Dynamic Network Slice Reconfiguration by Exploiting Deep Reinforcement Learning

Fengsheng Wei, Gang Feng, *Senior Member, IEEE*, Yao Sun, Yatong Wang, Ying-Chang Liang, *Fellow, IEEE*
University of Electronic Science and Technology of China
E-mail:fenggang@uestc.edu.cn

***Abstract*—It is widely acknowledged that network slicing can tackle the diverse usage scenarios and connectivity services that the 5G-and-beyond systems need to support. To guarantee performance isolation while maximizing network resource utilization under traffic uncertainty, 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 the numerous variables. In this paper, we investigate network slice reconfiguration with aim of minimizing long-term resource consumption by exploiting Deep Reinforcement Learning (DRL). To address the curse of dimensionality of the problem, we propose to incorporate the Branching Dueling Q-network (BDQ) into DRL, to avoid some unnecessary calculations of Q-value by separating the Q-network into a shared value branch and a number of distributed advantage branches. Furthermore, the value branch and the advantage branch of each dimension are aggregated to derive the corresponding dimension's sub-Q-value. Then the best reconfiguration action is composed of the sub-actions in individual dimensions which are selected by ϵ-greedy policy. Finally, we design an intelligent online network slice reconfiguration policy based on BDQ and extensive simulation experiments are conducted to validate the effectiveness of the proposed slice reconfiguration policy.**

I. Introduction

It is widely acknowledged that network slicing is a key enabler to address the diverse requirements that the 5G-and-beyond systems are envisioned to support. Unlike the traditional wireless network, network slices not only need customized capabilities that are crucial for corresponding services, but also need the capability to adapt to the changing traffic requirements [1]. Traffic load variations or traffic uncertainty in network slice may degrade the 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 reducing resource utilization. On the other hand, it can also cause Slice 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 while meeting slice service quality.

This work has been supported by the R&D Program in Key Areas of Guangdong Province (Grant No. 2018B010114001), the General Program of National Natural Science Foundation of China (Grant No. 61871099), and the Fundamental Research Funds for the Central Universities (Grant No. ZYGX2019J122).

Recently, a body of research work use a common paradigm which periodically optimize Network Slice Reconfiguration Problem (NSRP) instances to meet the instantaneous traffic demands of the network slice [2], [3]. However, this class of solutions may cause frequent reconfigurations to maintain the optimality of resource allocation, due to the lack of a prediction mechanism of future traffic requirements. In the long run, these solutions may cause substantial overhead 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 [4], etc.

NSRP with long-term optimization objective is essentially a sequential decision problem, which involves two interrelated aspects of network slice reconfiguration. On the one hand, the optimal network slice embedding should be addressed based on the instantaneous traffic demands of the flows in the network slice. On the other hand, future traffic demands should be also taken into account to reduce the number of unnecessary reconfigurations. Fortunately, recent emerging Reinforcement Learning (RL) provides an effective tool to solve such sequential decision problems under complex environments. However, for NSRP in a substrate network with non-trivial network slice, traditional table-based RL is infeasible due to its complicated state space and the multi-dimensional discrete action space.

In this paper, we resort to Deep Reinforcement Learning (DRL) to solve NSRP. Since NSRP involves two interrelated networks, i.e., the substrate network and network slice, it is difficult to extract an effective set of features to represent the environment [5]. To model NSRP as a Markov Decision Process (MDP), we simplify the representation of the environment based on the problem properties rather than directly representing the networks. We first utilize depth-first-search (DFS) to identify all the physical paths that can serve the Service Function Chain (SFC). Then we use their capacities together with virtual flow rates and history information of reconfiguration to represent the state of the environment. After a trial and error process of designing, we model the NSRP as an MDP.

Nevertheless, it is still very challenging to solve NSRP by employing the most commonly used DRL algorithms such as Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG). Although these algorithms can be used to solve MDPs with multi-dimensional state space, unfortunately they

fail in converging for the problems with multi-dimensional action spaces, let alone the action is also discrete [6]. To cope with this convergence issue, we exploit the Branching Dueling Q-network (BDQ) to evaluate the state-action values and propose an Intelligent Network Slicing Reconfiguration Algorithm (INSRA).

The remainder of the paper is organized as follows. Section II presents the system model and the formulation of NSRP. In Section III, we present the MDP modeling for the NSRP and elaborate our proposed INSRA. In Section IV, we present the numerical results and we conclude the paper in Section V.

II. System Model and Problem Formulations

We model the substrate network as a weighted directed graph $G^s = (N^s, L^s)$, where N^s and L^s denote the sets of nodes and links respectively. The physical nodes, denoted by $N^s = \{1, 2, \cdots, n\}$, can be classified into two types: VNF-capable nodes and common nodes. The former can provide Network Functions (NFs) while the latter are only used for packet forwarding. The VNF set is denoted by $\mathcal{F}$. In particular, some common nodes can serve as source nodes and/or destination nodes, which have no VNF functionality and are denoted by $\mathcal{S}$ and $\mathcal{D}$ respectively. The residual computational capacity of the node $i \in N^s$ is denoted by $R^s_{c,t}(i)$ and the residual bandwidth capacity of the link $(i,j) \in L^s$ is denoted by $R^s_{b,t}(i,j)$.

The service provided by the network slice is defined by its SFC. We assume the SFC of the network slice consists of h ordered VNFs denoted as $f_1 \rightarrow \cdots \rightarrow f_h$. Suppose there are N flows in the network slice. The kth flow is denoted by $(s_k, d_k, \mu_{k,t})$, which represent the source node, destination node, and the rate of the flow, respectively. Note that we use subscript t to denote the time throughout the paper.

To avoid coordination overhead caused by service splitting, we assume that a flow receives each service function by exactly one VNF-capable node as in [7]. Let the binary variable $x_{i,f}(k) \in \{0,1\}$ denote whether node i provides function f for flow k (i.e., $x_{i,f}(k) = 1$ if node i provides function f for flow k, otherwise $x_{i,f}(k) = 0$). To ensure that only one physical node serves VNF f for flow k, we have the following constraint:

$$\sum_{i \in N^s} x_{i,f}(k) = 1, \forall k, \forall f \in \mathcal{F}. \tag{1}$$

To embed the network slice, each flow needs three types of resources: the computational resources for VNF function, the computational resources for packet forwarding, and the bandwidth resources for packet transmission. Without loss of generality, similar to that in [7], we assume that one unit data flow consumes one unit computational resource, ω_b units bandwidth and ω_f units forwarding resource.

As that in [7], the amount of computational resources for the VNF required by flow k is:

$$R^c_k(t) = \sum_{i \in N^s, f \in \mathcal{F}} x_{i,f}(k)\mu_{k,t}. \tag{2}$$

The amount of computational resources consumption for VNF of node i is:

$$R_c(i) = \sum_{1 \le k \le N, f \in \mathcal{F}} x_{i,f}(k)\mu_{k,t}. \tag{3}$$

The required bandwidth and the forwarding resources are related to the length of the path onto which the flow is mapped [7]. We use $r_{ij}(k)$ to denote the data rate of flow k on link (i,j). Please note that the flows are not split in our model. Therefore, $r_{ij}(k)$ only takes two possible values for k. In particular, flow k is mapped to the path that contains link (i,j) if $r_{ij}(k) = \mu_{k,t}$, otherwise $r_{ij}(k) = 0$. The variable $r_{ij}(k)$ should satisfy the flow conservation law on all nodes. In particular, for the intermediate nodes, we have the following constraint:

$$\sum_{j \in N^s} r_{ji}(k) - \sum_{j \in N^s} r_{ij}(k) = 0, \forall i \in N^s, \forall k. \tag{4}$$

For source node s_k and destination node d_k of flow k, we have

$$\sum_{j \in N^s} r_{ij}(k) = \mu_{k,t}, \forall k, i = s_k \tag{5}$$

and

$$\sum_{i \in N^s} r_{ij}(k) = \mu_{k,t}, \forall k, j = d_k, \tag{6}$$

respectively.

The amount of computational resources used for forwarding flow k is:

$$R^f_k(t) = \sum_{(i,j) \in L^s} \omega_f r_{ij}(k). \tag{7}$$

Because the destination nodes do not forward flows, the amount of resources used for forwarding on node i is:

$$R_f(i) = \begin{cases} \sum\limits_{1 \le k \le N, (i,j) \in L^s} \omega_f r_{ij}(k), & \text{if i} \notin \mathcal{D} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

The total bandwidth consumed by flow k is:

$$R^b_k(t) = \sum_{(i,j) \in L^s} \omega_b r_{ij}(k). \tag{9}$$

Similarly, the total bandwidth consumption of link (i,j) can be computed as:

$$R_b(i,j) = \sum\nolimits_k \omega_b r_{ij}(k). \tag{10}$$

To accommodate the traffic dynamics of flows in a network slice, we need to adaptively reconfigure the network slice with the aim to minimize 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.

To define the embedding cost, we assume the pricing functions for computational and bandwidth resources are $\varphi_c(\cdot)$

and $\varphi_b(\cdot)$, respectively [3]. Accordingly, the cost of embedding the network slice, i.e., the *embedding cost* is defined as:

$$C_{res}(t)=\varphi_c\left[\sum_{k=1}^{N}\left(R_k^c(t)+R_k^f(t)\right)\right]+\varphi_b\left[\sum_{k=1}^{N}R_k^b(t)\right]. \quad (11)$$

The amount of reconfiguration resources is quantified as a function of the state difference of a slice before and after reconfiguration [3]. Similar as in [3], the state of a network slice is reflected by the node mapping variable $\boldsymbol{x}$ and link mapping variable $\boldsymbol{r}$. To reflect the resource consumption for reconfiguring the network slice, we define the *reconfiguration cost* as

$$C_{conf}(t)=\boldsymbol{\delta}_x^T\cdot\boldsymbol{I}(\boldsymbol{x}-\boldsymbol{x}')+\boldsymbol{\delta}_r^T\cdot\boldsymbol{I}(\boldsymbol{r}-\boldsymbol{r}'), \quad (12)$$

where $\boldsymbol{x}', \boldsymbol{r}'$ are the resource allocation variables at the previous time, $\boldsymbol{\delta}_x^T$ and $\boldsymbol{\delta}_r^T$ 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:

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

$$s.t.\ (1),(4)-(6) \quad (14.1)$$

$$R_f(i)+R_c(i)\leq R_{c,t}^s(i),\forall i\in N^s \quad (14.2)$$

$$R_b(i,j)\leq R_{b,t}^s(i,j),\forall(i,j)\in L^s \quad (14.3)$$

$$r_{ij}(k)\in\{0,\mu_{k,t}\},\forall(i,j)\in L^s,\forall k\in\{1,\cdots,N\}. \quad (14.4)$$

$$x_{i,f}(k)\in\{0,1\},\forall i\in N^s,\forall f\in\mathcal{F},\forall k. \quad (14.5)$$

NSRP is an MIP which turns out to be NP-hard. The proof is based on the polynomial time reduction from the multi-way separator problem [8].

III. 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 Modeling for NSRP

An MDP is defined by $\langle t_i, S, A, T, R\rangle$, where t_i, S, A, T, R denote the time step, state space, action space, state transition and reward function, respectively [9]. Our MDP formulation of NSRP is defined as follows. Under our model of NSRP, the agent needs to make a decision upon traffic demand variations of the flows in the network slice. To avoid frequent reconfigurations, we define the decision time as the time when there are W flows whose traffic variation exceeds $\zeta\cdot\bar{\mu}_k$, where ζ is a predefined factor and $\bar{\mu_k}$ is the nominal rate of flow k.

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

$$s(t)=\{c_1,\cdots c_M;\mu_1,\cdots,\mu_N;C_{conf}(t-1)\}. \quad (15)$$

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 of all, the information of the substrate network is necessary for decision making. However, the number of features used to represent the substrate network is extremely large and the time complexity of constructing a single feature is as high as $O(n^3)$ [5]. Consequently, it is infeasible to address NSRP by using DRL which represents the substrate network directly. Instead, we propose a novel approach to simplify the complex representation of the networks by exploiting the properties of NSRP. Our approach is based on the following observations:

- The number of possible physical paths that can serve the network slice, which we call the *candidate path*, is not too large, especially for the network slice which is based on coarse-grained NFV implementation [10].
- 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 also crucial to make 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 the 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 included in the state as well.

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

$$\boldsymbol{a}(t)=(p_1,p_2,\cdots,p_N), \quad (16)$$

where p_i is the path onto which flow i is mapped. Since our model contains N flows, the action space is therefore a 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 flow variations, random substrate node failure, etc.

Reward Function: The reward is defined as:

$$r(t) = \begin{cases} -C(t), & mapping\ succeeds \\ -\sigma, & otherwise \end{cases} \tag{17}$$

We state that a mapping is successful if it does not violate constraints (14.1) to (14.5). Recall that in NSRP, our objective is to *minimize* the total resource consumption. However, under the RL framework, 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 $-\sigma$.

According to the above analysis, we see that the state space of this MDP is a continuous space with $M+N+1$ dimensions and the action space is an N-dimensional discrete space with each dimension takes values in $\{1, \cdots, M\}$. Consequently, the discrete-action reinforcement learning algorithm, the Dueling Double Deep Q-Network (Dueling DDQN), is very suitable to address this problem. However, Dueling DDQN becomes ineffective in solving NSRP as the number of the flows increases, which leads to an exponentially increasing size of the action space, i.e., $|\mathcal{A}| = M^N$. 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 INSRA in the following subsection.

B. Compression of Action Space with BDQ

According to the above modeling, both the state space and the action space of the reinforcement learning model for solving NSRP are very large. Since Dueling-DDQN can automate the feature construction of the substrate network through the powerful representation of the multi-layer neural network, it can thus tackle MDPs with very large state space. However, Dueling DDQN cannot address multi-dimensional action spaces which are naturally discrete.

The action branching architecture proposed in [11] 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 [11] 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 environments 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 discretized-BDQ which can be used to compress the naturally multi-dimensional action space of NSRP. The architecture of BDQ is illustrated in Fig. 1. 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 provides a certain degree of autonomy to each sub-action. Specifically,

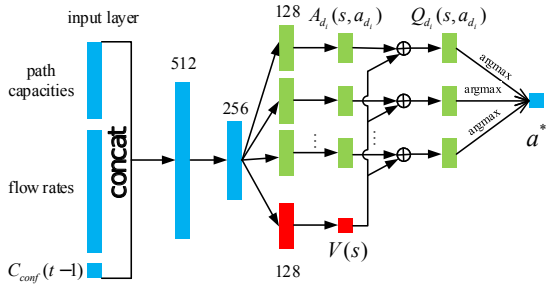


Fig. 1. The BDQ network architecture used in solving NSRP

the action 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 $a = (a_1, a_2, \cdots, a_N)$ is split into N sub-actions, and each sub-action has $|\mathcal{A}_d| = n$ discrete choices. According to [11], 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:

$$Q_d(s, a_d) = V(s) + (A_d(s, a_d)) - \frac{1}{n} \sum_{a'_d \in \mathcal{A}_d} A_d(s, a'_d)). \tag{18}$$

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

$$a = (\arg\max_{a'_{d1}} Q_{d1}(s, a'_{d1}; \boldsymbol{\theta}), \cdots, \arg\max_{a'_{dN}} Q_{dN}(s, a'_{dN}; \boldsymbol{\theta})). \tag{19}$$

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

$$y = r + \gamma \frac{1}{N} \sum_d \hat{Q}_d(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d(s', a'_d)). \tag{20}$$

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

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}}[\frac{1}{N} \sum_d [y_d - Q_d(s, a_d; \boldsymbol{\theta})]^2]. \tag{21}$$

To incorporate prioritized experience replay, the prioritization error is set to the sum across a transition's absolute, distributed TD-errors [11]:

$$\delta_i(s, a, r, s') = \sum_d |y_d - Q_d(s, a_d)|, \tag{22}$$

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 particular, the number of the actions that need to be evaluated can be reduced from M^N to $N \cdot M$ in NSRP. As a result,

Algorithm 1 Intelligent network slice reconfiguration algorithm (INSRA)

Input: M, 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 (15)
  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 (19). The agent gets 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. The prioritization error is computed as in (22).
    Compute the TD-target as in (20). Perform a gradient descent step on (21) w.r.t. $\boldsymbol{\theta}$.
    Every Y steps set $\boldsymbol{\theta}^- = \boldsymbol{\theta}$.
  end if
end for

the time complexity of the training increases linearly with N, making it effective in solving NSRP. Therefore, we incorporate action branching architecture into Dueling DDQN to derive the BDQ network to solve NSRP and give our INSRA in Algorithm 1.

IV. NUMERICAL RESULTS

In this section, we evaluate the performance of our proposed INSRA. We also compare INSRA with three other heuristic algorithms, namely cheapest path first (CPF), largest flow chooses the cheapest path first (LF-CPF) and random path mapping (RPM). Among all the paths that satisfy the constraints, CPF chooses the path with the largest capacity for each flow sequentially with the aim to avoid frequent reconfiguration while LF-CPF chooses the path with largest capacity for the largest flow. Additionally, RPM algorithm randomly selects a path for each flow without constraint violations.

A. Simulation Settings

For simulation experiments, we construct a substrate network with 13 nodes and 19 direct links. There are four types of VNFs in the system and 8 nodes are VNF-capable. Please note the BDQ agent does not need to be modified to fit different network scenarios[1]. The computing capacity of each node is randomly set in $[2, 10]$ units and the bandwidth capacity of each link is uniformly distributed in $[1, 8]$ units. Let the network slice contain 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 [12], [13] variable $\eta \sim N(0, 1)$ to represent the perturbation of the traffic. Since η may approaches to ∞, we use a truncated version of η to simulate the traffic variation by confining its value to $[0, 5]$. We set the penalty parameter as $\sigma = 50$ when the resource constraints are violated. In addition, we set $\omega_b = 0.5$ and $\omega_f = 0.1$. The pricing functions for computational and bandwidth resources are both linear functions. We set the cost coefficient parameter $\boldsymbol{\delta_x}$ and $\boldsymbol{\delta_r}$ both to 2.

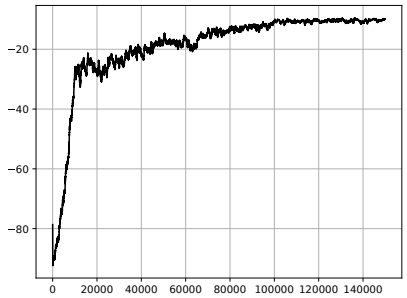


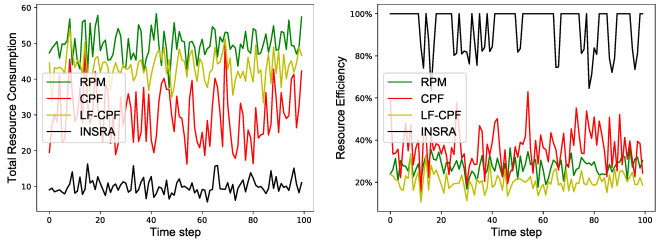
Fig. 2. Convergence of INSRA

We use TensorFlow to build a BDQ whose architecture is illustrated in Fig. 1. 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. 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 $\alpha = 10^{-4}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to update $\boldsymbol{\theta}$. To avoid correlation between the action-values and target values, we copy the weights of 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 as 64. Furthermore, we use prioritized replay with $\alpha = 0.6$ and β annealed from 0.4 to 1 in 10^5 time steps.

B. Numerical Results

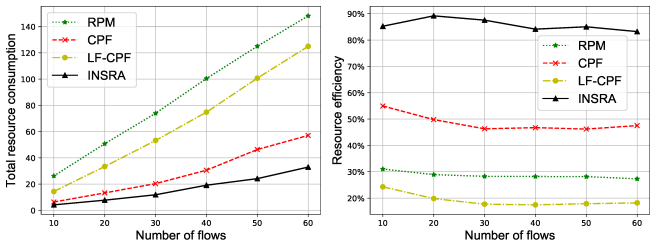
First, we verify the convergence properties of our proposed INSRA by depicting its *learning curve* (the curve of the reward *vs*. the learning time steps). As shown in Fig. 2, INSRA converges to the optimal policy within 10^5 scheduling time steps. Therefore, it can be effectively realized as an online network slice reconfiguration algorithm. Second, we compare the long-term total resource consumption for INSRA, LF-CPF, CPF and RPM in Fig. 3a. Please note that the results of INSRA are obtained after convergence. We can

[1]Since different network scenarios only lead to different number of candidate paths and/or different number of flows, thus we can preset the hyper-parameter M and N in BDQ to a large value, say M_{pre} and N_{pre}. Then if the number of candidate paths M_{act} and/or the number of flows N_{act} are less than the presets, we can set the rest $M_{pre} - M_{act}$ and/or $N_{pre} - N_{act}$ state components to zero.



(a) (b)

Fig. 3. Long-term resource consumption and resource efficiency



(a) (b)

Fig. 4. Long-term resource consumption and resource efficiency *vs.* the number of flows

observe that our proposed INSRA can maintain a fairly low resource consumption compared with the baseline algorithms. Therefore, it can minimize resource consumption effectively. Furthermore, we verify the performance of INSRA by comparing the resource efficiency of these algorithms. We define the *resource efficiency* as the ratio of embedding cost to the long-term total cost. Formally,

$$r_{eff}(t) = \frac{\sum_{\tau=0}^{t} C_{res}(\tau)}{\sum_{\tau=0}^{t} C(\tau)} \tag{23}$$

As illustrated in Fig. 3b, the resource efficiency of INSRA can be as high as 100%, which is much higher than that of the baseline algorithms. Moreover, we find that although the embedding cost of LF-CPF is the best, its resource efficiency is the worst. Consequently, statically reconfigure the network slice without considering the future information of the traffic demands is not optimal. In contrast, our proposed algorithm can not only predict the traffic variations implicitly, but also reconfigure the resource allocation ahead of time.

Finally, we examine the performance of the proposed INSRA with the baseline algorithms for different number of flows. The reconfiguration cost coefficient is set to 2 and the number of the flows in the network slice takes values in $\{10, 20, \cdots, 60\}$. The results are averaged in 1000 time steps to avoid randomness which are plotted in Fig. 4. In Fig. 4a, we can see the total resource consumption increases approximately linearly with the number of flows among all the algorithms. Notably, the slop of the curve of INSRA is the lowest compared with the baseline algorithms, which suggests INSRA can effectively minimize the total resource consumption even in the network slice with a large number of flows. In Fig. 4b, it can be observed that INSRA can keep high resource efficiency compared with baseline algorithms. Moreover, these results indicate that our proposed INSRA can effectively tackle the system with 10 candidate paths 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.

V. Conclusion

Network slicing is one of the most promising architectural technologies to meet the diversified requirement in future wireless networks. Intelligently automating the reconfiguration of the network slice is one of the most urgent problems in network slicing. In this paper, we have modeled the NSRP as an MDP and solved it by exploiting DRL. To address its multi-dimensional discrete action space, we utilize the BDQ network to compress its action space and then propose the INSRA algorithm. Numerical results reveal that the proposed algorithm can avoid unnecessary reconfigurations by implicitly predicting the future traffic demands of the flows in the network slice thus to minimize the long-term resource consumption.

References

[1] S. Zhang, "An Overview of Network Slicing for 5G," IEEE Wireless Communications, vol. 26, no. 3, pp. 111–117, Jun. 2019.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] C. Qi, Y. Hua, R. Li, Z. Zhao, and H. Zhang, "Deep Reinforce- ment Learning With Discrete Normalized Advantage Functions for Resource Management in Network Slicing," IEEE Communications Letters, vol. 23, no. 8, pp. 1337–1341, Aug. 2019.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.