5G Network Slicing and Drone-Assisted Applications: A Deep Reinforcement Learning Approach

Linh Le, Tu N. Nguyen, Kun Suo, and Jing (Selena) He*
Kennesaw State University, Marietta, GA 30060, USA
{linh.le, tu.nguyen, ksuo, selena.he}@kennesaw.edu

ABSTRACT

5G radio access network (RAN) slicing is gaining momentum in finding applications in a wide range of domains, especially those requiring high-speed data transmissions such as space-air-ground integrated networks (i.e., drone-based systems). A key challenge in building a robust RAN slicing to support these applications is, therefore, designing a RAN slicing (RS)-configuration scheme that can utilize information such as resource availability in substrate networks as well as the interdependent relationships among slices to map (embed) virtual network functions (VNFs) onto live substrate nodes. With such motivation, we propose a machine-learning-powered RAN slicing scheme that aims to accommodate maximum numbers of slices (a set of connected VNFs) within a given request set. We present a deep reinforcement scheme that is called Deep Allocation Agent (DAA). In short, DAA utilizes an empirically designed deep neural network that observes the current states of the substrate network and the requested slices to schedule the slices of which VNFs are then mapped to substrate nodes using an optimization algorithm. DAA is trained towards the goal of maximizing the number of accommodated slices in the given set by using an explicitly designed reward function. Our experiment study shows that, on average, DAA is able to maintain a rate of successfully routed slices above 80% in a resource-limited substrate network, and about 60% in extreme conditions, i.e., the available resources are much less than the demands.

KEYWORDS

5G RAN slicing, deep reinforcement learning, embedding, VNFs mapping, UAV/drone-assisted communications.

1 INTRODUCTION

Unmanned Aerial Vehicle (UAV) communications can assist the existing cellular communications for the rapid service recovery in a high-speed fashion as shown in Fig. 1. However, networks with components mounted on UAV's may suffer from problems such as the lack of computational resources as well as endurance capabilities. Therefore, fault-tolerant communication schemes that can quickly recover from failures are of importance for UAV-assisted communications.

*Corresponding author: Tu N. Nguyen. This research was supported in part by US NSF grants CNS-2103405, AMPS-2229073, CPS-2103459, SHF-2210744, and was also supported in part by a grant from SunTrust/Truist Banks, Inc. and the KSU Institute for Cybersecurity Workforce Development.

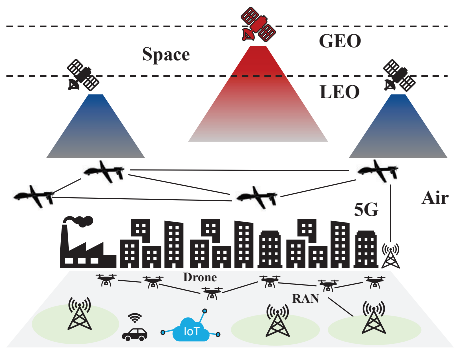


Figure 1: DAA deep neural network architecture

Radio Access Network (RAN) slicing is a critical technology for the new generation 5G network. There have been efforts to develop an integrated network architecture from the drone-assisted, air-based network and ground-based network where RAN slicing plays a critical role [1–4]. In particular, a RAN slice that is independent from others consists of a set of VNFs. Upon receiving requests from the enterprise, the mobile network operator (MNO) works with the RAN enforcement to allocate the substrate network resources to VNFs and virtual links between VNFs.

There have been efforts to design algorithms for the RAN resource allocation problem in drone-based systems that tend to consider only the available resources of the substrate network to design a mapping plan and disregard the VNF connectivity and bandwidth requirements [5–12]. However, mapping VNF strategy will not only hinge on substrate nodes resource allocation but also rely on the substrate connection and the bandwidth requirement of the virtual links between VNFs. This is a key research challenge we will tackle in the proposed research. In particular, for an RAN slicing (RS)-configuration in a cellular network, we would like to explicitly account for the *mapping plan* of VNFs, with the goal of providing *stable* performance for drone-based applications.

With such motivation, in this paper, we present a deep reinforcement scheme, *Deep Allocation Agent (DAA)*, that is able to embed VNFs of a slice while being aware of the interdependency among slices. We model the problem of maximizing the number of accommodated network slices as a reinforcement learning problem: **1) Environment States** include information on the current substrate network, i.e., available resources in all substrate nodes, and a set of pending slices to be accommodated, i.e., VNFs in each slice and their demands. **2) Episodes** are the accommodation of all slices in a given set. An episode ends by allocating resources to all slices in the given set, or by entering an environment state in which no pending slices can be further resolved. Each step in an episode refers to the allocation of a slice. **3) Actions** that the agent makes at each step include selecting then allocating resources to VNFs in the slice. **4) Rewards** that the agent receives after an action are designed to guide the agent to generate schedule that maximize the number of allocated slices in a given set.

The neural network of DAA first observes the current environment state to select a pending slice to accommodate. Its VNFs are then mapped to substrate nodes using an algorithm that maximizes the flexibility of the substrate network and the allocability of the remaining VNFs. We train the deep network of DAA as a deep reward network (DRN). In this case, the neural network predicts the true reward obtained if a slice is selected. The training of DRN is guided by the previously mentioned reward function. Our experiment study shows that, on average, DAA models are able to maintain a rate of successfully accommodated slices about 80% in a resource-scarce case (i.e., the available resources are less than the demands from the slices), and approximately 60% in extreme conditions of network (available resources are much less than demands). Specifically, the paper has the following **contributions** and **intellectual merits**:

(1) We tackle the problem of maximizing the number of slices in network slicing using deep reinforcement learning to support the space-air-ground integrated networks such as drone-assisted systems. Specifically, we propose a reinforcement learning model with specific designs of environments, actions, and rewards, that guide the agents towards a schedule that fulfills the most slices in a given set.
(2) We present DAA, a deep reinforcement scheme that consists of an empirically designed deep neural network that schedules slices and an allocation algorithm that maps VNFs to substrate nodes while being aware of the network's flexibility. The deep network of DAA is trained as a deep reward network to maximize the number of accommodated slices. Furthermore, DAA enjoys the scalability of a deep architecture (polynomial time) that is an advantage over traditional optimization methods which is highly important especially in the context of UAV/drone-assisted communications.

Organization: The rest of the paper is organized as follows. Section §2 discusses the concepts related to our work, and mathematically formalizes our research problem. We discuss DAA in Section §3 then present the experiment study in Section §4. Finally, we conclude our paper in Section §5 and discuss future application in Section §6.

2 MODEL AND RESEARCH PROBLEM

In the following sections, an overview of the quantum network and basic mathematical notations are introduced. Formal research problem is also defined to demonstrate the key ideas behind the proposed model and methodology.

2.1 Network Model

Given a substrate network $\mathcal{G} = \{\mathcal{N}, \mathcal{R}\}$ in which we have a set of nodes $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \dots, \mathcal{N}_n\}$ and a set of resources $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_n\}$. A node $\mathcal{N}_i$ has maximum resources available of $\mathcal{R}_i$. In other words, any node $\mathcal{N}_i$ can allocate a maximum amount $\mathcal{R}_i$ of resources to VNFs. A VNF in a specific slice is not available and accessible by other network slices for the isolation purpose.

Let $G = \{G^1, G^2, \dots, G^l\}$ be the set of l RAN slices of which resources need to be allocated in $\mathcal{G}$. Each slice $G^i = \{V^i, R^i\}$ consists of a VNF set $V^i = \{v^i_1, v^i_2, \dots, v^i_{l_i}\}$ and a resource set $R^i = \{r^i_1, r^i_2, \dots, r^i_{l_i}\}$ in which the demanded resources for VNF v^i_j is r^i_j. In other words, v^i_j needs to be assigned to a substrate node $\mathcal{N}_k$ with $\mathcal{R}_k \geq r^i_j$ to be considered successfully mapped. To mathematicize the allocation of a VNF v^i_j to a substrate node $\mathcal{N}_k$, we introduce a binary variable $m^{i,j}_k$. $m^{i,j}_k = 1$ when v^i_j is mapped to $\mathcal{N}_k$, and $m^{i,j}_k = 0$ otherwise. A VNF can only be allocated to one substrate node, therefore we have the constraint $\sum_{k=1}^{n} m^{i,j}_k = 1$. The maximum resources available in each substrate node cannot be exceeded when allocating VNFs, which leads to another constraint $\sum_{i=1,j=1}^{l,l_i} m^{i,j}_k * r^i_j \leq \mathcal{R}_k$. We further assume the slice G^i to be successfully allocated only when all $v^i_j \in V^i$ are successfully allocated. Consequently, we introduce a binary variable A^i. $A^i = 1$ when $\sum_{j=1,k=1}^{l_i,n} m^{i,j}_k = l_i$, and $A^i = 0$ otherwise. In other words, $A^i = 1$ when slice G^i has all its VNFs successfully mapped to the substrate network.

2.2 Problem Formulation

In this work, we advocate a novel allocation scheme – dubbed *Deep Reinforce Allocation Scheme ($Q_{\mathcal{AS}}$)* – that implements a deep reinforcement learning model to circumvent the computability limitations of *heuristic* conventional resource allocation schemes, while concurrently maximizing serviceability of the network. In other words, our allocation scheme aims to maximize the number of network slices that are successfully allocated with resources to operate properly.

Given $G = \{G^1, G^2, \dots, G^l\}$ be the set of l RAN slices of which resources need to be allocated in $\mathcal{G}$. Our goal is to maximize $Q_{\mathcal{RS}}$ as follows:

$$Q_{\mathcal{AS}} = \max_{i \in \{1,2,\dots,l\}} \sum A^i \tag{1}$$

We further constraint the number of VNFs in all slices in G to be equal $l_i = s\ \forall\ i$. It should be noted that this constraint

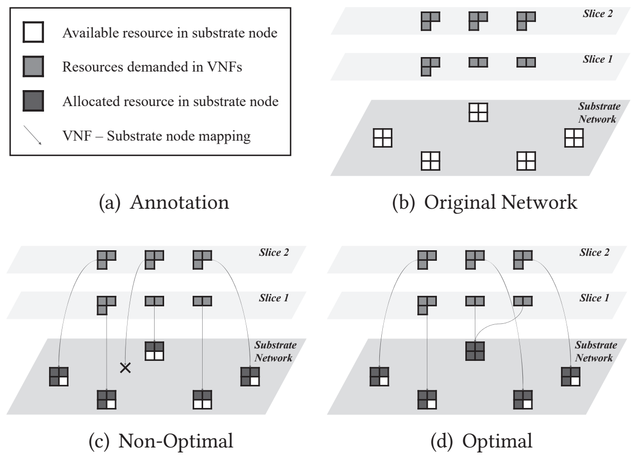

Available resource in substrate node
Resources demanded in VNFs
Allocated resource in substrate node
VNF – Substrate node mapping
Slice 2
Slice 1
Substrate Network
×
(a) Annotation
(b) Original Network
(c) Non-Optimal
(d) Optimal

Figure 2: An example on a VNFs embedding problem and its non-optimal and optimal solutions

does not reduce the generality of the research problem, since a slice j VNFs with $j < s$ can be padded with VNFs of 0 demands to meet the size requirement. In this problem, we also do not constraint that all slices in G can be allocated successfully in $\mathcal{G}$. This means that a complete allocation solution may not exist for a mapping from G to $\mathcal{G}$.

We break the problem into two tasks which are 1) to schedule slices to allocate and 2) to map VNFs in the selected slice to substrate nodes. Fig. 2(b) shows the substrate network $\mathcal{G}$ of five nodes, each with 4 resource blocks. There are two slices that need accommodations. Fig. 2(c) demonstrates the case in which slice 1 is accommodated first with the VNFs are randomly mapped to substrate nodes that satisfy the resource constraint. This solution is not optimal as slice 2 cannot be accommodated anymore. Fig. 2(d) shows the optimal solution in which slice 2 is accommodated first. Slice 1 can then be accommodated successfully as well. This example shows the importance of scheduling the RAN slices as well as a mapping strategy that conserves resources based on future unallocated slices and VNFs.

3 DEEP ALLOCATION AGENT

Given a substrate network $\mathcal{G}$ with the node set $\mathcal{N}$ and the resource set $\mathcal{R}$, and a set of RAN slices G, we refer to the accommodation of all slices G^i in G as an *episode*. At each step t, DAA makes an action $a^{(t)}$ to allocate resources to *one* slice G^* based on the current environment state including the current available resources $\mathcal{R}^{(t)}$ and the demands of pending slices $R^{(t)}$. An episode ends when all slices are resolved, or when no pending slices can be accommodated. In the following sections, we first discuss our VNF mapping algorithm. Then, we describe the DNN architecture of DAA in terms of input states, output actions, reward function, architecture, and training.

3.1 VNF Allocation Algorithm

The allocation of resources to a VNF, i.e., mapping a VNF to a substrate node, may significantly alter the performance of network slicing. Consequently, we derive mapping algorithm that select a substrate node for a given VNF while considering the **flexibility** of the substrate network and the **allocability** of the remaining VNFs. We first describe the concepts of flexibility and allocability in the context of our problem.

Substrate Network Flexibility. We define the flexibility of a substrate network as the capability to accommodate pending VNFs. Mathematically, given a substrate node $\mathcal{N}_k$ with available resource $\mathcal{R}_k^{(t)}$ at step t, and a RAN slice set G with VNFs $\{v_j^i | i \in \{1, l\}; j \in \{1, s\}\}$, the flexibility $\mathcal{F}(\mathcal{N}_k)$ of $\mathcal{N}_k$ is defined as

$$\mathcal{F}(\mathcal{N}_k) = \sum_{i=1,j=1}^{l,s} \mathbf{1}_{i,j}^k \tag{2}$$

where $\mathbf{1}_{i,j}^k$ is a binary indicator as follows

$$\mathbf{1}_{i,j}^k = \begin{cases} 1 & r_j^i \leq \mathcal{R}_k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The flexibility $\mathcal{F}^{\mathcal{S}}$ of the substrate network is then calculated as the sum of all substrate nodes' flexibility:

$$\mathcal{F}^{\mathcal{S}} = \sum_{k=1}^{n} \mathcal{F}(\mathcal{N}_k) \tag{4}$$

VNF Allocability. We define the allocability of a VNF as the number of substrate nodes that can accommodate the VNF. Mathematically, the allocability $\mathcal{A}(v_j^i)$ of a VNF v_j^i is defined as

$$\mathcal{A}(v_j^i) = \sum_{k=1}^{n} \mathbf{1}_{i,j}^k \tag{5}$$

with $\mathbf{1}_{i,j}^k$ defined previously. It can be seen that the lower $\mathcal{A}(v_j^i)$, the less substrate nodes can accommodate v_j^i, and when $\mathcal{A}(v_j^i) = 0$, v_j^i, and subsequently G^i, are completely blocked from being successfully allocated with resources. Therefore, to minimize the possibility of a VNF and its slice being blocked, as well as further increase the possible solutions for future mappings of remaining VNFs, we will derive mapping solution based on the minimum allocability across all VNFs at each step t

$$\mathcal{A}^{\mathcal{S}}(v_j^i) = \min_{i,j} \mathcal{A}(v_j^i) \tag{6}$$

VNF Allocation Algorithm. Given a VNF that needs resources, a substrate node is selected so that, **after** the allocation, both the substrate network flexibility $\mathcal{F}^{\mathcal{S}}$ and the $\mathcal{A}^{\mathcal{S}}(v_j^i)$ of the current network are maximized. More specifically, at step t, given a VNF v_j^i of demanding resource r_j^i and the substrate nodes $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_n\}$ with resources $\mathcal{R}^{(t)} = \{\mathcal{R}_1^{(t)}, \mathcal{R}_2^{(t)}, \ldots, \mathcal{R}_n^{(t)}\}$, the process is as follows.

(1) Temporarily allocate v_j^i to each $\mathcal{N}_k$ that satisfies $\mathcal{R}_k \geq v_j^i$ and update $\mathcal{R}^{(t)}$ to obtain $\mathcal{R}_{(k)}^{(t+1)}$ with $\mathcal{R}_{k(k)}^{(t+1)} \leftarrow \mathcal{R}_k^{(t)} - r_j^i$. Calculate the flexibility $\mathcal{F}^{\mathcal{S}}{}_{(k)}$ and allocability $\mathcal{A}^{\mathcal{S}}{}_{(k)}$
(2) Sort the substrate nodes $\mathcal{N}$ in descending order of $\mathcal{A}^{\mathcal{S}}{}_{(k)}$ then $\mathcal{F}^{\mathcal{S}}{}_{(k)}$ ($\mathcal{A}^{\mathcal{S}}{}_{(k)}$ is prioritized)
(3) Select the first substrate node $\mathcal{N}_*$ in the sorted $\mathcal{N}$ to be the mapping solution for v_j^i

This process ensure that $\mathcal{N}_*$ is selected so that the substrate network at $t+1$ has the maximum capability to accommodate the remaining VNFs, as well as has the lowest chance to block any VNFs and their slices completely.

3.2 Deep Reward Network Architecture

Input State. At step t in an episode $\mathcal{E}$, an input state to DAA consists of 1) the current available resource in each substrate node $\mathcal{R}^{(t)}$ and 2) the current demanding resources from all slices $R^{(t)}$. The data representations for $\mathcal{R}^{(t)}$ and $R^{(t)}$ are straightforward. $\mathcal{R}^{(t)}$ is modeled as a vector

$$\mathcal{R}^{(t)} = \{\mathcal{R}_1^{(t)}, \mathcal{R}_2^{(t)}, \ldots, \mathcal{R}_n^{(t)}\} \tag{7}$$

whereas $R^{(t)}$ is modeled as a $l \times s$ matrix

$$R^{(t)} = \begin{bmatrix} r_1^{1(t)} & r_2^{1(t)} & \ldots & r_s^{1(t)} \\ \ldots & \ldots & \ldots & \ldots \\ r_1^{l(t)} & r_2^{l(t)} & \ldots & r_s^{l(t)} \end{bmatrix} \tag{8}$$

If a slice G^i is already accommodated, its corresponding row $R_i^{(t)}$ in $R^{(t)}$ is replaced with a $\mathbf{0}$ vector. Slices of less than s VNFs can be padded with 0's to ensure the input matrix size.

Output Action. At each step t in an episode, DAA selects one among the pending slices $G^{(t)}$ then mapping its VNFs to the substrate network. We utilize a hybrid approach, that is to use the deep network DRN for scheduling the slices, and the algorithm presented previously to map the VNFs.

In terms of scheduling, DRN outputs a reward vector $\rho^{(t)}$ of size l: $\rho^{(t)} = \{\rho_0^{(t)}, \rho_1^{(t)}, \ldots, \rho_l^{(t)}\}$ in which $\rho_i^{(t)}$ represents the reward at the end of step t if DAA selects slice i to accommodate. Then, the *pending* slice with the highest reward value is selected in each step, $a^{(t)} = \text{argmax}_{i \in G_P^{(t)}}(\rho_i^{(t)})$, where $G_P^{(t)}$ is the set of pending slices at t. With $a^{(t)}$ selected, the VNFs in the chosen slice are sorted in descending order of demanding resources. Then, each VNF is mapped to the substrate network using the allocation algorithm that is described in Section §3.1. The reason we first sort the VNFs in descending order of demanding resources is that, VNFs of higher demands will have less allocability and thus less mapping options. Therefore, they should be accommodated first to avoid being blocked.

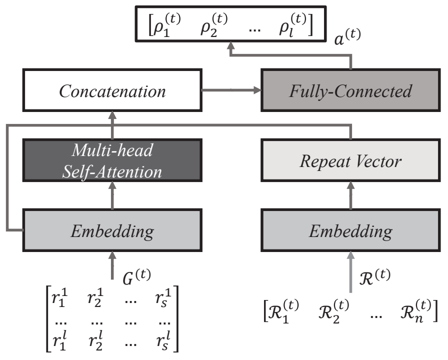

$[\rho_1^{(t)} \ \rho_2^{(t)} \ \ldots \ \rho_l^{(t)}]$
$a^{(t)}$
Concatenation
Fully-Connected
Multi-head Self-Attention
Repeat Vector
Embedding
Embedding
$G^{(t)}$
$\mathcal{R}^{(t)}$
$\begin{bmatrix} r_1^1 & r_2^1 & \ldots & r_s^1 \\ \ldots & \ldots & \ldots & \ldots \\ r_1^l & r_2^l & \ldots & r_s^l \end{bmatrix}$
$[\mathcal{R}_1^{(t)} \ \mathcal{R}_2^{(t)} \ \ldots \ \mathcal{R}_n^{(t)}]$

Figure 3: DAA deep neural network architecture

At the end of each action, the input states are updated for the next step $t+1$. First, the row that associates to the slice resolved by $a^{(t)}$ in $R^{(t)}$ is replaced with 0's to obtain $R^{(t+1)}$. Then, $\mathcal{R}^{(t)}$ is updated with the new available resource values after accommodating all VNFs in the selected slice.

Reward Function. The reward function is specifically designed to increase the number of accommodated slices by the substrate network. Mathematically, let the reward at step t be $P^{(t)}$, then

$$P^{(t)} = n_r^{(t)}\alpha + (l - n_r^{(t)})\beta f + \lambda P^{(t+1)}(1 - f) \tag{9}$$

where $n_r^{(t)}$ is the number of slices that are resolved from the beginning of the episode through step t, f is a binary indicator that is 1 when t is the ending step of and episode and 0 otherwise, $\alpha > 0$ is the reward term, $\beta < 0$ is the penalty term, and $\lambda \in [0, 1]$ is a discount factor. Both α, β, and λ are hyper-parameters to be selected during the training phase. Overall, the reward function design encourages the agent to find schedules that accommodate more slices, and penalizes schedules that end too early - the less slices solved, the heavier the penalties.

Reward Network Architecture. In this subsection, we describe the architecture of DRN. First, each input component among $\mathcal{R}^{(t)}$ and $R^{(t)}$ are fed into a different embedding network. In short, an embedding network consists of multiple fully-connected layers which transform an input vector into embedding vectors, usually of lower dimensionality. Let the mappings represented by the embedding networks for $\mathcal{R}^{(t)}$ and $R^{(t)}$ be $Emb_{\mathcal{R}}(\cdot)$ and $Emb_R(\cdot)$, respectively, then

$$\begin{aligned} Emb_{\mathcal{R}}(\mathcal{R}^{(t)}) &= U_{\mathcal{R}}^{(t)} = \begin{bmatrix} u_1^{\mathcal{R}(t)} & u_2^{\mathcal{R}(t)} & \ldots & u_{e_{\mathcal{R}}}^{\mathcal{R}(t)} \end{bmatrix} \\ Emb_R(R^{(t)}) &= U_R^{(t)} = \begin{bmatrix} u_{1,1}^{(t)} & u_{1,2}^{(t)} & \ldots & u_{1,e_R}^{(t)} \\ \ldots & \ldots & \ldots & \ldots \\ u_{l,1}^{(t)} & u_{l,2}^{(t)} & \ldots & u_{l,e_R}^{(t)} \end{bmatrix} \end{aligned} \tag{10}$$

where $e_{\mathcal{R}}$ is the size of the output embedding for $\mathcal{R}^{(t)}$, and e_R is the size of the output embedding for $R^{(t)}$.

For DRN to further learn the interrelationships among all the pending slices, $U_R^{(t)}$ is input into a *Multi-Head Self-Attention* (MHSA) block [13]. In short, a MHSA block outputs a "context" matrix in which each row represents the "context score" of a slice given the rest in $R^{(t)}$. The MHSA block allows DAA to further analyze all pending slice requests while selecting the optimal one to accommodate. Let the mapping by the MHSA block be $S_R^{(t)}$, then

$$S_R^{(t)} = \begin{bmatrix} \sigma_{1,1}^{(t)} & \dots & \sigma_{1,l}^{(t)} \\ \dots & \dots & \dots \\ \sigma_{l,1}^{(t)} & \dots & \sigma_{l,l}^{(t)} \end{bmatrix} \tag{11}$$

where $\sigma_{i,j}^{(t)}$ is the context score of G^i with respect to slice G^j. We repeat $u_{\mathcal{R}^{(t)}}$ l times, then concatenate them with the rows in $U_R^{(t)}$ and $S_R^{(t)}$ to form the embedding for each slice:

$$U^{(t)} = \begin{bmatrix} u_1^{\mathcal{R}(t)} \dots & u_{e_\mathcal{R}}^{\mathcal{R}(t)} & u_{1,1}^{(t)} \dots & u_{1,e_R}^{(t)} & \sigma_{1,1}^{(t)} \dots & \sigma_{1,l}^{(t)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ u_1^{\mathcal{R}(t)} \dots & u_{e_\mathcal{R}}^{\mathcal{R}(t)} & u_{l,1}^{(t)} \dots & u_{l,e_R}^{(t)} & \sigma_{l,1}^{(t)} \dots & \sigma_{l,l}^{(t)} \end{bmatrix} \tag{12}$$

Finally, U is input into a block of fully-connected layers that output a vector of $|\mathcal{D}|$ values $\rho^{(t)} = \{\rho_0^{(t)} \dots \rho_l^{(t)}\}$ that represent the reward if DAA selects each slice. The architecture of DAA's deep neural network is shown in Fig. 3.

Training Algorithm. We train DAA deep neural network as a supervised deep neural network. More specifically, DRN is trained as a supervised model to predict the true reward of taking an action. In an episode, we first randomize the slice set G. Then, for each step in an episode, the input states are fed to the model to generate actions then updated for the next step as described in Section §3-B. At the end of an episode, the reward values for each step are computed backward using equation (9).

In terms of training objective, we utilize *Mean Squared Error* function. Because the model only knows the reward of actions that it has taken, we utilize a *mask* vector $m^{(t)}$ of size l in which $m_i^{(t)} = 1$ if slice i is selected in $a^{(t)}$, and 0 otherwise. $m^{(t)}$ prevents non-selected slices from contributing to the loss value. The loss for one episode $\mathcal{E}_i$ is computed as

$$\mathcal{L}_i = \sum_{t \in \text{episode}} (\hat{P}^{(t)} * m^{(t)} - P^{(t)} * m^{(t)})^2 \tag{13}$$

where $\hat{P}^{(t)}$ is the predicted reward vector that is output by DRN, and $*$ represents an element-wise multiplication.

To improve performance of the training process, we further apply the Experience Relay strategy. More specifically, after being used, data (input states and output rewards) of an episode is stored in a memory dataset. After the memory has more than n_b episodes, with n_b being the training batch size, $(n_b - 1)$ among the generated episodes are randomly sampled and combine with the current episode to train DRN. Finally, the loss of the batch is averaged across its episodes.

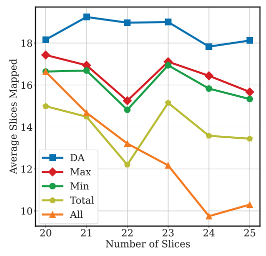
4 EXPERIMENT AND EVALUATION

We extensively test DAA in slicing a substrate network in different conditions from easy to more difficult. More specifically, we start from substrate networks of 100 nodes of which available resources are randomized in $[10, 30]$ and average at 20, and a slice set of 20 slices each of which consists of 10 VNFs with demands randomized in $[1, 19]$ and averaged at 10. As both the available resources and demands total at $2,000$, this is the case where resources are plenty. Then, we test DAA in use cases that resources are scarcer compared to the demands from VNFs in the slices that need accommodations:

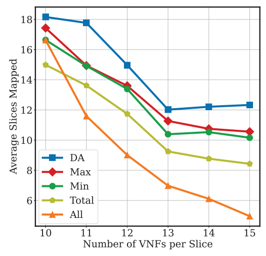
(1) Number of slices increases from 20 to 25
(2) Number of VNFs per slice increases from 10 to 15
(3) Amount of demanding resources from the VNFs increases, from averaging at 10 to averaging at 15
(4) Available resources in the substrate nodes decrease, from averaging at 20 to averaging at 15

After fine-tuning, the selected deep network architectures are as 1) all embedding blocks have three fully-connected layers; each layer in the substrate embedding block has a size of n, and each layer in the slice embedding block has a size of s. 2) the MHSA has five attention heads of each has a size of s. And 3) the fully-connected block that outputs return values has three layers, each has $3(n + s)$ neurons. The hyper-parameters α, β, and λ, for the reward function in Eq. (9) are set at 0.2, -1, and 0.9, respectively. Finally, the mini-batch size in all experiments, n_b is 256. In all cases, DRN is trained with ADAM optimization algorithm with learning rate of 10^{-6} for 500 epochs.

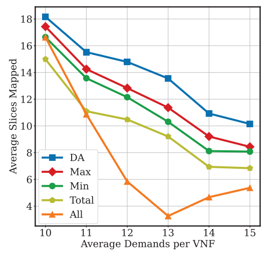
We implement four baseline strategies to compare with DAA. The first (All) performs VNF mapping purely based on their demands: all VNFs across all slices are sorted in descending order then allocated to substrate nodes using the algorithm in Section §3A. The second, third, and forth baselines sort the slices in descending order of their VNFs' max demands (Max), min demands (Min), and total demands (Total). Then, in the currently selected slice, the VNFs are embedded in descending order of their individual demands. We use the average number of accommodated slices across 100 testing episodes as the evaluation metric. The results of all experiments are illustrated in Fig. 4. As shown in Fig. 4, in all experiment settings, DAA yields an accommodation rate that is significantly higher than that of all the naive baselines. In Fig. 4(a) DAA maintains an accommodation rate of over 18 in all cases, whereas the other four's performances drop gradually. In Fig. 4(b) and (c), the decreasing patterns of all models are fairly similar (except for the All baseline); however, DAA still outperforms the others. In Fig. 4(d), we can see DAA has the lowest decreasing rate while still yielding the highest performances.



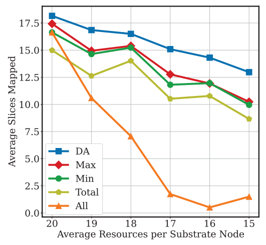
(a) Increasing slices



(b) Increasing VNFs per slice



(c) Increasing VNF demands



(d) Decreasing sub. resources

Figure 4: Models' performance on slicing networks with increasingly scarcer resources

5 CONCLUSION

Given a high demand in the applications, where UAV/drone communications can assist the cellular networks, RAN slicing plays a critical role in these applications. This paper addresses a fundamental challenge in RAN slicing: how to enhance the embedding performance of RAN slicing to support the space-air-ground integrated networks in terms of mapping VNFs while concurrently meeting the variation of resource demands and requirement of the RAN slices. In terms of architecture, we proposed a deep reinforcement scheme (DAA) that consists of two components, an empirically designed deep neural network that was used to observe the current input states to decide the schedule, and VNFs of the selected slice were then determined by an allocation algorithm that was aware of the substrate network's flexibility and the VNFs' allocability. We further trained the deep network of DAA as a deep reward network to predict reward the agent took after selecting and accommodating a slice. Experiment results show that, on average, DAA is able to maintain a fulfilling rate at above 80% in a resource-limited network, and about 60% in extreme conditions, i.e., insufficient resources at the substrate.

6 APPLICATIONS AND BEYOND

With the rapid growth of new services and Internet applications, especially massive Internet of Things such as smart grid, traditional cellular networks are now facing a major challenge of supporting applications with multitude of connections such as between smart devices and the controller in the smart grid or between UAV/drones and land vehicles in drone-assisted communications systems. The new paradigm proposed in this research offers promising new approaches for developing stable mapping plans for VNFs that are *central to any drone-mounted infrastructures and systems*, with enhanced capabilities to satisfy the increasing recoverability and stability demands of services and Internet applications, while meeting their service level objectives under both normal operations and abnormal operations with failures in the network. In addition, this research provides new *AI and ML based architectures* for dynamic operations in drone communications using network slicing technologies. The proposed work has the potential to impact and strengthen the development of different applications such as smart grid and UAV/drone-assisted communications.

REFERENCES

[1] P. R. et al., "Network slicing to enable scalability and flexibility in 5g mobile networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, 2017.

[2] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112–119, 2017.

[3] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5g network slicing resource utilization," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[4] H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. 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] H. Halabian, "Optimal distributed resource allocation in 5g virtualized networks," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 28–35.

[6] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on ran: Flexibility and resources abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

[7] M. R. Rahman and R. Boutaba, "Svne: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 105–118, 2013.

[8] S. e. a. D'Oro, "The slice is served: Enforcing radio access network slicing in virtualized 5g systems," *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2019.

[9] S. D'Oro, F. Restuccia, and T. Melodia, "Toward operator-to-waveform 5g radio access network slicing," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 18–23, 2020.

[10] M. Richart, J. Baliosian, J. Serrat, and J. Gorricho, "Resource slicing in virtual wireless networks: A survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 462–476, 2016.

[11] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frimpong, "Network resource management and qos in sdn-enabled 5g systems," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–7.

[12] Y. Jia, H. Tian, S. Fan, P. Zhao, and K. Zhao, "Bankruptcy game based resource allocation algorithm for 5g cloud-ran slicing," in *2018 IEEE Wireless Communications and Networking Conference*, 2018, pp. 1–6.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.