A Multi-Agent Reinforcement Learning Approach for Capacity Sharing in Multi-Tenant Scenarios

Irene Vilà, Jordi Pérez-Romero, Oriol Sallent, and Anna Umbert

***Abstract*—5G is envisioned to simultaneously provide diverse service types with heterogeneous needs under very different application scenarios and business models. Therefore, network slicing is included as a key feature of the 5G architecture to allow sharing a common infrastructure among different tenants, such as mobile communication providers, vertical market players, etc. In order to provide the Radio Access Network (RAN) with network slicing capabilities, mechanisms that efficiently distribute the available capacity among the different tenants while satisfying their needs are required. For this purpose, this paper proposes a multi-agent reinforcement learning approach for RAN capacity sharing. It makes use of the Deep Q-Network algorithm in a way that each agent is associated to a different tenant and learns the capacity to be provided to this tenant in each cell while ensuring that the service level agreements are satisfied and that the available radio resources are efficiently used. The consideration of multiple agents contributes to a better scalability and higher learning speed in comparison to single-agent approaches. In this respect, results show that the policy learnt by the agent of one tenant can be generalised and directly applied by other agents, thus reducing the complexity of the training and making the proposed solution easily scalable, e.g., to add new tenants in the system. The proposed approach is well aligned with the on-going 3GPP standardization work and guidelines for the parametrization of the solution are provided, thus enforcing its practical applicability.**

***Index Terms*—RAN slicing, capacity sharing, multi-agent reinforcement learning, Deep Q-Network.**

I. Introduction

ONE of the main features of the 5G architecture is network slicing, which allows the creation of multiple end-to-end logical networks (i.e., network slices) on top of the same physical infrastructure, so that each slice can be optimised to the requirements (e.g., data rates, latency, availability and reliability) of specific service and application domains (e.g., public safety, industrial, corporate). Each network slice can then be allocated to a different tenant (e.g., a communication provider, a mobile virtual network operator (MVNO), a vertical industry player), who can use it to provide services to its own users [1], [2]. The network slice allocated to each tenant includes a 5G core subnet instance and a Radio Access Network (RAN) subnet instance, denoted as RAN slice.

The deployment of RAN slices on a Next Generation (NG)-RAN infrastructure needs to deal with the management of the common pool of radio resources available in the existing cells in order to provide multiple and diverse RAN behaviours and, at the same time, to fulfil the requirements of the different services [3]. This management needs to consider that the traffic requirements of a RAN slice vary with time and can be different in each cell. To deal with all these variations, the amount of radio resources allocated to each RAN slice in each cell needs to be dynamically modified through capacity sharing mechanisms that ensure both the fulfilment of the RAN slice requirements and an efficient use of the available radio resources. This dynamic capacity sharing is the main problem addressed by this paper.

A. Related Work

The problem of capacity sharing in RAN slicing scenarios has been addressed by some prior works using different techniques and under different assumptions. The capacity sharing from a single cell perspective has been considered in [4]–[8]. Specifically, the problem is addressed in [4] by defining an exponential smoothing model, while in [5] it is formulated as an optimisation problem based on Karush Kuhn Tucker (KKT) conditions, and in [6] a biconvex problem is solved considering jointly the radio resources, caching and backhaul capacities in the RAN slicing process. Moreover, [7] and [8] establish the capacity provided to each tenant in a cell based on market-oriented models that aim at maximising the infrastructure provider's revenue. Other solutions address the problem of capacity sharing in multi-cell scenarios by using heuristic approaches [9]–[12]. In particular, capacity sharing is modelled in [9] as a winner bid problem solved by means of dynamic programming, while [10] uses an integer mathematical program and proposes a low complexity heuristic algorithm for associating resources to users. In turn, [11] proposes a fisher market game and [12] an iterative algorithm that adjusts the per-cell capacity provided to each tenant to be used for admission control.

Given the complexity of 5G networks and the inherent dynamic uncertainty of the wireless environment, Reinforcement Learning (RL) methods are potential candidates to deal with the capacity sharing problem, as they allow optimising dynamic

Manuscript received November 16, 2020; revised March 3, 2021, April 30, 2021, and June 23, 2021; accepted July 16, 2021. Date of publication July 27, 2021; date of current version September 17, 2021. This work was supported in part by the Spanish Research Council and FEDER funds under SONAR 5G Grant ref. TEC2017-82651-R, in part by the European Commission's Horizon 2020 5G-CLARITY project under Grant 871428, and in part by the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia under Grant 2020FI_B2 00075. The review of this article was coordinated by Dr. Feng Ye. *(Corresponding author: Irene Vilà.)*

The authors are with the Department of Signal Theory and Communications, Universitat Politècnica de Catalunya (UPC), 08034 Barcelona, Spain (e-mail: irene.vila.munoz@upc.edu; jordi.perez-romero@upc.edu; sallent@tsc.upc.edu; anna.umbert@upc.edu).

Digital Object Identifier 10.1109/TVT.2021.3099557

decision-making problems in real time [13]. Among these methods, Deep Reinforcement Learning (DRL) approaches, which combine deep neural networks (DNNs) with RL, are particularly promising due to their capability to support large state and action spaces. The success of DRL started with the Deep Q-Network (DQN) [14] that combines DNN with Q-learning and since its launch successive extensions have been proposed such as Double DQN (DDQN) [15] and prioritized experience replay [16], among others, which mainly aim at enhancing the speed of learning and the stability of DQN but at the cost of increasing the complexity of the solution and the number of hyperparameters to configure [17]. Moreover, going beyond Q-learning, DNN have also been applied to other RL approaches, like the policy gradient-based methods used for continuous action spaces, such as the Deep Deterministic Policy Gradients (DDPG) algorithm [18], or the actor-critic algorithms, such as the Asynchronous Advantage Actor Critic (A3C) algorithm [19].

DRL methods have already been used to approach the capacity sharing problem in multi-cell RAN slicing scenarios in some previous works [20]–[30]. In [20] and [21] the aggregated capacity reserved to each slice at network level is provided, respectively, by means of DQN and Deterministic Policy Gradients (DPG) combined with K-Nearest Neighbours (K-NN). A similar problem is addressed in [22] through Generative adversarial network (GAN)-DDQN. In turn, [23] and [24] provide the cell capacity share by first computing the aggregate capacity reserved to each slice at network level using a DQN agent associated to each slice and, then, applying a heuristic algorithm to obtain the cell capacity for each slice. Similarly, [25] provides the aggregate capacity allocated to each tenant at network level by means of a DQN agent and, then, the cell capacity for each tenant is computed by a knapsack algorithm. In [26], a DQN agent is used to assign the cell capacity to all tenants in a single cell with virtualised Device to Device (D2D) communications. In [27] the RAN slice configuration and PRB allocation in a cell is performed by a two-level hierarchy scheme by using, respectively, DDPG and DDQN agents. The allocation of the cell capacity to all tenants in a single cell when considering DQN, DDQN and DDPG-based agents is assessed in [28]. Moreover, [29] and [30] propose an Ape-X-based approach to solve the capacity sharing problem, where the capacity provided to each slice in a single cell is provided by a different actor and all the actors base their policy on a common learner.

While the DRL-based capacity sharing approaches in [20]–[22] and [26]–[28] propose a single-agent to provide the joint solution for all the tenants, the approaches in [23]–[25], [29] and [30] tackle the problem as a Multi-Agent Reinforcement Learning (MARL) approach [31]. In MARL, multiple agents interact with a common environment in order to learn each agent's policy according to RL methods. The use of MARL to tackle the capacity sharing problem by associating one agent to each tenant exhibits relevant advantages with respect to a single-agent approach that allocates the capacity to all the tenants. On the one hand, it is more scalable as it allows easily adding/removing new tenants in the scenario simply by adding/removing the corresponding agent and without modifying the structure of the DNN, which in the single-agent case would depend on the number of tenants. On the other hand, it increases the speed of learning, since the dimensions of the state-action spaces are more reduced as the agent only needs to account for the states and actions related to the tenant and the training of the different agents can be performed independently of the others. The benefits of MARL have been exploited in [23]–[25], where each agent determines the aggregated assigned capacity over all the cells to a tenant, as well as in [29] and [30], where each agent determines the capacity assigned to a tenant in a single cell. However, none of the previous works have proposed a MARL capacity sharing solution where an agent is able to directly provide the capacities assigned to its associated tenant in the different cells in a multi-cell scenario. This is relevant for the management of RAN slices in multi-cell scenarios from a system-level perspective because, on the one hand, Service Level Agreements (SLA) are defined for geographical areas covering multiple cells and, on the other hand, the traffic of one tenant across the different cells may exhibit time and space heterogeneities. Therefore, it is important that an agent learns how to make joint decisions for multiple cells and this paper intends to fill this gap. Besides, another gap to be filled is that the previous MARL-based approaches in [23]- [25] provide limited insights about multi-agent design features that are relevant from a practical perspective, such as the interaction between the different agents or the training-related aspects of the multi-agent approach. Only the work in [29] and [30] provides some details in this respect for the Ape-X solution, but being restricted to the operation on a single cell basis.

B. Main Contributions

In this paper, a MARL capacity sharing solution for multi-tenant and multi-cell scenarios is proposed, where the capacities to be provided to each tenant in the different cells are obtained by associating one DQN agent to each tenant. The main contributions and novelties with respect to previous works are summarized in the following:

- In the proposed MARL approach a DQN agent learns the policy for jointly assigning the capacities to be provided to a tenant in the different cells of the scenario. This is a difference with respect to prior multi-agent approaches in which an agent either assigns the capacity for a single cell, [29] and [30], or assigns the aggregate capacity for all cells, [23]–[25]. Furthermore, a synchronous and cooperative operation of the different agents is considered in the proposed solution since their decisions are performed at the same time and each agent is designed to find a solution that benefits jointly all the tenants. This also constitutes a relevant difference with respect to other approaches, such as [23] and [24].
- The proposed MARL approach addresses the capacity sharing when considering the SLA for each tenant as an aggregate across the multiple cells in the scenario in order to capture the total amount of capacity to be provided to each tenant. Instead, other approaches such as [20]–[30] just consider the SLA specified in terms of the QoS parameters defined at user level, but without enforcing any

aggregate capacity per tenant. In this way, the definition of the state in the MARL solution proposed here includes the parameters of the SLA, which allows the agents to adapt to different SLA requirements without the need of performing a new training when the SLA values change, as required in [23]–[26].

- A key feature of the proposed solution is that the policy learning process can be conducted by a single agent and then the learnt policy can be generalised and directly applied by the other agents, thus reducing the complexity of training in multi-agent scenarios. This leads to a scalable solution where e.g., the addition of a new tenant does not require any re-training. To the authors' best knowledge, little effort has been conducted by previous works on capacity sharing to address these training-related aspects in detail. In this respect, [29] and [30] use Ape-X for learning a single policy that jointly uses the experiences of multiple tenants. Instead, the general policy considered in the present paper can be learnt based on the experiences of a single tenant, thus simplifying the learning process.
- The practicality of the proposed approach is enforced with respect to previous capacity sharing solutions by formulating the solution as a Self-Organising Network (SON) function, which is integrated in a RAN slicing management framework well aligned with the on-going 3GPP standardisation work on management and orchestration of network slicing [32]–[34].

A first approach of the proposed model was firstly introduced in our recent conference paper [35]. This paper substantially extends this previous work. First, it provides a complete and detailed description of the algorithm, which has been reformulated with a new definition of the state that facilitates the capability of generalising the learnt policies and a new definition of the reward to better capture the SLA fulfilment and resource utilisation targets. Moreover, the solution has been aligned with 3GPP standardisation by reformulating it as a SON function. Finally, this paper provides an extensive set of results to demonstrate the generalisation capability of the learnt policies, to analyse the impact of the configuration parameters of the MARL approach and to assess the performance in relation to the optimum solution.

The rest of the paper is organised as follows. Section II presents the system model and formulates the problem of capacity sharing in a multi-cell scenario and Section III describes the proposed multi-agent DQN approach. Based on this, Section IV describes the considered scenario for evaluation, assesses the capability of the agents in the MARL solution to learn a general policy applicable by other tenants and discusses the scalability of the solution when adding a new tenant in the scenario. An analysis of the impact of different model parameter configurations on the achieved performance and an analysis of the optimality of the solution closes this section. Finally, Section V summarises the conclusions and future work.

II. System Model and Problem Definition

Let us consider an Infrastructure Provider (InP) that owns a NG-RAN infrastructure, which is composed of N cells with diverse deployment characteristics (i.e., cell radius, transmission power, frequency of operation). Assuming 5G New Radio (NR) technology, each cell n has a total of W_n Physical Resource Blocks (PRBs) with a PRB bandwidth B_n, which provide a total cell capacity c_n (b/s), defined as $c_n = W_n \cdot B_n \cdot S_n$, where S_n is the average spectral efficiency at cell n. Then, the total system capacity C is obtained by aggregating c_n for all the cells $n = 1 \ldots N$. The InP shares its NG-RAN infrastructure among K tenants by providing each tenant k with a RAN Slice Instance (RSI), which corresponds to the Network Slice Subnet Instance (NSSI) for the RAN part in the 3GPP terminology.

In order to satisfy the service requirements, a SLA is established between the InP and each of the tenants. Based on this SLA, the following requirements are established for the k-th tenant:

- Scenario Aggregated Guaranteed Bit Rate ($SAGBR_k$): the aggregated capacity to be provided across all cells to tenant k, if requested.
- Maximum Cell Bit Rate ($MCBR_{k,n}$): Maximum bit rate that can be provided to tenant k in cell n. This parameter is defined by the InP to avoid that a single tenant uses all the capacity in a cell under highly extreme heterogeneous spatial load distributions of tenants demanding excessive capacity in certain cells.

These requirements are considered in the creation process of each RSI and its fulfilment is responsibility of the InP. For this purpose, it relies on the RSI Lifecycle Management (LCM) for creating, modifying, optimising and terminating RSIs. According to the 3GPP management model [32], the RSI LCM falls under the scope of the Network Slice Subnet Management Function (NSSMF) for the RAN part of a network slice, denoted hereafter as RAN NSSMF.

The RAN NSSMF is in charge of the correct operation and dynamic configuration of the RSIs. In particular, this paper considers the capacity sharing function as part of the RAN NSSMF to optimise the amount of radio resources (and associated capacity) to be provided to each RSI, which is a challenging task. This provided capacity needs to be dynamically updated depending on the RSI traffic demands, which are not homogeneous among the different cells in the NG-RAN and they can fluctuate over the time. Therefore, the capacity sharing solution considered in this paper is designed as a SON function that automatically and dynamically adjusts the capacity provided to each RAN slice across the different cells. Indeed, this vision is aligned with the recent study conducted at 3GPP on the SON functionalities for 5G [33]. This study identifies the so-called cross-slice network resource optimization use case intended to optimise the allocation of physical and virtual resources across multiple network slice instances. Therefore, the proposed approach in this paper can be regarded as a specific solution to this use case for the RAN.

Specifically, the proposed capacity sharing SON function dynamically tunes the capacity share for each tenant in time steps of duration Δt (in the order of minutes) in order to adapt to the spatial and temporal traffic variations among the different cells, minimise SLA breaches (i.e., violations) in the system and optimise the resource utilisation of the different cells in the

system. The capacity share $\boldsymbol{\sigma_k(t)}$ of tenant k in time step t is defined as $\boldsymbol{\sigma_k(t)} = [\ \sigma_{k,1}(t), \ldots, \sigma_{k,n}(t), \ \ldots, \sigma_{k,N}(t)]$, where each component $\sigma_{k,n}(t)$ corresponds to the proportion of the total capacity (i.e., proportion of the total PRBs W_n) in cell n provided to that tenant during time step t and ranges $0 \leq \sigma_{k,n}(t) \leq MCBR_{k,n}/c_n$. Note that the capacity share solution in a cell cannot exceed the total capacity of the cell, so that $\sum_{k=1}^{K} \sigma_{k,n}(t) \leq 1$. The capacity share $\boldsymbol{\sigma_k(t)}$ needs to be upgraded periodically on a per-minutes basis in order to adapt to the traffic demands. Following the current 5G Network Resource Model (NRM) defined by 3GPP [32] the capacity share can be configured on a cell using the RRMPolicyRatio attribute, which specifies the percentage of radio resources (e.g., PRBs) to be allocated on a per-slice basis. The basic abbreviations and notations of the proposed capacity sharing SON function are summarized in Table I.

III. Multi-Agent DQN Approach

The capacity sharing SON function has been addressed as a MARL approach in order to deal with the complexity of the computation of $\boldsymbol{\sigma_k(t)}$ in multi-cell scenarios. In the proposed approach, each RL agent is associated to a tenant k in the system and centrally learns the policy π_k to tune $\boldsymbol{\sigma_k(t)}$ dynamically by interacting with the environment. The selected RL method for deriving the policy π_k at the k-th agent is the DQN algorithm due to three main reasons. First, DQN has been designed to support high dimension state and action spaces, which is achieved with the use of DNN. This is convenient for the capacity sharing problem since the consideration of multiple cells and the randomness in the traffic demands in each of the cells can result in large state and action spaces. Second, the learning in DQN is performed by bootstrapping, i.e., the policy is progressively updated by considering single samples of experience instead of considering all the samples until reaching a certain goal or finishing an episode, like in Monte Carlo simulations [36]. This is suitable for the case of capacity sharing since a continuous learning of π_k is desired, rather than in episodes. Third, in relation to other DRL approaches such as DDQN or DDPG, preliminary results in a previous study [28] showed that, despite having differences in terms of the practicality of the implementation (e.g., speed of the training process, number of hyperparameters to configure, etc.), their performance in a single-agent and single-cell scenario was very similar to that of DQN. In this respect, DQN was considered as a good design choice, considering the trade-off between practicality and achieved performance.

The scheme of the proposed solution is shown in Fig. 1. The proposed capacity sharing SON function falls within the scope of the RAN NSSMF of the 3GPP management system. The function is composed of K DQN agents, each one associated to one tenant. The InP provides as input the service profile parameters associated to each tenant, which include the SLA parameters. Moreover, the SON function includes a monitoring module, which collects performance measurements of the tenants in the different cells and provides them to the processing module. The performance measurements of each tenant can

TABLE I
List of Abbreviations and Notations

Abbreviation/ Notation	Definition
NG-RAN	Next-Generation- Radio Access Network.
MARL	Multi-Agent Reinforcement Learning.
DNN	Deep Neural Network.
DQN	Deep Q-Network.
SLA	Service Level Agreement.
SON	Self-Organising Network.
InP	Infrastructure Provider.
N	Total number of cells.
PRB	Physical Resource Block.
W_n	Total number of PRBs.
B_n	PRBs' bandwidth.
c_n	Total cell capacity.
S_n	Average spectral efficiency.
C	System capacity.
K	Total number of tenants.
$SAGBR_k$	Scenario Aggregated Guaranteed Bit Rate.
$MCBR_{k,n}$	Maximum Cell Bit Rate.
RSI	RAN Slice Instance.
NSSMF	Network Slice Subnet Management Function.
Δt	Duration of a time step.
$\boldsymbol{\sigma_k(t)}$	Capacity share of tenant k in time step t.
$\sigma_{k,n}(t)$	Proportion of c_n of cell n provided to tenant k during time step t.
π_k	Policy learnt by the agent associated to tenant k.
$o_{k,n}(t)$	Offered load of tenant k in cell n.
$\rho_{k,n}(t)$	Resource usage of tenant k in cell n.
$T_k(t)$	Aggregated throughput of tenant k across all cells.
$\boldsymbol{s_k(t)}$	State of tenant k in time step t.
$s_{k,n}(t)$	State of tenant k in cell n in time step t.
$\boldsymbol{a_k(t)}$	Action selected for tenant k in time step t.
$a_{k,n}(t)$	Action selected for tenant k in cell n in time step t.
Δ	Action increase step.
$r_k(t)$	Reward obtained by tenant k in time step t.
$\rho_n^A(t)$	Fraction of available PRBs not used by any tenant in the cell.
$\sigma_n^A(t)$	Available capacity share in cell n not assigned to any tenant.
$\delta_k^{(1)}(t), \delta_k^{(2)}(t)$	Reward factors.
φ_1, φ_2	Reward factors' weights.
$O_k(t)$	Aggregated offered load of tenant k across all cells.
$O(t)$	Aggregated offered load of all tenants in all cells.
$\beta_k(t)$	Guaranteed capacity not required by other tenants at time step t.
γ	Discount factor.
$Q_k(\boldsymbol{s_k}, \boldsymbol{a_k}, \theta_k)$	Q-network associated to tenant k with weights θ_k.
D_k^l	Experience dataset of agent associated to tenant k with length l.
$U(D_k^l)$	Mini-batch of experiences.
π_k^ε	ε-Greedy policy with probability of selecting a random action ε.
$L(\theta_k)$	Average mean squared error loss.
τ	Learning rate.
$e_{k,j}$	Experience j of tenant k.
$A_k(t)$	Assigned capacity to tenant k at time step t.
R_k	Average reward of tenant k.
SS_k	Average SLA satisfaction of tenant k.
U	Average system utilization.

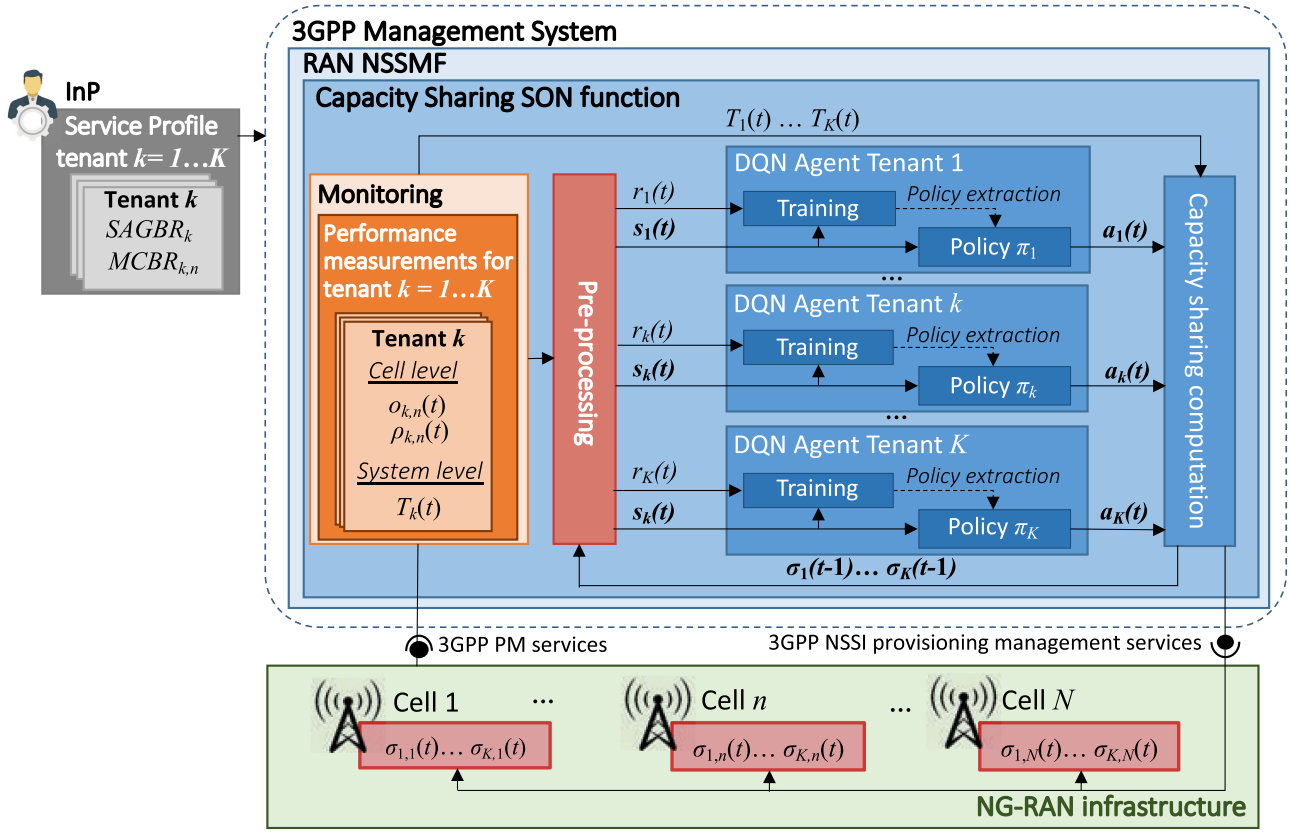

Fig. 1. MARL solution scheme.

be obtained in each time step e.g., using the NSSI performance data file reporting service defined in [34] through Performance Management (PM) services interface. The performance measurements considered here for tenant k include:

- Offered load ($o_{k,n}(t)$): Requested capacity by the tenant in the n-th cell during the last time step, i.e., period (t-Δt,t]. This parameter is obtained by aggregating the capacity requirement by all the users in cell n that belong to tenant k.
- Resource usage ($\rho_{k,n}(t)$): Fraction of PRBs occupied by the k-th tenant in the n-th cell during the last time step. It is computed as $\min(\sigma_{k,n}(t), o_{k,n}(t)/c_n)$.
- Throughput ($T_k(t)$): Aggregated throughput experienced by tenant k across all cells during the last time step. This is obtained by aggregating the throughput in each cell n given by $\min(o_{k,n}(t), \sigma_{k,n}(t)\cdot c_n)$.

Furthermore, a pre-processing module is also proposed to be included within the SON function, which computes the inputs to each DQN agent based on the performance measurements provided by the monitoring module and on the allocated capacity share $\sigma_k(t\text{-}1)$ during the last time step t-1.

Then, at each time step t, the DQN agent associated to tenant k obtains the state $\boldsymbol{s_k(t)}$ of the environment. Based on $\boldsymbol{s_k(t)}$, the agent selects an action $\boldsymbol{a_k(t)}$ according to the trained policy π_k by the DQN agent. Once all the actions for all the agents have been obtained, the capacity sharing computation module computes the resulting $\sigma_k(t)$ for all the tenants, avoiding unfeasible $\boldsymbol{\sigma_k(t)}$ solutions (e.g., allocation of more capacity than available in a cell). Next, the computed capacity sharing values $\boldsymbol{\sigma_k(t)}$ are communicated to the different cells in the NG-RAN environment through 3GPP interfaces for NSSI provisioning management services [37]. Moreover, the k-th agent is also provided with the reward signal $r_k(t)$ as a result of the last performed action $\boldsymbol{a_k(t\text{-}1)}$, which jointly with the $\boldsymbol{s_k(t)}$ are used for training the DQN agent associated to tenant k. The specific definitions of the state, action and reward signals in the proposed DQN agents and the operation of the capacity sharing computation module are described in the following.

A. State

The state obtained by the k-th tenant's agent at time t from the network environment is denoted as $\boldsymbol{s_k(t)} = [s_{k,1}(t), \ldots, s_{k,n}(t), \ldots, s_{k,N}(t), SAGBR_k/C, \sum_{k'=1,k'\neq k}^{K} SAGBR_{k'}/C]$, where each component $s_{k,n}(t)$ corresponds to the state of the tenant in cell n, given by $< \rho_{k,n}(t), \rho_n^A(t), \sigma_{k,n}(t\text{-}1), \sigma_n^A(t-1), MCBR_{k,n}/C >$. The component $\rho_n^A(t)$ is the fraction of available PRBs not used by any tenant in the cell, that is:

$$\rho_n^A(t) = 1 - \sum_{k=1}^{K} \rho_{k,n}(t) \tag{1}$$

Similarly, $\sigma_n^A(t)$ is the available capacity share in cell n not assigned to any tenant, given by:

$$\sigma_n^A(t) = 1 - \sum_{k=1}^{K} \sigma_{k,n}(t) \tag{2}$$

B. Action

At time t, the k-th tenant's agent selects the joint action $\boldsymbol{a_k(t)} = [a_{k,1}(t), \ldots, a_{k,n}(t), \ldots, a_{k,N}(t)]$, which is composed of the cell-specific actions $a_{k,n}(t)$ for each cell $n = 1 \ldots N$. The cell-specific action determines the variation in the capacity share $\sigma_{k,n}(t)$ to be applied in the following time step t in cell n. In order to achieve a gradual tuning of the capacity share, the cell-specific action can take three possible values $a_{k,n}(t) \epsilon \{\Delta, 0, -\Delta\}$, corresponding to increasing the capacity share in a step of Δ, maintaining it or decreasing it in a step of Δ. Consequently, the action space for $\boldsymbol{a_k(t)}$ corresponds to all the possible combination vectors of the three possible action values for each of the cells, which results in 3^N possible actions for each tenant. It is worth noting that other approaches in which the cell-specific action could take more than three possible values could also be feasible. However, they would imply a much larger action space, and thus, a longer process for learning the policy π_k. The actions $\boldsymbol{a_k(t)}$ are provided to the capacity sharing computation module to obtain the resulting capacity share solution $\boldsymbol{\sigma_k(t)}$, as detailed in Section III.E.

C. Reward

In order to assess to what extent the last action $\boldsymbol{a_k(t\text{-}1)}$ was adequate for the previous state $\boldsymbol{s_k(t\text{-}1)}$, a reward $r_k(t)$ is provided to the k-th tenant agent, which is defined as:

$$r_k\ (t) = \delta_k^{(1)}(t)^{\varphi_1} \cdot \delta_k^{(2)}(t)^{\varphi_2} \tag{3}$$

which considers two main factors, $\delta_k^{(1)}(t)$ and $\delta_k^{(2)}(t)$, defined in the following, and their corresponding weights, φ_1 and φ_2. These weights φ_1 and φ_2 are assumed to be defined by the InP according to its own criteria in establishing the importance of the factors $\delta_k^{(1)}(t)$ and $\delta_k^{(2)}(t)$.

The first factor, denoted as $\delta_k^{(1)}(t)$, intends to promote the satisfaction of the SLA of tenant k and the provisioning of enough capacity to satisfy its offered load. It is defined as:

$$\delta_k^{(1)}(t) = \\ = \begin{cases} \frac{T_k(t)}{O_k(t)} & \text{if } O(t) \leq C \\ \min\left(\frac{T_k(t)}{min(SAGBR_k + \beta_k(t), O_k(t))}, 1\right) & \text{Otherwise} \end{cases} \tag{4}$$

where $O_k(t)$ corresponds to the aggregated offered load of tenant k among all the cells at time step t, bounded by the $MCBR_{k,n}$ of each of the cells, given by:

$$O_k(t)\ = \sum_{n=1}^{N} \min(o_{k,n}(t), MCBR_{k,n}) \tag{5}$$

Moreover, $O(t)$ in (4) corresponds to the aggregated offered load $O_k(t)$ of all tenants at time step t while $\beta_k(t)$ captures the guaranteed capacity not required by other tenants at time step t, given by:

$$\beta_k\ (t) = \sum_{\substack{k'=1 \\ k' \neq k}}^{K} \max(SAGBR_{k'} - O_{k'}(t), 0) \tag{6}$$

which reaches $\beta_k\ (t) = 0$ when the offered load $O_k(t)$ of all the tenants is higher or equal than their $SAGBR_k$. Based on these parameters, the definition of $\delta_k^{(1)}(t)$ in (4) considers two different situations. The first condition in (4) considers the situation when there is enough capacity to satisfy the offered load of all the tenants $O(t)$ and, hence, $\delta_k^{(1)}(t)$ is given by the ratio between the throughput obtained by tenant k and its offered load, being maximum, i.e., $\delta_k^{(1)}(t) = 1$, when the throughput equals the offered load. In turn, the second condition in (4) considers the situation when the total system capacity C is not enough to satisfy the offered load requirement of all the tenants $O(t)$. In this situation, $\delta_k^{(1)}(t)$ is given by the ratio between the throughput of tenant k and the minimum between its offered load and its $SAGBR_k$ increased by the unused capacity left by the other tenants $\beta_k(t)$. Then, in this case $\delta_k^{(1)}(t)$ will be maximum (i.e., $\delta_k^{(1)}(t) = 1$) when the throughput equals the offered load $O_k(t)$ or, in the case that the offered load is higher than $SAGBR_k + \beta_k(t)$ when at least the throughput ensures this later value.

The fact that the definition of $\delta_k^{(1)}(t)$ depends on the aggregated offered load of all tenants $O(t)$ contributes to a collaborative behaviour between tenants, as in situations of overload this factor will promote actions that allow assuring the SLA of all tenants, avoiding those actions that would only benefit the k-th tenant at the cost of degrading the performance of the rest.

The second factor in the reward, $\delta_k^{(2)}(t)$, aims at measuring the degree of capacity overprovisioning and is defined by the ratio between the system throughput provided to the k-th tenant $T_k(t)$ and its provided capacity during the last time step, that is:

$$\delta_k^{(2)}(t)\ = \frac{T_k(t)}{\sum_{n=1}^{N} c_n \cdot \sigma_{k,n}(t-1)} \tag{7}$$

D. DQN Agent

The DQN agent of tenant k centrally learns the policy π_k that determines the actions to be executed in each cell. The proposed DQN agent executes the DQN algorithm of [14] but particularised to the state, action and reward signals previously introduced.

The objective of DQN, as a value-based RL algorithm, is to find the optimal policy π_k* that maximises the discounted cumulative future reward $d_k(t)$, computed as:

$$d_k(t)\ = \sum_{j=0}^{\infty} \gamma^j r_k(t+j+1) \tag{8}$$

where γ is the discount factor, which ranges $0 \leq \gamma \leq 1$, used to place more emphasis on immediate rewards. Finding π_k* can be performed by obtaining the optimal action-value function $Q_k{}^*(\boldsymbol{s_k}, \boldsymbol{a_k})$, which is the maximum expected discounted cumulative reward starting at time step t from $\boldsymbol{s_k}$, taking the action $\boldsymbol{a_k}$ and following the policy π_k:

$$Q_k{}^*\ (\boldsymbol{s_k}, \boldsymbol{a_k}) = \max_{\pi_k} E\left[d_k(t) \middle| \begin{matrix} \boldsymbol{s_k(t) = s_k} \\ \boldsymbol{a_k(t) = a_k} \\ \pi_k \end{matrix} \right] \tag{9}$$

This last expression can be decomposed into the Bellman Equation, which allows expressing $Q_k{*}(\boldsymbol{s_k},\boldsymbol{a_k})$ in a recursive form:

$$Q_k{}^*(\boldsymbol{s_k},\boldsymbol{a_k}) = E\left[r_k(t+1)+\gamma \max_{\boldsymbol{a_k}'} Q_k{}^*(\boldsymbol{s_k(t+1)},\boldsymbol{a_k}') \,\middle|\, \begin{matrix}\boldsymbol{s_k(t)}=\boldsymbol{s_k}\\ \boldsymbol{a_k(t)}=\boldsymbol{a_k}\end{matrix}\right] \tag{10}$$

According to $Q_k{}^*(\boldsymbol{s_k},\boldsymbol{a_k})$, the optimal policy can be found by selecting $\boldsymbol{a_k}$ greedily for each state $\boldsymbol{s_k}$ that is:

$$\pi_k{}^* = \underset{\boldsymbol{a_k}}{\operatorname{argmax}}\, Q_k{}^*(\boldsymbol{s_k},\boldsymbol{a_k}) \tag{11}$$

Generally, RL algorithms approximate the optimal $Q_k{}^*(\boldsymbol{s_k},\boldsymbol{a_k})$ by updating the approximated function $Q_k(\boldsymbol{s_k},\boldsymbol{a_k})$ iteratively based on the time difference (TD) error at each time t. The TD error is defined as the difference between $r_k(t+1)+\gamma\max_{\boldsymbol{a_k}'} Q_k{}^*(\boldsymbol{s_k(t+1)},\boldsymbol{a_k}')$, denoted as TD target, and the approximated value $Q_k(\boldsymbol{s_k},\boldsymbol{a_k})$. In the case of DQN, a non-linear approximation of $Q_k{}^*(\boldsymbol{s_k},\boldsymbol{a_k})$ is performed by using a DNN, denoted as Q-network, with weights θ_k, so that $Q_k{}^*(\boldsymbol{s_k},\boldsymbol{a_k}) \approx Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$. The inputs of the Q-network are the different components of the state $\boldsymbol{s_k}$, while the outputs correspond to the values of $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ for each possible action $\boldsymbol{a_k}$. Therefore, the policy π_k selects the action that maximises the output of the Q-network as:

$$\pi_k = \underset{\boldsymbol{a_k}}{\operatorname{argmax}}\, Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k) \tag{12}$$

The use of a DNN to approximate $Q_k{}^*(\boldsymbol{s_k},\boldsymbol{a_k})$ allows dealing with large state and action spaces. However, the use of non-linear approximation functions such as DNNs can imply instabilities or even divergence in the learning process due to: (i) correlations in sequential observations; (ii) correlations between action-values $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ and the TD-target; (iii) the fact that small updates of $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ may change the policy, dramatically, which can lead to changes of the distribution of the data collected from the environment. In order to avoid these effects, a DQN agent is composed of different elements, listed in the following:

- **Evaluation DNN** ($Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$): corresponds to the main approximation function of the expected reward function $Q_k(\boldsymbol{s_k},\boldsymbol{a_k})$. This function is trained off-line and is used to extract the policy π_k to select the actions to be performed in the environment. When starting the learning process, the weights θ_k are initialised randomly.
- **Target DNN** ($Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k{}^-)$)**:** this is another Q-network with the same structure as the evaluation DNN but with weights $\theta_k{}^-$. It is used to obtain the TD-Target as $r_k(t)+\gamma\max_{\boldsymbol{a_k}'} Q_k(\boldsymbol{s_k(t)},\boldsymbol{a_k}',\theta_k{}^-)$. Instead of updating the weights $\theta_k{}^-$ every time step, they are updated every M time steps with the weights of the evaluation DNN $\theta_k{}^- = \theta_k$. Consequently, the computation of the TD error, which depends on the target DNN, is no longer dependant on rapidly fluctuating estimates of the Q-values, as the target DNN only changes every M time steps but remains fixed the rest of the time.
- **Experience Dataset** (D_k^l)**:** a dataset D_k^l of length l is used to store the experiences of each agent. The stored experience for the agent associated to tenant k at time step t is represented by the experience tuple $<\boldsymbol{s_k(t-1)}, \boldsymbol{a_k(t-1)}, r_k(t), \boldsymbol{s_k(t)}>$. The use of the dataset allows randomly selecting mini-batches of experiences $U(D_k^l)$ with length J to update of the weights θ_k of the evaluation DNN. The use of D_k^l and the random minibatches for the learning of $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ is called *Experience Replay* and has several benefits. First, randomly selecting experiences from the dataset D_k^l breaks the temporal correlations in the training data, which may lead to inefficient learning. Second, past experiences can be reused, allowing for a greater data efficiency. Third, when not using experience replay and updating $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ on-policy, i.e., based on real time experiences, the current values of $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ determine the action that will lead to the next state, which will be used to update θ_k and will determine all the future experiences. This may lead to unwanted feedback loops where the values $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ can get stuck in poor local minimum or diverge. This effect is smoothed when using experience replay, since the data used for training is averaged over many of the previous states.

The training operation of the different DQN agents in the MARL solution has been summarized in *Algorithm* 1. The process of training of each DQN agent can be split into two main processes: the data collection (lines 4–13 of *Algorithm* 1) and the update of the weights θ_k of the evaluation DNN (lines 14–22 of *Algorithm* 1).

For the DQN agent associated to tenant *k*, the process of data collection consists in gathering experiences from the network environment and storing them in the experience dataset D_k^l , which is performed in time steps of Δt. For each time step *t,* the DQN agent observes the state of the environment $\boldsymbol{s_k(t)}$ and, accordingly, triggers an action $\boldsymbol{a_k(t)}$ based on an ε-Greedy policy π_k^ε that chooses actions according to (12) with probability 1-ε and a random action with probability ε. By using the ε-Greedy policy π_k^ε , the agent can explore new states that would not be visited, which improves the learning behaviour of the policy π_k. Moreover, the reward $r_k(t)$ is obtained, which assesses the suitability of the last performed action $\boldsymbol{a_k(t\text{-}1)}$ for the last state $\boldsymbol{s_k(t\text{-}1)}$. Then, the experience $<\boldsymbol{s_k(t-1)}, \boldsymbol{a_k(t-1)}, r_k(t), \boldsymbol{s_k(t)}>$ is stored in the dataset D_k^l. When the dataset D_k^l is full (i.e., l experiences are stored), old experiences are removed from the dataset to save new ones. Note that during the initial steps of the data collection, the actions are selected completely randomly (i.e., $\varepsilon = 1$ is considered in the π_k^ε) from the environment in order to explore several states and start filling the dataset D_k^l with experiences.

The process of updating the weights θ_k of the evaluation DNN is based on the experiences stored in the experience dataset. For each update of $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$, a minibatch of J experiences $U(D_k^l)$ is firstly selected from the dataset D_k^l. This selection is performed by randomly choosing experiences $e_{k,j} = <\boldsymbol{s_{k,j}}, \boldsymbol{a_{k,j}}, r_{k,j}, \boldsymbol{s_{k,j}}*>$ for $j = 1 \ldots J$. Then, $Q_k(\boldsymbol{s_k},\boldsymbol{a_k},\theta_k)$ is updated based on the mini-batch gradient descent of $U(D_k^l)$. For this

Algorithm 1: MARL DQN Training.

1 Initialize DNN counter $m = 0$.
2 **For** $t = 0 \ldots$ MaxNumberOfTrainingSteps
3 **For** $k = 0 \ldots K$
4 Collect global state $\boldsymbol{s_k(t)}$ by obtaining $s_{k,n}(t)$ for $n = 1 \ldots N$ cells.
5 Generate random ε' (ε' = 1 for the initial steps).
6 **If** $\varepsilon' < \varepsilon$
7 Choose randomly action $\boldsymbol{a_k(t)}$.
8 **Else**
9 Obtain action according to π_k in (12).
10 **End if**
11 Obtain reward $r_k(t)$ as a result of last action $\boldsymbol{a_k(t\text{-}1)}$.
12 If $D_l(k)$ is full (l samples are stored), remove the oldest one.
13 Store experience $< \boldsymbol{s_k(t-1)}, \boldsymbol{a_k(t-1)}, r_k(t), \boldsymbol{s_k(t)} >$ in D_k^l.
14 Randomly sample a minibatch of experiences $U(D_k^l)$ from D_k^l of length J.
15 Compute the loss function $L(\theta_k)$ by (13).
16 Compute the mini-batch gradient descent $\nabla L(\theta_k)$ by (14).
17 Update weights θ_k of evaluation DNN by (15).
18 **If** $m == M$
19 Update the weights of target DNN $\theta_k{}^- = \theta_k$ and set $m = 0$.
20 **Else**
21 $m = m+1$
22 **End if**
23 **End for**
24 Compute $\boldsymbol{\sigma_k(t)}$ for $k = 1 \ldots K$ by applying *Algorithm* 2 for all cells.
25 **End for**

purpose, the average mean squared error (MSE) loss over all the J experiences $e_{k,j}$ in $U(D_k^l)$, denoted as $L(\theta_k)$, is firstly computed as:

$$L(\theta_k) = E_{e_{k,j} \in U(D_k^l)}[(r_{k,j} = \gamma \max_{\boldsymbol{a_k}'} Q_k(\boldsymbol{s_{k,j}}*, \boldsymbol{a_k}', \theta_k{}^-) - Q_k(\boldsymbol{s_{k,j}}, \boldsymbol{a_{k,j}}, \theta_k))^2] \quad (13)$$

Then, the mini-batch gradient descent of $L(\theta_k)$, denoted as $\nabla L(\theta_k)$, is obtained by differentiating $L(\theta_k)$ with respect to θ_k, which yields:

$$\nabla L(\theta_k) = E_{e_{k,j} \in U(D_k^l)}[\left(r_{k,j} + \gamma \max_{\boldsymbol{a_k}'} Q_k(\boldsymbol{s_{k,j}}^*, \boldsymbol{a_k}', \theta_k{}^-) - Q_k(\boldsymbol{s_{k,j}}, \boldsymbol{a_{k,j}}, \theta_k)\right) \nabla_\theta Q_k(\boldsymbol{s_{k,j}}, \boldsymbol{a_{k,j}}, \theta_k)] \quad (14)$$

Finally, the weights in the $Q_k(\boldsymbol{s_k a_k}, \theta_k)$ network are updated according to:

$$\theta_k \rightarrow \theta_k + \tau \nabla L(\theta_k) \quad (15)$$

where τ is the learning rate. After each update of θ_k, the obtained $Q_k(\boldsymbol{s_k a_k}, \theta_k)$ can be used by the ε-Greedy policy to trigger actions in the network environment. Moreover, during the update of weights θ_k, the DQN agent has a counter m of the number of time steps since the last target DNN update and, when $m = M$, the weights in the target DNN are updated as $\theta_k{}^- = \theta_k$ and m is initialised again.

E. Capacity Sharing Computation

In the considered MARL approach, the DQN agents associated to the different tenants in the system interact with a common network environment in a collaborative and coordinated manner in order to trigger the actions that the capacity sharing computation module will use to configure the capacity share $\boldsymbol{\sigma_k(t)}$ of each tenant.

Initially, the capacity shares of each tenant in each of the cells $\sigma_{k,n}(t = 0)$ are initialised proportionally to the $SAGBR_k$ of the tenant, according to:

$$\sigma_{k,n}(t = 0) = SAGBR_k \Big/ \left(\sum_{k'=1}^{K} SAGBR_{k'}\right) \quad (16)$$

Then, $\sigma_{k,n}(t)$ is dynamically tuned according to the selected actions $\boldsymbol{a_k(t)}$ by each agent at each time step. Even though the process of data collection by each DQN agent is performed independently of the others, the different DQN agents trigger their actions and store the experiences synchronously. At each time step t, each agent k is provided with its state $\boldsymbol{s_k(t)}$ and accordingly selects an action $\boldsymbol{a_k(t)}$ as previously explained. Next, the capacity sharing computation module gathers the selected actions $\boldsymbol{a_k(t)}$ by all the agents and computes the resulting capacity share solution $\boldsymbol{\sigma_k(t)}$. Specifically, the capacity share of tenant k at cell n, $\sigma_{k,n}(t)$, is updated according to:

$$\sigma_{k,n}(t) = \begin{cases} \sigma_{k,n}(t-1) + a_{k,n}(t) & \text{if } 0 \le \sigma_{k,n}(t-1) + a_{k,n}(t) \le \\ & \le \frac{MCBR_{k,n}}{c_n} \\ \sigma_{k,n}(t-1) & \text{Otherwise} \end{cases} \quad (17)$$

The formulation of (17) assures that $\sigma_{k,n}(t)$ is within its bounds and considers that the last capacity share value $\sigma_{k,n}(t\text{-}1)$ is maintained when $a_{k,n}(t)$ forces $\sigma_{k,n}(t)$ to be out of its bounds.

In some special situations, the fact that the actions of each tenant are triggered independently can lead to capacity sharing solutions that excess the total cell capacity in some cells (i.e., $\sum_{k=1}^{K} \sigma_{k,n}(t) > 1$). When this occurs in any cell, the capacity sharing computation module obtains the capacity shares in the cell by executing *Algorithm* 2. The algorithm firstly applies the actions of tenants aiming at decreasing or maintaining the capacity share in the cell (i.e., $a_{k,n}(t) \in \{-\Delta, 0\}$) and computes the resulting available cell capacity $\sigma_n^A(t)$ (lines 1–2). If there is no available capacity (i.e., $\sigma_n^A(t) = 0$), the actions of tenants willing to increase its capacity in the cell are not applied (lines 3-4). In turn, when there is available capacity (i.e., $\sigma_n^A(t) > 0$), the capacity share is obtained by distributing $\sigma_n^A(t)$ among those tenants with $a_{k,n}(t) = \Delta$ proportionally to their $SAGBR_k$ value as long as they are not already provided with more than $SAGBR_k$ (lines 5-6) according to:

Algorithm 2: Capacity Sharing Computation for Cell n for Cell Capacity Excess Situations.

1 Compute $\sigma_{k,n}(t)$ of tenants with $a_{k,n}(t) \in \{-\Delta, 0\}$ according to (17).
2 Compute available cell capacity share $\sigma_n^A(t)$ according to (2).
3 **If** $\sigma_n^A(t) = 0$
4 Set $\sigma_{k,n}(t) = \sigma_{k,n}(t\text{-}1)$ of tenants with $a_{k,n}(t) = \Delta$.
5 **Else**
6 Compute $\sigma_{k,n}(t)$ by distributing $\sigma_n^A(t)$ among tenants with $a_{k,n}(t) = \Delta$ and $T_k(t) \leq SAGBR_k$ proportionally to their $SAGBR_k$ according to (18).
7 **End if**

$$\sigma_{k,n}(t) = \sigma_{k,n}(t-1) + \frac{\sigma_n^A(t) \cdot SAGBR_k}{\sum_{\substack{k'=1 \\ a_t(k',n)=\Delta, \\ T_t(k') \leq SAGBR_{k'}}}^{K} SAGBR_{k'}} \tag{18}$$

IV. Performance Evaluation

This section evaluates the proposed capacity sharing SON function by assessing its performance in a scenario with given offered loads. First, after describing the considered scenario in section IV.A and the Key Performance Indicators (KPIs) in section IV.B, the capability of the agent of one tenant to learn a general policy applicable to any other tenant in the system is assessed in section IV.C. This is followed by a discussion of the scalability of the solution by adding a new tenant in the considered scenario for evaluation in section IV.D. Finally, the impact of different model parameter configurations on the achieved performance is provided in section IV.E and an analysis on the optimality of the SON function is included in section IV.F.

A. Considered Scenario

The assumed scenario comprises a NG-RAN infrastructure with five cells that serve the users of two different tenants, denoted as Tenant 1 and Tenant 2. The configuration of the scenario is presented in Table II, including the cells configuration and the SLA parameters established for each tenant.

The model has been developed in Python by using the library *TF-Agents* [38], which provides tools for the development of DRL models, including DQN. The developed model has been trained according to the parameters of Table III. The dataset considered for training is composed of 1400 synthetically generated offered load patterns of Tenant 1 and Tenant 2 in the different cells during one day, considering different combinations of $SAGBR_k$ values for both tenants.

After the model has been trained, the resulting policies π_k are evaluated using the offered load patterns shown in Fig. 2. The figure plots the aggregated offered loads among all the cells of Tenant 1, $O_1(t)$, and Tenant 2, $O_2(t)$, during one day.

TABLE II
Scenario Configuration

Parameter		Value
Number of tenants (K)		2
Number of cells (N)		5
PRB Bandwidth (B_n)		360 kHz
Number of cell available PRBs (W_n)		65 PRBs
Average spectral efficiency (S_n)		5 b/s/Hz
Total cell capacity (c_n)		117 Mb/s
Total system capacity (C)		585 Mb/s
$SAGBR_k$	Tenant 1	351Mb/s (corresponding to 60% of C)
	Tenant 2	234Mb/s (corresponding to 40% of C)
$MCBR_{k,n}$	Tenant 1	93.6 Mb/s (corresponding to 80% of c_n)
	Tenant 2	

TABLE III
MARL Model Parameters

Parameter		Value
Initial collect steps		5000
Maximum number of time steps for training		$2 \cdot 10^6$
Experience Replay buffer maximum length (l)		10^7
Mini-batch size (J)		256
Learning rate (τ)		0.0001
Discount factor(γ)		0.9
ε value (ε-Greedy)		0.1
DNN configuration		Input layer: 17 nodes 1 full connected layer: 100 nodes Output layer: 243 nodes
Reward weights	φ_1	0.5
	φ_2	0.4
Time step duration	Δt	3 min
Action step	Δ	0.03

Fig. 2. Offered loads of Tenant 1 and 2 during a day.

The figure also includes the values of $SAGBR_1$ and $SAGBR_2$, the total system capacity C and the aggregated offered loads of both tenants $O(t)$. Note that the offered loads of both tenants exceed their $SAGBR_k$ at some point during the day and the system offered load $O(t)$ is higher than C during the time period from 900 min to 1300 min. Moreover, a uniform distribution of the load among the different cells has been considered.

Regarding the complexity of the proposed approach, it has been assessed in terms of the execution time of the MARL model over a machine with 2 CPU AMD Opteron 4386 operating

with Ubuntu 18.04, configured to use 2 cores and 8G RAM. Specifically, the execution of one trained DQN agent in one time step during evaluation stage lasts 3.8 ms on average, which is a sufficiently low value that would enable the operation in a practical system. Concerning the training stage, it is obtained that the execution of a time step lasts approximately 36 ms. The larger duration of the time step in the training stage compared to the evaluation stage is motivated by the additional operations required during the training for updating the weights of the DNNs.

B. Key Performance Indicators

In this section, different KPIs are defined in order to assess the performance of the model:

- Assigned capacity to tenant k at time step t ($A_k(t)$): It is measured in bps and is obtained from the capacity share $\boldsymbol{\sigma_k(t)}$ provided by the SON function and the capacity of each cell c_n as:

$$A_k(t) = \sum_{n=1}^{N} c_n \cdot \sigma_{k,n}(t) \tag{19}$$

- Average reward of tenant k (R_k): It is computed as the average of the reward $r_k(t)$ obtained by the tenant over a duration of G time steps.

$$R_k = \frac{1}{G} \sum_{t=0}^{G-1} r_k(t) \tag{20}$$

- Average SLA satisfaction of tenant k (SS_k): It measures the average of the ratio between the provided throughput $T_k(t)$ to tenant k and the minimum between the aggregated offered load of the tenant $O_k(t)$ and its $SAGBR_k$ value over a duration of G time steps, that is:

$$SS_k = \frac{1}{G} \sum_{t=0}^{G-1} \min\left(\frac{T_k(t)}{\min(O_k(t),\ SAGBR_k)}, 1 \right) \tag{21}$$

  which ranges $0 \leq SS_k \leq 1$, taking $SS_k = 0$ value when the SLA is not satisfied and $SS_k = 1$ when it is fully satisfied. Note that the definition of SS_k considers that when $O_k(t)$ is lower than the $SAGBR_k$, $O_k(t)$ needs to be provided, whereas in the case that $O_k(t)$ is greater than $SAGBR_k$, at least $SAGBR_k$ needs to be provided.
- Average system utilization (U): It is computed as the average ratio between the aggregated throughput provided to all tenants and the total system capacity C over a duration of G time steps, that is:

$$U = \frac{1}{G} \sum_{t=0}^{G-1} \frac{1}{C} \sum_{k=1}^{K} T_k(t) \tag{22}$$

C. Generalization of the Learnt Policies

According to the proposed approach, the DQN agent of each tenant learns its own policy during the training and then this policy is applied during the evaluation. However, considering that the training of the different tenants has been done under very different situations of their own load and the load of the others and for different SLA parameters, in this section, we intend to analyse to what extent there are significant differences between the policies learnt by the different tenants. In this way, the main goal is to assess whether it is possible or not to generalise a policy leant by one tenant so that it can be also used by another tenant.

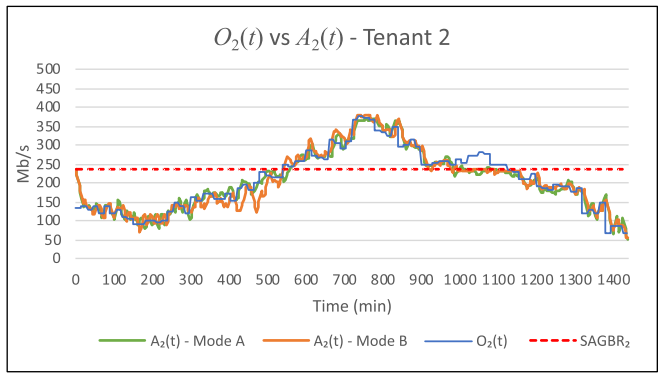

Fig. 3. Offered load vs assigned capacity for Tenant *2* for tenant policy application *Mode A* and *B*.

TABLE IV
KPIs FOR BOTH POLICY APPLICATION MODES

Policy application mode		Mode A	Mode B
Average reward	Tenant 1 (R_1)	0.9673	0.9674
	Tenant 2 (R_2)	0.9541	0.9483
SLA Satisfaction	Tenant 1 (SS_1)	0.9725	0.9742
	Tenant 2 (SS_2)	0.9705	0.9577
Average utilisation (U)		0.8885	0.8861

To conduct the analysis, the capacity sharing solution for the offered loads of Fig. 2 is obtained under two different policy application modes. In *Mode A*, the DQN agent of each tenant applies its trained policy, i.e., the DQN agent of Tenant 1 applies policy π_1, and the DQN agent of Tenant 2 applies policy π_2. Both policies are the ones obtained after $200{\cdot}10^4$ training steps. In turn, *Mode B* considers that the DQN agents of both Tenant 1 and Tenant 2 apply the same policy π_1 learnt for Tenant 1.

Fig. 3 presents the temporal evolution of the offered load of Tenant 2 against its assigned capacity $A_2(t)$ for policy application *Mode A* and *Mode B*. In addition, the $SAGBR_2$ of Tenant 2 is included. The assigned capacity for both policy application modes generally adapts to the offered load for all the situations where the total offered load $O(t)$ (seen in Fig. 2) does not exceed the system capacity C. In turn, when $O(t)$ exceeds the system capacity, the assigned capacity to Tenant 2 is kept in the $SAGBR_2$ value. The figure shows that very little differences are observed in the assigned capacity $A_2(t)$ when applying the policies according to *Mode A* and *Mode B*.

Moreover, to quantitatively assess the differences between both modes, Table IV provides the average reward R_k and the SLA satisfaction SS_k for Tenant 1 and Tenant 2 in addition to the average system utilisation U. The obtained values show that the achieved performance for both policy application modes is

very similar, with differences lower than 1% for all the analysed KPIs.

First, it is worth pointing out that for the case of Tenant 1 there are some slight differences in Table IV although the policy applied for this tenant is the same in both modes. The reason is that the actions taken by the DQN agent of Tenant 2, which can be slightly different when changing the applied policy, impact on the state seen by the DQN agent of Tenant 1 and, thus, on the selected actions by this Tenant. In any case, the impact of these different actions on the performance is negligible as seen in the table.

Concerning Tenant 2, the reason that similar behaviour is obtained for policy application *Mode A* and *Mode B* is that, although the training of policies π_1 and π_2 has been performed independently for both tenants, this has considered a dataset composed of several offered load situations, exploring different complementarities between the two tenants, jointly with diverse combinations of $SAGBR_k$ and $MCBR_{k,n}$ values. This allows the agents to learn equivalent policies that can be generalised to many offered load situations and SLA requirements. This observation has important positive implications on the practicality of the proposed approach, because it means that a single training process carried out by one DQN agent using a dataset that covers a wide range of offered load situations and SLA requirements can be sufficient to obtain a policy that is valid for multiple tenants. As a consequence, a reduction of the complexity of the training process will be achieved in a multi-agent scenario. Moreover, this also facilitates the scalability of the model to add new tenants in the scenario, because the addition of a tenant can be done without retraining the previous learnt policies, as it will be studied in the next sub-section.

D. Addition of a New Tenant

Following the generalization capability of the trained policies that has been observed in previous section, this section aims at assessing the association of already trained policies to new tenants that are added in the scenario, without neither training new policies for the new tenants nor retraining (i.e., training again) the policies from the existing tenants. To this end, a new tenant, denoted as Tenant 3, is introduced to the scenario of Table II. Instead of performing a separate training for the new Tenant 3, the previously trained policy for Tenant 1, π_1, is used for this new tenant as well as for Tenant 1 and 2. Since the $SAGBR_k$ of Tenants 1 and 2 use the total system capacity of Table II, in order to support the new tenant, the capacity in the system is extended by increasing the cell bandwidth from 25 MHz to 30 MHz. As a result, the number of PRBs in each cell is increased to $W_n = 78$ PRBs, providing a total cell capacity $c_n = 143.2$ Mb/s and, thus, a total system capacity of $C = 716$ Mb/s. The SLA established for Tenant 3 considers $SAGBR_3 = 93.6$ Mb/s and $MCBR_{3,n} = 114.56$ Mb/s, corresponding to 80% of the cell capacity. The $SAGBR_k$ of Tenant 1 and 2 remain the same as in Table II, whereas the $MCBR_{k,n}$ of those tenants is updated to $MCBR_{1,n} = MCBR_{2,n} = 114.56$Mb/s given that the cell capacity has increased.

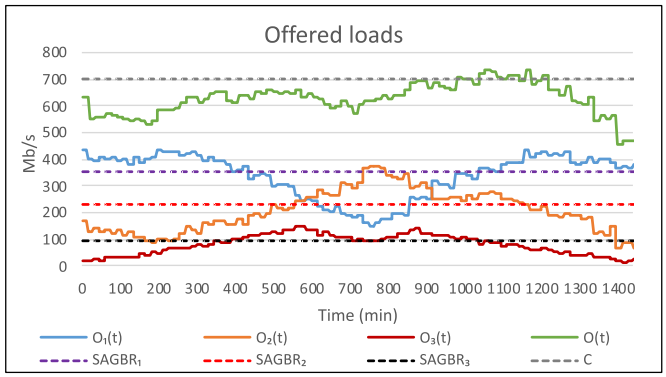

Fig. 4. Offered load of Tenant 1, 2, and 3 during a day.

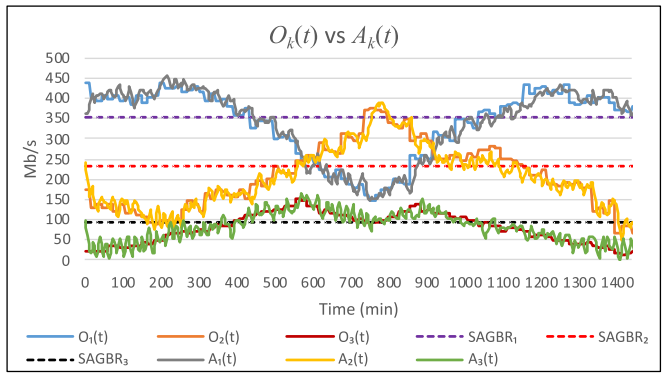

Fig. 5. Offered load vs assigned capacity for Tenant 1, 2, and 3.

The offered loads considered for evaluation during one day are plotted in Fig. 4 together with the $SAGBR_k$ values, the aggregated offered load in the system $O(t)$ and the total system capacity C. The offered loads of Tenant 1 and Tenant 2, $O_1(t)$ and $O_2(t)$, are the same as in Fig. 2, and the offered load of Tenant 3, $O_3(t)$, presents lower values than the other tenants, reaching its higher values at $t = 570$ min and $t = 880$ min when its $SAGBR_3$ is exceeded. Despite introducing Tenant 3, the total offered load of the three tenants only slightly exceeds the system capacity from $t = 1000$ min to $t = 1200$ min.

Fig. 5 shows the offered loads $O_k(t)$ against the assigned capacity $A_k(t)$ of Tenant 1, 2 and 3, in addition to their $SAGBR_k$ values. Since most of time there is enough capacity to fulfil the offered load of the three tenants, the offered loads are satisfied nearly all the day. When the overall offered load $O(t)$ exceeds the system capacity, the tenants that required more capacity than their $SAGBR_k$ are assigned with lower capacity than their offered load, such as Tenant 2 from $t = 1035$ min to $t = 1115$ min. In the case of Tenant 3, the offered load $O_3(t)$ is generally satisfied the tenants or the need of re-training when the capacity in the system changes. These results also provide evidence of the advantage of a multi-agent approach with respect to a single-agent approach, where a single agent manages all the since in the periods when the $O_3(t)$ is larger than $SAGBR_3$, there is enough capacity in the system to satisfy all the tenants. These results show qualitatively that the policy learnt by one tenant is general enough to properly assign the capacity to the other tenants according to their offered

TABLE V
PERFORMANCE PARAMETERS

Applied policy		Tenant- specific policies	Tenant 1 policy
Average reward	Tenant 1 (R_1)	0.964	0.967
	Tenant 2 (R_2)	0.939	0.949
	Tenant 3 (R_3)	0.873	0.859
SLA Satisfaction	Tenant 1 (SS_1)	0.986	0.979
	Tenant 2 (SS_2)	0.957	0.961
	Tenant 3 (SS_3)	0.901	0.893
Average utilisation (U)		0.843	0.845

loads and SLA requirements and, additionally, performs satisfactorily in front of changes in the system capacity, since the internal parameters of the DQN agent (i.e., state, reward factors, actions, etc.) are defined in relative values. This highlights the capability of scaling of the proposed solution, as the trained policies can be used by new tenants in the scenario without the need of performing a separated training for each of tenants and the addition of a new tenant would imply the training of the whole solution again. Moreover, this would require a larger training duration due to the larger state and action spaces resulting from the additional state and action dimensions for the new tenant. Instead, in the proposed approach, a new tenant can be added just by associating an already trained policy.

To further illustrate the scaling capability of the proposed approach, the performance achieved when applying the policy π_1 for all tenants is compared against the case of applying the policies π_1, π_2 and π_3 specifically trained for each tenant. To this end, policies π_1, π_2 and π_3 have been trained according to model parameters detailed in Table III by considering the scenario with Tenant 3 and extended capacity. Table V includes the resulting average SLA satisfaction SS_k and the average reward for Tenant 1, 2 and 3 and the average system utilisation U when applying the specifically trained polices for each tenant, and when applying the policy π_1. Once again, the achieved values for the different assessed indicators for both cases are really close, presenting differences below 1.5% for all indicators. This highlights the scaling capability of the proposed approach since the application of a previously trained policy is able to adapt to the offered loads of the new tenant and the loads of the other tenants when the system capacity increases.

Regarding the achieved values for the different analysed performance indicators in Table V, although high values are obtained for all the tenants, Tenant 1 achieves the highest, closely followed by Tenant 2, and Tenant 3 presents the lowest values. The reason is that the offered load values of Tenant 3 are much lower than the ones for Tenant 1 and Tenant 2, and thus, the analysed performance indicators are more affected by the increases and decreases in steps of Δ. This means that decreasing the assigned capacity of Tenant 1 by $\Delta = 0.03$ has a lower impact on R_k and SS_k of Tenant 1 than for Tenant 3. Since the offered load levels of Tenant 2 are similar to the ones of Tenant 1, lower differences are obtained between them as a result of this effect. Therefore, in order to achieve higher performance values for Tenant 3, lower values of Δ would be more appropriate since

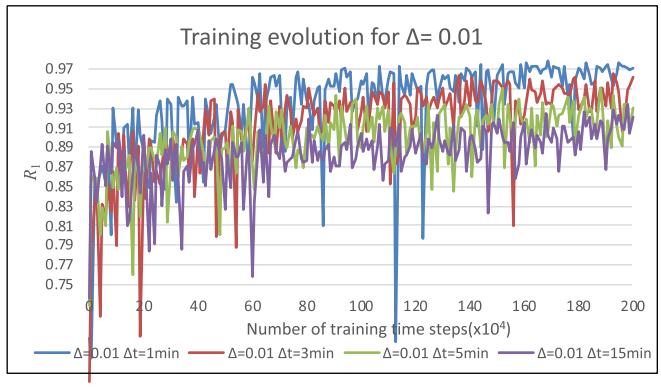

Fig. 6. Average aggregated reward every 10000 time steps during the training for $\Delta = 0.01$ and $\Delta t = \{1, 3, 5, 1, 5\}$ min.

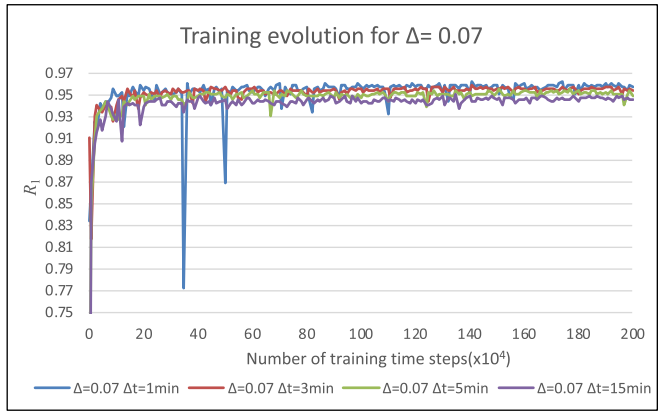

Fig. 7. Average aggregated reward every 10000 time steps during the training for $\Delta = 0.07$ and $\Delta t = \{1, 3, 5, 1, 5\}$ min.

they would have a lower impact on the assessed parameters. This reveals that the selection of Δ needs to jointly consider the traffic type and levels of all tenants in order to choose a value that best satisfies all of them given the clear impact of Δ on the achieved performance, as is studied in the following section.

E. Impact of Action and Time Step

This section aims at analyzing the impact of the values of time step duration Δt and action increment/decrement step Δ on the proposed SON function. For this purpose, the policy π_1 of Tenant 1 has been trained in the scenario of Table II by considering different parameter configurations. The assessed configurations include all the combinations between time step $\Delta t = \{1, 3, 5, 15\}$min and action increment/decrement step $\Delta = \{0.01, 0.02, 0.03, 0.05, 0.07, 0.09\}$. The rest of the parameters used for training are those specified in Table III.

First, the impact Δt and Δ on the training evolution process is studied. To this end, the policy π_1 for Tenant 1 obtained by the training every 10^4 time steps has been evaluated by applying it on the offered load of Tenant 1 of Fig. 2. This allows capturing how the training process progressively updates the learnt policy when increasing the number of training steps. Fig. 6 and Fig. 7 show the evolution of the average reward R_1 of Tenant 1 for $\Delta t = \{1,$

TABLE VI
KPIs FOR THE DIFFERENT COMBINATIONS OF Δ AND Δt

Action step value (Δ)	Average SLA satisfaction								Average system Utilisation (U)			
	Tenant 1 (SS_1)				Tenant 2 (SS_2)							
	Δt=1min	Δt=3min	Δt=5min	Δt=15min	Δt=1min	Δt=3min	Δt=5min	Δt=15min	Δt=1min	Δt=3min	Δt=5min	Δt=15min
0.01	0.972	0.972	0.960	0.933	0.961	0.976	0.958	0.971	0.877	0.879	0.848	0.823
0.02	0.984	0.978	0.982	0.956	0.972	0.976	0.974	0.969	0.897	0.894	0.884	0.855
0.03	0.982	0.974	0.973	0.975	0.973	0.958	0.966	0.979	0.894	0.886	0.881	0.883
0.05	0.981	0.973	0.973	0.960	0.956	0.944	0.946	0.958	0.885	0.878	0.873	0.856
0.07	0.963	0.970	0.968	0.959	0.934	0.935	0.916	0.949	0.861	0.869	0.863	0.865
0.09	0.942	0.960	0.952	0.961	0.919	0.908	0.908	0.922	0.822	0.848	0.843	0.850

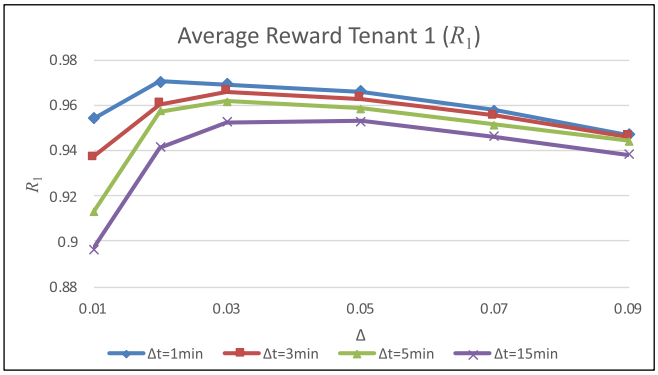

Fig. 8. Average aggregated reward between $100{\cdot}10^4$ and $200{\cdot}10^4$ training time steps for the different considered configurations.

3, 5, 15}min when considering, respectively, a small action step $\Delta = 0.01$ and a large action step $\Delta = 0.07$. For both values of Δ, higher average reward is achieved when reducing Δt, since the policy is triggered more frequently and thus it can more easily react to changes. However, evident differences are observed between the training evolution when using $\Delta = 0.01$ and $\Delta = 0.07$. For $\Delta = 0.07$, the average reward presents an initial period of around $50{\cdot}10^4$ training time steps where it increases and presents high fluctuations until the average reward stabilises to a value in the range between 0.94 and 0.96 depending on Δt and the fluctuations decrease drastically, reflecting that the algorithm has achieved convergence. Instead, for $\Delta = 0.01$, the average reward keeps increasing during the whole analysed training period and it exhibits larger fluctuations than for $\Delta = 0.07$, so convergence takes longer. The reason is that with low values of Δ, the actions at each time step have a low impact on the next state and the reward obtained, so the training has more difficulties to converge.

Fig. 8 depicts the average reward R_1 of Tenant 1 obtained between $100{\cdot}10^4$ and $200{\cdot}10^4$ training time steps for the different analysed combinations of Δt and Δ. The highest reward is observed for $\Delta t = 1$ min and the reward decreases when increasing Δt. Regarding the effect of the action step Δ, the highest reward is achieved for $\Delta = 0.02$ for $\Delta t = 1$ min while the maximum is achieved for $\Delta = 0.03$ for the rest of time step durations and, for higher values than 0.03, a decreasing trend is observed, since large values of Δ make more difficult the adjustment of the assigned capacity to the offered load of Tenant 1 in Fig. 2. These results together with the fluctuations obtained in Fig. 6 and Fig. 7 reveal that a trade-off exists when selecting the value of Δ: a higher reward is achieved for low values of Δ but at the cost of slower convergence. In addition, Fig. 8 shows that the combination of large values of Δt with low values of Δ (e.g., $\Delta = 0.01$ and $\Delta t = 15$ min) lead to poor performance as those combinations do not allow adapting to the dynamics of the offered load.

Finally, the impact of Δ and Δt on the performance metrics is analysed in Table VI. It presents the average SLA satisfaction SS_k and system utility U obtained for different values of Δ and Δt when applying the trained policies for Tenant 1 after $200{\cdot}10^4$ training time steps to the offered loads of Tenant 1 and Tenant 2 in Fig. 2. Focusing on the average SLA satisfaction, the capacity sharing SON function achieves high values above 0.9 for both tenants and all the combinations of Δ and Δt. Higher values are generally obtained for combinations with low values of Δ and Δt, since they adapt better to the changes in the offered loads. Also, a decreasing trend in the SLA satisfaction is observed when increasing Δ for values of Δ beyond 0.03. Comparing the SLA satisfaction obtained for Tenant 1 and Tenant 2, slight differences are obtained between them. About the impact of Δ and Δt on the average system utilization U, similar effects than in the case of the SLA satisfaction are obtained, achieving the highest values for $\Delta = 0.02$ for all values of Δt in exception of $\Delta t = 15$ min, which the maximum is achieved for $\Delta = 0.03$, and being the utilization reduced when increasing Δ beyond the maximum.

Based on the obtained results, it is concluded that the selection of the Δ and Δt values has a clear impact on the training evolution of the policies and their achieved performance and an adequate selection of these values that jointly considers the specific traffic dynamics of the different tenants is fundamental for ensuring an accurate learning process and a good compromise between the different KPIs achieved by the capacity sharing SON function.

F. Optimality Analysis

In this section, the optimality of the capacity sharing SON function is analyzed by comparing its performance to the optimum. The optimum has been obtained by an exhaustive search algorithm that evaluates in each time step all the possible values of capacity share $\boldsymbol{\sigma_k(t)}$ of Tenant 1 and Tenant 2, discretized in steps of Δ, and selects the one that achieves the maximum aggregate reward of both tenants. To assess the optimality in a wide range of offered load situations, results have been obtained for a set of 240 offered load temporal patterns of one day duration, which include diverse offered load behaviours with

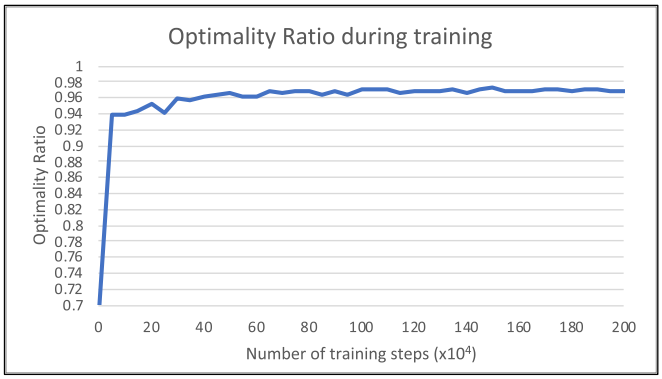

Fig. 9. Optimality ratio during training.

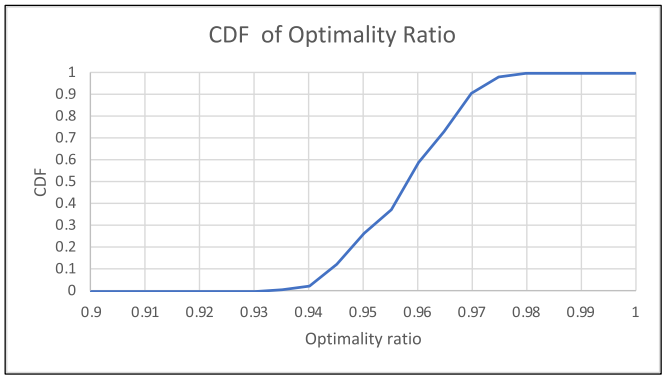

Fig. 10. Cumulative Density Function (CDF) of optimality ratio.

diverse complementarities between the offered loads of Tenant 1 and Tenant 2. For each pattern, results have been obtained using the scenario configuration in Table II and by applying the trained policy π_1 of Tenant 1 to both tenants. For each pattern, results are obtained in terms of the optimality ratio, which is defined as the average of the aggregate reward of Tenant 1 and Tenant 2 obtained with the SON function divided by the average optimum reward over all the time steps in an offered load pattern.

Fig. 9 presents the evolution of the optimality ratio during the training process for the offered load pattern of Fig. 2. This has been obtained by evaluating the policy π_1 every $5{\cdot}10^4$ training steps and computing the optimality ratio. It is observed that, initially, the optimality ratio increases abruptly with the number of training steps and, after approximately $5{\cdot}10^4$ training steps, it achieves values higher than 0.94. Then, it increases slowly with the number of training steps and stabilises to a value of around 0.97, corresponding to the situation when the algorithm has converged. The figure also reflects that no significant improvements are obtained by increasing the number of training steps beyond $50{\cdot}10^4$.

To analyze the optimality ratio under a broader range of situations, Fig. 10 shows the Cumulative Density Function (CDF) of the optimality ratios obtained for the different offered load patterns with the policy learnt after $200{\cdot}10^4$ time steps. Results reveal that the optimality ratio for all the analysed offered load patterns range between 0.94 and 0.98. Moreover, it has been obtained that the average optimality ratio is 0.96. Overall, results reveal that the proposed MARL approach achieves a behaviour very close to the optimum and highlight the capability of the trained policy π_1 to adapt to diverse offered loads. It is also worth noting that this near optimal results are obtained with very small execution times of the trained policy, as previously discussed in section IV.A, while the exhaustive search method requires to assess all the combinations for each time step, which is highly time consuming and requires execution times higher in several orders of magnitude than the MARL approach.

V. Conclusion and Future Work

This paper has presented a capacity sharing approach for multi-tenant and multi-cell scenarios. The proposed solution consists in a collaborative multi-agent reinforcement learning approach, where each agent is based on the Deep Q-Network (DQN) technique. The different DQN agents, which interact with a common network environment in a coordinated and cooperative manner, provide the capacity to be assigned to their associated tenant in each of the cells of the network environment in order to satisfy the Service Level Agreement (SLA) of the tenant and making an efficient use of the available resources. The approach has been contextualised within the 3GPP management framework and has been proposed as a Self-Organizing Network (SON) function.

The behaviour of the proposed capacity sharing SON function has been assessed in a multi-cell scenario with two tenants by considering different parameter configurations of the multi-agent reinforcement learning solution, which differ in the values of actions and the time step that the actions are triggered. Results have shown that: (i) The capacity sharing SON function satisfactorily adapts the capacity assigned to each tenant to their traffic and SLA requirements; (ii) The policies learnt by the agents associated to each tenant are generalizable to any tenant, given that the dataset used for training is composed of a wide range of traffic requirement situations and SLA requirements; (iii) The proposed approach is easily scalable to deal with the addition of new tenants simply by associating to the new tenant a new DQN agent with a previously learnt policy and to support the operation when the system capacity changes. (iv) The values of actions and the time step need to be jointly configured to better adapt to the variability characteristics of the traffic requirements of the different tenants in order to improve the SLA satisfaction and maximise the utilisation of the available resources. (v) The trained policies are able to provide results very close to the optimum when they are applied to diverse offered load patterns, with observed optimality ratios ranging between 0.94 and 0.98.

Overall, the results presented here reflect the potential and adequacy of the proposed DQN-based multi-agent approach for capacity sharing. However, research efforts on deep reinforcement learning have provided new advanced techniques, such as DDQN or DDPG, which have also shown a promising behaviour in terms of convergence when being applied to capacity sharing in a single-cell and single-agent scenario [28]. Therefore, it is worth considering as future work the detailed assessment and comparison of these techniques against the DQN-based approach of this paper in multi-cell and multi-agent

scenarios, taking into account both performance and practicality aspects.

REFERENCES

[1] P. Rost *et al.*, "Mobile network architecture evolution toward 5G," *IEEE Commun. Mag.*, vol. 54, no. 5, pp. 84–91, May 2016.

[2] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 32–39, Jul. 2016.

[3] R. Ferrús, O. Sallent, J. Pérez-Romero, and R. Agustí, "On 5G radio access network slicing: Radio interface protocol features and configuration," *IEEE Commun. Mag.*, vol. 56, no. 6, pp. 184–192, May 2021.

[4] A. S. D. Alfoudi, S. H. S. Newaz, A. Otebolaku, G. M. Lee, and R. Pereira, "An efficient resource management mechanism for network slicing in a LTE network," *IEEE Access*, vol. 7, pp. 89441–89457, 2019.

[5] D. Marabissi and R. Fantacci, "Highly flexible RAN slicing approach to manage isolation, priority, efficiency," *IEEE Access*, vol. 7, pp. 97130–97142, 2019.

[6] P. L. Vo, M. N. H. Nguyen, T. A. Le, and N. H. Tran, "Slicing the edge: Resource allocation for RAN network slicing," *IEEE Wireless Commun. Lett.*, vol. 7, no. 6, pp. 970–973, Dec. 2018.

[7] J. Pérez-Romero, O. Sallent, R. Ferrús, and R. Agustí, "Profit-Based radio access network slicing for Multi-tenant 5G networks," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Valencia, Spain, 2019, pp. 603–608.

[8] Ö. U. Akgül, I. Malanchini, and A. Capone, "Dynamic resource trading in sliced mobile networks," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 1, pp. 220–233, Mar. 2019.

[9] J. Shi, H. Tian, S. Fan, P. Zhao, and K. Zhao, "Hierarchical auction and dynamic programming based resource allocation (HA&DP-RA) algorithm for 5G RAN slicing," in *Proc 24th Asia-Pacific Conf. Commun.*, Ningbo, China, 2018, pp. 207–212.

[10] J. Gang and V. Friderikos, "Optimal resource sharing in multi-tenant 5G networks," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Barcelona, 2018, pp. 1–6.

[11] P. Caballero, A. Banchs, G. De Veciana, and X. Costa-Pérez, "Network slicing games: Enabling customization in multi-tenant mobile networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 662–675, Apr. 2019.

[12] J. Pérez-Romero, O. Sallent, R. Ferrús, and R. Agustí, "Self-optimized admission control for multitenant radio access networks," in *Proc. IEEE 28th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun.*, Montreal, QC, 2017, pp. 1–5.

[13] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 44–52, Jun. 2019.

[14] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] H. van Hasselt *et al.*, "Deep reinforcement learning with double Q-Learning," in *Proc. 30th AAAI Conf. Artif. Intell. (AAAI-16)*, USA, 2016, pp. 2094–2100.

[16] T. Shaul *et al.*, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Representations (ICLR 2016)*, USA, 2016, pp. 1–21.

[17] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. 32th AAAI Conf. Artif. Intell. (AAAI-2018)*, USA, 2018, pp. 3215–3222.

[18] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *4th Int. Conf. Learn. Representations (ICLR 2016)* Jul. 2016, pp. 1–14.

[19] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33th Int. Conf. Mach. Learn.*, USA, 2016, pp. 1928–1937.

[20] R. Li *et al.*, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[21] C. Qi, Y. Hua, R. Li, Z. Zhao, and H. Zhang, "Deep reinforcement learning with discrete normalized advantage functions for resource management in network slicing," *IEEE Commun. Lett.*, vol. 23, no. 8, pp. 1337–1341, Aug. 2019.

[22] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "GAN-Powered deep distributional reinforcement learning for resource management in network slicing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 334–349, Feb. 2020.

[23] 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. 45758–45772, 2019.

[24] G. Sun *et al.*, "Autonomous resource provisioning and resource customization for mixed traffics in virtualized radio access network," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2454–2465, Sep. 2019.

[25] T. Li, X. Zhu, and X. Liu, "An End-to-End network slicing algorithm based on deep Q-Learning for 5G network," *IEEE Access*, vol. 8, pp. 122229–122240, 2020.

[26] G. Sun, G. O. Boateng, D. Ayepah-Mensah, G. Liu, and J. Wei, "Autonomous resource slicing for virtualized vehicular networks with D2D communications based on deep reinforcement learning," *IEEE Syst. J.*, vol. 14, no. 4, pp. 4694–4705, Dec. 2020.

[27] J. Mei, X. Wang, K. Zheng, G. Bondreau, A. Bin, and H. Abou-zeid, "Intelligent radio access network slicing for service provisioning in 6G: A hierarchical deep reinforcement learning approach," *IEEE Trans. Commun.*, to be published, doi: 10.1109/TCOMM.2021.3090423.

[28] V. García, "Deep reinforcement learning based approaches for capacity sharing in radio access network slicing," Master's thesis, Dept. Signal Theory and Communications, Barcelona, Spain, Jul. 2020.

[29] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno, "Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning," *IEEE Access*, vol. 8, pp. 68183–68198, 2020.

[30] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno, "Radio resource allocation method for network slicing using deep reinforcement learning," in *Proc. Int. Conf. Inf. Netw.*, 2020, pp. 420–425.

[31] L. Busoniu, R. Babuska, and B. De Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in Multi-Agent Systems and Applications –1* D. Srinivasan and L.C. Jain, eds., vol. 310 of Stud. Comput. Intell., Berlin, Germany: Springer, 2010, Ch. 7, pp. 183–221.

[32] Management and orchestration; 5G in Network Resource Model; Stage 2 and 3 (Release 17) document in 3GPP TS 28.541 v17.3.0, Jun. 2021.

[33] Telecommunication management; Study on the self-organizing networks (SON) for 5G networks (Release 16) document in 3GPP TR 28.861 v1.1.0, Dec. 2019.

[34] Management and orchestration; Performance assurance (Release 16) document in 3GPP TS 28.550 v16.7.0, Dec. 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/128500_128599/128550/16.05.00_60/ts_128550v160500p.pdf

[35] I. Vilà, J. Pérez-Romero, O. Sallent, and A. Umbert, "A novel approach for dynamic capacity sharing in Multi-tenant scenarios," in *Proc. IEEE 31st Int. Symp. Pers., Indoor Mobile Radio Commun.,* London, U.K., 2020, pp. 1–6.

[36] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[37] Management and orchestration; Provisioning (Release 16) document in 3GPP TS 28.531 v17.0.0, Jun. 2021. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/128500_128599/128531/16.06.00_60/ts_128531v160600p.pdf

[38] S. Guadarrama *et al.*, *TF-Agents: A Library For Reinforcement Learning in TensorFlow* (2018).

Irene Vilà (Graduate Student Member, IEEE) received the B.E. degree in telecommunication systems engineering and the M.E. degree in telecommunication engineering from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 2015 and 2017, respectively. In 2018, she joined the Mobile Communication Research Group (GRCM), Department of Signal Theory and Communications (TSC), UPC, where she is currently a Ph.D. Student, supported with an FI AGAUR grant by the Government of Catalunya. Her current research interests include radio access network (RAN) slicing, network virtualization and the application of artificial intelligence and, particularly, machine learning to radio resource management.

Jordi Pérez-Romero (Member, IEEE) received the degree in telecommunications engineering and the Ph.D. degree from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1997 and 2001, respectively. He is currently a Professor with the Department of Signal Theory and Communications, Universitat Politècnica de Catalunya. He is working in the field of wireless communication systems, with a particular focus on radio resource management, cognitive radio networks and network optimization. He has been involved in different European projects with different responsibilities, such as researcher, work package leader, and Project Responsible, has participated in different projects for private companies and has contributed to the 3GPP and ETSI standardization bodies. He has authored or coauthored more than 250 papers in international journals and conferences, three books and has contributed to seven book chapters. He has an h-index of 31 in Google Scholar. He is an Associate Editor for the IEEE VEHICULAR TECHNOLOGY MAGAZINE and *EURASIP Journal on Wireless Communications Networks*.

Oriol Sallent is currently a Professor with the Universitat Politècnica de Catalunya, Barcelona, Spain. He has participated in a wide range of european and national projects, with diverse responsibilities as a Principal Investigator, Co-ordinator, and Workpackage Leader. He regularly serves as a Consultant for a number of private companies. He has been involved in the organization of many different scientific activities, such as Conferences, Workshops, and Special Issues in renowed international journals. He has contributed to standardisation bodies such as 3GPP, IEEE, and ETSI. He is the coauthor of 13 books and has authored or coauthored more than 250 papers, mostly in high-impact IEEE journals and renowed international conferences. His research interests include 5G RAN (Radio Access Network) planning and management, artificial intelligence-based radio resource management, virtualisation of wireless networks, cognitive management in cognitive radio networks and dynamic spectrum access and management among others.

Anna Umbert (Member, IEEE) received the Engineering and Ph.D. degrees in telecommunications from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1998 and 2004, respectively. In 2001, she joined UPC, as an Assistant Professor, and became an Associate Professor in 2017 which is her current status. She has authored or coauthored more than 50 papers in international journals and conferences. Her research interests include radio resource and QoS management in the context of heterogeneous wireless networks, cognitive management in cognitive radio networks, dynamic spectrum access and management, self-organised networks and network optimisation. Since 1997, she has been participated in several projects founded by both public and private organisations.