Delay Minimization for Edge Computing with Dynamic Server Computing Capacity: A Learning Approach

Burak Yılmaz
Technische Universität Darmstadt
b.yilmaz@nt.tu-darmstadt.de

Andrea Ortiz
Technische Universität Darmstadt
a.ortiz@nt.tu-darmstadt.de

Anja Klein
Technische Universität Darmstadt
a.klein@nt.tu-darmstadt.de

Abstract*—The offloading decisions of K mobile users (MUs) aiming at minimizing the execution delay in a Mobile Edge Computing (MEC) scenario with non-orthogonal multiple access is considered. In this work, we assume a time-varying MEC server computing capacity which exploits additional computing resources that are freed over time, but are not known beforehand. In this setting, the optimal offloading decision depends on the different tasks of MUs, their channel fading processes and the MEC server computing capacity. We first formulate the optimization problem and identify two main challenges, namely, how to exploit the given incomplete knowledge to minimize the delay and how to handle the high dimensionality of the problem. To address these challenges, we propose a novel reinforcement learning (RL) algorithm, termed combinatorial offloading learning (COL). The name stands for its ability to handle the combinatorial nature of the solutions. Exploiting the available knowledge, we learn the offloading decision policy aiming at minimizing the delay. Furthermore, we handle the curse of dimensionality, typical of combinatorial problems, by splitting the learning task, solving $K+1$ smaller RL problems and using linear function approximation. Through numerical simulations, we show that COL could perform similar to a short term optimal solution with complete information and exhaustive search, and outperforms known strategies like the greedy approach.

I. Introduction

Mobile Edge Computing (MEC) is a key technology that exploits its proximate access to users to enable applications with latency critical requirements like augmented reality and connected cars [1]. Computation offloading addresses the limitation of available resources in mobile and smart devices. Though offloading computational tasks to a more powerful server is desirable to save energy and computation time for users, MEC requires a policy to decide when to offload computational tasks to efficiently use the limited computing and communication resources available [2].

As an important application of 5G communications and beyond, strategies for computation offloading scenarios are being studied extensively in multiple user settings [3]. In [4], the energy-latency trade-off in multiuser MEC systems is investigated, which jointly manages the available radio and computing resources. The authors of [5] investigate the latency minimization problem with a prescribed resource utilization constraint. They propose a polynomial-time approximate solution with guaranteed performance. In [6], an integer optimization problem is formulated to minimize the energy consumption and offloading latency, then game theoretic techniques are applied to develop a distributed algorithm. In [7], a heuristic offloading decision algorithm is proposed. It jointly optimizes the offloading decision and communication and computing resource allocation to minimize the latency and energy consumption. The authors of [8] investigate joint radio and computing resource allocation, where they consider interference between multiple virtualizations for computation in the MEC server. They maximize joint throughput and minimize energy consumption. In [9], in order to minimize the energy and delay cost, offloading decisions and the allocation of communication resource are jointly optimized. In [10], the authors propose an online algorithm that optimally adapts offloading decisions and wireless resource allocations to the time-varying wireless channel conditions. The authors of [11] investigate joint resource allocation and offloading decision optimization, using mixed integer nonlinear programming.

Even though the mentioned studies contributed significantly to the development of computation offloading scenarios, they also have their own limitations. In this paper, we focus on the computation offloading decisions made for K mobile users (MUs) in a scenario with a single MEC server with computing resources. First, in contrast to [7], [10], [11], we consider a variety of tasks with different computing resource requirements. This task variety is an important aspect to account for because the tasks are associated to the users' interests and needs [12]. The studies in [4], [6], [9] focused on the effects of channel reuse and channel competition for offloading scenarios. However, the effect of the limited computing resources in the considered offloading server was not taken into account. Therefore secondly, in our work we consider a MEC server with limited computing resource for offloading and we assume that information of this limited resources is an unknown parameter for offloading. This assumption is an open challenge that is pointed out in [3] and could be explained with an exemplary scenario. Consider two groups of users with different priorities served by a single MEC server. When the high priority group needs more resources, it would be allocated a larger portion of the MEC server computing resources, where

This work has been performed in the context of the DFG Collaborative Research Center (CRC) 1053 MAKI and the Center emergenCITY of the LOEWE initiative (Hessen, Germany).

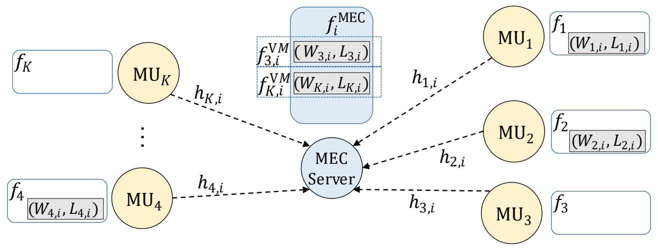

Fig. 1: MEC scenario with computation offloading.

the low priority group is served with a dynamic computing capacity which is left for their offloaded computation. Within this example, the low priority group would be served with time-varying computing resources, and if some additional computing resources are freed while the given tasks of the low priority group are being transmitted to the MEC, these resources could be also allocated in order to further speed up the offloaded tasks. Third, we consider non-orthogonal multiple access (NOMA) for offloading transmissions in our scenario. NOMA is a promising technology of accommodating several users within the same resource block. By doing so, significant bandwidth efficiency enhancement can be attained over conventional orthogonal multiple-access techniques [13].

In this paper, we first formulate the optimization problem for the offloading decisions and identify two main challenges. The first is to find an offloading decision policy aiming at minimizing the execution delay experienced by MUs, considering a scenario in which no knowledge about the amount of computing resources that would be allocated for offloading is available. This dynamic MEC server computing capacity is assumed to be unknown and time-varying. The second is to handle the high dimensionality of the problem. This is due to the combinatorial nature of the offloading decisions and the infinite number of fading channel conditions the MUs can experience. To address these challenges, we formulate the offloading decision problem as a Multi-armed Bandit (MAB) problem and use reinforcement learning (RL) approach. Specifically, we propose an RL algorithm termed combinatorial offloading learning (COL). The name of the algorithm represents its ability to handle the combinatorial nature of the offloading decision solutions. Our algorithm is inspired by the strategy proposed in [14] for energy harvesting multiple access scenarios. We combine our proposed algorithm with linear function approximation to manage the infinite number of states. The advantage of our proposed algorithm is its ability to split the original RL problem into $K+1$ smaller problems, thus greatly increasing the computational efficiency.

The rest of the paper is organized as follows. In Sec. II, the system model is presented. The execution delay minimization is formulated in Sec. III and the implemented algorithm is explained in Sec. IV. Simulation results are presented in Sec. V and Sec. VI concludes the paper.

II. System model

A MEC scenario consisting of a single server and K MUs, termed MU$_k$, $k = 1, ..., K$, as depicted in Fig. 1, is considered. Time is divided into time slots (TSs) and each TS is identified by its index i. In each TS, every MU receives a task that it wants to compute with minimum execution delay. This delay is defined as the time between the moment a computational task is received at MU$_k$ and the result of this computation is available at MU$_k$. We assume the tasks are non-splittable, i.e., MUs can either compute their tasks locally or completely offload their task.

We consider a centralized controller at the MEC server, which is referred to as just MEC server from here on, that is allocating communication and computing resources to each MU. For this goal, system information consisting of the channel state information and computation requests of the MUs is required. This information is assumed to be obtained by the MEC server in each TS. Based on the collected information, MEC server decides which MUs are to offload their tasks to the MEC server in TS i. Then it informs the MUs about the offloading decisions. When MU$_k$ offloads its task to the MEC server, additional steps have to be considered, i.e., the transmission of the task from MU$_k$ to the MEC server, the execution of said task in the MEC server and the transmission of the result back to the MU$_k$. All of these processes are described in detail in the following.

In the offloading case, the first step is the transmission of the task to the MEC server. In each TS i, J_i MUs offload and we assume indices of these offloading MU$_k$ are ordered as $k = 1, ..., J_i$, $J_i \leq K$. The fading channel from MU$_k$ to the MEC server is described by the channel coefficient $h_{k,i} \in \mathbb{C}$ which remains constant for TS i. We assume a NOMA system with no decoding errors. The offloading MUs are transmitting using the whole TS and the full available bandwidth B. The noise at the MEC server is assumed to be independent and identically distributed (i.i.d.) zero-mean additive white Gaussian with variance σ^2. The transmit power of MU$_k$ is denoted by p_k. We consider two cases without and with Successive Interference Cancellation (SIC). The transmission rate of offloading MU$_k$ in the case without SIC is calculated using Shannon's capacity formula as

$$R_{k,i}^{\mathrm{noSIC}} = B \log_2 \left(1 + \frac{p_k |h_{k,i}|^2}{\sigma^2 + \sum_{j=1, j \neq k}^{J_i} p_j |h_{j,i}|^2} \right). \quad (1)$$

For the second case, the MEC server can employ SIC to subtract the interference of the already decoded signals from the received signal [13]. While applying SIC, we assume the MU signals are decoded one-by-one with no decoding errors. Again we assume offloading MUs are indexed first, $k = 1, ..., J_i$, and indexed according to the decoding order. Then, after decoding first $k-1$ MUs' signals first, the resulting transmission rate for the next MU$_k$ to be decoded is

$$R_{k,i}^{\mathrm{SIC}} = B \log_2 \left(1 + \frac{p_k |h_{k,i}|^2}{\sigma^2 + \sum_{j=k+1}^{J_i} p_j |h_{j,i}|^2} \right). \quad (2)$$

The time required to transmit the task of MU$_k$ is dependent on the task properties as well as the transmission rate. We characterize a computational task of MU$_k$ in TS i with the task size $W_{k,i}$ in Bits, and the task complexity $L_{k,i}$ in CPU cycles [12], see Fig. 1. With these parameters, the time required to transmit the task of MU$_k$ in TS i to the MEC server is

$$T_{k,i}^{\mathrm{Tx}} = \frac{W_{k,i}}{R_{k,i}}. \tag{3}$$

In the case of computation offloading, the second step is the computation of the offloaded tasks in the MEC server. Let f_i^{MEC} denote the MEC server computation power in TS i, in CPU cycles per second. This is related to the portion of computing resources of the MEC server that are allocated for the computation of the offloaded tasks and changes in each TS. For offloading MUs, virtual machines (VMs) are allocated in the MEC server. The computing power of a VM allocated to MU$_k$ in TS i is represented with $f_{k,i}^{\mathrm{VM}}$, also measured in CPU cycles per second. The allocated VM computing powers fulfill the condition

$$f_i^{\mathrm{MEC}} = \sum_{k=1}^{J_i} f_{k,i}^{\mathrm{VM}}. \tag{4}$$

Then, the time required to compute the offloaded task of MU$_k$ in TS i is

$$T_{k,i}^{\mathrm{VM}} = \frac{L_{k,i}}{f_{k,i}^{\mathrm{VM}}}. \tag{5}$$

Lastly, after the computations are completed at the MEC server, the results are transmitted back to the offloading MUs. The result is expected to have a negligible size compared to the original task size $W_{k,i}$ and the time needed for the reception of the result is not considered, similar to [15].

The execution delay $T_{k,i}$ experienced by MU$_k$ is dependent on the offloading decision. In case that MU$_k$ performs its computation locally, the execution delay represents solely the time spent for the local computation at MU$_k$. Let f_k denote the computing power of this device in CPU cycles per second. Then, the time required to locally compute the task of MU$_k$ in TS i is given by

$$T_{k,i}^{\mathrm{local}} = \frac{L_{k,i}}{f_k}. \tag{6}$$

Accordingly, the execution delay of offloading the task of MU$_k$ is the sum of the transmission time $T_{k,i}^{\mathrm{Tx}}$ to the MEC server and the computation time $T_{k,i}^{\mathrm{VM}}$ at the MEC server, i.e.,

$$T_{k,i}^{\mathrm{offload}} = T_{k,i}^{\mathrm{Tx}} + T_{k,i}^{\mathrm{VM}}. \tag{7}$$

III. Problem formulation

In this section, we formulate the execution delay minimization problem for the scenario of Section II as a Multi-armed Bandit (MAB) problem and identify the main challenges to be addressed. First, we identify the objective of our problem and then, we present the MAB formulation.

A. Objective function

A computation offloading decision depends on the tasks and computational capabilities of the MUs, the channel fading and interference processes, and the computational capability of the MEC server. The computation offloading decision in TS i is given by the vector $\mathbf{X}_i \in \{0,1\}^K$. The k^{th} element of $\mathbf{X}_i$ is denoted by $x_{k,i} \in \{0,1\}$ and corresponds to the individual offloading decision for MU$_k$. $x_{k,i} = 0$ indicates a local computation and $x_{k,i} = 1$ indicates computation offloading for MU$_k$. The execution delay $T_{k,i}$ of MU$_k$ in TS i is

$$T_{k,i} = \begin{cases} T_{k,i}^{\mathrm{local}}, & \text{if } x_{k,i} = 0, \\ T_{k,i}^{\mathrm{offload}}, & \text{if } x_{k,i} = 1. \end{cases} \tag{8}$$

The execution delay minimization problem with respect to the computation offloading decisions is given by

$$\begin{aligned} \underset{\mathbf{X}_i}{\text{minimize}} \quad & \sum_{k=1}^{K} T_{k,i} \\ \text{subject to} \quad & \mathbf{X}_i \in \{0,1\}^K \end{aligned} \tag{9}$$

We identify (9) as a non-linear knapsack problem. This is due to the fact that the solution of this problem is a combination of the offloading decisions for all MU$_k$, $k = 1, \ldots, K$. Note that the offloading decisions for MUs cannot be considered independently because the execution delay of an offloading MU is dependent on the other MUs that are offloading. Also, this dependency is non-linear due to the logarithmic function of transmission rates, see (1), (2).

The dimension of the problem in (9) increases with the variety of tasks available to MU$_k$ and grows exponentially with K. Specifically, in TS i, the number of feasible solutions is bounded by 2^K. Moreover, it is assumed that the allocated MEC server computing capacity for offloading computation, i.e. f_i^{MEC} is not known when the offloading decision is made at the MEC server. Therefore, the information of only fading channel conditions and tasks of MUs is not sufficient to calculate and compare possible execution delays. To overcome these challenges, we propose an online learning algorithm to exploit past observations and past offloading decisions, to learn the optimal computation offloading decision policy.

B. MAB formulation

The multi-armed bandit (MAB) problems are problems of sequential decision making under uncertainty and constitute a special case of RL [16]. In TS i, the resulting execution delay depends on the offloading decisions made, considering sizes $W_{k,i}$ and complexities $L_{k,i}$ of the tasks, the channel coefficients $h_{k,i}$, and the MEC server computing capacity f_i^{MEC}. Consequently, the system under consideration can be modeled as a MAB, and specifically as a contextual MAB problem, also called bandits with side information [17].

A contextual MAB is defined by a context or state set $\mathcal{S}$ that is revealed to the agent, a set $\mathcal{A}$ of actions, and a set $\mathcal{R}$ of rewards [16]. In TS i, the state $S_i \in \mathcal{S}$ corresponds to the tasks and channel states of all the MUs and it is given by the tuple $S_i = \{W_{1,i}, \cdots, W_{K,i}, L_{1,i}, \cdots, L_{K,i}, h_{1,i}, \cdots, h_{K,i}\}$. It should be noted that $h_{k,i}$ can take any value in a continuous range. Therefore, the set $\mathcal{S}$ contains infinitely many possible states. The set $\mathcal{A}$ contains the offloading decisions $\mathbf{X}_i$ and in our model, $\mathcal{A}$ is a finite and discrete set which grows exponentially with respect to K. Last, the reward $R_i \in \mathcal{R}$

is a numerical value that indicates the cost of selecting A_i in S_i and it is given by the sum of execution delays as

$$R_i = \sum_{k=1}^{K} T_{k,i}. \tag{10}$$

A solution to a MAB problem is a policy to select the actions throughout the sequence of TSs. In our case, it is the controller algorithm that makes offloading decisions for MUs. The policy π can be evaluated using the so called action-value function $Q^\pi(S_i, A_i)$ which is the expected reward when starting with state-action pair (S_i, A_i) and following policy π after that [16]. The optimal policy π^* yields a state-action value Q^* which is greater than or equal to the state-action value of any other policy for all $S_i \in \mathcal{S}$. Determining Q^* is important because it leads to π^*.

The system considered is able to continue its operation as long as all MU_k, $k = 1, \ldots, K$ continue to receive new tasks in TS i. Hence, minimizing the total cost of using a certain decision policy is not a feasible approach since it could be infinitely large. Therefore, we use the average cost R, which minimizes the average cost per action as

$$R = \mathbb{E}\left[\lim_{I\to\infty} \frac{1}{I} \sum_{i=1}^{I} R_i\right]. \tag{11}$$

IV. COL: Combinatorial offloading learning

Our algorithm, termed combinatorial offloading learning (COL), is motivated by the challenge posed by the unknown variable of the MEC server computing capacity f_i^{MEC}. Therefore, a comparison of the local and remote execution delays, see (8), is not possible. Our algorithm handles the problem of the unknown system information and finding the offloading decision policy at the same time through online learning. In this section, we first present the strategy that we propose for our problem. Then, we describe the implementation of linear function approximation and the proposed algorithm.

A. Action selection strategy

Decision making in MAB problems involves a fundamental choice between exploitation and exploration, i.e., a decision between using the best action known given current information or gathering more information through a less tried action. However, problems with infinite state spaces pose a greater challenge that require algorithms that also scale well with the number of states [16].

To handle the challenge of the combinatorial nature of their solution, the authors of [14] decompose their RL problem into smaller ones. We apply the same strategy to introduce K additional RL problems for each MU_k (MU-RLP) while terming the first problem we already have in (9) as joint RL problem (joint-RLP) [14]. Each MU-RLP is associated with one MU and its sole task is to learn to make offloading decisions for that MU, i.e., minimizing the execution delay of that particular MU independent of other MUs. The decision for each MU is done simultaneously and independently in

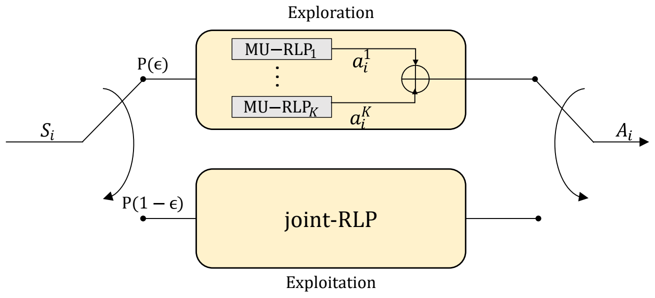

Fig. 2: Schematic of proposed strategy to RL problems [14].

each MU-RLP. Note that MU-RLPs do not produce optimal decisions due to joint effects of their actions. However, by utilizing the knowledge gathered from individual rewards per user, using MU-RLPs efficiently explore different offloading decision combinations in comparison to the selection of pure random actions. The joint-RLP selects the greedy action A_i which yields the highest Q-value for a given state S_i in TS i.

Figure 2 shows the flow of the action selection algorithm in TS i. Given state S_i, the algorithm can select the action A_i either using the MU-RLPs or the joint-RLP. With probability of ϵ, MU-RLPs are used in combination to produce the action A_i. Each MU-RLP learns to provide an individual offloading decision in the best interest of its corresponding MU. With probability $1-\epsilon$, joint-RLP is used for action selection. While joint-RLP uses the best known action given the state, MU-RLP actions efficiently explore the actions. Also in MU-RLP, the action that yields the highest Q-value is selected with probability $1 - \epsilon_{\text{mu}}$ and a random action is selected with probability ϵ_{mu}.

B. Linear Function Approximation

Even though the exploration of the large action space is handled by the COL efficiently, still the challenge of managing the infinite number of states remains. This results from the fact that channel conditions $h_{k,i}$ can take any value in a continuous range. Since $\mathcal{S}$ is infinitely large, the action value function Q^π has also an infinite number of values. In this case, linear function approximation can be used to represent Q^π as a weighted sum of feature functions [16]. The feature functions are used to map state-action pairs (S_i, A_i) onto feature values. The action-value function is approximated as

$$Q^\pi(S_i, A_i) \approx \hat{Q}^\pi(S_i, A_i) = \mathbf{f}^{\mathrm{T}}(S_i)\mathbf{w}, \tag{12}$$

where $\mathbf{f}$ is a vector formed by all the feature values and $\mathbf{w}$ is a vector of weights containing the contribution of each feature [16]. In this paper, tile coding is used as approximation technique due to its flexibility, computational efficiency and suitability for multi-dimensional continuous spaces [16]. In tile coding, number of tiles and grids G can be altered to find a suitable resolution for the approximation.

C. COL Algorithm

The proposed algorithm is composed of $K+1$ RL problems. The weights $\mathbf{w}$ that are used in linear function approximation of the Q-values are updated following a gradient descent approach with learning rate α, so that the error between the action value function Q^π and its estimate $\hat{Q}^\pi$ is reduced. The weights $\mathbf{w}$ of the global RL problem are updated as

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha \left(R_i + \max_A \hat{Q}^\pi(S_{i+1}, A_{i+1}, \mathbf{w}) - \hat{Q}^\pi(S_i, A_i, \mathbf{w}) \right) \mathbf{f} \quad (13)$$

The weights $\mathbf{w}^k$ of the MU-RLPs are updated as

$$\mathbf{w}^k_{i+1} = \mathbf{w}^k_i + \alpha \left(R^k_i + \max_a \hat{Q}^\pi(S_{i+1}, a^k_{i+1}, \mathbf{w}^k) - \hat{Q}^\pi(S_i, a^k_i, \mathbf{w}^k) \right) \mathbf{f}. \quad (14)$$

Algorithm 1 COL algorithm

```
1: initialize $\alpha, \epsilon, \epsilon_{\mathrm{mu}}$
2: initialize weights for joint-RLP and MU-RLPs
3: observe $S_i$
4: while there are tasks to compute do
5:     generate random number $r$
6:     if $r \geq \epsilon$ then                         ▷ joint-RLP is preferred
7:         select $A_i$ w.r.t. highest $\hat{Q}(S_i, A_i)$
8:     else                                              ▷ MU-RLPs are preferred
9:         for each MU-RLP do
10:            select decisions $a_i^k$ using $\epsilon_{\mathrm{mu}}$-greedy
11:        end for
12:        $A_i = (a_i^1, \ldots, a_i^K)$
13:    end if
14:    take action $A_i$
15:    calculate costs $R_i$                             ▷ Eq. (10)
16:    observe $S_{i+1}$
17:    update joint-RLP weights                          ▷ Eq. (13)
18:    update MU-RLP weights                             ▷ Eq. (14)
19:    $S_i = S_{i+1}$
20: end while
```

The proposed COL algorithm is summarized in Algorithm 1. The learning parameters are initialized first (line 1). These parameters are the learning rate α and the exploration probabilities $(\epsilon, \epsilon_{\mathrm{mu}})$. Additionally, the weights used for the action-value function approximation for the joint-RLP and the MU-RLPs are initialized (line 2). Then, the initial state S_i of the MU conditions is observed (line 3). A random number $r \in [0, 1]$ is generated according to a uniform distribution and compared to ϵ to decide if the MU-RLP or the joint-RLP is used for selecting the action (lines 5-6). If the joint-RLP is preferred, the action that yields the maximum value of the action-value function is selected (line 7). If the MU-RLPs are preferred, ϵ_{mu}-greedy is used to make offloading decisions for each user separately. $\mathbf{X}_i$ is obtained by combining the single actions (lines 9-12). Afterwards, the selected offloading decisions are applied and computation executions are performed (line 14). The corresponding costs are calculated with respect to (9) (line 15). After the offloading process, the new state S_{i+1} is observed (line 16). This state includes the new tasks that arrived at the MUs and the channel states. Next, the weights for the action-value function approximations are updated using (13) and (14) (lines 17-18). Finally, the current value of S_i is updated (line 19) and the procedure described above is repeated as long as there are tasks to compute.

V. Simulation results

In this section, numerical results for the evaluation of COL are presented. The results are obtained by generating $EN = 50$ independent random tasks, allocated MEC server computing resources and channel realizations. Each realization is an episode where the MUs receive tasks for a limited number of TSs I, $I = 500$. Tasks are selected randomly from the set $\mathcal{T}(\mathrm{size}, \mathrm{complexity}) = \{(4 \times 10^6, 330), (5 \times 10^5, 960), (4 \times 10^6, 1900), (5 \times 10^5, 2100)\}$ following a uniform distribution for every user in each TS. The MUs are assumed to have constant positions during TS i and have constant CPU performances selected for whole simulation randomly from the $f_k \in [0.9, 1.1]$GHz following a uniform distribution. The MEC server has a CPU performance selected randomly from a uniform distribution $f_i^{MEC} \in [5, 7]$GHz in TS i, that is allocated to offloading users equally as VM computing powers. We assume that the offloading user signals occupy the same spectrum with bandwidth $B = 20$MHz. The duration τ_i of TS i is set to be the maximum of the possible local computation times of MUs regarding their tasks in that TS. The channel coefficients $h_{k,i}$ are taken from an i.i.d. Rayleigh fading process with zero mean, unit variance and a path loss exponent of three. The noise variance at the server is set to $\sigma^2 = 1$. When performing linear function approximation, each dimension forming the state space is divided into three tiles and 32 grids are considered. The learning rate α is set as 0.001 for the MU- and joint-RLPs. The exploration parameters, i.e., ϵ and ϵ_{mu}, are defined with respect to the episode number EN as $\epsilon = \epsilon_{\mathrm{mu}} = (0.9)^{EN}$, $EN = 1, \ldots, 50$. For comparison, we consider two approaches: A greedy strategy where the MU with the strongest channel condition is selected to offload to avoid interference. Additionally, a short term optimal policy is considered where an exhaustive search on all possible actions is carried with the knowledge of allocated MEC server computation resource f_i^{MEC} in TS i.

In Figure 3, the average normalized execution delay per TS versus the episode numbers for $K = 8$ MUs is shown. The execution delay is normalized to the local computation time, where a normalized execution delay of local computation equals to 1 and any execution delay shorter than the respective local computation delay corresponds to a value in $[0, 1]$. At the 50th episode, our proposed algorithm converges close to the optimal algorithm, performing 32% better than the greedy policy and only 1% worse than the optimal algorithm which has additional information available. Note that in the beginning, the COL algorithm performs comparable to the greedy policy because it starts with sub-optimal decisions due to lack of system knowledge. However, as time passes, it gathers more experience through exploration.

Figure 4 shows the average normalized execution delay per TS versus the varying number K of MUs when there is no interference cancellation. Our proposed algorithm achieves an average execution delay only 5% higher than the optimal policy considering $K = 10$ MUs while performing 20% better than greedy policy. We also observe that the performances of our solution and the reference policies degrade with increasing number of MUs. This is an expected result because the MEC server has limited resources that must be shared among offloading MUs. The quality of service, i.e., average execution delay, degrades when the available resources must be split

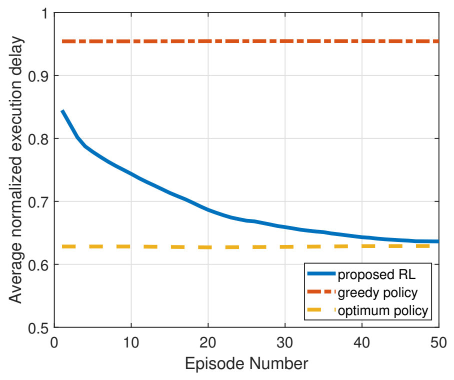

Fig. 3: Average execution delay vs. learning episodes.

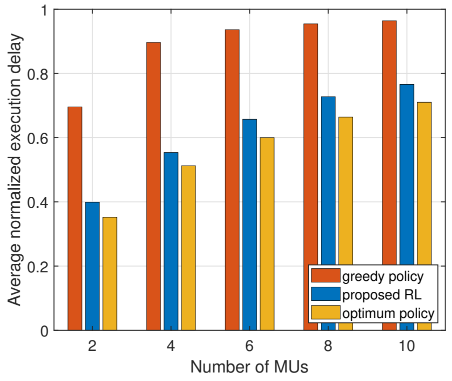

Fig. 4: Average execution delay vs. number of mobile users when SIC is not applied.

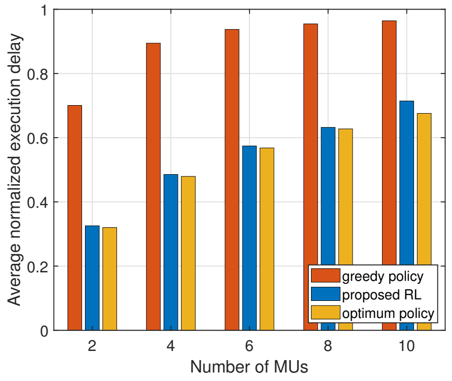

Fig. 5: Average execution delay vs. number of mobile users when SIC is applied.

between more users.

Figure 5 shows the average normalized execution delay per TS versus the varying number K of MUs when SIC is applied. At $K = 8$, we achieve a performance only 2% worse than the optimum algorithm. As mentioned before, the action space $\mathcal{A}$ suffers from the curse of dimensionality. Therefore, our algorithm's performance degrades eventually when more MUs are considered. However, in the case $K = 10$, the proposed algorithm still achieves a performance close to the optimum algorithm with only 4% difference, and outperforming the greedy algorithm by 25%. Besides, the proposed algorithm neither knows the MEC computing capacity information in TS i nor uses the computationally expensive exhaustive search for combinatorial actions. Still the proposed algorithm is comparable to the optimum algorithm in performance.

VI. Conclusions

A MEC scenario with a single MEC server and multiple mobile users was considered. In addition, a variety of given tasks are assumed and they are offloaded with non-orthogonal multiple access. Our goal was to find an offloading decision policy when the MEC server computing capacity is unknown. To this aim, we formulated the execution delay minimization problem as an MAB and identified the challenges posed by combinatorial nature of the problem, incomplete knowledge of system information, and infinite number of states. Following that, we proposed an RL algorithm which exploits the past observations of the system as well as the past offloading decisions, to learn optimal computation offloading decision policy. The proposed algorithm efficiently handles the exploration of the exponentially growing solution space. Additionally, we applied linear function approximation to address the issue of continuous states in a computationally efficient manner while maintaining an accurate representation. Through numerical simulations we showed that the proposed learning algorithm, with the unknown variable of MEC server computing capacity, achieves a similar performance to the optimal solution and it outperforms the other reference scheme.

References

[1] ETSI, "Mobile Edge Computing - A Key Technology Towards 5G," White Paper, Mobile-edge Computing Industry Initiative, Tech. Rep., 2015.

[2] T. X. Tran, A. Hajisami, and et.al., "Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, 2017.

[3] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Commun. Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[4] Y. Mao, J. Zhang, and et.al., "Power-Delay Tradeoff in Multi-User Mobile-Edge Computing Systems," in *IEEE Global Commun. Conf. (GLOBECOM)*, 2016, pp. 1–6.

[5] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai, "Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing," *IEEE Trans. Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, Nov 2017.

[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," *IEEE/ACM Trans. Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[7] X. Lyu, H. Tian, and et.al., "Multiuser Joint Task Offloading and Resource Optimization in Proximate Clouds," *IEEE Trans. Veh. Technology*, vol. 66, no. 4, pp. 3435–3447, April 2017.

[8] Z. Liang, Y. Liu, and et.al., "Multiuser Computation Offloading and Downloading for Edge Computing With Virtualization," *IEEE Trans. Wireless Commun.*, vol. 18, no. 9, pp. 4298–4311, 2019.

[9] M. Chen, M. Dong, and B. Liang, "Joint offloading decision and resource allocation for mobile cloud with computing access point," in *IEEE Int. Conf. Process. on Acoustics, Speech and Signal Process. (ICASSP)*, 2016, pp. 3516–3520.

[10] L. Huang, S. Bi, and Y. J. Zhang, "Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks," *IEEE Trans. Mobile Computing*, pp. 1–1, 2019.

[11] A. Khalili, S. Zarandi, and M. Rasti, "Joint Resource Allocation and Offloading Decision in Mobile Edge Computing," *IEEE Commun. Lett.*, vol. PP, pp. 1–1, 02 2019.

[12] A. P. Miettinen and J. K. Nurminen, "Energy Efficiency of Mobile Clients in Cloud Computing," in *USENIX Conference on Hot Topics in Cloud Computing*, 2010, p. 4.

[13] K. Higuchi and A. Benjebbour, "Non-orthogonal Multiple Access (NOMA) with Successive Interference Cancellation for Future Radio Access," *IEICE Trans. on Commun.*, vol. E98.B, pp. 403–414, 03 2015.

[14] A. Ortiz, T. Weber, and A. Klein, "Resource Allocation in Energy Harvesting Multiple Access Scenarios via Combinatorial Learning," in *IEEE Int. Workshop Signal Process. Advances Wireless Commun. (SPAWC)*, 2019, pp. 1–5.

[15] D. Nowak, T. Mahn, and et.al., "A Generalized Nash Game for Mobile Edge Computation Offloading," in *IEEE Int. Conf. Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2018, pp. 95–102.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[17] S. Bubeck and N. Cesa-Bianchi, "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems," 2012.