Joint Scheduling-Offloading policies in NOMA-based Mobile Edge Computing Systems

Ibrahim Djemai
SAMOVAR, Telecom SudParis, IP Paris
91011 Evry, France
ibrahim.djemai@telecom-sudparis.eu

Mireille Sarkiss
SAMOVAR, Telecom SudParis, IP Paris
91011 Evry, France
mireille.sarkiss@telecom-sudparis.eu

Philippe Ciblat
LTCI, Telecom Paris, IP Paris
91120 Palaiseau, France
philippe.ciblat@telecom-paris.fr

***Abstract*—We consider a Non Orthogonal Multiple Access (NOMA)-based wireless network where User Equipments (UEs) are connected to a Base Station (BS) equipped with a Mobile Edge Computing (MEC) server. The UEs can process their buffered data packets with strict delay either locally or by offloading them to the base station's MEC server. In order to minimize the dropped packets due to buffer overflow or delay violation, the scheduling-offloading problem is formulated as a Markov Decision Process (MDP) and solved using various optimal and Reinforcement Learning (RL) algorithms. The output of each policy is, for each user, the number of packets to be processed and the type of processing (locally or remotely). The decisions rely on the channel state information and the buffers states. The numerical results show the great advantage of using NOMA compared to Orthogonal Multiple Access (OMA). We further analyze the scalability capabilities of the used algorithms, which validates the benefits of using Deep Reinforcement Learning (DRL) techniques.**

***Index Terms*—Scheduling, MEC, NOMA, MDP, RL**

I. Introduction

Emerging 5G and future generation networks are faced with new challenges in terms of higher data rates, lower energy and higher reliability with the ever-increasing number of connected devices and the exponential growth of mobile data traffic. To efficiently use the bandwidth resources and enable high data-rates, Non-Orthogonal Multiple Access (NOMA) was introduced as a promising technique to replace Orthogonal Multiple Access (OMA) techniques. It accommodates the high demands by allowing users to transmit in time and frequency simultaneously. On the other hand, an increase in computational complexity is perceived for the new applications and services at the end devices. As their current processing power is insufficient to cope with these demands, Mobile-Edge Computing (MEC) was proposed as a work-around. It allows the devices to offload their intensive processing to a powerful computational server located at the Base Station (BS), to be processed and sent back. As a result, the load on the end devices can be reduced and their processing sped up whilst preserving their battery life.

In such NOMA-based 5G system with MEC server at the network edge, efficient resource allocation-offloading policies are required to organise the flow of packets and take advantage of the aforementioned technologies. The policy controls the actions to be decided by the users to optimize some performance metrics. Sequential Decision Making (SDM) and Reinforcement Learning (RL) are tools that are used to devise the best policies in given known or unknown environments. To mention some of the recent works in this scope, the authors in [1] proposed a heuristic algorithm that minimizes the task average execution delay by optimizing NOMA's Successive Interference Cancellation (SIC) ordering and resource allocation in IoT networks. An online packet offloading algorithm is also developed in [2] that maximizes the long term system throughput and minimizes the signaling overhead. A multi-base stations and multi-users framework is considered in [3] where a multi-agent Deep RL (DRL) is used to jointly optimize the packet offloading, sub-station and sub-channel resource allocation, and minimize the energy consumption in the system. In [4], joint optimization of offloading policy (including a packet splitting for partial offloading) and channel resource allocation (*i.e.* NOMA/OMA sub-carrier allocation) using DRL is also proposed. Moreover, a DRL-based online packet offloading in multi-carrier NOMA-enabled MEC networks is developed in [5] to optimize sub-carrier allocation. Finally, the work in [6] focused on an ultra-dense network, with small BSs and clustering, to jointly optimize user clustering, and resource and power allocation.

In this work, we investigate a multi-user system with NOMA users and a BS equipped with MEC functionalities. We aim at devising efficient resource scheduling-offloading policies that minimize the total packet loss, under the strict delay constraint for the task, which was introduced first in [7]. Unlike this latter work (which considers only OMA), we allow NOMA communications. Therefore, this paper focuses on the benefit of using NOMA when comparing to an OMA-based scenario, i.e Time Division Multiple Access (TDMA) scenario, even with the computational complexity that comes with it. In addition, using sequential decision making methods, we propose offline optimal and reinforcement learning algorithms, and show their superior results compared to other naive scheduling methods. Finally, we experiment with a higher state space configuration that illustrate how well the used algorithms scale, and show that Deep Q-Learning is the method of choice when expanding the system state space.

The remainder of the paper is organized as follows. In Section II, we describe the system model. In Section III, we formulate the optimization problem as an MDP and solve it using various offline and reinforcement learning algorithms.

In Section IV, we present and analyze the numerical results. Finally, we give some concluding remarks in Section V.

II. System Model

We consider 2 User Equipments (UEs) served by a BS with MEC server. The UEs are equipped with buffers to store the application data that need to be processed within a strict delay. The decision for offloading or processing locally is made at the BS level at the beginning of each time slot of size T^s. The decision is then broadcasted to the UEs for free. Therefore, at each time slot, the UEs process either their data packets *locally* using their processor, or *remotely* by offloading them to be processed at the BS and then receive the results. When both UEs offload their packets to the MEC server, NOMA is considered as the multiple access technique. In the sequel, we provide more details on the buffer model and strict delay constraint, the channel model, the NOMA rates in uplink and downlink, and the scheduling decisions with their corresponding time constraints.

A. UE Buffer Model

The buffer at each UE$_i$, $i \in \{1,2\}$, is modeled as a vector $\mathbf{b}_i$ of size B_d. Each position is labeled with the age of the data packet in it, while -1 is used to represent an empty slot. The data packets are ordered from oldest to newest, and we refer to the age of the data packet at time slot n at the position j as $b_i^j[n]$, therefore we have $b_i^j[n] \geq b_i^{j'}[n]$ if $j < j'$. The number of packets present in the buffer is denoted q_i. Fig. 1 shows the model described.

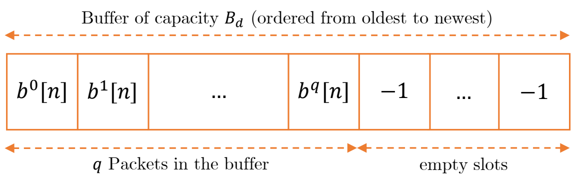
Buffer of capacity B_d (ordered from oldest to newest)
$b^0[n]$
$b^1[n]$
...
$b^q[n]$
-1
...
-1
q Packets in the buffer
empty slots

Fig. 1. Buffer Model

At each time slot, a new batch of data packets $d_i[n]$ arrives at the buffer with age 0 and increases after each time step. The packet arrival is assumed to follow the Poisson Random Distribution, with mean λ_d :

$$p(d_i[n] = d) = e^{-\lambda_d} \cdot \frac{(\lambda_d)^d}{d!} \tag{1}$$

If the number of empty slots is smaller than number of arrived packets at a given time, i.e., $B_d - q_i < d_i[n]$, then the difference number of packets will be dropped due to *buffer overflow*. Additionally, after updating the age of the packets in the buffer, each data packet that reaches the maximum strict delay K_0 will be dropped due to *delay violation*.

B. Channel Model

The channel between the UEs and the BS is modeled as a Rayleigh flat-fading channel, with an Additive White Gaussian Noise (AWGN) of power spectral density N_0. The bandwidth is denoted W^{DL} for the downlink (DL) and W^{UL} for the uplink (UL). The channel response is constant during a time slot, with h_i the complex amplitude of the channel for UE$_i$, $i \in \{1,2\}$, and $g_i = |h_i|^2$ its gain. We assume no channel correlation across successive time slots. We model the variation of the channel gain by the continuous random variables $g_i = g$ following the exponential distribution with mean ξ:

$$p(g) = \frac{1}{\xi} e^{\frac{-g}{\xi}} \tag{2}$$

We assume that the BS makes its decision based on a quantized version of the channel. In practice, the BS broadcasts a training sequence, the UEs then estimate their channel, and send back a quantized version in order to limit the size of the overhead. The quantized channel gain for UE$_i$ is defined as $\tilde{g}_i = Q(g_i)$, where Q is the quantization function and $\tilde{g}_i$ is the lower value of the interval in which h_i belongs to. We also denote $\tilde{g}_i^+$ as the upper value of that interval. Notice that $\tilde{g}_i^+$ can be deduced from $\tilde{g}_i$. The finite set of the quantized channel states is $\mathcal{G}$.

C. Non-Orthogonal Multiple Access (NOMA)

NOMA is a multiple access technique that allows the superposition of UEs signals in time and frequency, using distinct codes (Code Domain NOMA), or distinct power levels (Power Domain NOMA), thus achieving a higher spectral efficiency compared to Orthogonal Multiple Access methods. Successive Interference Cancellation (SIC) is used to decode the signals at the receiving side. In our case, we consider Power Domain NOMA.

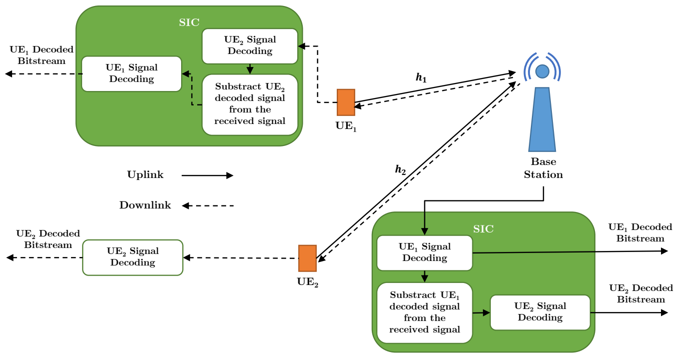

Fig. 2. Uplink and Downlink NOMA scenarios

1) Uplink Case: In our scenario with 2 UEs, we consider UE$_1$ to have a better channel gain than UE$_2$: $\tilde{g}_1 > \tilde{g}_2$. (This order can be swapped between the UEs at a given time step as the equations remain the same but with interchangeable indices. This holds true throughout the article). UEs send their signals x_i with offloading powers P_i^o, for $i \in \{1,2\}$. The BS receives a combination of both signals expressed as:

$$y = \sum_{i=1}^{2} \sqrt{P_i^o} h_i x_i + w \tag{3}$$

where w is the AWGN noise. The BS proceeds into decoding the signals using SIC, as illustrated in Fig. 2. The obtained rate expressions are given by the equations below. The use of

$\tilde{g}_2^+$ in Eq. (4) is to consider the worst-case scenario for the interference term when calculating the rate for UE$_1$. :

$$C_1^{\text{NOMA, UL}} = W^{\text{UL}} \log_2\left(1 + \frac{P_1^o \cdot \tilde{g}_1}{P_2^o \cdot \tilde{g}_2^+ + W^{\text{UL}} N_0}\right) \quad (4)$$

$$C_2^{\text{NOMA, UL}} = W^{\text{UL}} \log_2\left(1 + \frac{P_2^o \cdot \tilde{g}_2}{W^{\text{UL}} N_0}\right) \quad (5)$$

2) Downlink Case: The BS has power P_s, where $\alpha_i P_s$ is allocated to UE$_i$ with $\sum_{i=1}^{2} \alpha_i = 1$. The users UE$_1$ and UE$_2$ receive y_i for $i \in \{1, 2\}$:

$$x = \sum_{i=1}^{2} \sqrt{\alpha_i P_s} \cdot x_i, \quad y_i = h_i \cdot x + w_i \quad (6)$$

With SIC decoding as in Fig. 2, the corresponding rate expressions are as following, with $\alpha_1 = \alpha$ and $\alpha_2 = 1 - \alpha$:

$$C_1^{\text{NOMA, DL}} = W^{\text{DL}} \log_2\left(1 + \frac{\alpha P_s \cdot \tilde{g}_1}{W^{\text{DL}} N_0}\right) \quad (7)$$

$$C_2^{\text{NOMA, DL}} = W^{\text{DL}} \log_2\left(1 + \frac{(1-\alpha) P_s \cdot \tilde{g}_2}{\alpha P_s \cdot \tilde{g}_2 + W^{\text{DL}} N_0}\right) \quad (8)$$

D. Scheduling Decisions and Time Constraints

Three different decisions can be made at the beginning of a time slot n of duration T^s: *Idle*, *Local Processing* or *Offloading* for each UE.

- *Idle* : The UE will not execute any packet, $u_i = 0$.
- *Local Processing* : The UE can process locally up to U^ℓ packets during the time slot.
- *Offloading* : The UE will offload up to U^o packets to the BS, to be processed and returned, all within the time slot.

In the offloading case, the operation includes 4 phases (transmission, waiting, reception and decoding) indicated in the equations below, and has to be done within the time slot duration T^s. We distinguish two sub-cases:

- *Only one of the users* UE$_i$, $i = 1$ or $i = 2$, offloads $u_i < U^o$ packets to the MEC server:

$$u_i\Big(\underbrace{\frac{L^{UL}}{C_i^{\text{TDMA, UL}}}}_{\text{Transmission - } P_i^o} + \underbrace{T^w}_{\text{Waiting time}} + \underbrace{\frac{L^{DL}}{C_i^{\text{TDMA, DL}}}}_{\text{Signal reception time}} + \underbrace{\frac{1}{\beta} \cdot \frac{L^{DL}}{C_i^{\text{TDMA, DL}}}}_{\text{Decoding time}} \Big) \leq T^s \quad (9)$$

where $L^{\text{UL}}, L^{\text{DL}}$ are the number of bits transmitted (UL) and received (DL) respectively, $\beta \in (0, 1]$ is the decoding efficiency, and $C_i^{\text{TDMA, UL}}$, $C_i^{\text{TDMA, DL}}$ are the respective rates for UL and DL defined by:

$$C_i^{\text{TDMA, UL}} = W^{\text{UL}} \cdot \log_2\left(1 + \frac{P_i^o \cdot \tilde{g}_i}{W^{\text{UL}} \cdot N_0}\right) \quad (10)$$

$$C_i^{\text{TDMA, DL}} = W^{\text{DL}} \cdot \log_2\left(1 + \frac{P^s \cdot \tilde{g}_i}{W^{\text{DL}} \cdot N_0}\right). \quad (11)$$

- *Both users* offload : In this case, NOMA will be used and the execution time at each UE will depend on the NOMA rates previously defined in Eqs. (4, 5, 7, 8).

$$u_1\Big(\underbrace{\frac{L^{UL}}{C_1^{\text{NOMA, UL}}}}_{\text{Transmission - } P_1^o} + \underbrace{T^w}_{\text{Waiting time}} + \underbrace{\frac{L^{DL}}{\min(C_2^{\text{NOMA, DL}}, C_1^{\text{NOMA, DL}})}}_{\text{Signal reception time}} + \underbrace{\frac{1}{\beta} \cdot \frac{L^{DL}}{C_1^{\text{NOMA, DL}}} + \frac{1}{\beta} \cdot \frac{L^{DL}}{C_2^{\text{NOMA, DL}}}}_{\text{Decoding time}} \Big) \leq T^s \quad (12)$$

$$u_2\Big(\underbrace{\frac{L^{UL}}{C_2^{\text{NOMA, UL}}}}_{\text{Transmission - } P_2^o} + \underbrace{T^w}_{\text{Waiting time}} + \underbrace{\frac{L^{DL}}{C_2^{\text{NOMA, DL}}}}_{\text{Signal reception time}} + \underbrace{\frac{1}{\beta} \cdot \frac{L^{DL}}{C_2^{\text{NOMA, DL}}}}_{\text{Decoding time}} \Big) \leq T^s \quad (13)$$

As described in section II-C, we assume that UE$_1$ has a better channel gain than UE$_2$, $\tilde{g}_1 > \tilde{g}_2$. Therefore, when receiving both signals using the slowest rate of the two, $\min(C_2^{\text{NOMA, DL}}, C_1^{\text{NOMA, DL}})$, UE$_1$ has to decode UE$_2$'s signal before decoding its own signal, which explains the two terms of the decoding time in Eq. (12). Moreover, by setting the equality in Eqs. (9), (12) and (13), we can extract the optimal offloading powers P^o, which cannot exceed a threshold $P^{\max}$.

III. Problem Formulation and Resolution

A. Problem Formulation

In this section, we formulate our scheduling problem (how many packets to schedule for computing and at which level: locally or remotely) as a Markov Decision Process (MDP). The problem boils down to exhibit policies that minimize the overall data packet loss due to either buffer overflow or delay violation. These policies output an action to be made at the beginning of each time step, given some information on the system (e.g. channel conditions, number of packets in the buffer and their age, multiple access). Here the policies we developed correspond to the optimal ones, based on the MDP model, as well as those obtained thanks to the model-free Reinforcement Learning approach.

The MDP models an environment using a state space $\mathcal{S}$, an action space $\mathcal{A}$, a (state) transition probability matrix $\mathcal{T}$, and an expected cost matrix $\mathcal{C}$. Consequently, at each time step n, an agent is at a state $\mathbf{s}[n]$, takes an action $a[n]$, leading to a new state $\mathbf{s}[n+1]$ according to the matrix $\mathcal{T}$ (independent of time), while receiving a cost $c[n+1]$. In our setup, the state space represents all the useful information required for making a relevant decision on the scheduling. From that, this space $\mathcal{S}$ contains the information on the buffer and channel status for both users, and a state is defined as:

$$\mathbf{s} = (\mathbf{b}_1, \mathbf{b}_2, \tilde{g}_1, \tilde{g}_2), \quad \mathbf{s} \in \mathcal{S}.$$

The actions are the following ones: offloading packets or processing locally packets or idle, and the number of packets u_i to be processed for each UE$_i$ within the limit defined for each type (U^o for offload and U^ℓ for local).

The problem at hand is considered as MDP, since the transition from a state $\mathbf{s}[n]$ to a state $\mathbf{s}[n+1]$ depends only on the current state $\mathbf{s}[n]$ and the current action $a[n]$. We denote $w_i[n] = \max(u_i[n], r_i[n])$ as the number of packets that leave the buffer of user i after taking an action $a[n]$, where $r_i[n]$ is the number of packets that have reached the maximum delay K_0 after incrementing the age of all packets by 1.

The state transition probability for our model is given by:

$$p(\mathbf{s}'|\mathbf{s}, a) = p(\mathbf{b}_1'|\mathbf{b}_1, a) \cdot p(\mathbf{b}_2'|\mathbf{b}_2, a) \cdot p(\tilde{g}_1') \cdot p(\tilde{g}_2') \tag{14}$$

where $p(\tilde{g}_i')$ is the distribution of the channel gain $\tilde{g}_i'$, and $p(\mathbf{b}_i'|\mathbf{b}_i, a)$ is the probability transition between two buffer states for user i.

We hereafter define the range of possible actions and next state transitions, where for a state $\mathbf{s}$, each next state $\mathbf{s}'$ and action a have to satisfy the below-mentioned conditions.

- The offloading powers corresponding to the chosen action a are less than or equal to the maximum offloading power, $P_i^o \le P^{\max}$.
- The number of processed packets is less than or equal to the size of the buffer, $u_i \le q_i$.
- The age difference between the same buffer slot in the states $\mathbf{s}$ and $\mathbf{s}'$ is less than or equal to 1,
$$b_i^{j'} \le b_i^j + 1,\ j = \{0, \cdots, B_d\}.$$
- The size of the next buffer state is greater than or equal to the difference between the size of the current buffer and the number of packets leaving the buffer, $q_i' \ge q_i - w_i$.
- The age of the next state packets is bigger than the same current packets ages by 1,
$$\textbf{if } b_i^{j+w_i} \ne -1 \textbf{ then } b_i^{j'} = b_i^{j+w_i} + 1,\ j = \{0, \cdots, B_d - w_i\}.$$
- The age of the next state packets is less than or equal to 0 for empty slots in the current buffer state,
$$\textbf{if } b_i^{j+w_i} = -1 \textbf{ then } b_i^{j'} \le 0,\ j = \{0, \cdots, B_d - w_i\}.$$
- The empty slots in the current buffer, when $w_i \ne 0$ packets leave the buffer, have an age of 0 or less in the next state,
$$\textbf{if } q_i = B_d \textbf{ and } w_i \ne 0$$
$$\textbf{then } b_i^{j'} \le 0,\ j = \{B_d - w_i, \cdots, B_d\}.$$

If these conditions are satisfied, the probabilities of transition for the buffers are given by:

if $q_i' < B_d$ **then**:

$$p(\mathbf{b}_i'|\mathbf{b}_i, a) = e^{-\lambda_d} \cdot \frac{(\lambda_d)^{q_i' - q_i + w_i}}{(q_i' - q_i + w_i)!}$$

else:

$$p(\mathbf{b}_i'|\mathbf{b}_i, a) = 1 - e^{-\lambda_d} \sum_{j=0}^{B_d - q_i + w_i - 1} \frac{\lambda_d^j}{j!}$$

We consider the cost function as an infinite-horizon discounted function (with factor γ) taking into account the packet losses. Therefore, we get

$$J(\pi) = \lim_{N \to \infty} \mathbb{E}^\pi \Big[\sum_{n=0}^{N} \gamma^n (c^v(\mathbf{s}[n], a[n]) + c^o(\mathbf{s}[n], a[n])) \Big] \tag{15}$$

where

- π is the policy for which the cost function is evaluated.
- $c^v(\mathbf{s}[n], a[n]) = w_1[n] - u_1[n] + w_2[n] - u_2[n]$ is the instantaneous cost due to delay violation, i.e. the number of packets that reach the maximum delay K_0 and are discarded.
- $c^o(\mathbf{s}[n], a[n]) = \sum_{i \in \{1,2\}} \sum_{j = B_d - q_i[n] + w_i[n] + 1}^{+\infty} (q_i[n] - w_i[n] + j - B_d) \cdot e^{-\lambda_d} \cdot \frac{(\lambda_d)^j}{j!}$ is the instantaneous cost due to buffer overflow, which is the number of arrived packets following the Poisson distribution that could not enter the buffer.

Our goal is to find the optimal policy $\pi^\star$ according to the minimization of the cost function defined in Eq. (15):

$$\pi^\star = \arg\min_{\pi}\ J(\pi). \tag{16}$$

B. Problem Resolution

To solve this optimization problem, we use a variety of Sequential Decision Making SDM algorithms. SDM techniques are aimed at producing the best policies π that optimize the behavior of an agent in a given environment.

Model-based methods, e.g. Value Iteration (VI) and Policy Iteration (PI), rely on full knowledge of the environment dynamics (Transition model $\mathcal{T}$ and Cost model $\mathcal{C}$) to produce the optimal policy $\pi^\star$ in an offline procedure. In the case of VI, the optimization is executed by incrementally improving a randomly initialized value function $V(\mathbf{s})$ (a function that estimates the quality of being at each state) using the Bellman equation until converging to the optimal solution with a stopping criteria. It is described by the following equation:

$$V^k(\mathbf{s}) = \min_{a \in \mathcal{A}} \Big(\mathcal{C}[\mathbf{s}, a] + \gamma \sum_{\mathbf{s} \in \mathcal{S}} \mathcal{T}[\mathbf{s}'|\mathbf{s}, a] \cdot V^{k-1}(\mathbf{s}') \Big) \tag{17}$$

where $V^k(\mathbf{s})$ takes the minimum value possible by an action a at iteration k.

As for PI, the algorithm runs two phases iteratively until convergence: Policy Evaluation and Policy Improvement. The first phase evaluates the value function at iteration k using the previous iteration quantities of the value function $V^{k-1}(\mathbf{s})$ and the policy $\pi^{k-1}(a|\mathbf{s})$. It is described as following:

$$V^k(\mathbf{s}) = \sum_{a \in \mathcal{A}} \pi^{k-1}(a|\mathbf{s}) \Big(\mathcal{C}[\mathbf{s}, a] + \gamma \sum_{\mathbf{s} \in \mathcal{S}} \mathcal{T}[\mathbf{s}'|\mathbf{s}, a] \cdot V^{k-1}(\mathbf{s}') \Big) \tag{18}$$

The second phase improves on the policy $\pi^k(a|\mathbf{s})$ by choosing the actions a that minimize the value function at each state $\mathbf{s}$ according to Eq. (19). The algorithm converges when the policy stops changing, $\pi^\star = \pi^k = \pi^{k-1}$:

$$\pi^k(a|\mathbf{s}) = \begin{cases} 1 & \text{if } a = \arg\min_{a \in \mathcal{A}} V^k(\mathbf{s}) \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

On the other hand, model-free methods, e.g. Q-Learning (QL), attempt to explore the environment to fill its state-action value matrix $Q(\mathbf{s}, a)$. Similar to the value function, the state-action value function estimates the quality of being in a given state and taking a given action. The process is done in a trial-and-error manner, where a balance between exploration and exploitation is needed using the exploration rate ϵ to insure convergence after enough episodes. An episode corresponds to a sequence of time step starting from a random state with no terminal state in the system. Obviously, the $Q(\mathbf{s}, a)$ obtained at the end of an episode carries over to the beginning of the next episode. The update for the state-action value matrix $Q^{new}(\mathbf{s}, a)$ is detailed in the equation below:

$$Q^{new}(\mathbf{s}[n], a[n]) = Q^{old}(\mathbf{s}[n], a[n]) + \ell \cdot \Big(c[n+1] + \gamma \cdot \max_{a' \in \mathcal{A}} Q^{old}(\mathbf{s}[n+1], a') - Q^{old}(\mathbf{s}[n], a[n]) \Big) \quad (20)$$

The action $a[n]$ is chosen using an ϵ-Greedy approach, which selects the best available action with probability $1 - \epsilon$ and a random action otherwise. $c[n + 1]$ is the instantaneous cost (sum of dropped packets) after following action $a[n]$ and transitioning to a new state $\mathbf{s}[n + 1]$. ℓ is the learning rate that scales the correction step taken at each update.

Another variant is Deep Q-Learning (DQN) [8], a method that approximates the state-action value function using Neural Networks (NN). It scales well to larger state spaces as we shall demonstrate in the simulation results.

The training of the NN, called the Evaluation Network (EN), is done using a secondary network, called the Target Network (TN). The TN is a frozen copy of the EN and gets updated every few episodes to ensure the stability of the training and the convergence of the EN. The agent is run through the environment, where the current state $\mathbf{s}[n]$ is fed to the EN with a set of parameters θ. The EN outputs the state-action values $q_\theta(\mathbf{s}[n], a[n])$, and an action $a[n]$ is extracted following the ϵ-Greedy approach. The tuple $(\mathbf{s}[n], a[n], c[n + 1], \mathbf{s}[n + 1])$ gets stored in an over-writable data buffer to be used later. Once the buffer is filled sufficiently, a batch of randomly sampled tuples are chosen from it and exploited at the TN with the set of parameters $\hat{\theta}$ to calculate the quantity for each item in the batch $q_{\hat{\theta}}(\mathbf{s}[n + 1], a[n + 1])$. This quantity is then used to calculate the Bellman step for $q_{\hat{\theta}}(\mathbf{s}[n], a[n])$ as follows:

$$q_{\hat{\theta}}(\mathbf{s}[n], a[n]) = c[n+1] + \gamma \cdot \max_{a \in \mathcal{A}} q_{\hat{\theta}}(\mathbf{s}[n+1], a[n+1]) \quad (21)$$

We compute the Mean Square Error (MSE) loss between $q_{\hat{\theta}}(\mathbf{s}[n], a[n])$ and $q_\theta(\mathbf{s}[n], a[n])$, denoted as $\mathcal{L}(q_\theta, q_{\hat{\theta}})$, and we use it to update the weights of the EN with the mini-batch gradient descent method and a learning rate ℓ.

IV. Numerical Results

We consider a buffer of size $B_d = 3$, a maximum delay $K_0 = 2$ and a time slot duration of $T^s = 1\ ms$. The maximum packets that can be offloaded is $U^o = 3$, while the maximum that can be executed locally is $U^\ell = 1$. The offloaded packets are of size $L^{\text{UL}} = 1000\ bits$ and are sent with an offload power not exceeding $P^{\max} = 2\ mW$ through a quantized Rayleigh-faded channel with $m = 3$ channel gain states $\mathcal{G} = \{-20, -1.487, 1.492\}\ dB$. Uplink and downlink bandwidths are set to $W^{\text{UL}} = 1\ MHz$ and $W^{\text{DL}} = 5\ MHz$, respectively. The channel noise spectral density is $N_0 = -87\ dBm/Hz$. We consider $T^w = 0.1\ ms$. The size of the packets in downlink is $L^{\text{DL}} = 100\ bits$ and these packets are sent back to the UEs with a power $P^s = 50\ W$ and a power allocation coefficient $\alpha = 0.5$ (in the case of NOMA). The decoding efficiency β is set to 1.

Concerning the algorithms, the stopping criterion for the iterations on the Value Iteration algorithm is a gap on value function update of $\delta = 10^{-7}$. In addition, the Q-Learning was trained over $5 \cdot 10^4$ episodes of 5000 time steps, and the learning rate ℓ is decayed linearly, from 10^{-2} to 10^{-3}. The Deep Q-Learning (DQN) was trained over 10^3 episodes of 5000 time steps, with a fixed learning rate $\ell = 5 \cdot 10^{-3}$, a buffer size of $256 \cdot 10^2$ tuples, and a batch size of 256. Moreover, the used NN was a Multi-Layer Perceptron (MLP) with 3 hidden layers of 64 neurons. Both QL and DQN use an exploration rate ϵ that decreases exponentially along the training, from 1 to 0.01, to achieve better exploration of the environment. The discounted factor γ is set to 0.99.

In following figures, we consider the next algorithms: Value Iteration (VI), Policy Iteration (PI), Q-Learning (QL) and Deep Q-Learning (DQN), compared to some naive methods. These naive methods are: naive local (NL), where the UEs are only allowed to execute their packets locally, naive offload (NO), where the UEs can only offload their packets to the MEC server, and naive random (NR), where the UEs choose a random action at each time step. The figures display the overall discount cost averaged over 10^3 episodes of 20000 time steps.

In Fig. 3.top, we consider the NOMA case, i.e. both UEs can offload at the same time. We remind that VI and PI are optimal. We notice that Q-Learning and Deep Q-Learning offer remarkable performance while they are model-free, With Q-Learning performing better than Deep Q-Learning (since the latter is an approximation of the former). Previous algorithms outperform the naive methods, proving the effectiveness of using a sequential decision making or a reinforcement learning approach.

In Fig. 3.bottom, we consider only the TDMA case, where only one UE is allowed to offload at a given time while the other one can still process its packets locally. We have similar comments except that the Q-Learning and Deep Q-Learning achieve the optimality since the space to explore is strongly reduced without NOMA operations.

However, there is a noticeable difference between the performances of the used SDM techniques in the NOMA setting and the TDMA setting. Indeed, the introduction of NOMA in the system improves the overall discounted cost, and gives a cost up to 1.8 times lower compared to TDMA. Therefore, we showcase that the use of NOMA is beneficial for our system, even with a higher state space configuration, hence a slower convergence time.

In Fig. 4, we plot a pie chart for the percentage of the different actions taken by each algorithm. For instance, Value and Policy Iteration take opposite approaches, in the sense that Policy Iteration prefers local operations and offloading without NOMA, whereas Value Iteration has an "offload while you can" method utilizing more NOMA. Q-Learning and Deep Q-Learning balance the processing actions between local and offload. Therefore, NOMA is not used as often as in Value Iteration or rarely as in Policy Iteration.

We increase the state space size by considering a buffer size $B_d = 4$, a number of channel states $m = 4$, and a maximum packets delay $K_0 = 3$. The resulting state space is of size 78400 possible states compared to 3600 in the previous setup. Fig. 5 shows the scalability performance of Q-Learning and Deep Q-Learning (PI and VI are not included as they take exponentially longer time to converge). We can see that DQN still performs better than the naive methods, while Q-Learning falls behind the Naive Offload method. This result affirms that the use of Deep Q-Learning is essential when scaling the system setup.

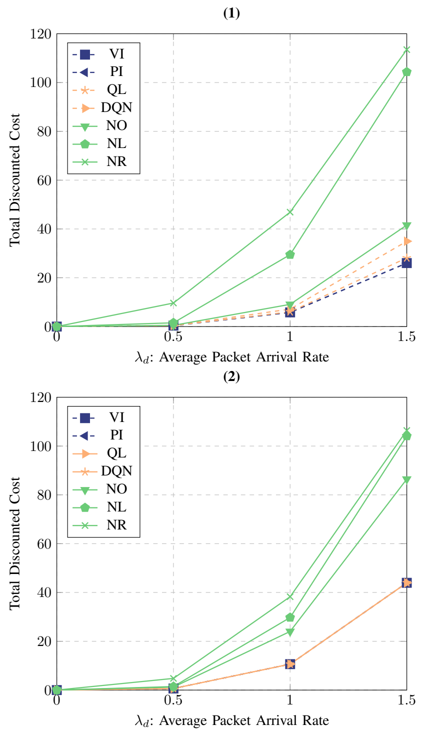

Fig. 3. Total discounted cost averaged over 1000 episodes vs average packet arrival rate λ_d, with NOMA (1/top), and TDMA (2/bottom)

V. Conclusion

This paper introduced the problem of jointly optimizing scheduling and offloading in a NOMA environment with 2 UEs and a MEC server at the base station. It is solved by using sequential decision making methods including optimal value iteration/policy iteration algorithms and other model-free reinforcement learning algorithms. In that context, NOMA technique has offered better performance than TDMA. Nevertheless, if the state space gets larger, scaling issue will arise with some of the used algorithms, which makes the use of Deep Q-Learning ever more important in larger systems.

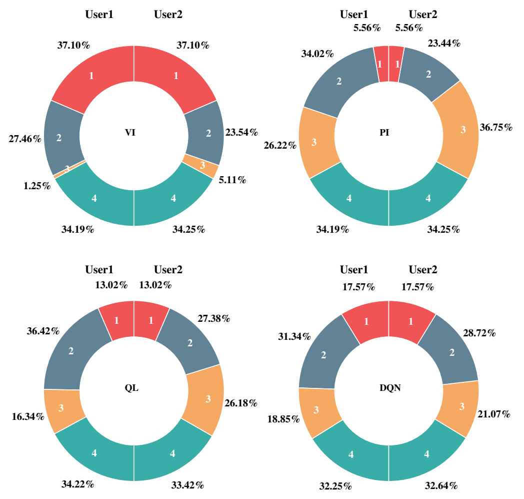

Fig. 4. Pie Chart of the percentage of actions taken during an episode with $\lambda_d = 1.0$ for VI (NW), PI (NE), DQL (SW), and QL (SE) algorithms. (1/red) = NOMA, (2/gray) = Regular Offload, (3/orange) = Local, (4/green) = Idle

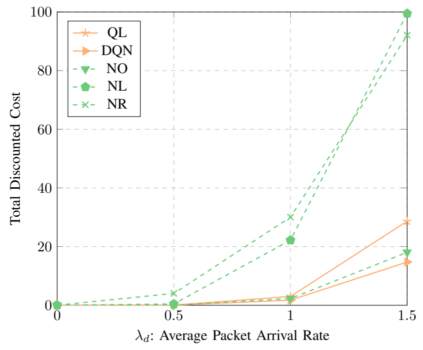

Fig. 5. Performance of Reinforcement Learning Algorithms : Q-Learning (QL) and Deep Q Learning (DQN) with a Bigger State Space Configuration $(B_d = 4, m = 4, K_0 = 3)$

References

[1] L. P. Qian, A. Feng, Y. Huang, Y. Wu, B. Ji, and Z. Shi. Optimal SIC Ordering and Computation Resource Allocation in MEC-aware NOMA NB-IoT Networks. *IEEE Internet of Things Journal*, 6(2):2806–2816, 2018.

[2] M. Hua, H. Tian, X. Lyu, W. Ni, and G. Nie. Online Offloading Scheduling for NOMA-aided MEC under Partial Device Knowledge. *IEEE Internet of Things Journal*, 2021.

[3] W. Liu, Y. He, J. Zhang, and J. Qiao. Deep Reinforcement Learning-Based MEC Offloading and Resource Allocation in Uplink NOMA Heterogeneous Network. In *2021 Computing, Communications and IoT Applications (ComComAp)*, pages 144–149. IEEE, 2021.

[4] T.P. Truong, T. Nguyen, W. Noh, S. Cho, et al. Partial Computation Offloading in NOMA-assisted Mobile-Edge Computing Systems Using Deep Reinforcement Learning. *IEEE Internet of Things Journal*, 8(17):13196–13208, 2021.

[5] M. Nduwayezu, Q.-V. Pham, and W.-J. Hwang. Online Computation Offloading in NOMA-based Multi-access Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Access*, 8:99098–99109, 2020.

[6] L. Li, Q. Cheng, X. Tang, T. Bai, W. Chen, Z. Ding, and Z. Han. Resource allocation for NOMA-MEC Systems in Ultra-Dense Networks: A Learning Aided Mean-Field Game Approach. *IEEE Transactions on Wireless Communications*, 20(3):1487–1500, 2020.

[7] I. Fawaz, M. Sarkiss, and P. Ciblat. Delay-optimal Resource Scheduling of Energy Harvesting-based Devices. *IEEE Transactions on Green Communications and Networking*, 3(4):1023–1034, 2019.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.