Network Slice Reconfiguration by Exploiting Deep Reinforcement Learning with Large Action Space

Fengsheng Wei, Gang Feng, *Senior Member, IEEE*, Yao Sun, Yatong Wang, and Shuang Qin, *Member, IEEE*, Ying-Chang Liang, *Fellow, IEEE*

***Abstract*—It is widely acknowledged that network slicing can tackle the diverse usage scenarios and connectivity services that the 5G-and-beyond system needs to support. To guarantee performance isolation while maximizing network resource utilization under dynamic traffic load, network slice needs to be reconfigured adaptively. However, it is commonly believed that the fine-grained resource reconfiguration problem is intractable due to the extremely high computational complexity caused by numerous variables. In this paper, we investigate the reconfiguration within a core network slice with aim of minimizing long-term resource consumption by exploiting Deep Reinforcement Learning (DRL). This problem is also intractable by using conventional Deep Q Network (DQN), as it has a multi-dimensional discrete action space which is difficult to explore efficiently. To address the curse of dimensionality, we propose a discrete Branching Dueling Q-network (discrete BDQ) by incorporating the action branching architecture into DQN, for drastically decreasing the number of estimated actions. Based on the discrete BDQ network, we develop an intelligent network slice reconfiguration algorithm (INSRA). Extensive simulation experiments are conducted to evaluate the performance of INSRA and the numerical results reveal that INSRA can minimize the long-term resource consumption and achieve high resource efficiency compared with several benchmark algorithms.**

***Index Terms*—network slice reconfiguration, deep reinforcement learning, core network slicing, Branch Dueling Q-network**

I. Introduction

THE next generation mobile network is envisioned to meet diversified service requirements for various scenarios, including enhanced Mobile BroadBand (eMBB), ultra Reliable Low Latency Communications (uRLLC), massive Machine Type Communication (mMTC) and other forthcoming applications. Such diverse applications have different or even contradictory requirements in terms of bandwidth, latency, energy efficiency, mobility, etc. For instance, mMTC supports a massive number of Internet of Things (IoT) devices which only send small data payloads sporadically [1], while eMBB devices are characterized by large steady payloads. Consequently, it is unreasonable to devise a one-size-fits-all network architecture to fulfill the diverging requirements. Benefiting from the development of Network Function Virtualization (NFV) and Software Defined Network (SDN) technologies, network slicing has been proposed as a key architectural technology to solve this problem [2]. As defined by the Next Generation Mobile Network Alliance (NGMN) [3], network slicing refers to a virtualization paradigm that enables multiple customized logical networks, i.e., network slices, to operate independently on top of the underlying physical network. A network slice can be viewed as a pre-defined virtual network through which its users can communicate with each other transparently as if they are using a physical network.

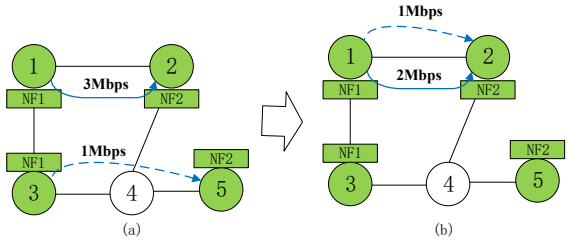


Fig. 1. The reconfiguration process of a network slice

A network slice consists of virtual nodes and virtual links, where virtual nodes can be implemented by NFV through virtual machines or containers running on general-purpose hardware [4]. A virtual link between a pair of virtual nodes can be established by SDN routers as interconnected physical links which may transverse several physical nodes. When a slice request is received, a network slice is initiated and configured by the management and orchestration (MANO) layer of network slicing. Unfortunately, the traffic carried by a network slice is inherently dynamic [5], [6]. An intuitive example is that network flow fluctuates following people's daily activities. Events like festival celebrations and sports events will incur a higher traffic demand compared to normal situations due to a large number of users simultaneously accessing the network slice.

Traffic load variations or traffic uncertainty in network slice may degrade resource utilization and deteriorate the Quality of Service (QoS). On the one hand, fluctuations in traffic demands will cause the optimal resource allocation to lose its optimality, thereby degrading resource utilization. On the other hand, it can also cause Service Level Agreement (SLA) violation and degrade the QoS of the slice. Consequently, it is necessary to perform slice reconfiguration that adjusts the resource allocation for a slice according to the variations of traffic demand, so as to maintain high resource efficiency

F. Wei, G. Feng, Y. Sun, Yatong Wang, S. Qin and Y. C. Liang are with the National Key Laboratory of Science and Technology on Communications, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the Center for Intelligent Networking and Communications (CINC), University of Electronic Science and Technology of China, Chengdu 611731, China. G. Feng is the corresponding author (email: fenggang@uestc.edu.cn; phone:+862861830292; fax: +862861830284).

This work was supported by Development Program in Key Areas of Guangdong Province under Grant 2018B010114001, and the National Science Foundation of China under Grant 61871099 and 61631005.

while meeting SLA. We refer to this problem as the Network Slice Reconfiguration Problem (NSRP). Let us use an example of Fig. 1 to illustrate the NSRP. The substrate network that serves a network slice is composed of five physical nodes and five physical links. All data flows in the network should be processed by the Service Function Chain (SFC) of the slice, which is an ordered sequence of Virtual Network Functions (VNFs) [7]. There are two physical paths that can serve the SFC of $nf1 \rightarrow nf2$. We assume that the capacity of the two paths (p_1: $1 \rightarrow 2$ and p_2: $3 \rightarrow 4 \rightarrow 5$) are both 3Mbps. Initially, flow f_1 (3Mbps, indicated by solid arrow) is routed on path p_1, and flow f_2 (1Mbps, indicated by dashed arrow) is routed on path p_2. As the demand of f_1 varies from 3Mbps to 2Mbps, it is possible to reconfigure the network slice by rerouting f_2 onto p_1 as in Fig. 1(b) so as to minimize resource consumption. We assume that after this operation, the demand of f_2 becomes 3Mbps at some time later. Therefore, the network slice needs another reconfiguration which reroutes f_2 onto p_2 to avoid SLA violation.

As shown in this example, reconfiguring network slice frequently is not always the best choice because the reconfiguration itself incurs certain resource overhead, such as control and management overhead in establishing and adjusting the links, rerouting overhead in routing disturbance, and retransmission overhead due to data loss [8], etc. Indeed, periodically solving the optimization problem instances to meet the instantaneous traffic demands may cause frequent reconfigurations to maintain the optimality of resource allocation, due to the lack of a prediction mechanism of future traffic requirements. Thus, an intelligent reconfiguration policy is urgently desired for network slicing to tackle the traffic uncertainties.

In this paper, we resort to Deep Reinforcement Learning (DRL) to solve NSRP. NSRP with long-term optimization objective is essentially a sequential decision problem, which is formulated as a Markov Decision Process (MDP) and can be solved by Reinforcement Learning (RL). However, for NSRP in a substrate network with non-trivial network slice, it is difficult to extract a set of effective features to represent the environment [9]. Moreover, traditional table-based RL is infeasible in solving NSRP due to its huge state space and the multi-dimensional discrete action space. To overcome these difficulties, we first simplify the representation of the environment based on the problem properties rather than directly representing the networks. Particularly, we use depth-first-search (DFS) algorithm to identify all the physical paths that can serve the SFC and use their capacities together with flow rates and history information of reconfiguration to represent the state of the environment. Second, we incorporate the action branching architecture into Dueling Double Deep Q-Network (Dueling DDQN) to address the large state space as well as the multi-dimensional discrete action space. Finally, we propose an intra-slice reconfiguration algorithm named Intelligent Network Slicing Reconfiguration Algorithm (INSRA) by combining the aforementioned algorithms.

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

1) We formulate the NSRP as an Integer Linear Programming (ILP) with aim of minimizing the long-term resource consumption and its NP-hardness is proved. To facilitate the representation of complex substrate network, we simplify its representation based on the problem properties and reformulate it as an MDP, thereby making it possible to use DRL to solve the NSRP.
2) We propose an intra-slice reconfiguration algorithm named INSRA to agilely reconfigure the core network slices. Specifically, we utilize a state-of-the-art neural network, i.e. the BDQ network to tackle the curse of dimensionality caused by the multi-dimensional discrete space. Simulation results demonstrate that BDQ can help compress the action space efficiently.
3) Simulation results reveal that the long-term cost of the network slice can be minimized compared with the benchmark algorithms. Meanwhile, the proposed algorithm can achieve a fairly stable performance under different network slice implementations and growing slice scales.

The remainder of the paper is organized as follows: Section II reviews the related work on network slice reconfiguration, highlighting the novelty of our contribution. In Section III and Section IV, we present the system model and the formulation of NSRP. We present the MDP modeling for the NSRP and elaborate our proposed INSRA in Section V. In Section VI, we present the numerical results, and finally we conclude the paper in Section VII.

II. Related Work

A body of related work has recently addressed the problem of traffic uncertainty in network slicing. Basically, these investigations fall into two categories, i.e., optimization-based and machine learning-based approaches. Since network slicing problem is a special case of virtual network embedding (VNE), which is NP-hard due to the capacity constraints [10], NSRP is generally formulated as an NP-hard problem. Therefore, most optimization-based solutions are carried out through approximate algorithms. On the other hand, to overcome the NP-hardness, explicit traffic prediction based on deep learning (DL) as well as implicit traffic prediction based on RL were exploited prevalently. In the following, we review the major related work on network slicing under traffic uncertainty.

A. Optimization-Based Approaches

NSRP is also called *dynamic network slicing* problem in the literature [11], [12]. The authors of [11] modeled the dynamic network slicing problem as a Mixed Integer Linear Programming (MILP) and solved it by using heuristic method. The authors of [13] formulated the network slicing problem as a mixed binary linear programming and proposed the penalty successive upper bound minimization algorithm. Wang *et al.* [14] modeled the network slice reconfiguration as a profit optimization problem and addressed it with a L_1-norm approximation method to maximize the profit of slice customers while reducing the reconfiguration cost.

Robust optimization is an effective tool to optimize problems with uncertainty constraints and objectives. Therefore, it is widely used in addressing the traffic uncertainty in

network slicing [15]–[17]. The authors of [15] designed a joint recovery and reconfiguration framework for network slicing by exploiting robust optimization. In addition, the authors of [16] addressed the traffic uncertainties in network slicing by employing Γ-robust uncertainty set. Similarly, the authors of [17] proposed a light-robustness optimization model for the network design and embedding problem considering the traffic uncertainties as well as the physical network failures.

Some researchers consider the network slicing problem as a VNF Forward Graph (VNF-FG) embedding problem. The authors of [18] formulated the network slicing problem based on queuing theory and a near-optimal heuristic algorithm was proposed to optimize the service delay. The authors of [19] investigated the network slicing problem with aim of optimizing the end-to-end throughput. Their model captures the key features in the 5G network, such as VNF interference, complex VNF-FGs, and the difference between edge cloud and core cloud. The authors of [20] proposed a two-step method for solving the VNF-FG design and VNF placement for 5G mobile networks, aiming at minimizing bandwidth consumption. In [21], the authors presented a methodology to make joint VNF placement and CPU allocation decisions in 5G.

These approaches are subject to some limitations. On the one hand, as we discussed earlier, these approaches may cause frequent reconfigurations due to the lack of prediction mechanism on the future traffic demands. On the other hand, some parameters can only be obtained during run-time, and thus the reconfiguration decisions are made after the traffic variations take place. Our proposed method can autonomously learn the future traffic demands and thus can proactively reconfigure the network slice.

B. Machine Learning-Based Approaches

To tackle the traffic dynamics due to user behaviors, a wide range of prediction mechanisms are adopted to predict the user behaviors as well as the resource demands. The authors of [22] used a neural-network to predict the number of instances of VNF thus to proactively perform the scaling and then an ILP was formulated to place these VNFs in the edge network. However, due to the limited bandwidth of the edge network, such a centralized learning mechanism is not practical since it needs to send numerous training data to the central controller. The authors of [23] proposed a VNF-FG embedding algorithm to meet the ever-changing resource availability of the physical servers and the continuous mobility of the users. To reduce the problem complexity, an optimized k-medoids clustering approach was applied to proactively partition the substrate network.

Network slice reconfiguration under traffic uncertainties is inherently a sequential decision problem which can be potentially solved by using RL. Recently, with the advances in computing power, RL is commonly used to automate the resource management in network slicing [24]–[29]. The authors of [24] proposed a learning-based framework for RAN slicing by jointly using Deep Learning (DL) and DRL. The authors of [25] proposed an online adaptive DRL approach to automatically embed the stochastically arrived SFC requests. However, their solutions perform poorly because the sequential processing of the VNF in a flow by flow fashion may reduce its optimality dramatically. The authors of [26] proposed an accelerated RL method that can learn proper VNF sizing and placement under various environments. However, these algorithms are inappropriate for the problem where the action space is naturally discrete [30]. To deal with the problem with large-scale discrete space, the authors of [28] modified the Deep Deterministic Policy Gradient (DDPG) with a heuristic algorithm to convert the continuous action to discrete feasible actions in VNF-FG embedding problem. However, this method is essentially a static solution to the optimization problem, disregarding the traffic dynamics. On the other hand, to the best of our knowledge, there is no prior work that addresses the fine-grained network slice reconfiguration with naturally discrete action space which is just the focus of our work in this paper.

III. System Model

In this paper, we consider the intra-slice reconfiguration problem for a specific slice, in which fine-grained reconfiguration of route paths, bandwidth, and the association of VNF instances are involved. Due to the nature of performance and resource isolation between slices [31], intra-slice reconfigurations can be performed for individual slices.

A sliced network consists of two logical parts, namely substrate network and network slice. The substrate network is an underlying physical network which is composed of forwarding servers and servers with specific VNF. A network slice can be fundamentally described as a set of traffic flows that traverse an SFC consisting of VNFs. Fig. 2 shows a substrate network and one service flow. In the following, we provide the substrate network model as well as the network slice model in the considered system. For ease of reference, the notations used in this paper are summarized in Table I.

Substrate Network: We model the substrate network as a weighted directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ denote the sets of nodes and links respectively. The physical nodes, denoted by $\mathcal{V} = \{1, 2, \cdots, N\}$, can be classified into two types: VNF-capable nodes and common nodes. The VNF-capable node can provide certain types of VNFs such as Mobility Management Entity (MME), Network Address Translation (NAT), Firewall, etc [32]. The common nodes have no VNF capability and are used only for packet forwarding. In addition, we denote the set of VNFs of the system by $\mathcal{F}$ and use the binary indicator $h_i(\pi) \in \{0, 1\}$ to indicate whether node $i \in \mathcal{V}$ is capable of VNF $\pi \in \mathcal{F}$.

Network Flows: There are M service flows in the system and the set of flows is denoted by $\mathcal{K}$. The kth flow is denoted by $(s_k, d_k, \mu_k(t))$, which represent the source node, destination node, and the rate of flow k at time t, respectively. Note that we use subscript t to denote the time throughout the paper. Different from the works [13], [14], [33] which allow flow splitting through physical nodes and/or physical links, we assume that each flow is mapped exactly onto one physical path in order to avoid coordination overhead caused

TABLE I
SUMMARY OF NOTATIONS

Notation	Definition
$\mathcal{V}, \mathcal{E}$	the physical nodes and physical links, respectively
$s_k, d_k, \mu_k(t)$	the source node, destination node, and the rate of flow k, respectively
$\mathcal{F}_k$	the SFC of flow k
π_m^k	the mth VNF of flow k
(k, π_m^k)	the virtual flow between virtual node π_m^k to π_{m+1}^k of flow k
$h_i(\pi)$	indicator variable that indicates whether or not node i is capable of VNF π
$x_{i,k}(\pi_m^k)$	node mapping variable indicating whether or not node i provides function π_m^k for flow k
$z_{ij}(k, \pi_m^k)$	link mapping variable indicating whether or not virtual flow (k, π_m^k) is mapped onto physical link (i, j)
$\alpha(\pi_m^k)$	unit computational resource consumption for VNF π_m^k
β	unit bandwidth consumption
$\boldsymbol{\delta}_x, \boldsymbol{\delta}_z$	cost coefficients of the reconfiguration on nodes and links, respectively

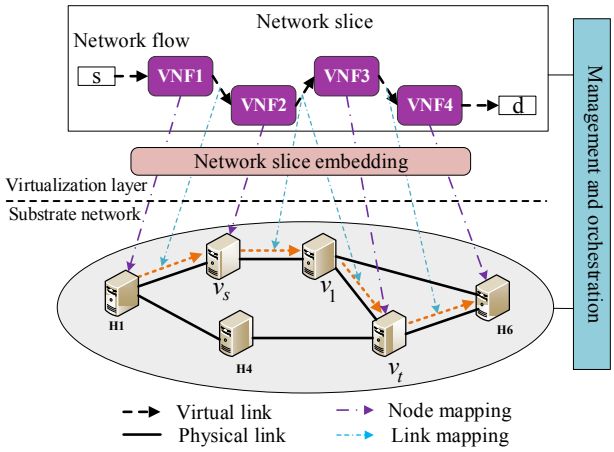


Fig. 2. System model

by flow splitting. Moreover, our model considers the dynamic traffic requirement instead of fixed flow rate. We assume that the SFCs of the flows in the considered network slice are the same, while the deployed nodes of VNFs could be different for each flow. Hence, we define the SFC of flow k as $\mathcal{F}_k = (\pi_1^k \rightarrow \cdots \rightarrow \pi_{L_k}^k)$. For ease of presentation, we define VNF π_m^k of flow k to be its *virtual node*. Accordingly, we define the flow segment between π_m^k and π_{m+1}^k to be its *virtual link*, which is denoted by (k, π_m^k).

Flow mapping: To model the virtual node mapping for flow k, we use the binary variable $x_{i,k}(\pi_m^k) \in \{0, 1\}$ to denote whether node i provides function π_m^k for flow k (i.e., $x_{i,k}(\pi_m^k) = 1$ if node i provides function π_m^k for flow k, otherwise $x_{i,k}(\pi_m^k) = 0$. For virtual node π_m^k, only the physical node which is capable of π_m^k can be its mapping target. Therefore, we have

$$x_{i,k}(\pi_m^k) \leq h_i(\pi_m^k), \forall i \in \mathcal{V}, \forall k \in \mathcal{K}, \pi_m^k \in \mathcal{F}_k. \quad (1)$$

As mentioned above, we present our model in the case where only one physical node serves VNF π_m^k for flow k. This translates into:

$$\sum_i x_{i,k}(\pi_m^k) = 1, \forall k \in \mathcal{K}, \pi_m^k \in \mathcal{F}_k. \quad (2)$$

In addition, we require each substrate node provides at most one VNF for each SFC [13], i.e.,

$$\sum_{\pi_m^k \in \mathcal{F}_k} x_{i,k}(\pi_m^k) \leq 1, \forall i \in \mathcal{V}, k \in \mathcal{K}. \quad (3)$$

To model the virtual link mapping for flow k, we propose to map its virtual links sequentially. In particular, the virtual link (k, π_m^k) is mapped to physical link (i, j) if $z_{ij}(k, \pi_m^k) = 1$, otherwise $z_{ij}(k, \pi_m^k) = 0$, where $z_{ij}(k, \pi_m^k)$ is a binary variable indicating the virtual link mapping. Please note that the binary variable $z_{ij}(k, \pi_m^k)$ ensures that flow k is not split by physical links. On the other hand, $z_{ij}(k, \pi_m^k)$ should satisfy the flow conservation law on all nodes. Since the placement of the source node s_k and the destination node d_k have specified by flow k, we only need to consider the mapping of the other VNFs of flow k. For s_k and d_k, we have

$$\sum_j z_{s_k,j}(k, \pi_m^k) = 1, \forall k \in \mathcal{K}, m = 0 \quad (4)$$

and

$$\sum_j z_{j,d_k}(k, \pi_m^k) = 1, \forall k \in \mathcal{K}, m = L_k \quad (5)$$

respectively, where π_0^k is a dummy VNF which is indeed s_k. For other nodes, the flow conversation constraints are related to the node mapping variables $x_{i,k}(\pi_m^k)$. In particular, if the virtual node π_m^k is mapped onto node i, there must exist a link incoming to node i to which the virtual link (k, π_{m-1}^k) is mapped and a link outgoing from node i to which the virtual link (k, π_m^k) is mapped, otherwise not. Therefore, we have the following constraints for all $(i, k, \pi_m^k) \in \mathcal{V} \times \mathcal{K} \times (\mathcal{F}_k \cup \{\pi_0^k\})$:

$$x_{i,k}(\pi_m^k) - x_{i,k}(\pi_{m+1}^k) = \sum_j z_{ij}(k, \pi_m^k) - \sum_j z_{ji}(k, \pi_m^k). \quad (6)$$

Capacity constraints: It is commonly assumed that the computational resource consumption of VNF is proportional to the flow rate [13], [34]. Therefore, we assume one unit data flow consumes $\alpha(\pi)$ units of computational resources for VNF type π. Specifically, the forwarding function can be seen as a special network function and its unit computational resource consumption is $\alpha(\pi_f)$. Similarly, we assume the unit bandwidth consumption is β. Thus, the amount of computational resources consumed on node i is

$$\begin{aligned} R_i^c(t) = &\sum_k \sum_{\pi_m^k} x_{i,k}(\pi_m^k) \mu_k(t) \alpha(\pi_m^k) \\ &+ \sum_k \sum_j \sum_{\pi_m^k} z_{ij}(k, \pi_m^k) \mu_k(t) \alpha(\pi_f). \end{aligned} \quad (7)$$

The bandwidth consumption on link (i, j) can be computed

as

$$R^b_{ij}(t) = \sum_k \sum_{\pi^k_m} z_{ij}(k, \pi^k_m)\mu_k(t)\beta. \quad (8)$$

The computational capacity of node i and the bandwidth capacity of link (i, j) is assumed to be C_i and B_{ij}, respectively. Thus, we have the following two capacity constraints for node i and link (i, j)

$$R^c_i(t) \leq C_i, \forall i \in \mathcal{V} \quad (9)$$

and

$$R^b_{ij}(t) \leq B_{ij}, \forall (i, j) \in \mathcal{E}, \quad (10)$$

respectively.

IV. Problem formulation and discussion

To accommodate the traffic dynamics of flows in a network slice, we need to adaptively reconfigure the network slice with aim of minimizing the long-term resource consumption. The resource consumption consists of two parts: the resource for embedding the network slice and that for reconfiguring the network slice. To formulate the NSRP, we define the *embedding cost* and the *reconfiguration cost* for resource provisioning of embedding the network slice and reconfiguring the network slice, respectively.

The resource for embedding the network slice is the sum of resources consumed by all flows in the network slice. To define the embedding cost, we assume the pricing functions for computational and bandwidth resources are $\varphi_c(\cdot)$ and $\varphi_b(\cdot)$, respectively [14]. Accordingly, the cost of embedding the network slice, i.e., the *embedding cost* is defined as:

$$C_{res}(t) = \varphi_c(\sum_i R^c_i(t)) + \varphi_b(\sum_{ij} R^b_{ij}(t)). \quad (11)$$

Slice reconfiguration itself incurs certain resource consumption because it may cause various overheads such as signaling and retransmission overhead. Thus, the reconfiguration resource is quantified as a function of the state difference of a slice before and after reconfiguration [14]. The state of a network slice is reflected by the node mapping variable $\boldsymbol{x}$ and link mapping variable $\boldsymbol{z}$. To reflect the resource consumption for reconfiguring the network slice, we define the *reconfiguration cost* as

$$C_{conf}(t) = \boldsymbol{\delta}^T_x \cdot \boldsymbol{I}(\boldsymbol{x}(t) - \boldsymbol{x}(t-1)) + \boldsymbol{\delta}^T_z \cdot \boldsymbol{I}(\boldsymbol{z}(t) - \boldsymbol{z}(t-1)), \quad (12)$$

where $\boldsymbol{x}(t-1), \boldsymbol{z}(t-1)$ are the decision variables at the previous time, $\boldsymbol{\delta}^T_x$ and $\boldsymbol{\delta}^T_z$ respectively are cost coefficients of the reconfiguration on nodes and links, and $I(\cdot)$ is an indicator function, i.e., if $x \neq 0$, $I(x) = 1$; otherwise $I(x) = 0$. Therefore, the total cost for an operation of reconfiguring the network slice at time t is:

$$C(t) = C_{res}(t) + C_{conf}(t). \quad (13)$$

Next, we formulate the NSRP as:

$$\min_{\boldsymbol{x}, \boldsymbol{z}} \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} C(t) \quad (14)$$

$$s.t.\ (1) - (6), (9) - (10) \quad (14.1)$$

$$x_{i,k}(\pi^k_m) \in \{0, 1\}, \forall i \in \mathcal{V}, k \in \mathcal{K}, \pi^k_m \in \mathcal{F}_k. \quad (14.2)$$

$$z_{ij}(k, \pi^k_m) \in \{0, 1\}, \forall (i, j) \in \mathcal{E}, k \in \mathcal{K}, \pi^k_m \in \mathcal{F}_k. \quad (14.3)$$

Since NSRP is indeed a sequential decision problem, we use the long-term average cost as the optimization objective in (14). Constraint (1) to (2) ensure that one VNF of a flow is served by at most one physical node. Constraints (4) to (6) are flow conservation constraints. Constraints (9) and (10) are capacity constraints of node and link respectively. Constraints (14.2) to (14.3) are the binary constraints on $x_{i,k}(\pi^k_m)$ and $z_{ij}(k, \pi^k_m)$, respectively.

Importantly, the constraints of NSRP can guarantee that each flow is processed by the VNFs consistent with the order in its SFC. More formally:

Theorem 1. *If there is a non-empty set S_f of feasible solutions that meet the constraints of NSRP, then for $\forall(\boldsymbol{x}, \boldsymbol{z}) \in S_f$ and $\forall k \in \mathcal{K}$, flow k is processed exactly in the order of the functions in $\mathcal{F}_k$ by the physical path p_k, where p_k is defined by the variables $\boldsymbol{x} = \{x_{i,k}(\pi^k_m)\}$ and $\boldsymbol{z} = \{z_{ij}(k, \pi^k_m)\}$.*

Due to space limit, we give the proof of Theorem 1 in Appendix A.

NSRP is an ILP which turns out to be NP-hard. The proof is based on a polynomial time reduction from the Generalized Assignment Problem (GAP) to NSRP.

Theorem 2. *Checking the feasibility of NSRP is NP-hard, and thus solving NSRP is NP-hard, too.*

Proof: We first construct an instance of NSRP as follows:

- We set the reconfiguration cost coefficients $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_z$ both to $\mathbf{0}$.
- The traffic rates of each flow remain the same as their nominal rates all the time.
- The substrate network is composed of J independent simple paths $\{p_i\}^J_{i=1}$ (i.e., $G = \bigcup^J_{i=1}$, and $\bigcap^J_{i=1} = \varnothing$), and all paths can provide the SFC of the slice.

The above instance of NSRP will reduce to a static integer linear programming with aim of minimizing the $C_{res}(t)$ under constraints (14.1) to (14.3). We refer this problem to P2 and will show its NP-hardness by a polynomial reduction from the GAP to P2.

GAP instance: In P2, the items and bins in GAP correspond to flows and candidate paths respectively. The size of bin p_i, denoted by ζ_i, is the maximal flow rate it can provide. The weight of item k is the rate of flow k, i.e., $\mu_k(t)$. Placing item k at bin p_i yields a cost c_{ik}, which is the embedding cost of placing flow k onto path p_i; also, each item shall be assigned to exactly one bin. The decision variables are binary flags ξ_{ik} indicating whether item k shall be assigned to bin i. The objective is to minimize the total cost.

Reduction: If there is a solution $\{\xi_{ik}\}$ of GAP that minimizes the total cost, we can construct a solution of P2 from $\{\xi_{ik}\}$ directly. On the other hand, if there is a solution of P2 defined by $\{x_{i,k}(\pi^k_m)\}$ and $\{z_{ij}(k, \pi^k_m)\}$, from Theorem 1, we can find a physical path p_k to which flow k is mapped. Since the substrate network consists of independent paths, p_k

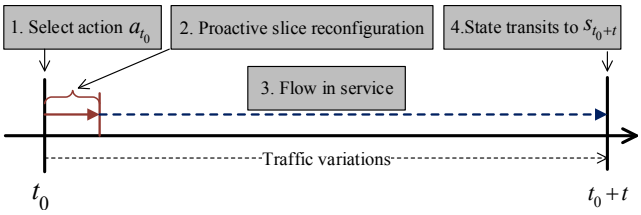


Fig. 3. The procedure of proactive slice reconfiguration

is therefore a solution of GAP. Thus, GAP has a solution if and only if P2 has a solution. In other words, GAP is reducible to P2. From the above analysis, the time complexity of the above reduction is polynomial. Since GAP is NP-hard, P2 and NSRP are NP-hard too. ■

It is interesting to notice that, in the proof of Theorem 2, we obtain a *simplified* description on the process of network slicing. Instead of mapping the virtual nodes and the virtual links of the flow separately, we simplified this process by directly mapping the entire flow onto the *candidate path*, which is a simple physical path onto which a flow can be mapped. Moreover, considering its long-term objective and to further take advantage of prediction information, NSRP is well suited to the RL framework. In the next section, we will reformulate it to an MDP and solve it by using DRL.

V. INSRA POLICY

In this section, we use MDP to model the long-term decision-making problem NSRP, and then solve it by using DRL.

A. Markov Decision Process

MDP is a framework that can be used to learn an optimal policy by interacting with the environment [35]. An MDP is defined as $\langle S, A, T, R, \tau \rangle$, where S, A, T, R, τ denote the state space, action space, state transition, reward function and discount factor, respectively. The agent interacts with the environment at each of discrete time steps. At each time step t, the agent receives the environment's state $s(t)$, and selects an action $a(t)$ based on that state. As a consequence of this action, the agent receives a numerical reward $r(t)$ and transits to a new state $s(t+1)$. Through these interactions, the agent can lean to achieve a goal by maximizing its long-term reward. In the following, we will present the definitions of each component of our MDP model.

B. Markov Decision Process Modeling for NSRP

In this paper, the agent corresponds to the MANO layer entity of the network slicing architecture [5]. The environment is comprised of the network flows and the substrate network. The rest components of our MDP model for NSRP are defined as follows.

Decision Epoch: Under our MDP model, the agent needs to proactively reconfigure the network slice upon traffic demand variations of the flows. To this end, the agent reconfigures the network slice for the future based on the historical traffic of the flows. The procedure of the network slice reconfiguration is shown as in Fig. 3.

The *decision epoch* (or *time step* [35]) is defined on a event-driven basis rather than fixed intervals of real-world time. At time t_0, a reconfiguration is made before the subsequent traffic variations. Since then, as the traffic variations accumulated up to a certain level, the previous reconfiguration may cause degradation on resource utilization or QoS in the forthcoming period. At this moment, a new reconfiguration is required. The next reconfiguration time, i.e., the decision epoch is define as:

$$t = \inf_{\tau} \{\tau | \int_{t_0}^{\tau} [\sum_{k \in \mathcal{K}} |\mu_k(\psi) - \mu_k(t_0)|] d\psi \leq (\tau - t_0)\vartheta_{th}\} \tag{15}$$

where, $\vartheta_{th} = \vartheta \cdot \sum_k \mu_k(t_0)$. The intuition behind this definition is as follows. Since the fluctuations of traffic reflects the change in resource demand, thus the accumulated traffic variations indicate the degree of resource over-provisioning or under-provisioning. When the accumulated traffic variations exceed the traffic *fluctuation threshold* ϑ_{th}, it indicates that there is a trend of imbalance between resource supply and demand. Therefore, a new reconfiguration is needed to avoid the forthcoming imbalance. In addition, the traffic threshold ϑ_{th} is the sum of the nominal flow rates multiplied by a coefficient ϑ.

State Space: We represent the state of NSRP by $M+N+1$ features, i.e.,

$$s(t) = \{c_1, \cdots c_M; \mu_1, \cdots, \mu_N; C_{conf}(t-1)\}. \tag{16}$$

The state contains three types of information. The first M elements $\{c_1, \cdots c_M\}$ represent the capacities of the candidate paths, and the subsequent N elements $\{\mu_1, \cdots, \mu_N\}$ represent the traffic demands of the N flows in the network slice. The last element, i.e., $C_{conf}(t-1)$ is the reconfiguration cost at $t-1$. Our motivations for constructing such a state for NSRP are as follows.

First, the state space must contain the information about the substrate network. However, the number of features used to represent the substrate network is extremely large and the time complexity of constructing a single feature of the substrate network is as high as $O(n^3)$ [9]. As a result, representing the substrate network directly will lead to slow convergence of the feature construction process, not to mention solving NSRP by RL. Therefore, we propose a novel approach to simplify the complex representation of the substrate network by exploiting the properties of NSRP. Our approach is based on the following observations:

- The number of candidate paths that can serve the network slice is not too large, especially for the network slice which is based on coarse-grained NFV implementation [2].
- The essence of network slice reconfiguration is to find an optimal mapping from the candidate paths for all the flows.

We can find out all the candidate paths by DFS in polynomial time. Suppose that all the M candidate paths have been found by DFS. Then the substrate network can be represented by the capacity of these paths, i.e., $\{c_1, \cdots c_M\}$. In this way, NSRP is

equivalent to *finding the optimal mapping from the candidate paths for all the flows with the aim to minimize the long-term cost*.

Second, the information about the network slice is of vital importance to make reconfiguration decisions. Because the flow variation does not change the source and the destination of the flow, we only need to use the flow rate μ_k to represent a flow. Therefore, the features that represent the network slice are set as $\{\mu_1, \cdots, \mu_N\}$.

Third, the historical reconfiguration cost $C_{conf}(t–1)$ reflects the number of traffic variations between two successive time steps. This information is important for predicting future slice changes and can help to reduce the future reconfiguration cost. For this reason, it is part of the state as well.

Action Space: Note that the agent aims to select the optimal mapping from the candidate paths for the N flows carried by the network slice. Therefore, the action is an N-dimensional vector, i.e.,

$$\boldsymbol{a}(t) = (p_1, p_2, \cdots, p_N), \tag{17}$$

where p_i is the path onto which flow i is mapped. Since there are N flows in the underlying model, thus the action space is an N-dimensional discrete space.

State Transitions: The state transition is considered to be stochastic because the next state depends on not only the selected action but also the external factors which are not controlled by the agent, such as stochastic traffic demand variations.

Reward Function: The reward is defined as:

$$r(t) = \begin{cases} -C(t), & mapping\ succeeds \\ -\sigma, & otherwise \end{cases} \tag{18}$$

We state that a mapping is successful if it does not violate constraints (14.1) to (14.3). Recall that in NSRP, our objective is to *minimize* the total resource consumption. However, in the general case, the objective of the agent is to *maximize* long-term reward. For this reason, we define the reward as the negative of the long-term total cost. In addition, to avoid constraint violation, we set the reward to $-\sigma$, where σ is a large penalty for the violation of the constraints.

According to the above analysis, the state space of this MDP is a continuous space with $M + N + 1$ dimensions. Moreover, its action space is an N-dimensional discrete space, and the sub-action of each dimension takes values in $\{1, \cdots, M\}$. Consequently, the discrete-action RL algorithm, the Dueling DDQN, is very suitable to address this problem. However, Dueling DDQN becomes ineffective in solving NSRP as the number of flows increases. This is because the size of the action space is exponentially related to the number of the flows, i.e., $|\mathcal{A}| = M^N$. Accordingly, the time complexity of the neural network increases exponentially with N, making it difficult for the neural network to converge. Thus, we propose to incorporate the action branching architecture into Dueling DDQN to compress the multi-dimensional discrete action space of the MDP and propose our intelligent network slice reconfiguration algorithm.

C. Handling the Large State Space with Dueling DDQN

We use Dueling DDQN as our learning algorithm because it can address MDP with large state space and naturally discrete action space. In addition, Dueling DDQN has a high sample efficiency and can acquire a good policy because it jointly utilizes the dueling architecture, Double DQN and prioritized experience replay to improve its performance on the basis of DQN.

General framework of DRL: DRL is a nonlinear value function based RL algorithm. It employs *deep Q-network* (DQN) as its value function approximator [36]. In particular, DQN exploits deep neural network to approximate the parameterized value function $Q(s, a; \boldsymbol{\theta})$. It uses the environment state s as its inputs and outputs the state-action values of each action a under current state s.

Like other value function approximation based RL algorithms, the training of DQN is essentially a process of supervised learning. DQN uses transition $\langle s, a, r, s' \rangle$, which is called *experience*, as its training sample. Experiences are obtained by iteratively interacting with the environment. The agent selects actions based on an ϵ-greedy policy, i.e., selecting a random action with probability ϵ and with probability $(1-\epsilon)$ to select

$$a = \arg\max_{a'} Q(s, a'; \boldsymbol{\theta}). \tag{19}$$

Then action a is executed and the agent gets reward r and the state transits to s'. Through this interaction, an experience $\langle s, a, r, s' \rangle$ is produced and is stored in the replay memory $\mathcal{Z}$. DQN uses experience replay, which randomly selects a batch of experiences to train the DQN, to stabilize the training process.

The objective of the training is to minimize the gap between the output of the DQN, i.e., the estimated Q-value $Q(s, a; \boldsymbol{\theta})$, and the target value which represents the real value of selecting action a under state s. To avoid the correlation between the estimated value and the target value, DQN uses two networks, i.e., Q-network with weights $\boldsymbol{\theta}$ and $\hat{Q}$-network with weights $\boldsymbol{\theta}^-$. The Temporal Difference (TD)-target, which can be seen as the real value that DQN aims to approximate, is computed as:

$$y^{DQN} = r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{\theta}^-). \tag{20}$$

The loss function is defined as the mean squared error between y^{DQN} and the estimated Q-value, i.e.,

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{Z}}[y^{DQN} - Q(s, a; \boldsymbol{\theta})]^2. \tag{21}$$

In general, DQN utilizes gradient descent to minimize the loss function in (21). Formally, the update rule of $\boldsymbol{\theta}$ is:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - [y^{DQN} - Q(s, a; \boldsymbol{\theta})] \nabla_{\boldsymbol{\theta}} Q(s, a; \boldsymbol{\theta}). \tag{22}$$

On the basis of DQN, Dueling DDQN leverages the following three techniques to improve its sample efficiency and the quality of the learned policy.

Double DQN: In DQN, the TD-target in (20) is the sum of the immediate reward and a discounted evaluation of the Q-value. The latter simply takes the maximum over the Q-values for all possible actions. However, such a TD-target may lead to overestimation of the actual Q-value [37]. Thus,

DDQN is applied to eliminate this overestimation. Similar with DQN, DDQN also uses two value functions with weights θ and θ^- respectively. But DDQN exploits a different method to evaluate the TD-target. In DDQN, the value function with weight θ is used to determine the greedy policy and the other is used to determine the Q-value. Formally, the TD-target of DDQN is computed as

$$y^{DDQN} = r + \gamma \hat{Q}(s', \arg\max_{a'} Q(s', a'; \boldsymbol{\theta}); \boldsymbol{\theta}^-). \tag{23}$$

Prioritized Experience Replay: Because of the large state space and action space of NSRP, numerous experiences should be stored in the replay memory to train the DQN. However, DQN tends to have a low sample efficiency since it samples the replay memory uniformly without differentiating the importance of individual experiences. Experiences with high TD-errors are indeed more important because they can speed up the learning progress and thus can make experience replay more efficient. Dueling DDQN employs prioritized experience replay (PER) [38] because it increases the replay probability of the experiences that have a high value of TD-error and thus leads to a high sample efficiency as well as a better policy.

In PER, the importance of the ith experience is measured by the absolute TD-error, which is given by:

$$\delta_i(s, a, r, s') = |y^{DDQN} - Q(s, a; \boldsymbol{\theta})|. \tag{24}$$

The larger this value is, the greater the probability that the experience will be replayed. Specifically, the probability of replaying experience i is defined as [38]:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{25}$$

where α determines how much prioritization is performed. We use *proportional prioritization* where $p_i = |\delta_i| + \zeta$ and ζ is a small positive value that prevents the edge-case of transitions not being replayed once δ_i is zero.

Since prioritized experience replay introduces a bias in estimating the Q-function [38], we compensate this bias by using weighted importance sampling (IS) weights:

$$\omega_i = (N \cdot P(i))^{-\beta}. \tag{26}$$

In practice, we do not use a fixed value of the exponent β but rather exploit a linear scheduler that anneals the exponent β from β_0 to 1.

Dueling Network Architecture: In problems with large action space such as NSRP, it is unnecessary to estimate the value of every action for certain states. The dueling network [39] separates the original DQN into value branch and advantage branch to avoid unnecessary estimation of the redundant and low-valued actions. These two branches are trained simultaneously by experience replay to get estimations of the state value $V(s; \boldsymbol{\theta})$ and the advantage function $A(s, a; \boldsymbol{\theta})$. At the output layer, the state-action value function $Q(s, a)$ is produced by an aggregation layer which combines the outputs of the value branch $V(s; \boldsymbol{\theta})$ and the advantage branch $A(s, a; \boldsymbol{\theta})$ as

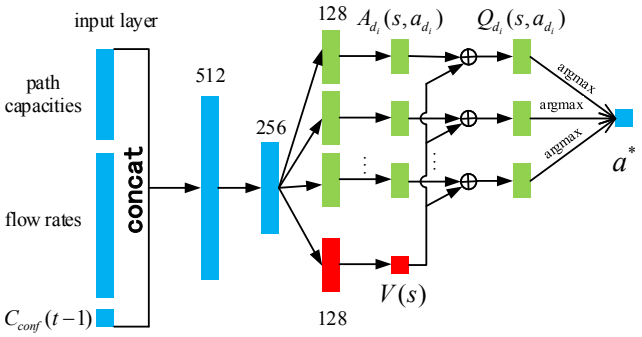


Fig. 4. The BDQ network architecture used in solving NSRP

$$Q(s, a; \boldsymbol{\theta}) = V(s; \boldsymbol{\theta}) + A(s, a; \boldsymbol{\theta}) - \frac{1}{|\mathcal{A}(s)|} \sum_{a'} A(s, a'; \boldsymbol{\theta}), \tag{27}$$

where $\mathcal{A}(s)$ is the available action set under state s. This architecture can discern valuable states without having to learn the value of each action under each state. Accordingly, it is employed to achieve a high-quality policy in Dueling-DDQN. As discussed earlier, although Dueling DDQN can tackle MDP with a large state space, it becomes ineffective to converge in the face of large discrete action space. Therefore, we incorporate action branching architecture into Dueling DDQN to address this issue in the following subsection.

D. Compression of Action Space with BDQ

The action branching architecture proposed in [30] provides an effective framework to solve MDPs with multi-dimensional discrete action space. The core notion of the architecture is to give a certain freedom of individual action dimension while sharing a common state-value estimator between these dimensions. Based on Dueling DDQN, the authors of [30] proposed a novel agent, called BDQ as an implementation of the action branching architecture. They verified the effectiveness of BDQ in problems with action spaces that contain as many as 6.5×10^{25} actions. However, the large discrete action spaces of these problems are discretized from the original continuous action spaces. The efficacy of the action branching architecture is not verified in problems with multi-dimensional action spaces which are inherently discrete. In this section, we incorporate the action branch architecture to Dueling DDQN to derive a discrete BDQ which can be used to compress the naturally multi-dimensional action space of NSRP.

The architecture of BDQ is illustrated in Fig. 4. Based on Dueling DDQN, BDQ further splits the advantage branch into N advantage branches while keeping a shared representation of the input state. In this way, the BDQ gives a certain degree of autonomy to each sub-action. Specifically, the action $\boldsymbol{a}$ of dimension N is split into N sub-actions and treated separately. The advantage of each sub-action, i.e., $A_d(s, a_d)$, is trained with the common state value $V(s)$ by experience replay. Similar with that in Dueling DDQN, the Q-value of each sub-action, $Q_d(s, a_d)$, is derived by aggregating the value branch and the corresponding advantage branch.

Formally, the action $\boldsymbol{a}(t) = (a_1, a_2, \cdots, a_N)$ is split into N sub-actions, and each sub-action has $|\mathcal{A}_d| = n$ discrete choices. According to [30], the value of the dth sub-action $a_d \in \mathcal{A}_d$ at state s is expressed in terms of the common state value $V(s)$ and the corresponding sub-action advantage $A_d(s, a_d)$ as:

$$Q_d(s, a_d) = V(s) + (A_d(s, a_d)) - \frac{1}{n} \sum_{a'_d \in \mathcal{A}_d} A_d(s, a'_d)). \quad (28)$$

Alternatively, the ϵ-greedy policy of BDQ is to select a random action with probability ϵ and with probability $(1-\epsilon)$ to select:

$$a = (\arg\max_{a'_{d1}} Q_{d1}(s, a'_{d1}; \boldsymbol{\theta}), \cdots, \arg\max_{a'_{dN}} Q_{dN}(s, a'_{dN}; \boldsymbol{\theta})). \quad (29)$$

BDQ exploits a TD-target similar as that in Dueling DDQN to avoid maximization bias, except that it is averaged across all the dimensions of the action:

$$y = r + \gamma \frac{1}{N} \sum_d \hat{Q}_d(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d(s', a'_d)). \quad (30)$$

The loss is the expected value of the mean squared error across the branches, i.e.,

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}}[\frac{1}{N} \sum_d [y_d - Q_d(s, a_d; \boldsymbol{\theta})]^2]. \quad (31)$$

To incorporate prioritized experience replay, the prioritization error is set to the sum across a transition's absolute, distributed TD-errors [30]:

$$\delta_i(s, a, r, s') = \sum_d |y_d - Q_d(s, a_d)|, \quad (32)$$

where $\delta_i(s, a, r, s')$ denotes the TD-error used to prioritize replay for experience (s, a, r, s').

BDQ achieves a linear increase of the number of estimated actions with the number of dimensions of the action space. In NSRP, the number of the actions that need to be evaluated is reduced from M^N to $N \cdot M$ by BDQ. As a result, the time complexity of the training increases linearly with N, making it effective in solving NSRP. By using BDQ, NSRP can be addressed in a large-scale network slice.

From the above analysis, we incorporate action branching architecture into Dueling DDQN to derive the BDQ network to solve NSRP and give our INSRA in Algorithm 1.

Remark 1: The equivalence between the MDP and NSRP should be clarified. Indeed,

- Each feasible solution of NSRP corresponds to an action of our MDP;
- Each action chosen by the agent corresponds to a feasible solution of the NSRP after INSRA converges.

Please refer to Appendix B for proof.

Remark 2: Some implementation issues should be clarified. First, the agent in INSRA is the MANO layer which has the information about the substrate network and network slice. Second, the actions performed by the agent are accomplished before the traffic variations of the flows take place. Because the mappings of the nodes and links are accomplished by SDN and NFV techniques, there is little signaling overhead [40]. Third, as shown in the next section, the number of reconfigurations and thus the signaling overhead can be significantly decreased by using INSRA compared with benchmark algorithms. Note that by exploiting a diminishing rule of ϵ in INSRA, an action chosen by the DRL agent corresponds to a feasible solution of the ILP.

Algorithm 1 Intelligent network slice reconfiguration algorithm (INSRA)

Input: N, ϵ, γ, α, β_0
Output: Desirable $\boldsymbol{a}(t)$

Establish two BDQ networks: trained network and target network with weights $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$, respectively. Initialize $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$ randomly and enable $\boldsymbol{\theta}^- = \boldsymbol{\theta}$, $C_{conf}(0) = 0$
for $t = 1, 2, \cdots$ **do**
  Construct $s(t)$: Run DFS to get $(c_1, \cdots, c_M)$. Construct $s(t)$ as in (16)
  if $t <= |\mathcal{Z}|$ **then**
    Randomly select an action $\boldsymbol{a}(t)$ to execute.
  else
    Choose an action $\boldsymbol{a}(t)$ with ϵ-greedy policy as in (29). The agent gets its reward $r(t)$ and the state of the environment transit to a new state $s(t+1)$. The agent stores the corresponding experience in the memory $\mathcal{Z}$.
    Perform prioritized experience replay according to (25) and (26). The prioritization error is computed as in (32).
    Compute the TD-target as in (30). Perform a gradient descent step on (31) w.r.t. $\boldsymbol{\theta}$.
    Every Y steps set $\boldsymbol{\theta}^- = \boldsymbol{\theta}$.
  end if
end for

VI. NUMERICAL RESULTS

In this section, we conduct simulation experiments based on TensorFlow to evaluate the performance of our proposed INSRA. We first examine the convergence performance of INSRA. Then we demonstrate the effectiveness of INSRA in minimizing long-term resource consumption by comparing its long-term performance with the other three benchmark algorithms. Finally, we show the performance of INSRA under different network slice implementations as well as different scales of network slice.

A. Simulation Settings

For simulation experiments, we construct a substrate network as illustrated in Fig. 5 which is widely used for performance evaluation of network slicing, such as that in [14], [41]. The parameters of the substrate network are listed in Table II. Seven kinds of VNFs are provided by the substrate network, i.e., $\mathcal{F} = \{$Proxy, IPS, Optimizer, Firewall, Billing, NAT, Transcoder$\}$. And 7 among 15 nodes are VNF-capable. Each VNF-capable node can randomly provide 3 VNFs in $\mathcal{F}$. The computing capacity of each node is randomly set

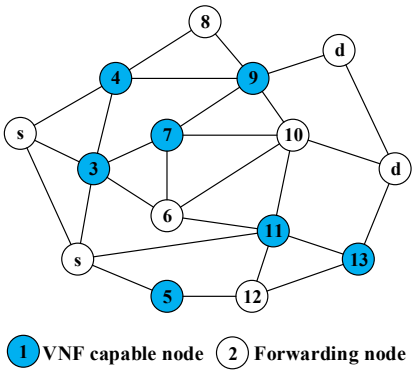


Fig. 5. Substrate Network

in $[10, 20]$ units and the bandwidth capacity of each link is uniformly distributed in $[5, 50]$ units.

In the simulations, we consider the scenarios with 5 slices over the substrate network. We run the slice reconfiguration algorithms for individual slices. The SFCs of these slices are defined in Table III [14], [18]. Each network slice carries 20 flows with varying traffic rates. The nominal traffic rate (i.e., $\bar{\mu}_{k,t}$) of the flows is uniformly generated in $[0.5, 3]$ units. We use the widely used Gaussian distributed variable $\eta \sim N(0, 1)$ to represent the perturbation of the traffic [16], [42], [43]. Since η may approaches to ∞, we use a truncated version of η to simulate the traffic variation by confining its value to $[0, 5]$. In other words, the rate of the kth flow is set as $\mu_{k,t} \cdot \widetilde{\eta}$, where $\widetilde{\eta}$ is a truncated standard Gaussian random variable. We set the penalty parameter as $\sigma = 50$ when the constraints are violated. In addition, we set $\alpha(\pi_m^k) = 0.5$, $\alpha(\pi_f) = 0.1$, and $\beta = 1$. The pricing functions for computational and bandwidth resources are both linear functions. We set the cost coefficient parameter both to 2 and the traffic fluctuation threshold to 0.5.

TABLE II
NETWORK PARAMETERS

parameter	Value
number of nodes	15
number of links	27
number of VNFs	7
$\alpha(\pi_m^k), \alpha(\pi_f)$	0.5, 0.1
β	1
nominal traffic rate	$U[0.5, 3]$
flow rate variations	truncated standard Gaussian distribution
node capacity	$U[10, 20]$
link capacity	$U[5, 50]$
cost coefficient	$\boldsymbol{\delta_x} = 2$, $\boldsymbol{\delta_z} = 2$
fluctuation threshold	$\vartheta = 0.5$

We use TensorFlow to build a BDQ. The architecture of the BDQ is illustrated in Fig. 4. We summarize the main parameters of the BDQ in Table IV. The front end of the network has two fully connected layers, each with 512 and 256 neurons respectively. The value branch consists of a fully connected layer with 128 neurons and outputs the state value.

TABLE III
SFC OF THE SLICES

Service	**SFC**
VoD	(UE)–Proxy–IPS–Optimizer–(Server)
Gaming	(UE)–Firewall–Billing–(Server)
Content Cache	(Cache)–Firewall–NAT–(Content)
VPN Access	(UE)–Firewall–IPS–NAT–(Server)
Video Chat	(UE)–Firewall–IPS–Transcoder–(UE)

Each advantage branch has one fully connected layer with 128 neurons. Note that all the neurons use Rectified Linear Unit (ReLU) as their activation functions. To balance exploration and exploitation, we apply an adaptive ϵ-greedy policy. The ratio of exploration, i.e. ϵ, is initialized as 0.5 and decreases as $\epsilon(t+1) = \max\{0.05, (1-0.0001)\epsilon(t)\}$. We use the Adam optimizer with a learning rate $\alpha = 10^{-4}$ and $\beta_1 = 0.9$, $\beta_2 = 0.999$ to update $\boldsymbol{\theta}$. To avoid the correlation between the action-values and target values, we copy the weights of the evaluation network $\boldsymbol{\theta}$ to the weights of the target network $\boldsymbol{\theta}^-$ every 500 training steps. In addition, we set the memory size as $|\mathcal{Z}| = 10^4$ and the batch size of gradient descent to 64. Furthermore, we use prioritized replay with $\alpha = 0.6$ and β annealed from $\beta_0 = 0.4$ to 1 in 10^5 time steps.

TABLE IV
HYPER PARAMETERS OF BDQ

parameter	Value
decision time parameter	$W = 5$, $\psi = 0.1$
penalty parameter σ	50
initial exploration ratio $\epsilon(t)$	10^{-2}
input layer	$M + N + 1$
minibatch size	64
discount factor γ	0.99
replay memory size $\lvert\mathcal{Z}\rvert$	1000
prioritized replay parameter	$\alpha = 0.6$, $\beta_0 = 0.4$
optimizer	Adam with learning rate 10^{-4}, $\beta_1 = 0.9, \beta_2 = 0.999$
target network update period Y	500

Because the models of existing research differ from NSRP in many aspects such as optimization objectives and constraints, it is unfair to compare their solutions with INSRA. Instead, we design the following three benchmark algorithms as comparison references for comprehensive performance evaluations:

1) Multi-Agent Reinforcement Learning based reconfiguration algorithm (MARL): In this algorithm, the learning model is a multi-agent system, where each flow acts as an agent to make decisions independently, and both the reward and state space remain the same as those in our proposed learning model of INSRA. By exploiting the idea of distributed reinforcement learning [44], these agents learn cooperatively to maximize the global reward.

2) Instantaneous Optimal Slice Reconfiguration (IOSR): In this algorithm, we exploit the Gurobi MIP solver to periodically solve NSRP instances to meet the instantaneous traffic demands. Due to the NP-hardness of NSRP, it is rather

time-consuming to obtain the optimal solution. Thus, we set the optimality gap to 0.05% to make a tradeoff between the optimality of the solution and the computation time.

3) Cheapest path first (CPF): In this algorithm, the flows are successively mapped to the least expensive candidate path to meet their instantaneous traffic demands without constraint violations.

After the slice is initialized, the traffic demands of the flows will be monitored and the reconfiguration algorithms will be executed to map the flows onto the candidate paths.

B. Numerical Results

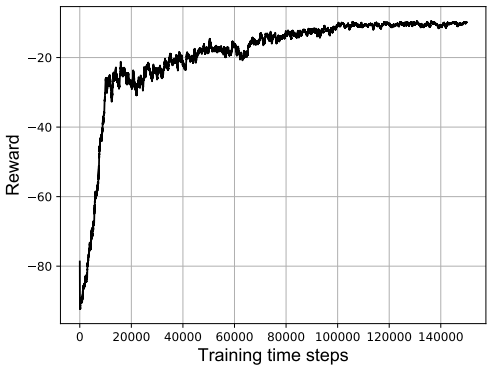


Fig. 6. Convergence of INSRA

1) Experiment 1 - Convergence performance: We verify the convergence properties of our proposed INSRA by depicting its *learning curve* (the curve of the reward *vs*. the learning time steps). As shown in Fig. 6, INSRA converges to the optimal policy within 10^5 scheduling time steps. Since intra-slice reconfiguration is performed at small time-scale [14], thus INSRA can be effectively realized as an online network slice reconfiguration algorithm.

2) Experiment 2 - Long-term Performance of INSRA: First, we compare the long-term total cost of INSRA, MARL, IOSR and CPF in Fig. 7. We can observe that our proposed INSRA can maintain a fairly low total cost compared with the benchmark algorithms. This indicates that INSRA can minimize long-term resource consumption effectively. Furthermore, we can also observe that the curve of INSRA is relatively stable compared with that of the other algorithms. This result demonstrates that INSRA can effectively predict the future traffic demands of the flows in the network slices, thereby avoiding frequent reconfigurations.

Next, we compare the reconfiguration cost as well as embedding cost of INSRA, MARL, IOSR and CPF in Fig. 8 and Fig. 9 respectively. We can observe that the reconfiguration cost of INSRA is the lowest. On the other hand, we can see the embedding cost of INSRA is slightly higher than that of CPF and IOSR. This is because the main objective of IOSR is to minimize the instantaneous embedding cost without considering the reconfiguration cost.

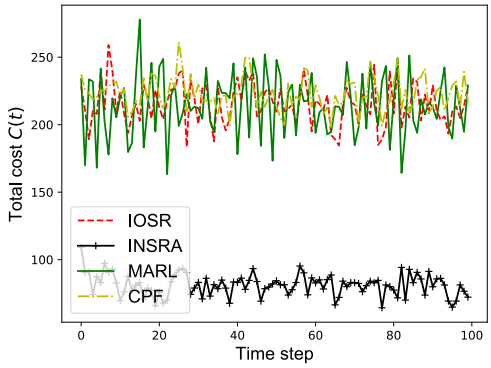


Fig. 7. Comparison of the total cost for 5 network slices in 100 time steps.

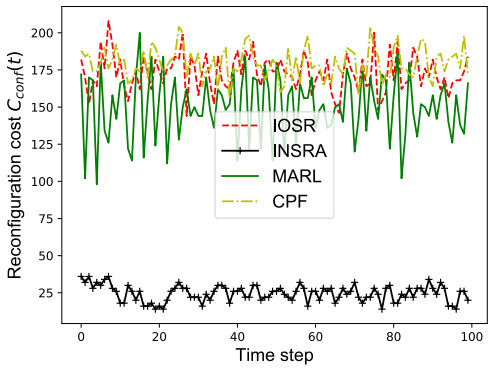


Fig. 8. Comparison of the reconfiguration cost for 5 network slices in 100 time steps.

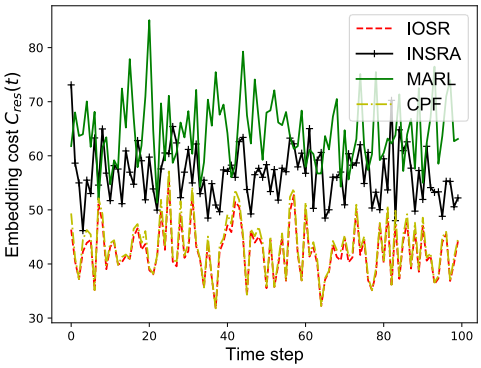


Fig. 9. Comparison of the embedding cost for 5 network slices in 100 time steps.

Finally, we compare the *resource efficiency* of these algorithms, which is defined as the ratio of accumulated embedding cost to the long-term total cost:

$$r_{eff}(t) = \frac{\sum_{\tau=0}^{t} C_{res}(\tau)}{\sum_{\tau=0}^{t} C(\tau)}. \tag{33}$$

As illustrated in Fig. 10, the resource efficiency of INSRA can be up to 80%, which is much higher than that of the benchmark algorithms. Moreover, we find that although the embedding cost of IOSR is the lowest, its resource efficiency is also low. Consequently, static optimization based slice reconfiguration

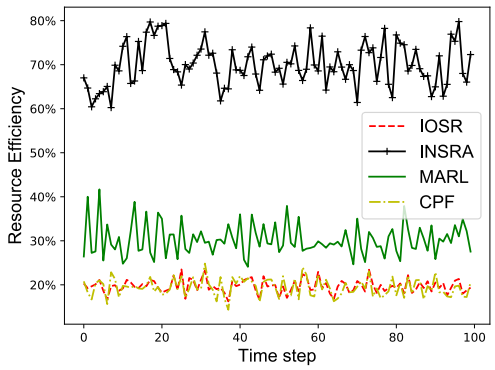


Fig. 10. Comparison of the resource efficiency of the system in 100 time steps.

algorithms without considering future traffic demands perform poorly. In contrast, the resource efficiency of INSRA is much better as it can predict the traffic variations by learning and can intelligently reconfigure the slices.

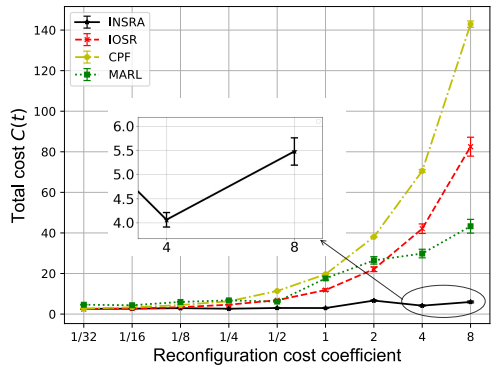


Fig. 11. Total cost *vs.* the reconfiguration cost coefficient with 95% confidence interval.

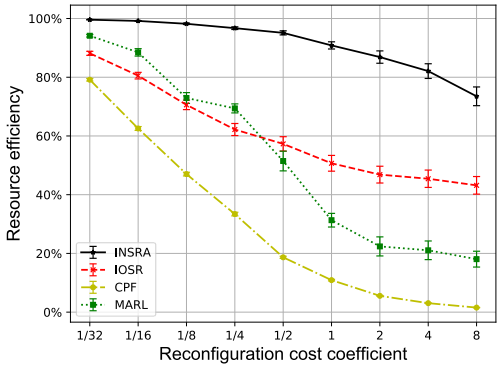


Fig. 12. Resource efficiency *vs.* the reconfiguration cost coefficient with 95% confidence interval.

3) Experiment 3 - Performance with different reconfiguration cost coefficient: This experiment primarily aims to demonstrate INSRA's performance in terms of reconfiguration cost coefficient. Similar with that in [14], we assume there are 20 flows in each network slice and we set the reconfiguration cost coefficient as one of the values in $\{\frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \cdots, 2, 4, 8\}$. The results are exhibited in Fig. 11 and Fig. 12. Each figure compares the performance of INSRA with three benchmark algorithms. We use the trained model to reconfigure the network slices in 2000 time steps. Both the mean and the 95% confidence interval of the results are plotted. Please note that in Fig. 11, the confidence interval of INSRA is almost invisible since the total cost of INSRA is too small. Thus, we magnify the curve of INSRA and attached it to the side of the main figure.

From Fig. 11, we can observe that the total cost of the three benchmark algorithms increases dramatically with the reconfiguration cost coefficient. In contrast, the reconfiguration cost coefficient has little impact on the total cost of INSRA. Therefore, this result suggests that INSRA can effectively avoid excessive reconfigurations in the long run. From Fig. 12, we can conclude that as the reconfiguration cost coefficient increases, the proportion of the resource that is used to embed the flows decrease moderately compared with the benchmark algorithms. Since different cost coefficients reflect different implementations of the network slice [14], these results indicate that our proposed INSRA can provide competitive performance under different network slice implementations.

4) Experiment 4 - Performance with different number of flows: In this experiment, we compare the performance of INSRA with the benchmark algorithms for different number of flows. The reconfiguration cost coefficient is set to 2 and the number of the flows in each network slice takes values in $\{10, 20, \cdots, 60\}$. Similar with that of the previous experiment, we simulate 2000 time steps to plot the mean and the 95% confidence interval of the results, which are shown in Fig. 13 and Fig. 14. Note that the curve of INSRA is magnified and attached to the side of the main figure since its value is too small.

In Fig. 13, we can see the total cost increases with the number of flows. Notably, the slope of the curve of INSRA is the lowest compared with the benchmark algorithms. This result suggests INSRA can effectively minimize the total resource consumption even in a slice with a large number of flows. In Fig. 14, it can be observed that INSRA can keep high resource efficiency as the number of flows increases. Moreover, these results indicate that our proposed INSRA can effectively tackle the system with more than 10 candidate paths (found by DFS) and 60 network flows, leading to an action space as large as $M^N = 10^{60}$. In short, this experiment demonstrates the effectiveness of INSRA in large-scale network slice.

VII. Conclusion

Network slicing is one of the most promising architectural technologies to meet the diversified requirements in future wireless networks. Intelligently reconfiguring the network slice is one of the most urgent problems in network slicing. In this paper, we model the intra-slice reconfiguration problem as an MDP and solve it through DRL. To address the large state space and the multi-dimensional discrete action space, we incorporate BDQ network to decrease the number of actions need to be evaluated in our proposed INSRA. Numerical results show that INSRA can minimize the long-term resource

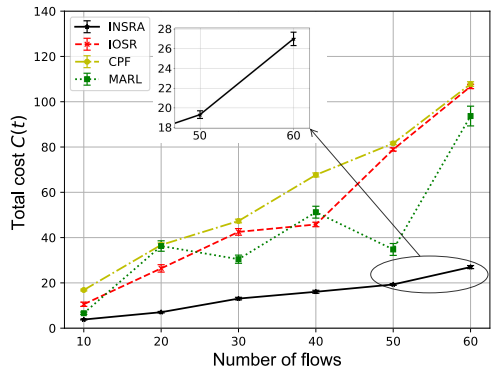


Fig. 13. Total cost vs. the number of flows with 95% confidence interval.

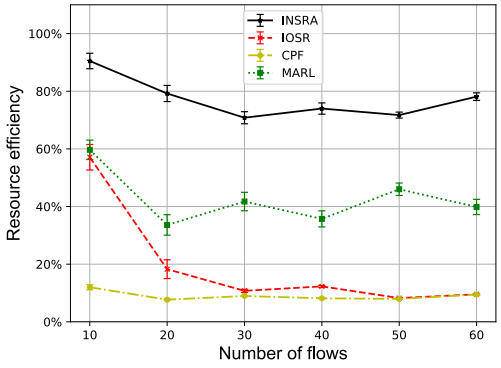


Fig. 14. Resource efficiency vs. the number of flows with 95% confidence interval.

consumption as well as predicting the future requirement of the slice thus to avoid unnecessary reconfigurations.

In this paper, we have investigated the resource reconfiguration under flow fluctuation within a network slice. In the future, it is interesting to investigate the automation of inter-slice reconfiguration. In addition, the hybrid slice reconfiguration which combines inter-slice and intra-slice reconfiguration could be considered. For example, we can first predict the slice traffic through deep learning (such as RNN), and then exploit robust optimization or stochastic programming to optimally perform the inter-slice reconfiguration as well as to compensate the prediction error.

Appendix A
Proof of Theorem 1

Proof: Equivalently, we should prove the following two statements:

- For flow k, we have $\sum_{\pi_m^k} x_{i,k}(\pi_m^k) = L_k$, and
- If flow k is processed by VNF π_m^k on node i_s (i.e., if $x_{i_s,k}(\pi_m^k) = 1$), it will be processed by VNF π_{m+1}^k on node i_t (i.e., to prove $x_{i_t,k}(\pi_{m+1}^k) = 1$), where $i_s \neq i_t$.

a) The first statement is a direct result of Constraint (2).

b) To prove the second statement, we need to prove that there exists a path $p = (i_s(v_0), v_1, \cdots, v_d, i_t(v_{d+1}))$ such that

$$z_{v_j,v_{j+1}}(k, \pi_m^k) = 1, \forall j \in \{0, \cdots, d\}, \tag{34}$$

and $x_{i_t,k}(\pi_{m+1}^k) = 1$.

First, there must exist at least one physical path from i_s to i_t. Otherwise, there will be no feasible solution since no solution can meet the flow conservation law on node i_s and node i_t since $i_s \neq i_t$. Second, since $x_{i_s,k}(\pi_m^k) = 1$, according to (2), there must be $x_{i_s,k}(\pi_m^k) = 0$. Therefore, we can get $\sum_j z_{i_s,j}(k, \pi_m^k) - \sum_j z_{j,i_s}(k, \pi_m^k) = 1$ according to (6). As a result, we can deduce that there must exist a link outgoing node i_s, say (i_s, v_1), such that $z_{i_s,v_1}(k, \pi_m^k) = 1$. Then if $x_{v_1,k}(\pi_{m+1}^k) = 1$, the path is found. Otherwise, by applying Constraint (6) on node v_1 (set $i = v_1$ in (6)), we reach a new node, say v_2, such that $z_{v_1,v_2}(k, \pi_m^k) = 1$. If $x_{v_2,k}(\pi_{m+1}^k) = 1$, then we have found the path. Otherwise, we repeat this process until we reach node v_t such that $x_{v_t,k}(\pi_{m+1}^k) = 1$. This process will terminate in finite steps since the objective function (14) can avoid cycles in the mapped paths. Refer to Fig. 2 for an illustration. In this way, we have found a path that satisfies (34). ■

Appendix B
Proof of Remark 1

Proof: To prove the first statement, we assume that $(\boldsymbol{x}, \boldsymbol{z})$ is a feasible solution of NSRP. From the definition of candidate path and Theorem 1, we know $(\boldsymbol{x}, \boldsymbol{z})$ defines candidate path p_k for all $k \in \mathcal{K}$, which constitute an action of the MDP according to (17).

To prove the second statement, we assume that $\boldsymbol{a}(t)$ is an action chosen by the agent after INSRA converges. From the definition of $\boldsymbol{a}(t)$ in (17), each path p_k in action $\boldsymbol{a}(t)$ is a candidate path onto which the flows can be mapped. Equivalently, p_k is a simple path that can process flow k exactly in the order of the functions in $\mathcal{F}_k$. Let $p_k = (v_0^k, \cdots, v_r^k)$. Without loss of generality, we assume that VNF π_m^k of flow k is instantiated on node v_m^k of p_k. For all $\pi_m^k \in \mathcal{F}_k$, by setting $x_{v_m^k,k}(\pi_m^k) = 1$ and $z_{i,j}(k, \pi_m^k) = 1, \forall (i,j) \in p_k$, we get the variables $\boldsymbol{x}$ and $\boldsymbol{z}$. From the above process, we see that $\boldsymbol{x}$ together with $\boldsymbol{z}$ satisfies constraint (1) - (6) and (14.2) - (14.3). For constraint (9) - (10), we see that after INSRA converges, the agent is impossible to chose an action that leads to unsuccessful mapping to get an $-\sigma$ reward. This is because that the agent chooses actions according to ϵ-greedy mechanism. By exploiting a diminishing ϵ implementation as in INSRA (i.e., $\epsilon \to 0$), the agent will choose an action without violating the resource capacity constraint (9) - (10) since such actions have larger Q-values. Therefore, we conclude that after INSRA converges, each action chosen by the agent corresponds to a feasible solution of the NSRP. ■

References

[1] B. Cao, Y. Li, L. Zhang, L. Zhang, S. Mumtaz, Z. Zhou, and M. Peng, "When Internet of Things Meets Blockchain: Challenges in Distributed Consensus," *IEEE Network*, 2019.

[2] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, May. 2017.

[3] NGMN Alliance, "Description of Network Slicing Concept," *NGMN 5G P*, vol. 1, 2016.

[4] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[5] V. Spyridon, G. Lazaros, L. Nikolaos, N. S. Ioannis, Q. Meiyu, S. Lei, L. Liu, D. Merouane, and S. P. Georgios, "The Algorithmic Aspects of Network Slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112–119, Aug. 2017.

[6] B. Cao, S. Xia, J. Han, and Y. Li, "A distributed game methodology for crowdsensing in uncertain wireless scenario," *IEEE Transactions on Mobile Computing*, 2020.

[7] I. Jang, D. Suh, S. Pack, and G. Dán, "Joint optimization of service function placement and flow distribution for service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2532–2541, Nov. 2017.

[8] M. H. Fan, Jinliang and Ammar, "Dynamic topology configuration in service overlay networks: A study of reconfiguration policies," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, Apr. 2006, pp. 1–12.

[9] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in *2016 12th International Conference on Network and Service Management (CNSM)*, Oct. 2016, pp. 10–18.

[10] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, 2016.

[11] M. R. Raza, M. Fiorani, A. Rostami, P. Öhlen, L. Wosinska, and P. Monti, "Dynamic Slicing Approach for Multi-Tenant 5G Transport Networks [Invited]," *Journal of Optical Communications and Networking*, vol. 10, no. 1, pp. A77–A90, Jan. 2018.

[12] J.-J. Chen, M.-H. Tsai, L. Zhao, W.-C. Chang, Y.-H. Lin, Q. Zhou, Y.-Z. Lu, J.-L. Tsai, and Y.-Z. Cai, "Realizing dynamic network slice resource management based on sdn networks," in *2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA)*. IEEE, 2019, pp. 120–125.

[13] N. Zhang, Y. F. Liu, H. Farmanbar, T. H. Chang, M. Hong, and Z. Q. Luo, "Network slicing for service-oriented networks under resource constraints," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2512–2521, Nov. 2017.

[14] G. Wang, G. Feng, T. Q. S. Quek, S. Qin, R. Wen, and W. Tan, "Reconfiguration in Network Slicing - Optimizing the Profit and Performance," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 591–605, 2019.

[15] R. Wen, G. Feng, J. Tang, T. Q. S. Quek, G. Wang, W. Tan, and S. Qin, "On Robustness of Network Slicing for Next-Generation Mobile Networks," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 430–444, Jan. 2019.

[16] A. Baumgartner, T. Bauschert, A. M. C. A. Koster, and V. S. Reddy, "Optimisation Models for Robust and Survivable Network Slice Design: A Comparative Analysis," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec. 2017, pp. 1–7.

[17] A. Baumgartner, T. Bauschert, A. A. Blzarour, and V. S. Reddy, "Network slice embedding under traffic uncertainties — A light robust approach," in *2017 13th International Conference on Network and Service Management (CNSM)*, Nov. 2017, pp. 1–5.

[18] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 433–446, Feb. 2019.

[19] Q. Zhang, F. Liu, and C. Zeng, "Adaptive Interference-Aware VNF Placement for Service-Customized 5G Network Slices," in *Proceedings - IEEE INFOCOM*, vol. 2019-April. Institute of Electrical and Electronics Engineers Inc., Apr. 2019, pp. 2449–2457.

[20] J. Cao, Y. Zhang, W. An, X. Chen, J. Sun, and Y. Han, "VNF-FG design and VNF placement for 5G mobile networks," *Science China Information Sciences*, vol. 60, no. 4, Apr. 2017.

[21] R. Solozabal, B. Blanco, J. O. Fajardo, I. Taboada, F. Liberal, E. Jimeno, and J. G. Lloreda, "Design of virtual infrastructure manager with novel VNF placement features for edge clouds in 5G," in *Communications in Computer and Information Science*, 2017.

[22] T. Subramanya and R. Riggio, "Machine Learning-Driven Scaling and Placement of Virtual Network Functions at the Network Edges," in *Proceedings of the 2019 IEEE Conference on Network Softwarization: Unleashing the Power of Network Softwarization, NetSoft 2019*. Institute of Electrical and Electronics Engineers (IEEE), Aug. 2019, pp. 414–422.

[23] O. A. Wahab, N. Kara, C. Edstrom, and Y. Lemieux, "MAPLE: A Machine Learning Approach for Efficient Placement and Adjustment of Virtual Network Functions," *Journal of Network and Computer Applications*, vol. 142, pp. 37–50, Sep. 2019.

[24] M. Yan, G. Feng, J. Zhou, Y. Sun, and Y.-C. Liang, "Intelligent Resource Scheduling for 5G Radio Access Network Slicing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7691–7703, Jun. 2019.

[25] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive Online Service Function Chain Deployment with Deep Reinforcement Learning," in *IWQoS '19: Proceedings of the International Symposium on Quality of Service*. Association for Computing Machinery (ACM), 2019, pp. 1–10.

[26] M. Nakanoya, Y. Sato, and H. Shimonishi, "Environment-adaptive sizing and placement of NFV service chains with accelerated reinforcement learning," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, 2019, pp. 36–44. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8717919

[27] R. Li, Z. Zhao, Q. Sun, C.-L. I, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep Reinforcement Learning for Resource Management in Network Slicing," *IEEE Access*, vol. 6, pp. 74 429–74 441, 2018.

[28] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for VNF Forwarding Graph Embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, Oct. 2019.

[29] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, "Intelligent Offloading in Multi-Access Edge Computing: A State-of-the-Art Review and Framework," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 56–62, 2019.

[30] P. Tavakoli, Arash and Pardo, Fabio and Kormushev, "Action Branching Architectures for Deep Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2018, pp. 4131–4138.

[31] X. Yang, Y. Liu, I. C. Wong, Y. Wang, and L. Cuthbert, "Effective isolation in dynamic network slicing," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–6.

[32] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, Aug. 2017.

[33] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "StEERING: A software-defined networking for inline service chaining," in *Proceedings - International Conference on Network Protocols, ICNP*. IEEE Computer Society, 2016.

[34] J. Gil Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[35] R. S. B. SUTTON, *REINFORCEMENT LEARNING : an introduction.* MIT press, 2018.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning." *Nature*, vol. 518, no. 7540, pp. 529–33, 2015.

[37] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[38] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," pp. 1–21, Nov. 2015. [Online]. Available: http://arxiv.org/abs/1511.05952

[39] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," no. 9, Nov. 2015. [Online]. Available: http://arxiv.org/abs/1511.06581

[40] R. Gu and J. Zhang, "GANSlicing: A GAN-Based Software Defined Mobile Network Slicing Scheme for IoT Applications," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May. 2019, pp. 1–7.

[41] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proceedings - IEEE INFOCOM*, 2013.

[42] R. Li, Z. Zhao, J. Zheng, C. Mei, Y. Cai, and H. Zhang, "The Learning and Prediction of Application-Level Traffic Data in Cellular Networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3899–3912, Jun. 2017.

[43] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1543–1557, Aug. 2019.

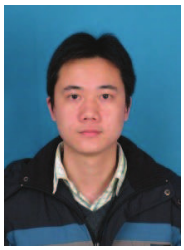
[44] J. R. Kok and N. Vlassis, "Collaborative multiagent reinforcement learning by payoff propagation," *Journal of Machine Learning Research*, 2006.

Fengsheng Wei received the B.E. and M.E. degree in School of Communication and Information Engineering at University of Electronic Science and Technology of China (UESTC) in 2012 and 2016, respectively. He is now pursuing his Ph.D. degree in School of Communication and Information Engineering at University of Electronic Science and Technology of China (UESTC). His current research interests include next generation mobile communication systems, machine learning and network slicing in mobile networks.

Gang Feng received his BEng and MEng degrees in Electronic Engineering from the University of Electronic Science and Technology of China (UESTC), in 1986 and 1989, respectively, and the Ph.D. degrees in Information Engineering from The Chinese University of Hong Kong in 1998. He joined the School of Electric and Electronic Engineering, Nanyang Technological University in December 2000 as an assistant professor and was promoted as an associate professor in October 2005. At present he is a professor with the National Laboratory of Communications, University of Electronic Science and Technology of China. Dr. Feng has extensive research experience and has published widely in computer networking and wireless networking research. His research interests include resource management in wireless networks, next generation cellular networks, etc. Dr. Feng is a senior member of IEEE.

Yao Sun received the B.S. degree in Mathematical Science, and the Ph.D. degree (Honors) in Communication and Information System from University of Electronic Science and Technology of China (UESTC), in 2014 and 2019, respectively. From Nov. 2017 to Nov. 2018, he was an international research visitor at School of Engineering, University of Glasgow. Dr. Sun is currently a research fellow at National Key Laboratory of Science and Technology on Communications, UESTC. Dr. Sun has extensive research experience and has published widely in wireless networking research area. He has won the IEEE Communication Society of TAOS Best Paper Award in 2019 ICC. His research interests include intelligent wireless networking, network slicing, blockchain system, internet of things and resource management in mobile networks.

Yatong Wang received the B.E. degree in School of Communication and Information Engineering at University of Electronic Science and Technology of China (UESTC) in 2018. She is now pursuing her Ph.D. degree in School of Communication and Information Engineering at University of Electronic Science and Technology of China (UESTC). Her current research interests include next generation mobile communication systems, machine learning and network slicing in mobile networks.



Shuang Qin received the B.E. degree in Electronic Information Science and Technology, and the Ph.D degree in Communication and Information System from University of Electronic Science and Technology of China (UESTC), in 2006 and 2012, respectively. He is currently an associate professor with National Key Laboratory of Science and Technology on Communications in UESTC. His research interests include cooperative communication in wireless networks, data transmission in opportunistic networks and green communication in heterogeneous networks.

Ying-Chang Liang (F'11) is currently a Professor with the University of Electronic Science and Technology of China, China, where he leads the Center for Intelligent Networking and Communications and serves as the Deputy Director of the Artificial Intelligence Research Institute. He was a Professor with The University of Sydney, Australia, a Principal Scientist and Technical Advisor with the Institute for Infocomm Research, Singapore, and a Visiting Scholar with Stanford University, USA. His research interests include wireless networking and communications, cognitive radio, symbiotic radio, dynamic spectrum access, the Internet-of-Things, artificial intelligence, and machine learning techniques. Dr. Liang has been recognized by Thomson Reuters (now Clarivate Analytics) as a Highly Cited Researcher since 2014. He received the Prestigious Engineering Achievement Award from The Institution of Engineers, Singapore, in 2007, the Outstanding Contribution Appreciation Award from the IEEE Standards Association, in 2011, and the Recognition Award from the IEEE Communications Society Technical Committee on Cognitive Networks, in 2018. He is the recipient of numerous paper awards, including the IEEE Jack Neubauer Memorial Award, in 2014, and the IEEE Communications Society APB Outstanding Paper Award, in 2012. He was elected a Fellow of the IEEE for contributions to cognitive radio communications. He is the Founding Editor-in-Chief of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS: COGNITIVE RADIO SERIES, and the Key Founder and now the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING. He is also serving as an Associate Editor-in-Chief for China Communications. He served as a Guest/Associate Editor of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE JOURNAL OF SELECTED AREAS IN COMMUNICATIONS, the IEEE Signal Processing Magazine, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORK. He was also an Associate Editor-in-Chief of the World Scientific Journal on Random Matrices: Theory and Applications. He was a Distinguished Lecturer of the IEEE Communications Society and the IEEE Vehicular Technology Society. He was the Chair of the IEEE Communications Society Technical Committee on Cognitive Networks, and served as the TPC Chair and Executive Co-Chair of the IEEE Globecom'17.