B_{u,v}, \forall (u,v) \in L \tag{6}$$

Similar to [22], each VNF in SFC can only be placed on a single physical node and cannot be served by multiple physical nodes. Thus, we have

$$\sum_{u \in N} z_{iju}(t) \le 1, \forall i \in I, \forall j \in F_i \tag{7}$$

In addition, the total delay of the transmission path should be less than the delay required by the request. Thus, we have

$$d_i \le D_i, \forall i \in I \tag{8}$$

where the transmission delay d_i of request i is defined as follows:

$$d_i = \sum_{f_{i,j} \in i} \sum_{(u,v) \in L} \tau_{uv} \cdot \pi_{u,v}^{f_{i,j}}(t) \tag{9}$$

3.3.2. Objective

To meet the changing demands of SFC, it is necessary to dynamically adjust the resources of physical nodes. How to meet user needs, reduce the cost of long-term dynamic adjustment, and make resource distribution more reasonable is what needs to be addressed in this paper.

The resource distribution is similar to the work [23], we use the standard deviation RD to measure the distribution of physical resources. RD can be calculated as:

$$RD = \sqrt{\frac{1}{|N|} \sum_{u=1}^{|N|} (RN_{avi}^u - \overline{RN_{avi}})^2} + \sqrt{\frac{1}{|L|} \sum_{l=1}^{|L|} (RL_{avi}^l - \overline{RL_{avi}})^2} \tag{10}$$

where $\overline{RN_{avi}}$ is the average value of the available resources of nodes, RN_{avi}^u is the available resources of a single node. $\overline{RL_{avi}}$ is the average value of available resources of links, and RL_{avi}^l is the available resources of a single link. RN_{avi}^u can be calculated by Eq. (11). RL_{avi}^l can be calculated by Eq. (12)

$$RN_{avi}^u = C_u - \sum_{i \in I} \sum_{j \in F_i} z_{iju}(t) \cdot r_j^i \tag{11}$$

$$RL_{avi}^l = B_{u,v} - \sum_{i \in I} \sum_{f_{i,j} \in F_i} b_i \cdot \pi_{(u,v)}^{f_{i,j}}(t) \tag{12}$$

And the total long-term cost of dynamic adjustment within T time slots can be defined as:

$$C = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} C_{total}(t) \tag{13}$$

where $C_{total}(t)$ can be calculated from Eq. (4).

Therefore the optimization objective can be organized as:

$$\begin{aligned} &\min C + RD \\ &s.t \ (5)\text{–}(9) \\ &z_{iju}(t) \in \{0,1\}, \forall i \in I, \forall j \in F_i, \forall u \in N, \forall t \in T \\ &\pi_{(u,v)}^{f_{i,j}}(t) \in \{0,1\}, \forall f_{i,j} \in i, \forall (u,v) \in L, \forall t \in T \end{aligned} \tag{14}$$

where $z_{iju}(t)$ is a binary variable, which equals 1 when the jth VNF of request i is instantiated on physical node u at time slot t, and value 0 otherwise. $\pi_{(u,v)}^{f_{i,j}}(t)$ is a binary variable, which equals 1 when the virtual link jth of i is mapped onto physical link $l_{uv} \in L$ at time t, and value 0 otherwise.

Theorem 1. *Our optimization problem is NP-hard.*

Proof. The Virtual Network Embedding (VNE) problem, which the primary objective is to allocate the virtual graph onto the physical graph, ensuring that the total capacities of virtual nodes/links do not exceed the capacities of their corresponding physical nodes and links, is acknowledged to be NP-hard [24]. Thus, simplifying the optimization problem to the VNE problem allows us to demonstrate its NP-hardness. In our optimization problem, there is a physical network $G = (N, L)$, where N is the set of physical nodes, and L is the set of physical links. We aim to identify an adjustment strategy characterized by two decision variables, $z_{iju}(t)$ and $\pi_{(u,v)}^{f_{i,j}}(t)$, ensuring compliance with node and link resource constraints as denoted by Eqs. (5) and (6), respectively. Among them, SFC can be regarded as a virtual graph. Additionally, within the context of VNE, the embedding cost, which pertains to the amount of consumed substrate resources for the embedding of Virtual Networks, serves as its objective [24]. Accordingly, we can reduce our objective function to consider only the adjustment cost in Eq. (14). By adopting this approach, our optimization problem is reducible to VNE. Since VNE is NP-hard, our optimization problem is NP-hard too.

4. The policy of dynamic adjustment

However, solving the adjustment policy under multi-objectives and multi-constraints is an NP-hard problem. So in this section, the solution process is modeled as MDP and solved using DDQN.

4.1. Markov decision process

MDP is a mathematical framework for describing and solving sequential decision problems. In reinforcement learning, MDP is widely used to model environments and formulate strategies of intelligent agent, then can learn the best strategy by interacting with the environment [25]. MDP can be represented by a quintuple (S, A, TP, R, γ), where S is a set of state space, A is a set of action space, TP is the state transition probability, R is the reward, and γ is the discount factor.

State Space: It indicates the specific state of the network at a certain moment. The state space contains two types of information in the ISFCDAA problem. It can be defined as Eq. (15), where $u(t) = \left[u_{n_1}(t), \ldots, u_{n_{|N|}}(t)\right]$ represents the vector of remaining resources on each physical server at time t, $\kappa(t) = \left[\kappa_{v_1}(t), \ldots, \kappa_{v_{|F|}}(t)\right]$ represents the vector indicating the successful placement of each type of VNF on the servers at time t. $\kappa_{v_x}(t) = \left[\kappa_{n_1}^{v_x}(t), \ldots, \kappa_{n_{|N|}}^{v_x}(t)\right]$ represents the mapping vector of VNF v_x on all servers at time t, where $\kappa_{n_i}^{v_x}(t) \in \{0, 1\}$ indicates whether VNF v_x is placed on server n_i at time t. |N| represents the number of severs, $|F|$ represents the number of VNF types, so the dimensions of the space is $|N| + |N| * |F|$. The integration of these information provides a more comprehensive description of the current state, which is important for decision-making on resource adjustment.

$$S(t) = \{u(t), \kappa(t)\} \tag{15}$$

Action Space: The agent selects the best adjustment strategy for the SFC of resource demand based on the current state $S(t)$, so the action in the ISFCDAA problem is an $|I|$-dimensional vector, it can be defined as:

$$A(t) = \left\{a_1(t), \ldots, a_{|I|}(t)\right\} \tag{16}$$

where $a_i(t)$ represents the scaling strategy chosen by SFC i for resource adjustment at time t. It should be noted that after selecting the action, the successful adjustment must satisfy the constraints Eqs. (5)–(9).

Transition Probability: It means that the probability distribution of the network transitioning from one state to another after executing an action in a given state. In this paper, the state transition probability is unknown because the transition to the next state depends not only on the chosen action, but also on external factors.

Reward: In MDP, each state transition is accompanied by a reward signal, which represents the evaluation of agent on that transition. The reward can be obtained immediately, it can be defined as:

$$r(t) = \begin{cases} -C_{total}(t) - RD(t), & \text{succeed} \\ -\sigma, & \text{otherwise} \end{cases} \tag{17}$$

In Eq. (17), we consider that a successful scaling operation is performed without violating constraints Eqs. (5)–(9). In our adjustment problem, its goal is to minimize the long-term cost and the standard deviation of network resources. But the goal of agent is to maximize the long-term reward, so the reward can be defined as the negative value of the total cost and resource standard deviation. In addition, a penalty can be set $-\sigma$, where σ is much larger than the total cost and resource standard deviation, which means the penalty for SFC adjustment failure. So it makes that the agent can obey the constraints while keeping the objective. This transformation allows us to consolidate multiple objectives into a single reward formula. Unlike the approach in [26], which may handle multiple objectives separately, our strategy streamlines the learning process for the agent by presenting a unified reward signal. This simplification is motivated by the desire to enhance the learning efficiency of the agent in navigating the complex trade-offs between cost and resource distribution. By framing the problem in this way, we aim to provide the agent with a clear and unified objective that facilitates effective learning in the pursuit of optimal adjustments.

In reinforcement learning problem, the goal of agent is usually to maximize long-term cumulative rewards, not just immediate rewards. The cumulative reward value at time slot t can be defined as:

$$R(t) = r(t) + \gamma \cdot r(t+1) + \gamma^2 \cdot r(t+2) + \cdots = \sum_{n=0}^{\infty} \gamma^n \cdot r(t+n) \tag{18}$$

where γ is the discount factor, and n is the number of iterations.

4.2. Deep reinforcement learning-based VNF dynamic adjustment approach

MDP is usually solved by using dynamic programming (DP) or deep reinforcement learning (DRL) [27]. However, when the DP method is applied to problems with a large state space, it may lead to a dimensional disaster, requiring the storage and computation of a large number of states and policies. In addition, the DP method needs to know the complete model information in advance. On the other hand, the model-free nature of the DRL method makes it more suitable for problems with a large state space, where the complete model does not need to be known in advance. It can learn the optimal policy by updating Q-values based on the collected experiences [28]. In the adjustment problem, the SFC demand changes randomly, the state transition probability cannot be determined, and it has a large-dimensional state space and discrete action space. Therefore, we use a DRL method to solve it.

(1) Related Background

DQN: DQN, proposed by DeepMind in 2013 [10], is a deep reinforcement learning algorithm that is highly relevant to solving MDP problems. Traditional reinforcement learning methods face challenges when dealing with problems with large state spaces and continuous action spaces, as they need to store and update the Q-value of each state–action pair. While DQN exploits deep neural network to approximate the parameterized value function Q, representing the expected cumulative reward given a state and action. There are two networks in DQN: the main network with network parameters denoted as θ, and the target network with network parameters denoted as θ'. DQN calculates the $Q(S_t, a; \theta)$ of each action through the deep neural network in the main network. The action a_t can be selected by using the ε-greedy strategy which guarantees a certain amount of exploration. Thus, we can obtain:

$$a_t = \begin{cases} \arg\max_a Q(S_t, a; \theta), & 1 - \varepsilon \\ random, & \varepsilon \end{cases} \tag{19}$$

Executing the action a_t may get the reward r, and the state turns to S_{t+1}. Through this interaction, an experience (S_t, a_t, S_{t+1}, r) is obtained. DQN employs experience replay, randomly selecting a batch of experiences for training, to stabilize the training process. During the training, S_{t+1} is input into the target network, and we obtain $Q(S_{t+1}, a_{t+1}; \theta')$ for each action. We select the $\max_{a_{t+1}} Q(S_{t+1}, a_{t+1}; \theta')$, and then we can get the real value $\hat{y}$ which can be defined as Eq. (20).

$$\hat{y} = r + \gamma \cdot \max_{a_{t+1}} Q(S_{t+1}, a_{t+1}; \theta') \tag{20}$$

where γ is the discount factor.

With this real value, the loss can be computed (Eq. (21)) and used to update the parameters of the neural network, allowing the network to gradually learn the optimal Q function estimate.

$$Loss = \left(\hat{y} - \max_{a_t} Q(S_t, a_t; \theta)\right)^2 \tag{21}$$

Double DQN: Although DQN performs well on many problems, it still has some problems, such as overestimation. In traditional DQN, a single neural network is used to estimate the Q values of all actions simultaneously. This can lead to overestimation of certain actions, and causing the learned policy to be biased towards actions with higher value. To address this issue, DDQN was proposed by DeepMind in 2015 [29]. DDQN employs two independent neural networks to

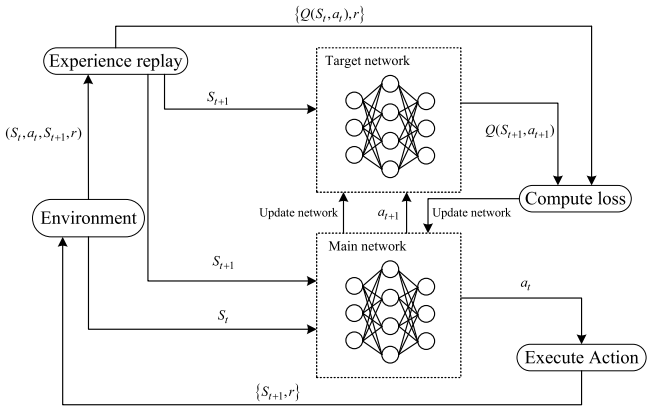


Fig. 3. The training process of DDQN.

estimate the optimal action and its corresponding Q value. Among them, the target network is used to select the optimal action, and the main network calculates the Q value of this action. The training process of DDQN is shown in Fig. 3. If (S_t, a_t, S_{t+1}, r) taken from the experience pool, then the $Q(S_t, a_t; \theta)$ can be obtained from the main network. When the next state S_{t+1} inputs to the main network, the action a_{t+1} corresponding to the $\max_{a_{t+1}} Q(S_{t+1}, a_{t+1}; \theta)$ can be selected. We input S_{t+1} to the target network, and we can obtain $Q(S_{t+1}, a_{t+1}; \theta')$. Thus, the DDQN actual value can be calculated as Eq. (22), while the predicted value is $Q(S_t, a_t; \theta)$. We use both of them to perform error back-propagation. The parameter of the main network is updated as Eq. (23). After several updates, the parameter of the main network is gradually synchronized to the target network.

$$\hat{y} = r + \gamma \cdot Q(S_{t+1}, a_{t+1}; \theta') \tag{22}$$

$$\theta \leftarrow \theta - [\hat{y} - Q(S_t, a_t; \theta)] \nabla_\theta Q(S_t, a_t; \theta) \tag{23}$$

(2) Intelligent Service Function Chain Dynamic Adjustment Algorithm

Since DDQN has strong stability and excellent performance in solving MDP, the DDQN-based ISFCDAA is proposed to address the challenges in SFC adjustment. ISFCDAA uses DDQN to dynamically adjust the allocation of VNF resources based on SFC demand. Considering the continuous changes in resource requirements and the need to maintain SLA, ISFCDAA is designed to optimize resource distribution while ensuring high-quality service delivery. We have introduced the mathematical model of the adjustment algorithm and discussed its key components, including state spaces, action selection, state transition probability, and reward computation. Furthermore, we have detailed the training process of the DDQN, which involves interacting with the environment and utilizing experience replay to improve data efficiency and enhance the learning process. Next we look at the details of ISFCDAA.

As shown in Algorithm 1, lines 1 to 22 describe the training process of ISFCDAA. At the beginning of the algorithm, an experience pool E is initialized to store (S_t, a_t, S_{t+1}, r) (line 1). At the same time, we initialize the parameters of the main network and the target network (lines 2–3). Then ISFCDAA is trained for $episode_{num}$ times, for each episode, first initializing the state and then doing the following steps:

1. The ε-greedy strategy is used to select an action A in the main network (line 7).
2. Based on the selected action, a specific sub-algorithm (Algorithm 2, Algorithm 3, or Algorithm 4) is called to adjust the SFC (lines 8–17).
3. After adjusting, we can obtained the reward r which is computed using Eq. (17), and the next state S_{t+1}. Then we store them in the experience pool E (line 18).
4. After accumulating enough experiences in the pool, the algorithm randomly samples a batch of samples and uses the gradient descent method to update the parameter of the main network, like Eq. (23) (line 19).
5. After updating Y times, the parameter of the main network is copied to the target network to stabilize and improve the learning process (line 20).

Once the training is complete, the trained network model is saved. The SFC adjustment strategy π is obtained by inputting SFCs into the main network (line 23).

Algorithm 1: Intelligent Service Function Chain Dynamic Adjustment Algorithm

Input: Penalty value $-\sigma$, initial exploration rate ε, discount factor γ, learning rate, update times T, SFCs
Output: Adjustment Policy π

```
1  Initialize experience pool E;
2  Initialize the parameter of the main network θ;
3  Initialize the parameter of the target network θ' = θ;
4  for episode ← 1 to episode_num do
5      Initialize state space state = {u_1, ..., u_N; κ_1, ..., κ_M};
6      for i ← 1 to sfc_num do
7          Take the ε-greedy strategy to select action A in the
             main network;
8          if A == 0 then
9              Call Algorithm 2;
10         else
11             if A == 1 then
12                 Call Algorithm 3;
13             end
14             if A == 2 then
15                 Call Algorithm 4;
16             end
17         end
18         Get reward r and next state S_{t+1}, and store
             (S_t, a_t, S_{t+1}, r) in the experience pool E;
19         Randomly sample a small batch from the experience
             pool E, and update the main network parameter θ
             using gradient descent;
20         Copy the main network parameter θ to the target
             network θ' after updating Y times;
21     end
22 end
23 Get the SFC adjustment strategy π through the main network;
```

During the execution of the action, if the selected action involves adjusting the resources of the VNF at the current node, Algorithm 2 is employed. If the demand of the VNF r_j^i is less than the remaining resources $C_u.avi$ on the current node, the resource adjustment is carried out (line 3). The state, reward, and policy after adjusting are then returned (line 8). However, if the adjustment requires more resources than what is available on the current node, the adjustment process fails (line 5).

When the selected action involves creating an instance of the VNF on a new node, as shown in Algorithm 3, the algorithm first searches for candidate nodes which meet the resource requirements, and then sorts them in descending order according to the remaining resources of the nodes (line 1). Then we screen the candidate nodes. If the link resources are insufficient or the delay constraint is violated after node instantiation, the node should be removed. Otherwise, the length of forwarding path after instantiation will be recorded (lines 3–8). If the list of candidate node is empty, the adjustment fails (lines 9–10). Conversely, it indicates a successful adjustment. The policy with the shortest forwarding path and the reward are returned (line 12).

Algorithm 2: Scale up

Input: SFCs
Output: Adjustment result π_i
for j *in* F_i **do**
  if $r_j^i < C_u.avi$ **then**
    Expand/shrink instance resources on the current node;
  else
    Return state, $-\sigma$, failure, π_i;
  end
end
Return state, r, success, π_i;

Algorithm 3: Instantiate VNF

Input: SFCs
Output: Adjustment result π_i
Find the candidate nodes $Candia\_node$ that satisfy VNF resources and have no identical instances, and sort them in descending order of remaining resources;
for *node in* $Candidate\_node$ **do**
  if *the link resource and delay are satisfied* **then**
    Record forwarding path;
  else
    Remove the candidate node;
  end
end
if $Candidate\_node == NULL$ **then**
  Return state, $-\sigma$, failure, π_i;
else
  Return state, r, success, π_i;
end

When the selected action involves migrating the VNF, except that the candidate node in Algorithm 4 is to find a node with the same instance, the other process is the same as Algorithm 3.

Algorithm 4: Migrate VNF

Input: SFCs
Output: Adjustment result π_i
Find the candidate nodes $Candia\_node$ that satisfy VNF resources and have the same instance, and sort them in descending order of remaining resources;
for *node in* $Candidate\_node$ **do**
  if *the link resource and delay are satisfied* **then**
    Record forwarding path;
  else
    Remove the candidate node;
  end
end
if $Candidate\_node == NULL$ **then**
  Return state, $-\sigma$, failure, π_i;
else
  Return state, r, success, π_i;
end

(3) Complexity Analysis

The time complexity of Algorithm 2 is $O(|F_i|)$, where $|F_i|$ represents the number of VNFs in each service request. The time complexity of Algorithm 3 and Algorithm 4 is both $O(|N|)$, where $|N|$ represents the number of physical nodes. $O(|d_{in}| \cdot |d_{out}|)$ represents the time complexity of the fully connected network, where $|d_{in}|$ is the input dimension and $|d_{out}|$ is the output dimension. Therefore, the time complexity of Algorithm 1 is $O(episode_{num} \cdot |I| \cdot |F_i| \cdot |N|^2 \cdot |d_{in}| \cdot |d_{out}|)$, where $|I|$ denotes the number of requests, $episode_{num}$ represents the times of training.

Table 3
Network parameters.

Parameters	Value
Number of nodes	17
Number of links	26
Number of VNF	6
Capacity of nodes	(30,50)
Bandwidth of links	10 Gbps
Unit delay of links	1 ms
Cost coefficient	$\alpha_1 = 1$, $\alpha_2 = 1$

Table 4
Parameters of SFC.

Network service	SFC	Bandwidth	Delay
Web service	NAT-FW-TM-WOC-IDS	100 kbps	500 ms
VoIP	NAT-FW-TM-FW-NAT	64 kbps	100 ms
Video streaming	NAT-FW-TM-WOC-IDS	100 kbps	500 ms

5. Experimental results and discussion

In this section, we perform simulation experiments based on Pytorch. Firstly, we discuss the simulation setup used to evaluate the performance of our proposed ISFCDAA. Secondly, we introduce two baselines and four evaluation indicators. Finally, we demonstrate the effectiveness of ISFCDAA in minimizing adjustment consumption, resource balancing and long-term profits by comparing its performance with the other two benchmark algorithms.

5.1. Simulation setup

The physical network used in the experiment is based on the Nobel-Germany topology from SDNlib [9]. It consists of 17 nodes and 26 links. Each CPU resource is initialized within the range of (30, 50), and the bandwidth of the physical link is set to 10 Gbps. The delay between adjacent physical nodes is proportional to the physical distance between them [11], and the unit delay for each link is 1 ms. When the constraint is violated, we set the penalty parameter $\sigma = 10000$. The cost of resource adjustment is calculated using a linear function, and the cost correlation coefficients are both to 1. The parameters of the substrate network are listed in Table 3

Under a single slot, we assume that there are already some SFCs placed in the network by using the algorithm in [30]. For generating SFCs, we have considered three real world requests [11] as shown in Table 4.

In this paper, we use Pytorch to build a 3-layer fully connected network, and its main parameters are shown in Table 5. The input layer is same as the state dimension, and the hidden layer contains 128 neurons that are activated by Rectified Linear Unit (ReLU). In DDQN, the role of Greedy Exploration is to balance exploration and utilization. However, if we only use it, the algorithm may fall into a local optimal solution and cannot find a better solution. Therefore, in order to ensure the exploratory nature of the algorithm, we introduce the ε-greedy strategy, with an initial exploration rate of 0.9. It will decrease as $\varepsilon(t+1) = \max(\varepsilon_{\min}, \varepsilon(t) - decay_{rate})$. In order to improve the sample utilization during the learning process and reduce the correlation between continuous samples, the size of the memory pool is set to 600. And we adopt the Adam optimizer, with a learning rate set to 0.015. A batch size set to 32. In addition, in order to improve the stability and reduce the deviation of the target value, we set the update period is to 200.

Table 5
Hyper parameters of ISFCDAA.

Parameters	Value
Penalty parameter σ	10 000
Initial exploration ratio $\epsilon(t)$	0.9
The minimum exploration ratio ϵ	0.05
Decay rate	0.005
Input layer	119
Batch size	32
Discount factor	0.95
Replay memory size	600
Optimizer	Adam with learning rate 0.015
Target network update period	200

5.2. Baseline

To evaluate the improvements and performance advantages of our ISFCDAA algorithm in a similar context, we compare it with the following two benchmark algorithms:

- Deep Q-Network(DQN): Based on the problem model of this paper, DQN is used to solve it;
- REAP [11]: A heuristic algorithm proposed to address the VNF reconfiguration problem in virtualized data centers. The goal of REAP is to minimize energy consumption and reconfiguration overheads in NFV infrastructure, considering scale up, scale out, and migration as reconfiguration solutions

5.3. Evaluation indicators

Similar to the work [23], we set up four evaluation indicators.

Physical resources distribution: Measure the balance of physical resource distribution. It can be obtained by Eq. (10).

Total cost: The total cost spent on adjusting the SFC in T time slots, which can be obtained by Eq. (13). C_b is set to 2 units and C_{ins} is set to 5 units.

Acceptance rate: The request acceptance rate in T time slots can be defined as:

$$AC(T) = \frac{\sum_{t=0}^{T} AC_{suc}(t)}{\sum_{t=0}^{T} |I|} \tag{24}$$

where $|I|$ is the number of SFC whose resources change, and $AC_{suc}(t)$ is the number of SFC that be satisfied at time slot t.

Long-term profit: The total profit earned by the network operator for serving requests in T time slots, which can be defined as:

$$Pro(T) = \int_{t=0}^{T} \sum_{i \in I} (Ben_{cpu} \sum_{j \in F_i} r_j^i + Ben_{bw} \sum_{f_{i,j} \in F_i} b_{f_{i,j}}) \tag{25}$$

where Ben_{cpu} is the revenue per CPU resource, and Ben_{bw} is the revenue per bandwidth resource. Both of them are set to 1.

5.4. Simulation results and analysis

(1) Experiment1—Performance with different number of SFCs

First of all, we use three algorithms to dynamically adjust the resources of different numbers of SFCs in a single time slot. As shown in Fig. 4, the proposed ISFCDAA achieves the most optimal resource distribution after adjusting resources under various numbers of SFCs. While the effect of DQN is slightly inferior to ISFCDAA, the resource distribution after applying REAP is the worst. Specifically, the mean standard deviation of the resource distribution of ISFCDAA is 2.41% lower than that of DQN and 9.90% lower than that of REAP. As shown in Fig. 5, under 50 SFCs, the standard deviation of resource distribution of ISFCDAA is 6.11% lower than that of DQN and 15.47% lower than that of ISFCDAA.

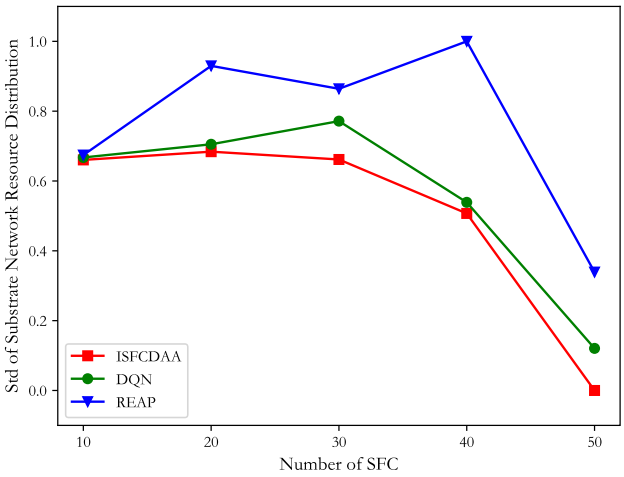


Fig. 4. Physical resources distribution.

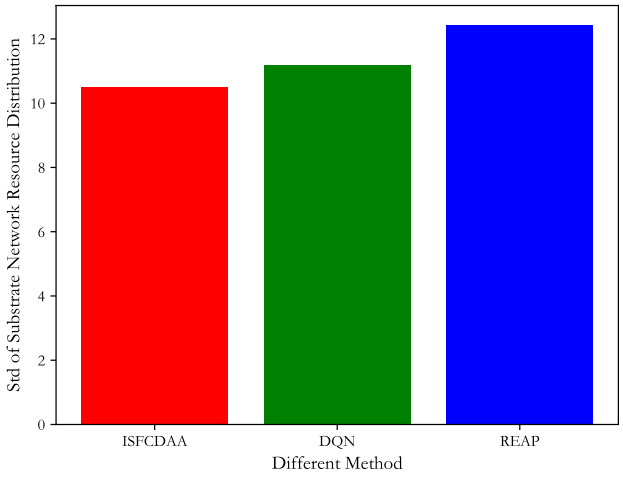


Fig. 5. Physical resources distribution with 50 SFCs.

However, as shown in Fig. 6, the adjustment cost of the algorithm in this paper is slightly higher than that of the REAP algorithm. When the number of SFCs increases to 20, the total cost of the ISFCDAA algorithm becomes lower than that of DQN but higher than that of REAP. On average, the total cost of ISFCDAA is 14.29% higher than REAP and 5.14% lower than DQN. This indicates that during the resource adjustment process, the ISFCDAA algorithm requires a higher cost to achieve the optimal resource distribution.

In this paper, our total cost includes instantiation cost and migration cost. As shown in Figs. 7 and 8, the ISFCDAA algorithm chooses two adjustment schemes: creating new instances and migration, which are relatively balanced, so that resource distribution is more effective. However, the REAP algorithm prefers to perform VNF instantiation which involve relatively little reconfiguration overhead, aligning with the experimental results of the study [11]. In pursuit of balanced resource distribution, the ISFCDAA algorithm might opt for strategies with slightly higher adjustment costs during the adjustment process. Therefore, almost all of the costs of ISFCDAA are higher than those of REAP.

(2) Experiment2—Long-term Performance

Since ISFCDAA can achieve the optimal resource distribution in a single time slot, but the cost is slightly higher than the benchmark algorithm REAP. In order to further verify the effectiveness of ISFCDAA, we compare the SFC acceptance rate of the three algorithms in 35 time slots. Among them, 20 SFCs generate resource changes in each

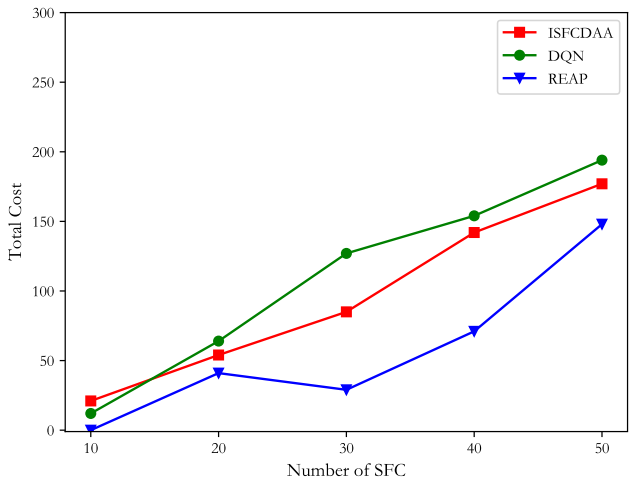


Fig. 6. Total cost.

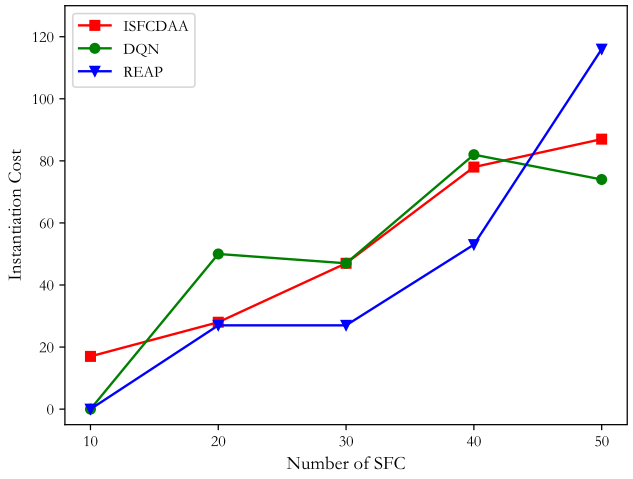


Fig. 7. Instantiation cost.

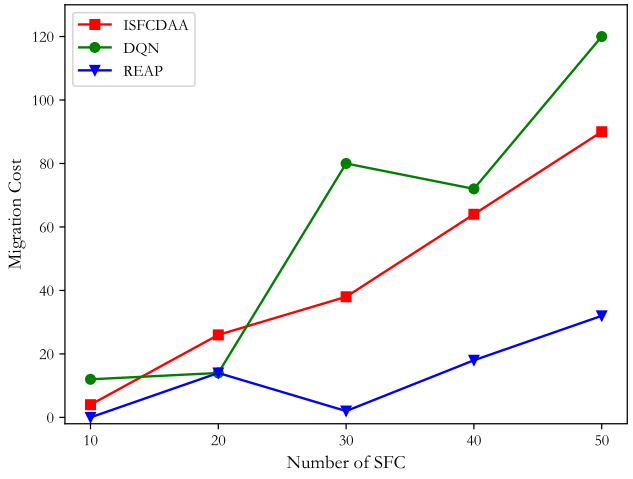


Fig. 8. Migration cost.

time slot. As shown in Fig. 9, the ISFCDAA demonstrates superior adaptability to changes in SFC resources, achieving a higher acceptance rate through resource adjustment. On average, the acceptance rate of ISFCDAA is 15.11% higher than that of DQN and 39.57% higher than that of REAP. Compared with REAP, DQN performs better. Although it

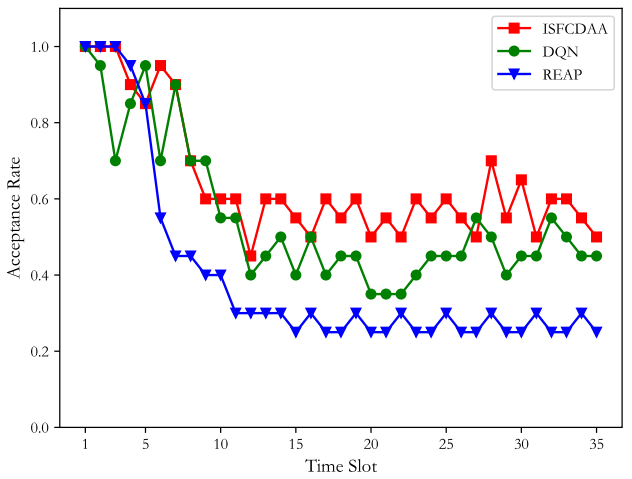


Fig. 9. Acceptance rate.

is not as good as DDQN, it can still adapt to the change of SFC resources to some extent and achieve a higher acceptance rate. Besides, the REAP algorithm performs poorly on SFCs with significant resource changes and cannot meet its resource requirements. Because the resource distribution of REAP is the worst, it cannot flexibly adapt to large resource changes. When the subsequent SFC requirements change, the remaining physical network resources are uneven. As a result, even if the SFC wants to migrate or instantiate a new VNF, its QoS cannot be satisfied, leading to a poor acceptance rate.

As shown in Fig. 10, the total cost of REAP is the lowest in the first two time slots, and then gradually increases, even surpassing the other two algorithms. After the fifth time slot, the total cost of REAP drops suddenly, and then remains at lower level. As for why the total cost of REAP in Fig. 10 reaches to zero, it is because, as indicated by Figs. 3, 4, and 6, REAP incurs the lowest cost during the reconfiguration process, but with the highest standard variance in resource distribution. This suggests that REAP leans more towards cost control, neglecting resource distribution and resulting in fragmented resources on each node. Over multiple time slots, as SFCs resources change, subsequent adjustments become challenging, potentially violating SLAs. Consequently, horizontal scaling and migration become unfeasible, leading to a zero adjustment cost. However, the total cost of ISFCDAA lies in the middle before the first 5 time slots, which is 16.92% lower than the total cost of DQN and 11.09% higher than REAP. After the fifth time slot, the total cost decreases and remains at a low level, although it surpasses REAP. It remains 7.99% lower than the total cost of DQN on average. However, the acceptance rate of REAP and DQN are much lower than that of ISFCDAA.

Therefore, this paper compares the benefits brought by the adjustments of each algorithm under 35 time slots, as shown in Fig. 11. It is evident that ISFCDAA yields the highest long-term profit, followed by DQN, while REAP has the lowest long-term profit. The average long-term profit of ISFCDAA is 21.47% higher than DQN and 42.92% higher than REAP. This proves the flexibility and superiority of ISFCDAA in the environment with dynamic resource. Besides, it underscores the performance advantages of DQN over REAP and the limitations of REAP in handling large resource changes.

6. Conclusion

This paper aims to address the challenges posed by dynamic demand of SFC in the context of NFV infrastructure. We propose an intelligent adjustment algorithm based on deep reinforcement learning to achieve a balanced distribution of network resources while minimizing the associated adjustment costs and meeting user requirements. We first

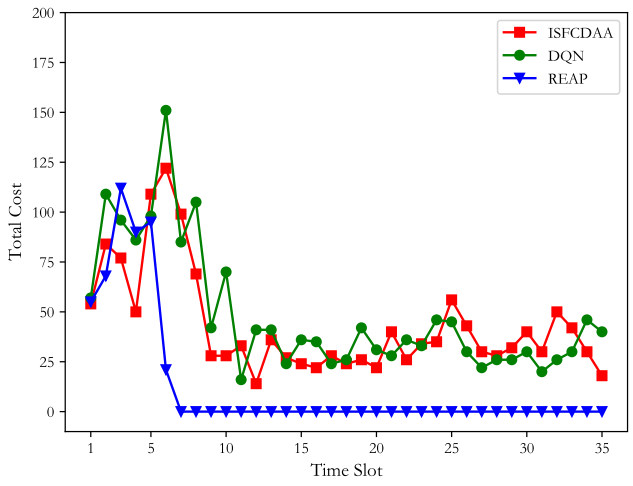


Fig. 10. Total cost.

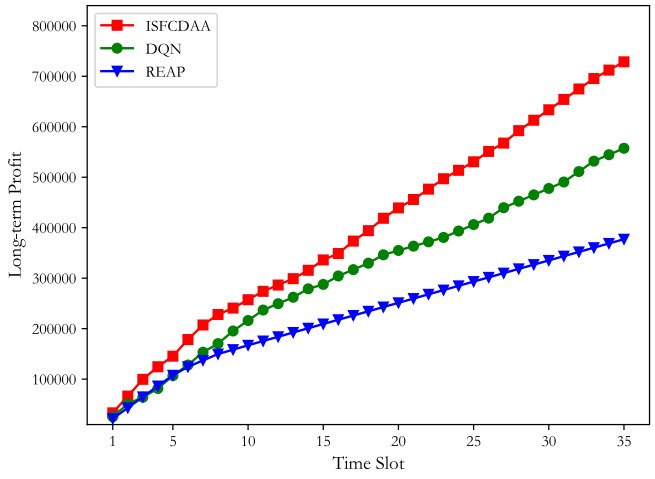


Fig. 11. Long-term profit.

formulate the problem as an ILP and transform the optimization process into MDP. We then propose an efficient algorithm for the problem, through jointly considering vertical scaling, creating new instances of VNF, and VNF migration. Finally, we evaluate the performance of ISFCDAA with other adjustment methods. Simulation results demonstrate that ISFCDAA achieves a more balanced resource distribution in a single time slot. Moreover, in the long term, the acceptance rate and long-term benefits of ISFCDAA are also significantly better than those of the DQN and REAP algorithms, providing an effective solution for resource management in NFV orchestrator. In future research, we plan to investigate a proactive strategy that incorporates a prediction mechanism to account for the volatility of demands. And inspired by the literature [21], we will add the impact of service interruption to find a more complete adjustment strategy.

CRediT authorship contribution statement

Yuantao Wang: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Zhaogang Shu:** Writing – review & editing, Supervision, Resources, Funding acquisition. **Shuwu Chen:** Supervision, Resources. **Jiaxiang Lin:** Supervision, Resources. **Zhenchang Zhang:** Supervision, Resources.

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.

Data availability

The authors do not have permission to share data.

Acknowledgments

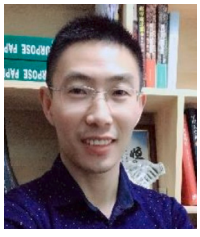
This work was supported by Natural Science Foundation by Technology Department of Fujian Province, China (No. 2020J01574), Industry-University-Research Innovation Fund for Future Network Technology by Education Department of China (No. 2021FNA05003), Industry-Research Project from Network Communication Company (No. KH230139A).

References

[1] Network functions virtualization-white paper3, https://portal.etsi.org/Portals/0/Tbpages/NFV/Docs/NFV_White_Paper3.pdf.
[2] F. Hu, Q. Hao, K. Bao, A survey on software-defined network and openflow: From concept to implementation, IEEE Commun. Surv. Tutor. 16 (4) (2014) 2181–2206.
[3] C. Bu, J. Wang, X. Wang, Towards delay-optimized and resource-efficient network function dynamic deployment for VNF service chaining, Appl. Soft Comput. 120 (2022) 108711.
[4] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, S. Davy, Design and evaluation of algorithms for mapping and scheduling of virtual network functions, in: Proceedings of the 2015 1st IEEE Conference on Network Softwarization, NetSoft, IEEE, 2015, pp. 1–9.
[5] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, 2010, pp. 267–280.
[6] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011, pp. 1–12.
[7] M. Savi, M. Tornatore, G. Verticale, Impact of processing-resource sharing on the placement of chained virtual network functions, IEEE Trans. Cloud Comput. 9 (4) (2019) 1479–1492.
[8] D. Zhao, G. Sun, D. Liao, S. Xu, V. Chang, Mobile-aware service function chain migration in cloud–fog computing, Future Gener. Comput. Syst. 96 (2019) 591–604.
[9] S. Orlowski, R. Wessäly, M. Pióro, A. Tomaszewski, SNDlib 1.0—Survivable network design library, Netw.: Int. J. 55 (3) (2010) 276–286.
[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013, arXiv preprint arXiv:1312.5602.
[11] S. Padhy, J. Chou, Reconfiguration aware orchestration for network function virtualization with time-varied workload in virtualized datacenters, IEEE Access 9 (2021) 48413–48428.
[12] H. Hawilo, M. Jammal, A. Shami, Orchestrating network function virtualization platform: Migration or re-instantiation? in: 2017 IEEE 6th International Conference on Cloud Networking, CloudNet, IEEE, 2017, pp. 1–6.
[13] T. Buh, R. Trobec, A. Ciglič, Adaptive network-traffic balancing on multi-core software networking devices, Comput. Netw. 69 (2014) 19–34.
[14] V. Eramo, M. Ammar, F.G. Lavacca, Migration energy aware reconfigurations of virtual network function instances in NFV architectures, IEEE Access 5 (2017) 4927–4938.
[15] S. Ayoubi, Y. Zhang, C. Assi, A reliable embedding framework for elastic virtualized services in the cloud, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 489–503.
[16] Q. Zhang, F. Liu, C. Zeng, Online adaptive interference-aware VNF deployment and migration for 5G network slice, IEEE/ACM Trans. Netw. 29 (5) (2021) 2115–2128.
[17] O. Houidi, O. Soualah, W. Louati, M. Mechtri, D. Zeghlache, F. Kamoun, An efficient algorithm for virtual network function scaling, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017, pp. 1–7.
[18] D. Harutyunyan, R. Behravesh, N. Slamnik-Kriještorac, Cost-efficient placement and scaling of 5G core network and MEC-enabled application VNFs, in: 2021 IFIP/IEEE International Symposium on Integrated Network Management, IM, IEEE, 2021, pp. 241–249.

[19] Z. Chen, H. Li, K. Ota, M. Dong, HyScaler: A dynamic, hybrid VNF scaling system for building elastic service function chains across multiple servers, IEEE Trans. Netw. Serv. Manag. (2023).
[20] V. Eramo, E. Miucci, M. Ammar, F.G. Lavacca, An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures, IEEE/ACM Trans. Netw. 25 (4) (2017) 2008–2025.
[21] G. Sun, R. Zhou, J. Sun, H. Yu, A.V. Vasilakos, Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization, IEEE Internet Things J. 7 (7) (2020) 6116–6131.
[22] R.A. Addad, D.L.C. Dutra, T. Taleb, H. Flinck, Ai-based network-aware service function chain migration in 5g and beyond networks, IEEE Trans. Netw. Serv. Manag. 19 (1) (2021) 472–484.
[23] H. Feng, Z. Shu, T. Taleb, Y. Wang, Z. Liu, An aggressive migration strategy for service function chaining in the core cloud, IEEE Trans. Netw. Serv. Manag. (2022).
[24] A. Fischer, J.F. Botero, M.T. Beck, H. De Meer, X. Hesselbach, Virtual network embedding: A survey, IEEE Commun. Surv. Tutor. 15 (4) (2013) 1888–1906.
[25] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[26] Y. Bi, C.C. Meixner, M. Bunyakitanon, X. Vasilakos, R. Nejabati, D. Simeonidou, Multi-objective deep reinforcement learning assisted service function chains placement, IEEE Trans. Netw. Serv. Manag. 18 (4) (2021) 4134–4150.
[27] J. Ye, Y.-J.A. Zhang, DRAG: Deep reinforcement learning based base station activation in heterogeneous networks, IEEE Trans. Mob. Comput. 19 (9) (2019) 2076–2087.
[28] B.R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A.A. Al Sallab, S. Yogamani, P. Pérez, Deep reinforcement learning for autonomous driving: A survey, IEEE Trans. Intell. Transp. Syst. 23 (6) (2021) 4909–4926.
[29] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, 2016.
[30] Y. Wang, Z. Shu, Y. Zhong, C. Qiu, J. Tian, Service function chain placement algorithm based on VNF instance sharing, Appl. Res. Comput. 40 (6) (2023) 1806–1811.

Yuantao Wang is currently pursuing the master's degree at computer science and technology, Fujian Agriculture And Forestry University. Her research interests include software defined networking and network function virtualization. She is now undertaking the research work on reducing the operating expense of network operators with the constrained optimization.



Zhaogang Shu is currently an Associate Professor at the College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, China. He also is the director of Department of Computer Science and Cloud Computing Lab, Fujian Agriculture and Forestry University. He received B.S. and M.S. degrees in computer science from Shantou University, China in 2002 and 2005 respectively. He also received Ph.D. degree from South China University of Technology, Guangzhou, China, in 2008. From Sept. 2008 to July 2012, he worked as a senior engineer and project manager at Ruijie Network Corporation, Fuzhou, China. From Oct. 2018 to Oct. 2019, he worked as a visiting professor in MOSIAC lab at the Department of Communications and Networking, Aalto University, Finland. He directed more than 10 research projects and was the author of more than 30 papers and 5 patents. His research interests include software-defined network, network function virtualization, 5G network and next generation network architecture, network security, machine learning based network optimization, cloud computing and edge computing. He serves as the reviewers of many famous journals on network and communications, including IEEE Network, IEEE/ACM Transactions on Networking, IEEE Transactions on Network Service and Management, ACM/Springer Mobile Networks, Elsevier Computer networks and so on. He also is the member of CCF (China Computer Federation) and Fujian Computer Society.

Shuwu Chen is currently a professor at the College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, China. He also is the director of Innovation Lab of IoT technology, Fujian Agriculture and Forestry University. He received bachelor's degree in industrial automation from Chang'an University, China, in 1998. And, he received master's degree in radio physics from Xiamen University, China, in 2003. He is the Co founder of Xiamen Four-Faith Communication Technology Co., Ltd., which focus on IoT technology and solutions. He directed dozens research projects and was the author of more than 10 patents. His research interests include IoT technology, edge computing and AI algorithm.

Jiaxiang Lin received the Ph.D. degree in Communication and Information System from Fuzhou University, China, in 2010. He is currently an Associate Professor with the College of Computer and Information Sciences, Fujian Agriculture and Forestry University, Fuzhou, China. He has hosted four national, provincial and ministerial level research projects, authored over 40 referred scientific papers and hold three patents of invention. His research interests include spatial data mining, artificial intelligence, and big data analysis.

Zhenchang Zhang is currently an Associate Professor at the College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, China. He has hosted and participated in a number of provincial and ministerial projects, including the fund project of Fujian Provincial Department of Education, the subsidy project of Fujian Provincial Marine Economy Development, the National Science and Technology Support Project, etc. He has obtained one national invention patent and more than 10 software copyrights. His research interests include parallel computing, blockchain, deep learning and data assimilation.