Efficient Embedding VNFs in 5G Network Slicing: 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 aims to logically split an infrastructure into a set of self-contained programmable RAN slices, with each slice built on top of the underlying physical RAN (substrate) is a separate logical mobile network, which delivers a set of services with similar characteristics. Each RAN slice is constituted by various virtual network functions (VNFs) distributed geographically in numerous substrate nodes. A key challenge in building a robust RAN slicing 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) 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 Virtual Network Functions - VNFs) within a given request set. More specifically, 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, drone-assisted wireless communications.

1 INTRODUCTION

The key aspect of RAN slicing is that the role of the mobile network operator(s) (MNOs) is to coordinate and allocate resources of the substrate network to ensure the harmonic coexistence of multiple RAN slices, while the role of the enterprise is to place slice requests and then manage the provided slices [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 MNO works with the RAN enforcement to allocate the substrate network resources (e.g., resource blocks in long-term evolution (LTE)) to VNFs and virtual links between VNFs, and provide an appropriate mapping plan, according to the proffered demand, RAN policy, connectivity, and available spectrum resources of the substrate nodes, also called resource blocks (RBs).

There have been efforts to design algorithms for the RAN resource allocation problem 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 RAN slicing-based applications.

With such motivation, in this paper, we present a deep reinforcement scheme that is called Deep Allocation Agent (DAA) that is able to embed VNFs of a slice while being aware of the interdependency among slices. We start with modeling the problem of maximizing the number of network slices to be fully accommodated as a reinforcement learning problem with the following settings: 1) Environment State includes 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.

DAA then solves the problem of using a combination of a deep neural network and an resource allocation algorithm. The neural network first observes the current environment
state to select a pending slice to accommodate. VNFs in 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 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. 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 and the VNFs’ allocability. The deep network of DAA is trained either as a deep reward network to maximize the number of accommodated slices. DAA is shown to obtain good slice-resolving rates even in resource-scarce substrate networks, and is quickly to converge during training.

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 NETWORK 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 \( G = \{ N, R \} \) in which we have a set of nodes \( N = \{ N_1, N_2, \ldots, N_n \} \) and a set of resources \( R = \{ R_1, R_2, \ldots, R_m \} \). A node \( N_i \) has maximum resources available of \( R_i \). In other words, any node \( N_i \) can allocate a maximum amount \( R_i \) of resources to VNFs. A VNF in a specific slice is not available and accessible by other network slices for the isolation purpose.

Let \( G = \{ G_1, G_2, \ldots, G_l \} \) be the set of \( l \) RAN slices of which resources need to be allocated in \( G \). Each slice \( G_i = \{ V_i, R_i \} \) consists of a VNF set \( V_i = \{ v_{i1}, v_{i2}, \ldots, v_{in} \} \) and a resource set \( R_i = \{ r_{i1}, r_{i2}, \ldots, r_{im} \} \) in which the demanded resources for VNF \( v_{ij} \) is \( r_{ij} \). In other words, \( v_{ij} \) needs to be assigned to a substrate node \( N_k \) with \( R_k \geq r_{ij} \) to be considered successfully mapped. To mathematicize the allocation of a VNF \( v_{ij} \) to a substrate node \( N_k \), we introduce a binary variable \( m_{ikj} \). \( m_{ikj} = 1 \) when \( v_{ij} \) is mapped to \( N_k \), and \( m_{ikj} = 0 \) otherwise. A VNF can only be allocated to one substrate node, therefore we have the constraint \( \sum_{k=1}^{n} m_{ikj} = 1 \). The maximum resources available in each substrate node cannot be exceeded when allocating VNFs, which leads to another constraint \( \sum_{i,j} m_{ikj} \cdot r_{ij} \leq R_k \). We further assume the slice \( G_i \) to be successfully allocated only when all \( v_{ij} \in V_i \) are successfully allocated. Consequently, we introduce a binary variable \( A_i \). \( A_i = 1 \) when \( \sum_{k=1}^{n} m_{ikj} = l_i \), 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 \( Q_{\alpha, S} \) – 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 \( G = \{ G_1, G_2, \ldots, G_l \} \) be the set of \( l \) RAN slices of which resources need to be allocated in \( G \). Our goal is to maximize \( Q_{\alpha, S} \) as follows:

\[
Q_{\alpha, S} = \max_{i \in \{1,2,\ldots,l\}} \sum_{l=1}^{n} A_i^l
\]

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 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 \( G \). This means that a complete allocation solution may not exist for a mapping from \( G \) to \( G \).

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. These two components make up the output of our \( Q_{\alpha, S} \) scheme. We shows an illustrative example in Fig. 1. Fig. 1(b) shows the substrate network \( G \) of five nodes,
Available resource in substrate node
Resources demanded in VNFs
Allocated resource in substrate node
VNF – Substrate node mapping

Figure 1: An example on a VNFs embedding problem and its non-optimal and optimal solutions

each with 4 resource blocks. There are two slices that need accommodations. Fig. 1(c) demonstrates the case in which slice 1 is accommodated first with the VNFs randomly mapped to substrate nodes that satisfy the resource constraint. This solution is not optimal as slice 2 cannot be accommodated anymore. Fig. 1(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 before allocating resources to them, as well as a mapping strategy that conserves resources based on future unallocated slices and VNFs.

3 DEEP ALLOCATION AGENT

In this section, we describe our deep reinforcement allocation scheme, henceforth referred to as Deep Allocation Agent (DAA). DAA utilizes a novel deep neural network to schedule slices and a VNF mapping algorithm to allocate resources to the selected RAN slice. The deep neural network is referred to as a deep reward network. Given a substrate network $G$ with the node set $N$ and the resource set $R$, and a set of RAN slices $G$, we refer to the accommodation of all slices $G^t$ in $G$ as an episode. At each step $t$, DAA makes an action $a^t$ to allocate resources to one slice $G^t$ based on the current environment state including the current available resources $R^t$ 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 Allocating Individual VNFs’ Resources

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 concept 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 $N_k$ with available resource $R_k^t$ at step $t$, and a RAN slice set $G$ with VNFs $\{v^i_j|i \in \{1, l\}; j \in \{1, s\}\}$, the flexibility $F(N_k)$ of $N_k$ is defined as

$$F(N_k) = \sum_{i=1, j=1}^{l} 1_{i,j}^k$$

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

$$1_{i,j}^k = \begin{cases} 1 & r^t_j \leq R_k \\ 0 & \text{otherwise} \end{cases}$$

The flexibility $F^S$ of the substrate network is then calculated as the sum of all substrate nodes’ flexibility:

$$F^S = \sum_{k=1}^{n} F(N_k)$$

**VNF Allocability.** We define the allocability of a VNF as the number of substrate nodes that can accommodate the VNF. Mathematically, the allocability $A(v^i_j)$ of a VNF $v^i_j$ is defined as

$$A(v^i_j) = \sum_{k=1}^{n} 1_{i,j}^k$$

with $1_{i,j}^k$ defined previously. It can be seen that the lower $A(v^i_j)$, the less substrate nodes can accommodate $v^i_j$, and when $A(v^i_j) = 0$, $v^i_j$, and subsequently $G^t$, 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 $t$

$$A^S(v^i_j) = \min_{i,j} A(v^i_j)$$

**VNF Allocation Algorithm.** Given a VNF that needs resources, a substrate node is selected so that, after the allocation, both the substrate network flexibility $F^S$ and the $A^S(v^i_j)$ of the current network are maximized. More specifically, at step $t$, given a VNF $v^i_j$ of demanding resource $r^t_j$ and the substrate nodes $N = \{N_1, N_2, \ldots, N_n\}$ with resources $R^t(i) = \{R^t_1(i), R^t_2(i), \ldots, R^t_n(i)\}$, the process is as follows.
(1) Temporarily allocate $v^i_j$ to each $N_k$ that satisfies $R_{R_k} \geq v^i_j$ and update $R^{(t)}$ to obtain $R_{R_k}^{(t+1)}$ with $R_{R_k}^{(t+1)} \leftarrow R_{R_k}^{(t)} - r^i_j$. Calculate the flexibility $F_{R_k}$ and allocability $A_{R_k}$.

(2) Sort the substrate nodes $N$ in descending order of $A_{R_k}$ then $F_{R_k}$ ($A_{R_k}$ is prioritized)

(3) Select the first substrate node $N_i$ in the sorted $N$ to be the mapping solution for $v^i_j$

This process ensures that $N_i$ 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 $E$, an input state to DAA consists of 1) the current available resource in each substrate node $R^{(t)}$ and 2) the current demanded resources from all slices $R^{(t)}$. The data representations for $R^{(t)}$ and $R^{(t)}$ are straightforward. $R^{(t)}$ is modeled as a vector

$$R^{(t)} = \{R_1^{(t)}, R_2^{(t)}, \ldots, R_n^{(t)}\}$$

whereas $R^{(t)}$ is modeled as a $1 \times s$ matrix

$$R^{(t)} = \begin{bmatrix} r_1^{(t)} & r_2^{(t)} & \ldots & r_s^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ r_1^{(t)} & r_2^{(t)} & \ldots & r_s^{(t)} \end{bmatrix}$$

If a slice $G^i$ is already accommodated, its corresponding row $R_i^{(t)}$ in $R^{(t)}$ is replaced with a 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 $G^i$ then mapping its VNFS to the substrate network. We utilize a hybrid approach, that is to use the deep network SN for scheduling the slices, and the algorithm presented previously to map the VNFS.

In terms of scheduling, SN outputs a reward vector $\rho^{(t)}$ of size $l$; $\rho^{(t)} = \{\rho_1^{(t)}, \rho_2^{(t)}, \ldots, \rho_l^{(t)}\}$ 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, $a^{(t)} = \text{argmax}_{i \in G^p} (\rho_i^{(t)})$, where $G^p$ 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 algorithm that is described in Section §3A. 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.

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, $R^{(t)}$ 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 $r^{(t)}$, then

$$r^{(t)} = n_r^{(t)} \alpha + (1 - n_r^{(t)}) \beta f + \lambda p^{(t+1)} (1 - f)$$

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 $\alpha$, $\beta$, and $\lambda$ 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 SN. First, each input component among $R^{(t)}$ 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 $R^{(t)}$ and $R^{(t)}$ be $Emb_{R^{(t)}}$ and $Emb_{R^{(t)}}$, respectively, then

$$Emb_{R^{(t)}}(R^{(t)}) = U^{(t)} \begin{bmatrix} R_1^{(t)} & R_2^{(t)} & \ldots & R_n^{(t)} \end{bmatrix}$$

$$Emb_{R^{(t)}}(R^{(t)}) = U^{(t)} \begin{bmatrix} u_{1,1}^{(t)} & u_{1,2}^{(t)} & \ldots & u_{1,e_R}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ u_{l,1}^{(t)} & u_{l,2}^{(t)} & \ldots & u_{l,e_R}^{(t)} \end{bmatrix}$$

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

![Figure 2: DAA deep neural network architecture](image-url)
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 rests 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

\[
S_R^{(t)} = \begin{bmatrix} \sigma_{1,1}^{(t)} & \ldots & \sigma_{1,L}^{(t)} \\ \ldots & \ldots & \ldots \\ \sigma_{L,1}^{(t)} & \ldots & \sigma_{L,L}^{(t)} \end{bmatrix}
\]

where \( \sigma_{i,j}^{(t)} \) is the context score of \( G_j \) with respect to slice \( G_i \). We repeat \( u_R^{(t)} \) \( l \) times, then concatenate them with the rows in \( U^{(t)} \) and \( S_R^{(t)} \) to form the embedding for each slice:

\[
U^{(t)} = \begin{bmatrix} u^{(t)}_{1,1} & \ldots & u^{(t)}_{1,n} & \ldots & u^{(t)}_{1,L} \\ \vdots & \ldots & \ldots & \ldots & \vdots \\ u^{(t)}_{L,1} & \ldots & u^{(t)}_{L,n} & \ldots & u^{(t)}_{L,L} \\ u^{(t)}_{1,1} & \sigma_{1,1}^{(t)} & \ldots & \sigma_{1,L}^{(t)} \\ \vdots & \ldots & \ldots & \ldots & \vdots \\ u^{(t)}_{L,1} & \sigma_{L,1}^{(t)} & \ldots & \sigma_{L,L}^{(t)} \end{bmatrix}
\]

Finally, \( U \) is input into a block of fully-connected layers that output a vector of \(|D| \) values \( r_1^{(t)}, r_2^{(t)}, \ldots, r_{|D|}^{(t)} \) that represent the reward if DAA select each slice. The architecture of DAA’s deep neural network is shown in Fig. 2.

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 \( m_i^{(t)} \) of size \( l \) in which \( m_i^{(i)} = 1 \) if slice \( i \) is selected in \( a_i^{(t)} \), and 0 otherwise. \( m_i^{(t)} \) prevents non-selected slices from contributing to the loss value. The loss for one episode \( \mathcal{L}_i \) is computed as

\[
\mathcal{L}_i = \sum_{t \in \text{episode}} (\hat{P}(i) \ast m_i^{(t)} - P(i) m_i^{(t)})^2
\]

where \( \hat{P} \) is the predicted reward vector that is output by DRN, and \( \ast \) 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 averages 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 averaged at 10 to averaged at 15
4. Available resources in the substrate nodes decrease, from 20 to 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 \( \alpha, \beta, \lambda \), for the reward function in Eq. (9) are set at 0.2, -1, and 0.9, respectively. The learning rate \( \lambda \) for training DRN as in Eq. (9) is 0.1. 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 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 are shown in Fig. 3. As shown in Fig. 3, in all experiment settings, DAA yields an accommodation rate that is significantly higher that than that of all the naive baselines. In Fig. 3(a) DAA maintains an accommodation rate of over 18 in all cases, whereas the other four’s performances drop gradually. In Fig. 3(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. 3(d), we can see DAA has the lowest decreasing rate while still yielding the highest performances.
This paper addresses a fundamental challenge in RAN slicing: how to enhance the embedding performance of RAN slicing in terms of mapping VNFs while concurrently meeting the variation of resource demands and requirement of the RAN slices. In terms of architecture, DAA 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, traditional cellular networks are now faced with a major challenge of supporting applications with multitude of connections such as between drones and land vehicles for smart mobility management. From a technical standpoint, 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 operation in drone communications using network slicing technologies. The proposed work is likely to have an immediate impact and will strengthen the development of applications with multitude of connections-assisted wireless communications for 5G and beyond.

ACKNOWLEDGEMENT

This research is supported in part by the US NSF grant CNS-2103405.

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] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Frimpong, “Network slicing based 5g and future mobile networks: Mobility, resource management, and challenges,” IEEE Communications Magazine, vol. 55, no. 8, pp. 138–145, 2017.
[3] 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.
[4] S. e. a. D’Oro, “The slice is served: Enforcing radio access network slicing in virtualized 5g networks,” in 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (INSM), 2019, pp. 28–35.
[5] A. Ksentini and N. Nekkaz, “Toward enforcing network slicing on ran: Flexibility and resources abstraction,” IEEE Communications Magazine, vol. 55, no. 6, pp. 102–108, 2017.
[6] 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.
[7] 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.
[8] S. e. a. D’Oro, S. Restuccia, and T. Melodia, “Toward operator-to-wavelength 5g radio access network slicing,” IEEE Communications Magazine, vol. 58, no. 4, pp. 18–23, 2020.
[9] 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.
[10] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frunzop, “Network resource management and qos in sdn-enabled 5g systems,” in 2015 IEEE Global Communications Conference (GLOBECOM), 2015, pp. 1–7.
[11] 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.
[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” Advances in neural information processing systems, vol. 30, 2017.