Slice Management in Radio Access Network via Deep Reinforcement Learning

Behnam Khodapanah*, Ahmad Awada†, Ingo Viering‡, Andre Noll Barreto§, Meryem Simsek¶, Gerhard Fettweis*
*Vodafone Chair Mobile Communications Systems, Technische Universität Dresden, Germany
Email:{behnam.khodapanah, gerhard.fettweis}@tu-dresden.de
†Nokia Bell Labs, Munich, Germany; Email: ahmad.awada@nokia-bell-labs.com
‡Nomor Research GmbH, Munich, Germany; Email: viering@nomor.de
§Barkhausen Institut, Dresden, Germany; Email: andre.nollbarreto@barkhauseninstitut.org
¶International Computer Science Institute, Berkeley, USA; Email: simsek@icsi.berkeley.edu

***Abstract*—In future 5G systems, it is envisioned that the physical resources of a single network will be dynamically shared between the virtual end-to-end networks called "slices" and the network is "sliced". The dynamic sharing of resources can bring about pooling gains, but different slices can easily influence each other. Focusing on slicing the radio access network, a slice management entity is required to steer the radio resource management (RRM) so that all of the slices are satisfied and negative inter-slice influences are minimized. The steering of RRM can be done by adjusting slice-specific control parameters in scheduler and admission controller mechanisms. We use a model-free reinforcement learning (RL) framework and train an agent as a slice manager. Simulation results show that such agents are capable of relatively quickly learning how to steer the RRM. Furthermore, a hybrid method of Jacobian-matrix approximation with RL approach has been devised and shown to be a practical and efficient solution.**

***Index Terms*—Network Slicing, 5G RAN Slicing, Radio Resource Management, Slice Orchestration, Reinforcement Learning**

I. Introduction

Fifth generation (5G) mobile networks are anticipated to have a big impact on society by connecting everyone and everything. Alongside the usual broadband users, it is expected that smart cities and industries' wireless connection requirements will be covered in 5G [1]. Therefore, these networks shall support a multitude of heterogeneous services, namely enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC) and massive Machine Type Communications (mMTC) [2]. Deploying multiple service-specific networks is not an efficient and financially feasible solution. Network slicing offers a flexible and scalable solution for accommodating these diverse services in a single physical network [3] [4].

The tenants of the single physical network specify their service requirements in terms of key performance indicators (KPIs) within a service level agreement (SLA) and the network operator instantiates the appropriate network slice to meet these SLAs [5]. At the same time, since the slices share the same physical infrastructure, they must be protected from each other such that dynamics of one slice do not adversely affect other slices [6].

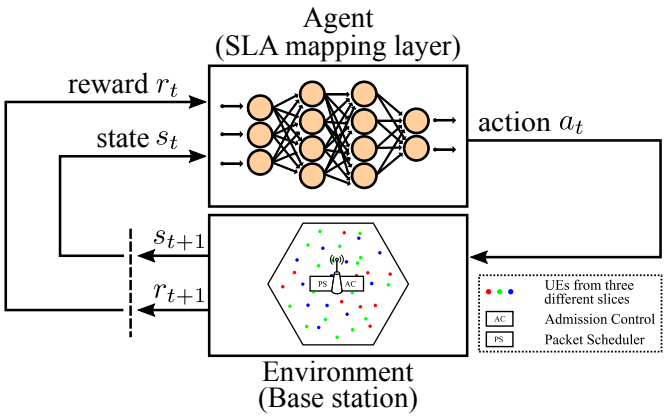


Fig. 1: Reinforcement learning in RAN slicing.

Since slices are end-to-end networks, the resources of both core network (CN) and radio access network (RAN) has to be sliced [5]. RAN slicing deals with the efficient sharing of the radio resources, e.g. spectrum resources in time and frequency. The radio resource management (RRM) mechanisms should simultaneously make sure that the resources are shared dynamically between the slices, while the slices are protected from negative influences among each other. In [7] we have presented an RRM framework, in which slice-specific control parameters for admission control and packet scheduler are defined and this approach has been shown to be able to steer the RRM. Tiling and puncturing are other mechanisms that can be used to steer the RRM in context of a sliced network [3] [8]. Since the entity that steers the RRM is trying to fulfill the SLAs of the slices, this entity is called "SLA mapping layer".

In our previous work [9] we have approached the problem by defining cost functions for KPIs and their respective targets and tried to minimize the cost. In [4], a heuristic based, reactive algorithm has been proposed. However, these approaches require accurate or approximate models of the inter-dependencies of slices in RAN, which might not be available in general. In this paper, we use deep reinforcement learning as a model-free algorithm, where no prior knowledge

TABLE I: Slice types

	Best Effort (BE)	Constant Bit-Rate (CBR)	Minimum Bit-Rate (MBR)
Description	The users belonging to this slice do not have any rigid requirements on their instantaneous throughput. However, the collective performance of the users are considered in the SLA. These users are always admitted, but also dropped if they linger more than the dropping threshold $\mathcal{T}_D$. Applications like web browsing can be considered as an example of this slice.	If the admission control has admitted an CBR user, regardless of the user's channel conditions,the constant throughput should be granted. Since the throughput is constant for all the users, the only KPI that is associated with this slice is the admission rate and is declared in the SLA. Voice-over-IP (VoIP) can be considered as a service which has similar requirement.	Similar to CBR users, if the admission control has admitted an MBR user, a minimum bit-rate has to be guaranteed for the MBR users. On the other hand, similar to BE users, the scheduler determines a share of unreserved resources to these users. Applications such as video streaming can be examples of this service since the video codecs require a minimum bit-rate to be able to stream with the lowest quality.
Input parameters	• Arrival rate λ_s • File size $\mathcal{F}_s$	• Arrival rate λ_s • Constant bit-rate $\bar{G}_s$ • File size $\mathcal{F}_s$	• Arrival rate λ_s • Minimum bit-rate $\bar{G}_s$ • File size $\mathcal{F}_s$
Control parameters	• Scheduler weights w_s	• Admission threshold th_s	• Scheduler weights w_s • Admission threshold th_s
Output KPIs	• Average throughput T_s • Fifth percentile throughput F_s • 1 - Dropping rate $1 - D_s$	• Admission rate A_s	• Average throughput T_s • Admission rate A_s
KPI targets	• $\bar{T}_s = 80$ Mbps • $\bar{F}_s = 3$ Mbps • $1 - \bar{D}_s = 97$ %	• $\bar{A}_s = 97.5$ %	• $\bar{T}_s = 90$ Mbps • $\bar{A}_s = 98$ %

about the RAN dynamics is required (Fig. 1). It should be noted that although reinforcement learning has been applied to the problem of resource management in [10] and [11], in their setup, the slices have similar targets and the only the resource allocation determines the achievement of the targets. In this work however, the focus has been on management of realistically diverse slices, in terms of different targets and different control mechanisms.

This paper is structured as follows. In Section II, we describe the system model of a sliced network with the presence of slices with different requirements and formulate the problem. Next, in Section III, we introduce the deep reinforcement learning (DRL) algorithm and show how to apply this to the SLA mapping layer. The evaluations can be found in Section IV, where the performance of the DRL in comparison with other algorithms is studied. Finally, in Section V we conclude the paper.

II. System Model and Problem Formulation

Let $\mathbb{S}$ be the set of all slices present in the network. The users of different slices arrive at a random time and location. The arrival process of the users of slice s is a Poisson-distributed random variable with an arrival rate λ_s [12]. Furthermore, the spatial distribution of the users is assumed to be uniform across the network and the imposed traffic load of slice s is determined by the arrival rate λ_s.

By definition, traffic models and KPIs from the users of different slices are different. Therefore, we have defined three exemplary slice types, which are summarized in Table I. Note that each of these slices have clearly defined KPIs as output parameters. To evaluate the performance, these KPIs are compared with the KPI targets defined in the SLA.

Section II-A and II-B describe the scheduling and admission processes, respectively. In Section II-C, the problem of the slice management in RAN in terms of the control parameters is explained.

A. Scheduling Process

To model the scheduling process in presence of different users of different slices, we first model the users' throughput. Based on Shannon's capacity formula, the throughput of user $i = 1, 2, \cdots, N_s$ from slice s is defined as

$$T_s^i = \omega_s^i \cdot B \cdot \log_2(1 + \psi_s^i), \tag{1}$$

where ω_s^i is the resource share of the user i in slice s, ψ_s^i is the average signal-to-interference-plus-noise-ratio (SINR) of user i and B is the total bandwidth.

For the CBR users, the throughput is constant and guaranteed and given in the SLA, i.e, $\bar{G}_s$. Consequently, the amount of resource share needed to fulfill the throughput for every user belonging to slice s in $\mathbb{S}_{\text{CBR}}$ is given by

$$\omega_s^i = \frac{\bar{G}_s}{B \cdot \log_2(1 + \psi_s^i)}. \tag{2}$$

The admitted CBR users take their share of resources first and collectively require

$$\Omega_{\text{CBR}} = \sum_{s \in \mathbb{S}_{\text{CBR}}} \sum_{i=1}^{N_s} \omega_s^i, \tag{3}$$

and the rest of the resources, i.e., $1-\Omega_{\text{CBR}}$, are shared between the MBR and BE users.

To model the scheduling of MBR and BE users, we propose a resource-fair scheduler with prioritization. A conventional resource-fair scheduler distributes the same amount of resources to each user. To enable prioritization of different slices, a weight vector is defined as $\mathbf{w} = [w_1, w_2, \cdots, w_{|\mathbb{S}_{\text{BE}} \cup \mathbb{S}_{\text{MBR}}|}]$, where $\mathbb{S}_{\text{BE}} \cup \mathbb{S}_{\text{MBR}}$ constitutes all the BE and MBR slices. The resource share of user $i = 1, 2, \cdots, N_s$ belonging to slice $s \in \mathbb{S}_{\text{BE}} \cup \mathbb{S}_{\text{MBR}}$ is defined as

$$\omega_s^i(\mathbf{w}) = \frac{w_s \cdot (1 - \Omega_{\text{CBR}})}{\sum\limits_{s' \in \mathbb{S}_{\text{BE}}} N_{s'} \cdot w_{s'} + \sum\limits_{s'' \in \mathbb{S}_{\text{MBR}}} N_{s''} \cdot w_{s''}}. \tag{4}$$

If we only use (4) for the MBR and BE users, there might be some MBR users that do not get enough resources to achieve their minimum throughput. To simultaneously use (4) and fulfill the MBR requirement, we propose an iterative scheduling. First, the resources are shared based on (4). If any of the MBR users has lower throughput than its minimum bit-rate, similar to (2), the minimum resources are determined and assigned to them. let $\breve{N}_{s''}$ be the number of users that have received this special treatment. The collective resource consumption of the users of these slices is

$$\breve{\Omega}_{\text{MBR}} = \sum_{s'' \in \mathbb{S}_{\text{MBR}}} \sum_{i=1}^{\breve{N}_{s''}} \omega_{s''}^i. \tag{5}$$

After this special treatment of some MBR users, the resource share of users of slices s in $\mathbb{S}_{\text{BE}} \cup \mathbb{S}_{\text{MBR}}$ is defined as

$$\omega_s^i(\mathbf{w}) = \frac{w_s \cdot (1 - \Omega_{\text{CBR}} - \breve{\Omega}_{\text{MBR}})}{\sum\limits_{s' \in \mathbb{S}_{\text{BE}}} N_{s'} \cdot w_{s'} + \sum\limits_{s'' \in \mathbb{S}_{\text{MBR}}} \hat{N}_{s''} \cdot w_{s''}}, \tag{6}$$

where $\hat{N}_{s''} = N_{s''} - \breve{N}_{s''}$ is the number of MBR users of slice s'' that have achieved the MBR only with the resources assigned to them by the PS. Note that after each iteration of the scheduler, using (6), there might be some MBR users whose resource share is not sufficient. Therefore, the iteration is repeated until all the MBR users are satisfied.

B. Admission Process

The role of admission control (AC) in the network is to regulate the incoming traffic. Tenants want the admission rate to be as high as possible. However, by admitting more users, the other KPIs of the network are affected, because the number of active users increase. This mechanism is especially crucial in sliced networks, since too many users from one slice might negatively impact the KPIs of the other slices. To implement an AC, we define slice-specific resource thresholds. For all of the MBR and CBR slices that are in set $\mathbb{S}_{\text{CBR}} \cup \mathbb{S}_{\text{MBR}}$, when a user from slice s appears, the admission policy is

$$\begin{cases} \text{If } \Omega_s \leq th_s & \text{grant admission} \\ \text{If } \Omega_s > th_s & \text{deny admission} \end{cases}, \tag{7}$$

where th_s is the resource threshold for slice s and $\Omega_s = \sum_{i=1}^{N_s} \bar{G}_s / B \cdot \log_2(1 + \psi_s^i)$ is the minimum amount of resources that is required to satisfy the MBR or CBR slice. In other words, if the new user increases the Ω_s above th_s, it is not admitted.

C. Problem Formulation and Previous Approaches

The aim of the SLA mapping layer is to steer the control parameters of the RRM mechanisms, so that the KPI targets defined in the SLA are achieved by the network. In our system model, the scheduler weights $w_s \ \forall s \in \mathbb{S}_{\text{BE}} \cup \mathbb{S}_{\text{MBR}}$ and admission control thresholds $th_s \ \forall s \in \mathbb{S}_{\text{CBR}} \cup \mathbb{S}_{\text{MBR}}$ are the control parameters to steer. One could define a function that maps the control parameters into the KPIs in the form of

$$\mathbf{y}_{\text{KPIs}} = f(\mathbf{x}_{\text{Controls}}). \tag{8}$$

A natural way to approach this problem is to solve an optimization problem, where the cost function is [9]

$$C = \left\| \mathbf{y}_{\text{KPIs}} - \mathbf{y}_{\text{Targets}} \right\|_2. \tag{9}$$

However, to solve this optimization problem, a model of the RAN is required, i.e., $f(\cdot)$ should be analytically described. Moreover, the KPIs of different nature and units create costs of different scales and summing them up as a single term in (9) implicitly and unfairly discriminates between KPIs.

Another approach is to use a coarse approximation of the Jacobian matrix of the (8) to iteratively react to violations of the SLA [4]. We require a reasonable approximation of the relationships between the X control parameters and K KPIs. This approximation can be represented in a $X \times K$ matrix defined as $\mathbf{J} = [j_{x,k}]$, where

$$j_{x,k} = \begin{cases} 0 & \text{increase in } x \text{ does not affect KPI } k \\ +1 & \text{increase in } x \text{ increases KPI } k \\ -1 & \text{increase in } x \text{ decreases KPI } k \end{cases}. \tag{10}$$

This matrix allows us to increase certain KPIs by increasing or decreasing the corresponding control parameters. Assuming that we have one instance of each slice type, one reasonable design for $\mathbf{J}$ matrix can be defined as

$$\mathbf{J} = \begin{array}{c} \begin{matrix} A_{\text{CBR}} & 1 - D_{\text{BE}} & T_{\text{BE}} & F_{\text{BE}} & T_{\text{MBR}} & A_{\text{MBR}} \end{matrix} \\ \begin{bmatrix} 0 & +1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 & -1 & +1 \end{bmatrix} \end{array} \begin{matrix} w_{\text{BE}} \\ th_{\text{CBR}} \\ th_{\text{MBR}} \end{matrix}. \tag{11}$$

For instance, if the T_{BE} is under its target, the w_{BE} should increase and th_{CBR} and th_{MBR} should decrease. To determine which KPI needs increasing, we define a $K \times 1$ violation vector $\mathbf{v} = H(\bar{\mathbf{y}} - \mathbf{y})$, where $H(\cdot)$ is the element-wise step function, i.e.,

$$v_k = H(\bar{y}_k - y_k) = \begin{cases} 1 & \text{if } y_k < \bar{y}_k \\ 0 & \text{if } y_k > \bar{y}_k \end{cases}, \tag{12}$$

where v_k, y_k and $\bar{y}_k$ are the kth KPI in the violation, output KPI and target KPI vectors, respectively. Using the violation vector $\mathbf{v}$, we know which KPIs are not satisfied and with the relationship matrix $\mathbf{J}$, we know which control parameters should be changed. Therefore, the update rule is defined as

$$\mathbf{x}' = \mathbf{x} + \delta \mathbf{J}\mathbf{v}, \tag{13}$$

where δ is the step size for the control parameter update. Here we have considered $\delta = 0.05$. The shortcoming of this approach is that the Jacobian matrix should be determined from heuristics beforehand, which might not be feasible. Moreover, inevitably, the inaccuracies of the approximation drive the control parameters to a space that might not be optimal. Furthermore, such algorithms require a reasonable starting point of the control parameters. The performance of this method is evaluated in Section IV. To avoid the drawbacks of the previous approaches, we formulate our problem in a reinforcement learning framework in Section III.

III. DRL BASED SLA MAPPING LAYER

In this section, first the concept of Reinforcement Learning (RL) is shortly introduced. Thereafter, the RL method is applied to the problem of controlling the slice-specific control parameters.

A. Deep Reinforcement Learning

RL is a subject of machine learning, in which an agent takes actions in an environment and receives rewards. The aim is to learn a policy for the agent such that the cumulative reward of the agent is maximized. To formulate the interactions between the environment and the agent, Markov Decision Process (MDP) is used, i.e., the agent at time step t selects an action a_t and the environment responds to these actions with moving to new states s_{t+1} and giving rewards r_{t+1} (Fig. 1). The action-value function is defined as

$$Q(s_t, a_t) = \mathbb{E}\{R|s_t, a_t\} = \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1}|s_t, a_t\}, \tag{14}$$

where R is the cumulative reward, r_t is the reward received at time step t and $\gamma \in [0, 1]$ is the discount factor. If the state space is discrete and finite, off-policy Q-learning with time difference (TD) update rule for $Q(s, a)$ is [13]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]. \tag{15}$$

After the training is finished, we choose the action that maximizes this function, so that we can collect maximum cumulative rewards.

In practice, however, the states can have high dimensions with continuous space. Therefore, the use of an approximate method that can handle large and continuous state dimensions is necessary. The deep Q-network (DQN) architecture uses artificial neural networks to approximate the $Q(s, a)$ function as $Q(s, a; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector containing all of the weights and biases in the neural network and parametrizes Q. $\boldsymbol{\theta}$ is the parameter that is updated during learning. At time step t, the DQN takes in the state s_t as input and outputs $Q(s_t, a; \boldsymbol{\theta}) \quad \forall a \in \mathbb{A}$, where $\mathbb{A}$ is the action space set. This means that the number of output nodes of the DQN is equal to number of possible actions. Each of the experiences between the agent and the environment can be summarized in a tuple $< s_t, a_t, s_{t+1}, r_{t+1} >$. According to [14], to avoid the correlation between the consecutive samples from the environment, we store the experiences in a replay buffer and in each training iteration a random mini-batch of the experiences is chosen for training the network. Furthermore, to avoid instabilities during training caused by a moving target in (15), it is suggested to create two neural networks, one which is updated every training step $Q(s, a; \boldsymbol{\theta})$ and another one, the target network $Q(s, a; \boldsymbol{\theta}^-)$, which is updated every τ steps [14] [15]. Besides, to avoid the influences of the replay buffer memory size on training, combined replay has been proposed in [16]. In this method, the most recent experience is added to the random mini-batch samples. The labels for training are

$$L(s_t, a) = \begin{cases} r_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \boldsymbol{\theta}); \boldsymbol{\theta}^-) & \text{If} \quad a = a_t \\ Q(s_t, a, \boldsymbol{\theta}) & \text{If} \quad a \neq a_t \end{cases}, \tag{16}$$

where $L(s_t, a)$ is the calculated label for all $a \in \mathbb{A}$ [15]. Algorithm 1 illustrate the process of training in more detail.

Initialize Q-Net. weights $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}_{\text{Random}}$;
Initialize Target Net. weights $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}_0$;
Populate replay buffer with minimum samples;
for $t \leftarrow 1$ **to** t_{max} **do**
  if $mod(t, \tau) = 0$ **then**
    Update Target Net. weights $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}_t$;
  end
  Sample a random number $\rho \leftarrow rand(0, 1)$;
  Based on greedy policy calculate $\epsilon(t)$;
  if $\rho < \epsilon(t)$ **then**
    Sample a random action $a_t \in \mathbb{A}$;
  else
    Q-Net. action $a_t = \arg\max_{a \in \mathbb{A}} Q(s_t, a; \boldsymbol{\theta}_t)$
  end
  Pass the a_t to the environment;
  Observe r_{t+1}, s_{t+1} from environment;
  Recent experience $\leftarrow < s_t, a_t, r_{t+1}, s_{t+1} >$;
  Add the recent experience to the replay buffer;
  Sample a mini-batch from the replay buffer;
  Augment the mini-batch with recent experience;
  Calculate the label according to (16);
  Train the network $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t+1}$;
end

Algorithm 1: DQN Training.

B. RL Application to SLA Mapping Layer

To apply the framework of RL to the problem of SLA mapping layer, we merely need to define the state, actions and rewards as follows

- *State*: Because we want the agent to be able to observe the environment in details, we define the states as the combination of the KPI reports and control parameters, i.e., at time step t the state vector is $s_t = [T^t_{\text{BE}}, F^t_{\text{BE}}, D^t_{\text{BE}}, A^t_{\text{CBR}}, T^t_{\text{MBR}}, A^t_{\text{MBR}}, w^t_{\text{BE}}, th^t_{\text{CBR}}, th^t_{\text{MBR}}]$.
- *Action*: Each control parameter is between 0 and 100 %. The action is to increase the control parameter by 5 %, decrease by 5 % or don't change it. Note that we further inhibit the control parameters to go above 95 % and below 5 %. Meaning that if the control parameter is at 95 %, increase action will not change the control parameter.
- *Reward*: for each KPI that is above its target, we assign +1 and -1 otherwise. Therefore, if K KPIs are being considered, the rewards are in range of $[-K, +K]$.

Note that for each of the control parameters, a separate agent is considered. This implies that for each agent, the other remaining agents are part of the environment and their actions are visible to the agent in the state vector.

IV. Evaluation

To evaluate the DQN algorithm, we first explain the experiment setup. The environment is a mobile network cell, in which the users from different slices appear, request a download of file-size $\mathcal{F}$. The users of CBR and MBR slices can be blocked and the users of BE slice can be dropped. We assume that there is one instance from each of the slice types. Therefore, the control parameters w_{BE} ($w_{\text{MBR}} = 1 - w_{\text{BE}}$), th_{CBR} and th_{MBR} are responsible for slice management in the RAN. Every $\mathcal{U}$ minutes the KPIs are compared with their corresponding target in the SLA and the rewards are calculated. Further details of the environment are given in Table II.

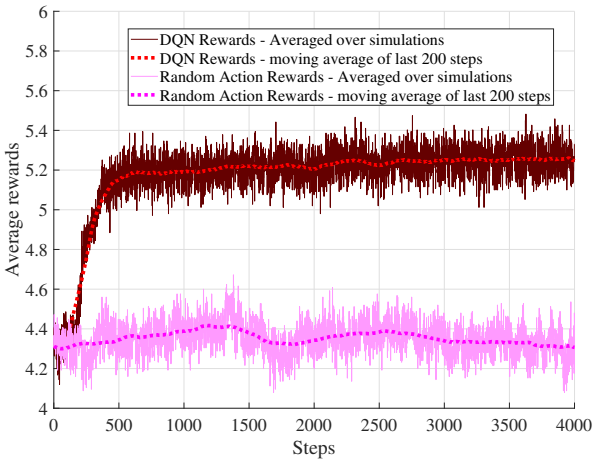
It is assumed that in each experiment the control parameters are initialized randomly. The maximum achieved rewards correspond to maximum possible fulfillment of the KPIs in the SLA. The list of tested algorithms are

1) *Random action*: at each time step the agent is taking random actions. This is a bad policy and determines the lower bound on the rewards.
2) *Genie agent*: the agent knows the best actions to drive the control parameters to the best possible states. The knowledge about the best states are derived via a brute force search.
3) *J-matrix*: the agent only reacts to the violations based on the approximate Jacobian matrix (Section II-C).
4) *DQN agent*: RL based agent (Section III)
5) *Hybrid Method*: to avoid lengthy random explorations required by the DQN, a hybrid of the J-matrix and the DQN is proposed. The J-matrix based increase/decrease actions are executed regardless of the DQNs decision. However, when the J-matrix is issuing no action, the DQN can execute its actions.

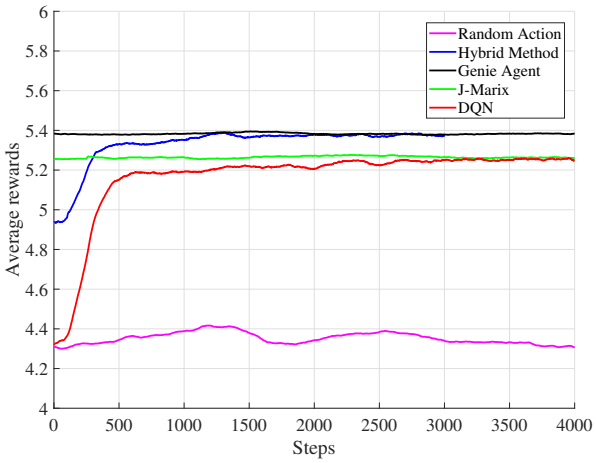
TABLE II: List of parameters

Environment parameters	
File size $\mathcal{F}$	16 Mb
Update interval $\mathcal{U}$	5 min
Drop time threshold $\mathcal{T}_D$	8 sec
Carrier frequency	2 GHz
Downlink transmit power	45 dBm
Noise power density	-174 dBm/Hz
Propagation model	Free-space path loss + Log-normal shadowing
Interference	Full interference from dummy cells
Total bandwidth	80 MHz
Cell radius	1 km
Shadowing std. dev.	8 dB
Antenna model	Isotropic
Slice Load λ_{BE}, λ_{CBR}, λ_{MBR}	10, 3, 4 users/s/cell
Guaranteed bit rate $\bar{G}_{\text{MBR}}$, $\bar{G}_{\text{CBR}}$	5, 5 Mbps
Agent parameters	
Neural network layer 1	Dense - 50 neurons
Neural network layer 2	Dense - 50 neurons
Neural network layer 3	Dense - 100 neurons
Activation function for all layers	Rectified Linear Unit (ReLU)
Batch size	32 samples
Initial replay buffer	160 samples
Target Net. update period τ	50 steps
Discount factor γ	0.9
Initial exploration rate	90 %
Final exploration rate	1 %

Fig. 2a illustrates the average reward collected over 250 simulations over 4000 steps for the DQN and the random action algorithms. For more clarity of the results, the running average over time with window size of 200 steps is drawn. Fig. 2b shows the running average of the rewards collected by different algorithms. The upper bound is found by brute force search where all the possible combinations of the control parameter, which we call "genie agent". The lower bound of the performance is found by following a complete random policy. Next, moving to the performance of the J-matrix, we see that the performance is constant. This is because this algorithm rarely changes the control parameter unless there is a violation. On the other hand the DQN starts with a worse performance, but after about 500 steps the performance is approximately the same as the J-matrix. Another observation is that the DQN has not found the global optimum yet and is stuck in a local minimum. The reason for this is simply because the exploration has not been enough for this agent. Furthermore, we observe that the hybrid method starts with a



(a) Average rewards over simulations and running average rewards.



(b) Running average rewards.

Fig. 2: Comparison of rewards collected by different algorithms.

worse performance than that of the J-matrix. This is because the exploration of the state-actions at the start degrades the performance, but after sufficient exploration, the DQN can start exploiting, i.e., drive the control parameters to some space that maximizes the cumulative rewards. Although the hybrid algorithm has the best performance, we should note that the J-matrix inhibits the exploration of the whole state space. This is because in some states, the SLAs are violated and the J-matrix quickly reacts to it and implicitly inhibits the exploration of such bad states. Therefore, one cannot simply remove the J-matrix and continue with the trained network, because to the eyes of the agent the J-matrix is part of the environment.

V. Conclusion

In this paper we have shown the application of reinforcement learning to the problem of slice-management in RAN. It has been shown that the RL agent can relatively quickly find a policy that has the same performance as the J-matrix, which is a carefully constructed algorithm based on heuristics. Furthermore, a hybrid algorithm has been introduces to avoid the initial random explorations of the DQN. This algorithm provides a combination of heuristics and learning algorithms, which is safe for initialization and further explorations of the DQN promise convergence of the control parameters where the maximal cumulative rewards can be assured.

Acknowledgments

We would like to thank the Center for Information Services and High Performance Computing (ZIH) at TU Dresden for generous allocations of computer time.

References

[1] S. E. Elayoubi *et al.*, "5G RAN Slicing for Verticals: Enablers and Challenges," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 28–34, January 2019.

[2] NGMN Alliance, "NGMN 5G White Paper," Tech. Rep., Feb. 2015.

[3] P. Rost *et al.*, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[4] B. Khodapanah *et al.*, "Slice management in radio access network via iterative adaptation," in *2019 IEEE International Conference on Communications (ICC): Communication QoS, Reliability and Modeling Symposium (IEEE ICC'19 - CQRM Symposium)*, Shanghai, P.R. China, May 2019.

[5] NGMN Alliance, "Description of Network Slicing Concept," Tech. Rep., Jan. 2016.

[6] I. da Silva *et al.*, "Impact of network slicing on 5G Radio Access Networks," in *2016 European Conference on Networks and Communications (EuCNC)*, June 2016, pp. 153–157.

[7] B. Khodapanah *et al.*, "Radio resource management in context of network slicing: What is missing in existing mechanisms?" in *2019 IEEE Wireless Communications and Networking Conference (WCNC) (IEEE WCNC 2019)*, Marrakech, Morocco, Apr. 2019.

[8] K. I. Pedersen *et al.*, "A flexible 5G frame structure design for frequency-division duplex cases," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 53–59, March 2016.

[9] B. Khodapanah *et al.*, "Fulfillment of Service Level Agreements via Slice-Aware Radio Resource Management in 5G Networks," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, June 2018, pp. 1–6.

[10] R. Li *et al.*, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74 429–74 441, 2018.

[11] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang, "Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks," *IEEE Access*, vol. 7, pp. 45 758–45 772, 2019.

[12] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Further advancements for E-UTRA physical layer aspects," 3rd Generation Partnership Project (3GPP), TR 23.203, Mar. 2017.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998. [Online]. Available: http://www.worldcat.org/oclc/37293240

[14] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature Publishing Group, a division of Macmillan Publishers Limited.*, 2015. [Online]. Available: https://www.nature.com/articles/nature14236

[15] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," 2015.

[16] S. Zhang and R. S. Sutton, "A Deeper Look at Experience Replay," *CoRR*, vol. abs/1712.01275, 2017. [Online]. Available: http://arxiv.org/abs/1712.01275