Multi-Agent Reinforcement Learning with Pointer Networks for Network Slicing in Cellular Systems

Feng Wang, M. Cenk Gursoy, and Senem Velipasalar
Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, 13244
E-mail: fwang26@syr.edu, mcgursoy@syr.edu, svelipas@syr.edu

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9839183

***Abstract*—In this paper, we present a multi-agent deep reinforcement learning (deep RL) framework for network slicing in a dynamic environment with multiple base stations. We first introduce the wireless network virtualization (WNV) and the interference channel model. Then, we formulate the network slicing problem in the dynamic environment in which fading varies, users have mobility, and requests are randomly generated over time. Subsequently, we propose a deep RL framework with multiple actors and centralized critic (MACC) to maximize the reward over all base stations instead of pursuing local optimization. The actors are implemented as pointer networks to fit the varying dimension of input. Finally, we evaluate the performance of the proposed deep RL algorithm via simulations to demonstrate its effectiveness.**

***Index Terms*—Network slicing, dynamic channel access, deep reinforcement learning, multi-agent actor-critic.**

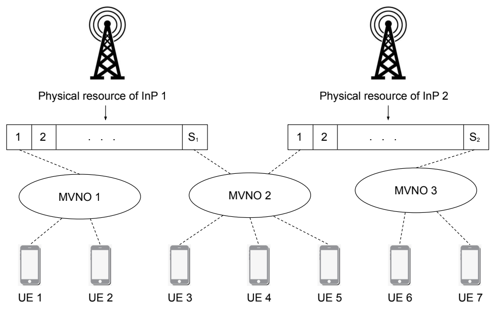

Fig. 1. WNV system model for allocating virtualized physical resources via MVNO.

I. Introduction

The emerging concept of network slicing enables further enhancements in 5G radio access network (RAN) service flexibility for novel applications with heterogeneous requirements [1]–[4]. In network slicing, the physical cellular network resources are divided into multiple virtual network slices to serve the end user, and thus network slicing is a vital technology to meet the strict requirements of each user by allocating the desired subset of network slices [5]. Instead of model-based allocation optimization that assumes the knowledge of traffic statistics [6]–[8], deep reinforcement learning (deep RL) [9]–[11] as a model-free decision making strategy can be deployed to optimize slice selection and better cope with the challenges such as dynamic environment, resource interplay, and user mobility [12]–[15]. Most existing studies in the literature assume the slice state to be identical for all different slices over time, and hence the selection problem reduces to assigning the number of slices to each request.

In this paper, we consider a more practical scenario of a cellular coverage area with multiple base stations and dynamic interference environment, analyze resource allocation in the presence of multiple users with random mobility patterns, and develop a multi-agent actor-critic deep RL agent to learn the slice conditions and achieve cooperation between base stations. We propose a learning and decision-making framework with multiple actors and a centralized critic (MACC) [16]–[18] that aims at maximizing the overall performance instead of local performance, to accelerate the training process. This multi-agent system requires the actors at each base station to communicate with a centralized critic at the data center/server to share the experience and update parameters in a centralized pattern during training process, and it subsequently switches to decentralized decision making following the training.

Typically, when 5G RAN achieves faster transmission rates with higher frequency bands, it also has smaller coverage, necessitating a relatively dense cellular network architecture. In such a setting, the dynamic environment with user mobility might have significant impact on the network slicing performance. Due to this, we introduce the pointer network [19] to implement the actor policy to handle the varying observations of the deep RL agents. We compare this proposed algorithm with different combinations of decentralized critic, feed-forward neural network (FNN) based actor, and other statistical algorithms to show the outstanding performance of the proposed novel approach.

II. System Model and Problem Formulation

In this section, we introduce the wireless network virtualization (WNV) and the interference channel model, and formulate the network slicing as an optimization problem.

A. Wireless Network Virtualization

WNV is well-known for enhancing the spectrum allocation efficiency. As shown in Fig. 1, infrastructure providers (InPs) own the physical infrastructure (e.g., base stations, resource blocks, back-haul, and core network), and operate the physical wireless network. WNV virtualizes the physical resources of InPs and separates them into slices to efficiently allocate these virtualized resources to the mobile virtual network operators (MVNOs). Therefore, MVNOs deliver differentiated services via slices to user equipments (UEs) with varying transmission requirements.

In this paper, we consider a service area that contains N_B base stations of InPs with inter-base-station interference, and N_M MVNOs that require services from the InPs. These MVNOs provide services to N_u UEs that are within the coverage area of at least one base station, and may move from the coverage of one base station to that of another. Users in the coverage area of multiple base stations that serve the MVNO can be assigned to either base station. Without loss of generality, we discuss the station-user pair for each MVNO in the remainder of this paper. Next, we introduce the dynamic channel environment.

B. Interference Channel Model

We consider a dynamic environment where N channels are available for each MVNO. The fading coefficient of the link between base station b and a UE u in a certain channel c at time t is denoted by $h_c^{b,u}(t) \sim \mathcal{CN}(0,1)$, an independent and identically distributed (i.i.d.) circularly symmetric complex Gaussian (CSCG) random variable with zero mean and unit variance. Each fading coefficient varies every T time slots as a first-order complex Gauss-Markov process, according to the Jakes fading model [20]. Therefore, at time $(n+1)T$, the fading coefficient can be expressed as

$$h_c^{b,u}((n+1)T) = \rho h_c^{b,u}(nT) + \sqrt{1-\rho^2} e_c((n+1)T) \quad (1)$$

where e_c denotes the channel innovation process, which is also an i.i.d. CSCG random variable. Furthermore, we have $\rho = J_0(2\pi f_d T)$, where J_0 is the zeroth order Bessel function of the first kind and f_d is the maximum Doppler frequency.

C. Network Slicing Problem Formulation

In the aforementioned environment, the resource allocation is performed in a first-come first-served fashion. Each base station may serve at most N_r requests from different users simultaneously, and the request queue can stack at most N_q requests. For each request, multiple channels can be allocated, and the transmission power is P_B in each channel. Different requests served by the same base station do not share the same channels, while requests to different base stations may share the same channel at the cost of inflicting interference. For all requests to each base station, transmission is allowed in no more than N_c channels, and consequently the total power is limited to $N_c P_B$. In this setting, if request k is allocated the subset C_k of channels at base station b, the sum rate r_k for this request is

$$\mathsf{r}_k = \sum_{c \in C_k} \mathsf{r}_c^{b,u}, \quad (2)$$

with

$$\mathsf{r}_c^{b,u} = \log_2\left(1 + \frac{P_B L^{b,u} |h_c^{b,u}|^2}{\sum_{b' \neq b} \mathbf{1}_c^{b,b'} P_B L^{b',u} |h_c^{b',u}|^2 + \sigma^2}\right), \quad (3)$$

where c denotes the index of a selected channel, $\mathsf{r}_c^{b,u}$ is the transmission rate from base station b to UE u in channel c, $\mathbf{1}_c^{b,b'}$ is the indicator function for transmission at base stations b and b' sharing channel c, σ^2 is the variance of the additive white Gaussian noise, and $L^{b,u}$ is the path loss:

$$L^{b,u} = \left(h_B^2 + (x_b - x_u)^2 + (y_b - y_u)^2\right)^{\alpha/2} \quad (4)$$

where h_B is the height of each base station, α is the path loss exponent, and $\{x_b, y_b\}$ and $\{x_u, y_u\}$ are the 2-D coordinates of base station b and user u, respectively.

Therefore, for each base station, the resource assignment including the selected subsets C_k, the number of selected channels n_c, the number of served requests n_r, and the number of requests in queue n_q must follow the following constraints:

$$C_{k_1} \cap C_{k_2} = \emptyset, \forall 1 \leq k_1, k_2 \leq n_r, \quad (5)$$

$$n_c = \left|\cup_{k=1}^{n_r} C_k\right|, \quad (6)$$

$$n_c \leq N_c, \quad (7)$$

$$n_r \leq N_r, \quad (8)$$

$$n_q \leq N_q. \quad (9)$$

If $n_r = N_r$ and $n_q = N_q$ at a base station, any incoming request to that base station will be denied service.

The features of each request k consist of minimum transmission rate m_k, lifetime l_k, and initial payload p_k before transmission starts. At each time slot t during transmission, the constraints are given as follows:

$$\mathsf{r}_k(t) \geq m_k(t), \quad (10)$$

$$l_k(t) > 0 \quad (11)$$

where $l_k(t)$ denotes the remaining lifetime at time t. Note that with this definition, we have $l_k(0) = l_k$. If any request being processed fails to meet the constraints (10) or (11), it will be terminated and be marked as failed. Any request in the queue that fails to meet constraint (11) will also be removed and marked as failure. Otherwise, the lifetime will be updated for each request being processed and each request in the queue as follows:

$$l_k(t+1) = l_k(t) - 1. \quad (12)$$

Each request being processed will transmit $\mathsf{r}_k(t)$ bits of the remaining payload:

$$p_k(t+1) = \max(p_k(t) - \mathsf{r}_k(t), 0). \quad (13)$$

Note that the initial payload is $p_k(0) = p_k$. If the payload is completed within the lifetime (i.e., the remaining payload is $p_k(t+1) = 0$ for $t+1 \leq l_k$), the request k is completed and marked as success.

For all aforementioned cases, if the request is completed successfully in time, the network slicing agent at the base station b receives a positive reward R_k equal to the initial payload p_k. Otherwise, it receives a negative reward of $R_k = -p_k$. Each user can only send one request at a time.

Afterwards, base station b records the latest transmission rate history of $\mathsf{r}_c^{b,u}$ into a 2 dimensional matrix $H^b \in \mathbb{R}_{\geq 0}^{N_u \times N}$. In each time slot t, for request k from user u that is allocated subset C_k of channels at base station b, the base station updates the entries corresponding to the channels that were selected for transmission:

$$H^b[u,c] = \mathsf{r}_c^{b,u}(t), \forall c \in C_k. \quad (14)$$

Note that the location $\{x_u, y_u\}$ of user u, the fading coefficient $h_c^{b,u}$, and the interference corresponding to $\mathbf{1}_c^{b,b'}$ vary over time, and therefore H_b is only a first-order approximation of the potential transmission rate $\mathsf{r}_c^{b,u}(t+1)$.

The goal of the network slicing agent at each base station is to find a policy π that selects n_c channels and assigns them to n_r requests, so that the sum reward of all requests at all base stations is maximized over time:

$$\underset{\pi}{\operatorname{argmax}} \sum_{t'=t}^{\infty} \left(\gamma^{(t'-t)} \sum_{b=1}^{N_B} \sum_{k \in K_b'} R_k\right), \quad (15)$$

where K_b' is the set of completed or terminated requests for base station b at time t, and $\gamma \in (0,1)$ is the discount factor.

III. Multi-Agent Deep RL with Multiple Actors and Centralized Critic (MACC)

To solve the problem in (15), we propose a multi-agent deep RL algorithm with multiple actors and centralized critic (MACC) where the actors utilize the pointer network to achieve the goal of attaining the maximal reward over all base stations by choosing the optimal subset of channels in processing each request. In the remainder of this paper, we denote the full observation at base station b as $\mathcal{O}_b$, the observation over all base stations as $\mathcal{O} = \cup_{b=1}^{N_B} \mathcal{O}_b$, and the channel selection at base station b as a matrix of actions $\mathcal{A}_b \in \{0,1\}^{n_r \times N}$. Each element in $\mathcal{A}_b$ is an indicator:

$$\mathcal{A}_b[k,c] = \begin{cases} 1 & \text{if channel } c \text{ is assigned to request } k, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

In each time slot when $n_c(t)$ channels are selected, $\sum_c \sum_k \mathcal{A}_b[k,c] = n_c(t)$. In this section, we first introduce the MACC framework, then describe the pointer network as the actor structure, and finally analyze the implementation on the considered network slicing problem.

A. Deep RL with Multiple Actors and Centralized Critic

In this section, we first introduce the actor-critic algorithm [21]. This deep RL algorithm utilizes two neural networks: the actor and critic. The two networks have separate neurons and utilize separate backpropagation, and they may have separate hyperparameters.

The multi-agent extension of actor-critic algorithm where each agent has separate actor and critic that aim at maximizing each individual reward is referred to as independent actor-critic (IAC) [16], [17]. When the action of each agent interferes with the others and the goal is to maximize the sum reward over all agents, the deep RL with MACC may be preferred [17], [18]. In this framework, the decentralized actors with parameter θ and policy $f^{\theta}(\mathcal{O}_b)$ are the same as in IAC, while there is only one centralized critic with parameter ϕ and policy $g^{\phi}(\mathcal{O})$ aiming at the sum reward by updating each decentralized actor. Therefore, MACC agents are more likely to learn coordinated strategies through interaction between agents and mutual environment, despite the scarcity of information sharing at the training phase. In the framework of MACC, the critic maps $\mathcal{O}$ to a single temporal difference (TD) error

$$\delta(t) = \sum_{b=1}^{N_B} \sum_{k \in K_b} R_k(t) + \gamma g^{\phi}\left(\mathcal{O}(t)\right) - g^{\phi}\left(\mathcal{O}(t-1)\right) \tag{17}$$

where K_b is the set of requests being processed and requests in the queue at time t, and $\gamma \in (0,1)$ is the discount factor. Then, the critic parameters ϕ and actor parameters θ are optimized by considering the least square temporal difference and policy gradient, respectively, as follows:

$$\phi^* = \operatorname*{argmin}_{\phi} (\delta^{g_\phi})^2, \tag{18}$$

$$\theta^* = \operatorname*{argmax}_{\theta} \nabla_\theta \log \pi_\theta(\mathcal{O}_b) \delta^{g_\phi}. \tag{19}$$

For a wireless resource allocation problem, the fast convergence in the training of neural networks is highly important. Therefore in our implementation, we not only recommend offline pre-training via simulations, but also speed up learning by sharing the neural network parameters among the agents, i.e., we use all the information to update one actor and one critic (as in IAC) during training and share the parameters to all agents.

B. Pointer Network

In this section, we introduce the pointer network to implement the actor policy f^{θ} for IAC and MACC.

Traditional sequence-processing recurrent neural networks (RNN), such as sequence-to-sequence learning [22] and neural

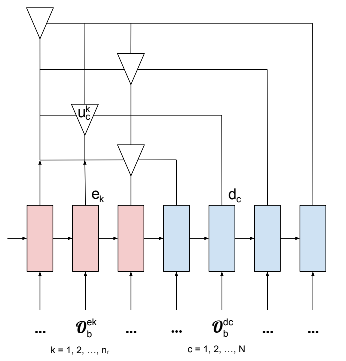

Fig. 2. Pointer network structure. At each step, the encoder RNN (red) converts each element of input $\mathcal{O}_b^{ek}$ into output e_k, while the hidden state and cell state are fed to the decoder RNN (blue). The decoder converts each element of input $\mathcal{O}_b^{dc}$ into output d_c at each step. The final output of the pointer network is generated by attention mechanism as the conditional probability $\mathcal{P}^c$, and each element are generated via e_k and d_c.

Turing machines [23], have a limit on the output dimensionality, i.e., the output dimensionality is fixed by the dimensionality of the problem so it is the same during training and inference. However in a network slicing problem, the actor's input dimensionality $\{N, n_c, n_r\}$ may vary over time, and thus the expected output dimension of $\mathcal{A}_b$ also varies according to the input. As opposed to the aforementioned sequence-processing algorithms, pointer network learns the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in the input sequence [19], and therefore the dimension of action $\mathcal{A}_b$ in each time slot depends on the length of the input.

Another benefit of pointer networks is that they reduce the cardinality of $\mathcal{A}_b$. General deep RL algorithms typically list out each combination as an element in action $\mathcal{A}'_b$, and each element indicates picking n_c channels out of N channels and assign each to one of n_r requests. Therefore $\mathcal{A}'_b$ have the dimension of $\binom{N}{n_c} n_r^{n_c}$. When the dimensions of N, n_c and n_r are high, this will require a prohibitively large network to give such an output, and require exponentially longer time to train. Comparatively, $\mathcal{A}_b$ generated from our proposed pointer network actor has only $n_r N$ output elements.

Therefore, we here introduce the pointer network as the novel architecture of the actor of the proposed agents as shown in Fig. 2. Similar to sequence-to-sequence learning, pointer network utilizes two RNNs called encoder and decoder whose output dimensions are h_e and h_d. The former encodes the sequence $\{\mathcal{O}_b^{e1}, \mathcal{O}_b^{e2}, \ldots, \mathcal{O}_b^{en_r}\}$ with $\mathcal{O}_b^{ek} \subset \mathcal{O}_b$ (where the index $k \in \{1, 2, \ldots, n_r\}$ corresponds to requests), and produces the sequence of vectors $\{e_1, e_2, \ldots, e_{n_r}\}$ with $e_k \in \mathbb{R}^{h_e}$ at the output gate in each step. The decoder inherits the encoder's hidden state and is fed by another sequence $\{\mathcal{O}_b^{d1}, \mathcal{O}_b^{d2}, \ldots, \mathcal{O}_b^{dN}\}, \mathcal{O}_b^{dc} \subset \mathcal{O}_b$ (where the index $c \in \{1, 2, \ldots, N\}$ corresponds to channels), and produces the sequence of vectors $\{d_1, d_2, \ldots, d_N\}, d_c \in \mathbb{R}^{h_d}$.

Furthermore, pointer network computes the conditional probability array $\mathcal{P}^c \in [0,1]^{n_r}$ where

$$\mathcal{P}^c[k] = P(\mathcal{A}_b[k,c] = 1 | \mathcal{O}_b^{d1}, \ldots, \mathcal{O}_b^{dc}, \mathcal{O}_b^{e1}, \ldots, \mathcal{O}_b^{en_r}) \tag{20}$$

using attention mechanism as follows:

$$\begin{aligned} u_k^c &= v^T \tanh(W_1 e_k + W_2 d_c), \\ \mathcal{P}^c &= \operatorname{softmax}([u_1^c, u_2^c, \ldots, u_{n_r}^c]^T) \end{aligned} \tag{21}$$

where softmax normalizes the vector of u_k^c to be an output distribution over the dictionary of inputs, and vector v and matrices W_1 and W_2 are trainable parameters of the attention model.

We note that the $n_r \times N$ matrix of the pointer network output $\mathcal{P} = [\mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^N]$ typically consists of non-integer values, and we obtain the decision of $\mathcal{A}_b$ by picking n_c maximum elements in each column:

$$\begin{aligned} M_1 &= [\max(\mathcal{P}^1), \max(\mathcal{P}^2), \ldots, \max(\mathcal{P}^N)], \\ M_2 &= \underset{M_1' \subset M_1, |M_1'| = n_c}{\operatorname{argmax}} \sum_{m \in M_1'} m, \\ \mathcal{A}_b[k, c] &= \begin{cases} 1 & \text{if } c \in M_2 \text{ and } k = \operatorname{argmax} \mathcal{P}^c, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{22}$$

In our implementation, we use two Long Short Term Memory (LSTM) [24] RNNs of the same size (i.e., $h_e = h_d$) to model the encoder and decoder. During the backpropagation phase of neural network training, the partial derivative of error functions with respect to weights in layers (encoder and decoder in our case) that are far from the output layer may be vanishingly small, preventing the network from further training, and this is called vanishing gradient problem [25]. However, the encoder and decoder are cardinal for the function of the pointer network while we demand fast convergence, so we set W_1 and W_2 in (21) as trainable vectors and set v as a single constant to reduce the depth of the pointer network and speed the training on both LSTMs.

C. Network Slicing Agent

In this subsection, we introduce the action and state spaces for MACC deep RL with pointer networks employed for the aforementioned network slicing problem.

1) Centralized Critic: The centralized critic agent $g^{\phi}(\mathcal{O})$ is implemented as a feed-forward neural network (FNN) that uses ReLU activation function for each hidden layer. This FNN takes the observation $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{N_B}\}$ as the input where $\mathcal{O}_b(t) = \{\mathcal{A}_b(t-1), H^b[\mathcal{U}_b(t), \mathcal{C}](t), \mathcal{I}_b(t)\}$. $\mathcal{A}_b(t-1)$ is the action matrix as described in (22) in the last time slot, $H^b(t)$ is the transmission rate history described in (14), $\mathcal{U}_b(t)$ is the set of users corresponding to the n_r requests served by base station b, and $\mathcal{C}$ is the set of all channels $\{1, 2, \ldots, N\}$. $\mathcal{I}_b$ is the information of requests, including the remaining payload p, minimum rate m, remaining lifetime l and absolute value of reward $|R| = p_k$ of the requests being processed and the requests in the queue: $\mathcal{I}_b = \mathcal{I}_b^{K_b}$ where $\mathcal{I}_b^k = \{p_k, m_k, l_k, p_k\}$.

The output of $g^{\phi}(\mathcal{O})$ is a single value that aims at minimizing the error $\delta(t)$ in (17). Note that in our implementation, the sum reward is not the instantaneous reward at time t, but the rewards of K_b, the set of all requests being processed and requests in the queue at time t. Therefore, we do not get the reward value or use the sample for training until all requests and those in queue are completed. Compared to the instantaneous reward, we use this reward assignment to strengthen the correlation between the action $\mathcal{A}_b$ and the outcome, and thus speed up the convergence.

2) Decentralized Actor with Pointer Network: The decentralized actor agent $f^{\theta}(\mathcal{O}_b)$ at each base station b is implemented as a pointer network. The initial hidden state and cell state of the encoder LSTM are generated by a separate FNN without hidden layers, whose input is the information of the requests in the queue $\{\mathcal{I}_b^{n_r+1}, \mathcal{I}_b^{n_r+2}, \ldots, \mathcal{I}_b^{n_r+n_q}\}$, and the output is a vector of two states with length $2h_e$.

The input that the encoder LSTM receives at each step is $\mathcal{O}_b^{ek}(t) = \{\mathcal{A}_b[k, \mathcal{C}](t-1), H^b[\mathcal{U}_b[k](t), \mathcal{C}](t), \mathcal{I}_b^k(t)\}$ whose components are the last action corresponding to request k, transmission rate history corresponding to the user $\mathcal{U}_b[k]$ of request k, and the information of request k.

The input that the decoder LSTM receives at each step is $\mathcal{O}_b^{dc}(t) = \{H^b[\mathcal{U}_b(t), c](t), \cup_{k=1}^{n_r} \mathcal{I}_b^k(t)\}$. The two components are the transmission rate history corresponding to all users $\mathcal{U}_b$ in channel c, and the information of all requests being processed.

Therefore as in (21), the output e_k and d_c of the encoder and decoder are fed into the attention mechanism to produce the conditional probability $\mathcal{P}$, and thus we obtain the output action $\mathcal{O}_b$ via (22).

Apart from the pointer network actor $f^{\theta}(\mathcal{O}_b)$, we consider two other decision modes to explore different actions and train MACC policies, including the random mode and the max-rate mode. The random mode randomly assigns n_c out of N channels to n_r requests without considering the observation, and the max-rate mode generates $\mathcal{O}_b$ via transmission rate history matrix $H^b[\mathcal{U}_b, \mathcal{C}]$ instead of $\mathcal{P}$:

$$\begin{aligned} M_1 &= [\max(H^b[\mathcal{U}_b, 1]), \ldots, \max(H^b[\mathcal{U}_b, N])], \\ M_2 &= \underset{M_1' \subset M_1, |M_1'| = n_c}{\operatorname{argmax}} \sum_{m \in M_1'} m, \\ \mathcal{A}_b[k, c] &= \begin{cases} 1 & \text{if } c \in M_2 \text{ and } k = \operatorname{argmax} H^b[\mathcal{U}_b, c], \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{23}$$

The agent follows an ϵ-ϵ_m-greedy policy as shown in Algorithm 1 below to update the neural network parameters. Specifically, it initially starts with a exploration probability $\epsilon = \epsilon_0 = 1$ and chooses the max-rate mode with probability ϵ_m, otherwise chooses the random mode. This probability decreases linearly to $\epsilon = \epsilon_1 \ll 1$ to follow mostly the actor policy $f^{\theta}(\mathcal{O}_b)$. When the training is over, ϵ is fixed to ϵ_1 in the testing duration T_{test}.

Algorithm 1 ϵ-ϵ_m-greedy exploration method

```
$\epsilon = \epsilon_0$
for i in range($T_{train}$) do
    for b in range($N_B$) do
        if random$(0,1) \geq \epsilon$ then
            Agent $b$ execute $f^{\theta}(\mathcal{O}_b)$ in (22)
        else
            if random$(0,1) \leq \epsilon_m$ then
                Agent $b$ execute max-rate mode in (23)
            else
                Agent $b$ execute random mode
            end if
        end if
    end for
    $\epsilon = \epsilon - (\epsilon_0 - \epsilon_1)/T_{train}$
end for
```

IV. Numerical Results

As shown in Fig. 3, we in the experiments consider a service area with $N_B = 5$ base stations and $N_u = 30$ users, and each cell radius is 2.5km. There are $N = 16$ channels available, and each base station picks at most $N_c = 8$ channels for transmission. The ratio between transmission power and the noise in each channel is $P_B/\sigma_k^2 = 6.3$. Each base station allows processing of $N_r = 4$ requests, and keeps at most $N_q = 2$ requests in the queue. When a request from one user

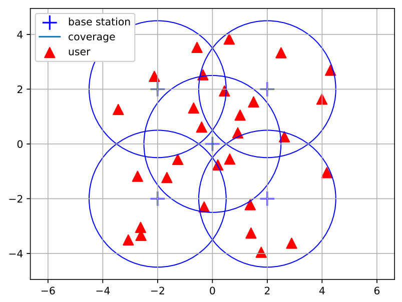

Fig. 3. Coverage map of service area with 5 base stations and 30 users.

is terminated or completed, the user will not be able to send another request within $T_r = 2$ time slots.

The channel varies with maximum Doppler frequency $f_d = 1$ and dynamic slot duration $T = 0.02s$, and the location of the users follows a random walk pattern. Each user may transfer between the coverage of different base stations, while staying within the coverage area of at least one of the base stations. All other parameters are listed in Table I.

TABLE I
USER AGENT TRAINING PARAMETERS

Number of Base Stations N_B	5
Height of Base Station h_B	0.5
Range of Base Station Coverage	2.5
Number of MVNO N_M	1
Number of Users N_u	30
Number of Channels N	16
Number of Channels for Transmission N_c	8
Maximum Doppler Frequency f_d	1
Dynamic Slot Duration T	0.02
Transmission Power to Noise Ratio P_B/σ_k^2	6.3
Path Loss Exponent α	-2
Maximum Number of Requests Processing N_r	4
Maximum Number of Requests in Queue N_q	2
Minimum Inter-arrival Time of Requests T_r	2
Random Range of Initial Payload p	(2, 4)
Random Range of Minimum Rate m	(0.6, 1)
Random Range of Lifetime l	$p/m+$ (3, 6)
Discount Factor γ	0.9
Adam Optimizer Learning Rate	10^{-6}
Critic FNN Hidden Layer and Nodes	1, 70
Encoder LSTM Hidden State Size h_e	70
Decoder LSTM Hidden State Size h_d	70
Training Time T_{train}	50000
Testing Time T_{test}	50000
Training Period T_t	10
Mini-batch	10
Initial Exploration Probability ϵ_0	1
Final Exploration Probability ϵ_1	0.005
Max-Rate Probability ϵ_m	0.2

In Table II, we provide the performance results achieved with the proposed MACC algorithm with pointer network during the testing phase, in terms of average sum reward per time slot and the ratio of completed requests over all requests. Note that we set the parameters of request generation as demanding as possible to test the algorithm in a challenging environment, in which some of the requests might be very difficult to complete. For comparison, we also show the performance of six other algorithms in the first six rows in the table.

The first three rows show the performance of statistical algorithms. The max-rate agent, which always executes the max-rate mode as in (23) and chooses user-channel pairs with maximal channel rate, completes 76.33% of the requests, and has an average reward of 21.29 bits/symbol. When one of the users is close to the station and has maximum rate in several channels, max-rate agent does not take the requirement of requests into account, and may assign more channels to the user than it needs. Therefore, the performance of the max-rate agent may not necessarily be efficient.

TABLE II
PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS DURING TEST PHASE

actor-critic	actor policy	average reward	complete ratio
None	max-rate	21.29	76.33%
None	FIFO	20.50	85.23%
None	hard slicing	19.91	79.42%
IAC	FNN	11.06	65.15%
MACC	FNN	9.03	62.38%
IAC	pointer network	26.14	84.28%
MACC	pointer network	34.59	95.31%

The first-in first-out (FIFO) agent also selects channels with high channel rate history, but it always assigns enough channel resources to the requests arriving earlier to fulfill their minimum transmission rate requirement. Compared to the max-rate agent, FIFO agent tends to complete the earlier requests so that the consumed resource in the past is less likely to be wasted by a failure. As a result, it obtains a slightly lower average reward but higher completion ratio than those of the max-rate one. However, the FIFO agent fails to take the overall reward $|R_k|$ of these requests and lifetime limit of requests in the queue into account, and hence it may fail during a surge of requests.

The hard slicing agent allocates channels to each request according to the maximum historical transmission rate $H^b[u, c]$, and the number of channels that are assigned to each request is proportional to the corresponding minimum transmission rate m_k. This hard slicing agent completes 79.42% of the requests, and the average reward is 19.91 bits/symbol. Compared to the FIFO agent, the hard slicing agent tends to pursue an average performance over different requests and neglect requests with high reward, and therefore it may perform well when every request can be completed, but not likely to do the same in a demanding environment where it has to discard some requests.

Compared to the three aforementioned statistical methods, we also test the IAC and MACC deep RL algorithms. To demonstrate the effectiveness of the pointer network actor, we also show the results when the actor policy is implemented as a single FNN. The input of this FNN is an extension of $\mathcal{O}_b$, i.e., when $n_r < N_r$, it is zero-padded to a full size of N_r requests. The output is $\mathcal{P}' \in [0,1]^{N_r \times N}$ and we post-process the output by picking a subset $\mathcal{P}$ which is the first n_r rows of $\mathcal{P}'$, and follow (22) to get $\mathcal{A}_b$. The training process also has ϵ-ϵ_m-greedy exploration method as in Algorithm 1. We note that both IAC and MACC with FNN actor have much lower performances compared to the max-rate, FIFO, or hard slicing agent, and therefore they fail to learn an effective policy. Compared to the pointer network, it is much harder for FNN to build a correlation between each single output element $\mathcal{P}^c[k]$ and the subset of input information $\mathcal{O}_b$ that corresponds to channel c and request k. Furthermore, FNN also suffers from the truncation from $\mathcal{P}'$ to $\mathcal{P}$ as most other

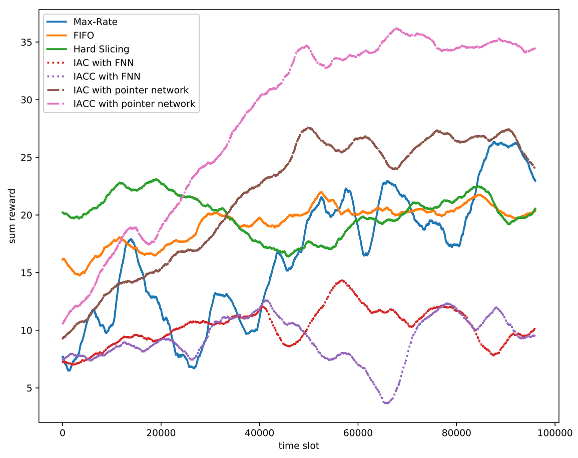

Fig. 4. Comparison of sum reward over all agents in Table II.

neural network structures with fixed dimensionality would do in this network slicing problem.

Finally, we compare IAC and MACC with pointer network based actors. We observe that MACC with pointer network actors obviously outperforms all other aforementioned algorithms with 95.31% requests completed, and the average reward is 34.59 bits/symbol. We see that IAC with pointer networks also performs well with similar complete ratio and higher average reward than the FIFO agent, and this shows that pointer networks are more effective in the learning of a desired actor policy. However, compared to that of MACC, either max-rate mode or IAC focuses on optimizing the local performance, and fails to cooperate in an interference channel environment within limited training time. Therefore, MACC with pointer networks has the best performance by optimizing the sum reward over all base stations.

In Fig. 4, we plot the moving average of the sum reward for all seven aforementioned agents. We can see that hard slicing agent has the best performance at the very beginning. On the other hand, the deep RL agents start with random parameters and has ϵ-ϵ_m-greedy policy which starts with a large probability $\epsilon(1-\epsilon_m)$ to explore the random actions, and consequently the performance starts low but gradually improves. We observe that in the test phase, MACC with pointer network based actors eventually outperform all other agents. Its sum reward converges to around 34 bits/symbol, while slightly oscillating due to the challenging dynamic environment with random channel states, varying user locations and randomly arriving requests.

V. Conclusion

In this paper, we designed network slicing agents using MACC multi-agent deep RL with pointer network based actors. We considered a multiple base station coverage area and dynamic environment with time-varying channel fading, mobile users, and randomly arriving service requests from the users. We described the system model, formulated the network slicing problem, and designed the MACC deep RL algorithm and pointer network based actor structure. We demonstrated the proposed agents' performance by comparing against three statistical algorithms and three other deep RL based algorithms. The proposed network slicing agents are shown to outperform by completing 95% of the requests and achieving the highest sum rate rewards under a demanding parameter setting.

References

[1] A. Ericsson, "5G systems enabling the transformation of industry and society," *Tech. Rep.*, 2017.
[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, 2017.
[3] N. Alliance, "Description of network slicing concept," *NGMN 5G P*, vol. 1, no. 1, 2016.
[4] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5G and future mobile networks: mobility, resource management, and challenges," *IEEE communications magazine*, vol. 55, no. 8, pp. 138–145, 2017.
[5] S. A. Kazmi, L. U. Khan, N. H. Tran, and C. S. Hong, *Network slicing for 5G and beyond networks*. Springer, 2019, vol. 1.
[6] Q. Ye, W. Zhuang, S. Zhang, A.-L. Jin, X. Shen, and X. Li, "Dynamic radio resource slicing for a two-tier heterogeneous wireless network," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9896–9910, 2018.
[7] H. Peng, Q. Ye, and X. Shen, "Spectrum management for multi-access edge computing in autonomous vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 7, pp. 3001–3012, 2019.
[8] J. Tang, B. Shim, and T. Q. Quek, "Service multiplexing and revenue maximization in sliced c-ran incorporated with urllc and multicast embb," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 4, pp. 881–895, 2019.
[9] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep reinforcement learning based resource allocation for v2v communications," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3163–3173, 2019.
[10] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.
[11] F. Wang, M. C. Gursoy, and S. Velipasalar, "Adversarial reinforcement learning in dynamic channel access and power control," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2021, pp. 1–6.
[12] R. Li, Z. Zhao, Q. Sun, I. Chih-Lin, 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.
[13] L. Zhang, J. Tan, Y.-C. Liang, G. Feng, and D. Niyato, "Deep reinforcement learning-based modulation and coding scheme selection in cognitive heterogeneous networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 6, pp. 3281–3294, 2019.
[14] Y. Shi, Y. E. Sagduyu, and T. Erpek, "Reinforcement learning for dynamic resource optimization in 5G radio access network slicing," in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2020, pp. 1–6.
[15] Y. Shao, R. Li, Z. Zhao, and H. Zhang, "Graph attention network-based drl for network slicing management in dense cellular networks," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2021, pp. 1–6.
[16] X. Lyu, Y. Xiao, B. Daley, and C. Amato, "Contrasting centralized and decentralized critics in multi-agent reinforcement learning," *arXiv preprint arXiv:2102.04402*, 2021.
[17] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *arXiv preprint arXiv:1706.02275*, 2017.
[19] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *arXiv preprint arXiv:1506.03134*, 2015.
[20] L. Liang, J. Kim, S. C. Jha, K. Sivanesan, and G. Y. Li, "Spectrum and power allocation for vehicular communications with delayed CSI feedback," *IEEE Wireless Communications Letters*, vol. 6, no. 4, pp. 458–461, 2017.
[21] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.
[22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
[23] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.
[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[25] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.