Mobility Load Management in Cellular Networks: A Deep Reinforcement Learning Approach

Ghada Alsuhli[†], Karim Banawan[‡], Kareem Attiah[*], Ayman Elezabi[†], Karim G. Seddik[†], Ayman Gaber[+], Mohamed Zaki[+], and Yasser Gadallah[†]

[†]Electronics and Communications Engineering Department, American University in Cairo, Cairo, Egypt

[‡]Electronics and Communications Engineering Department, Alexandria University, Alexandria, Egypt
[*]Department of Electrical and Computer Engineering, University of Toronto, Ontario, Canada

[+]Vodafone Egypt

Abstract—Balancing traffic among cellular networks is very challenging due to many factors. Nevertheless, the explosive growth of mobile data traffic necessitates addressing this problem. Due to the problem complexity, data-driven self-optimized load balancing techniques are leading contenders. In this work, we propose a comprehensive deep reinforcement learning (RL) framework for steering the cell individual offset (CIO) as a means for mobility load management. The state of the LTE network is represented via a subset of key performance indicators (KPIs), all of which are readily available to network operators. We provide a diverse set of reward functions to satisfy the operators' needs. For a small number of cells, we propose using a deep Q-learning technique. We then introduce various enhancements to the vanilla deep Q-learning to reduce bias and generalization errors. Next, we propose the use of actor-critic RL methods, including Deep Deterministic Policy Gradient (DDPG) and twin delayed deep deterministic policy gradient (TD3) schemes, for optimizing CIOs for a large number of cells. We provide extensive simulation results to assess the efficacy of our methods. Our results show substantial improvements in terms of downlink throughput and non-blocked users at the expense of negligible channel quality degradation.

1 Introduction

Global mobile data traffic is expected to reach around 38 exabytes per month in 2020 and is projected to grow to reach 160 exabytes per month in 2025. Traffic growth in cellular networks is driven by both the rising number of smartphone subscriptions and an increasing average data volume per subscription, fueled primarily by video content. This trend is expected to continue, as emerging media formats and applications, such as streaming high-quality video and augmented/virtual reality, will continue to drive traffic growth. Video traffic in mobile networks is forecast to grow by around 30% annually to account for 60% of all mobile data traffic in 2019 while traffic from social networking is also expected to rise annually by 20% until 2025 [2].

This traffic growth calls for a substantial network-level optimization. One venue for optimization is mobility load management. Total traffic carried by 4G networks is not uniformly distributed among the cells. As a rule of thumb, 15% of the network cells carry 50% of the generated traffic [3]. This increases the resource utilization in certain spots (hot spots) compared to the rest of the network. This raises the importance of offloading the congested cells via load balancing techniques. A typical approach is controlling the cell individual offset (CIO). The CIO is an offset that can be applied to alter the handover decision. This effectively forces some of the users to leave congested cells even if these cells may result in the highest user signal to interference and noise ratio (SINR). The CIO may be optimized based on various metrics, e.g. network throughput, utilization, or combinations thereof. Traditionally, network operators have relied on their subjective experience to devise load balancing decisions. To cope with the excessive traffic growth, network operators are shifting their attention to applying machine learning techniques for effective load balancing optimization. This is motivated by the complexity of the problem and the success of machine learning techniques in a wide range of telecommunications applications.

In this paper, we present a family of deep RL approaches, including several variations and enhancements, to solving the load balancing problem for LTE cellular networks. We choose the state of the environment to be a subset of KPIs, all of which are readily available to cellular operators. For the action, we use the CIO to trigger the handover (HO) procedure. Additionally, the reward function is chosen to match the needs of network operators, including total downlink throughput, the percentage of non-blocked users, and deviation from average utilization.

We start with the basic deep Q-learning technique [4], which works with a discrete set of actions. In traditional Q-learning, the agent constructs a table of the Q-values for each state-action pair where the Q-value assesses the quality of the decision at a certain state. These Q-values are updated through interaction with the environment. Since the chosen states in our formulation belong to an infinite space, we approximate the Q-values using a deep neural network. This leads to a deep Q-network (DQN) [4]. Next,

This work was supported by an internal research grant from the American University in Cairo, and was presented in part at the IEEE CCNC conference, Las Vegas, USA, January 2020 [1]

we propose applying various enhancements to the basic DQN including experience replay [5], double DQN (DDQN) [6], and state/target normalization [7]. These enhancements aim at reducing the generalization error[1], the bias, and the non-stationarity effects.

The above techniques are essentially classification-based[2] RL techniques. The action space dimensionality grows exponentially with the number of evolved nodes B (eNodeBs). This creates problems when we scale the system up to a higher number of cells. Moreover, we consider the general case when the CIO values belong to a continuum. To tackle both problems, we propose using actor-critic techniques [8]. Particularly, we adopt the deep deterministic policy gradient (DDPG), and its successor, the twin delayed deep deterministic policy gradient (TD3) [9]. Note that although actor-critic methods are sufficient to tackle the problem for discrete and continuous action space settings, the DQN/DDQN techniques remain extremely useful for small number of cells. This is due to the fact that DQN and DDQN requires less number of neural networks to implement and train (e.g., the DDQN requires two neural networks, while TD3 requires six). Hence, for a small number of cells, the DQN/DDQN outperforms the actor-critic methods in terms of the complexity, and the rate of convergence.

We empirically assess the performance of our techniques. To that end, we use NS-3 [10] to build a model of an LTE network. To model realistic users' mobility, we use the Simulation of Urban Mobility (SUMO) [11]. We use real imported maps from Fifth Settlement, Egypt, consistent with the data at network operators. This constructs the environment which the agent interacts with. We choose NS-3 as it supports the full protocol stack of the LTE system and can provide simulated, yet accurate, key performance indicators (KPIs). We present two case studies: First, a proof-of-concept scenario, where we consider a simple linear geometry of three cells in our cluster and a synthetically congested cell is available. We show using DDQN how can we effectively achieve a balanced load across the network. Next, we focus on a practical case study of a cluster from Fifth Settlement, Egypt with real site data. We show using DDPG, and TD3 that a significant boost in throughput can be achieved.

The rest of the paper is organized as following: Section 2 presents the system model, Section 3 presents the general RL framework. Next, we present the basic DQN in Section 4. Section 5 provides several improvements to the basic DQN. Next, we shift our attention to RL techniques for continuous action spaces in Section 6. We provide a short overview of the environment simulators and case studies setting in Section 7. Finally, Section 8 provides some numerical results and discussions, and Section 9 concludes the paper.

1. By generalization error, we mean the error in predicting the actual Q-value and/or optimal CIO value for previously unseen states.

2. Throughout this work, we use the notation classification-based RL and regression-based RL to shed light on the functionality of the neural network at the core of the RL agent. We refer to the DQN (and its related enhancements) as classification-based RL. Although that notation is usually associated with supervised learning, we use it in the context of the RL to emphasize the fact that the agent picks a CIO value from a discrete set of pre-defined CIO values. This is in the contrast to the actor-critic methods presented afterward, where the CIO values belong to a continuum. We refer to the actor-critic methods as regression-based RL. Nevertheless, we note that both techniques are fundamentally different as DQN is a value-based RL technique, where the actor-critic methods belong to the policy-gradient techniques [4].

1.1 Main Contributions

We summarize our contributions in the following points:

1) Recasting the mobility load-balancing problem as an MDP.
2) Novel definition of the MDP state based on the careful selection of KPIs. Our state definition parameters are readily available to any network operator. The state definition reflects the balance between the congestion level and the relative channel qualities of the users. This prevents the total DL throughput from declining as a result of load balancing.
3) Formulating a flexible optimization framework in terms of reward definitions including the total download throughput, the ratio of blocked users, and the deviation from the average load. This supports load balancing solutions for multiple, and sometimes conflicting, objectives, such as to maximize the network throughput or minimize the blockage.
4) Proposing several RL agents for discrete and continuous action spaces (CIO values). For the discrete action sets, we propose using DQN in addition to several enhancements to improve the performance. The enhancements include experience replay, DDQN, and state/target normalization. For the continuous action space, we propose using DDPG and TD3 agents.
5) Building a realistic system-level simulator for the cellular LTE network, which allows for controlling the CIO values. We use NS-3 to simulate the full protocol stack of LTE to exactly mimic real cellular networks. We also use real on-site data for the placement of the eNodeBs. Our simulators can be found in [12].

1.2 Related works

Unlike earlier generations of cellular networks, LTE networks are designed to support some self-optimization (SO) functionalities [13]. As a result, the past decade has witnessed a growing interest in realizing load balancing via self-tuning of handover (HO) parameters [14]–[18]. In [14], the authors present an algorithm to iteratively update the CIO. Reference [15] proposes a reactive load-balancing technique in which the CIOs of the serving and target cells are symmetrically updated with opposite signs by a specific value chosen from a predefined discrete subset by a reinforcement learning (RL). The chosen reward function is the negative of the number of unsatisfied users. However, the RL states are partially expressed in terms of the cell-edge user distribution, a piece of information that may not be readily known to a network operator in practice. In [16], the authors proposed a two-layer architecture in addition to a deep reinforcement learning technique to tackle the problem of load balancing in ultra-dense networks. The proposed off-policy deep RL-based algorithm can be trained via an asynchronous parallel learning framework and employ

multiple behavior policies for joint exploration to improve learning. There have also been a few attempts to automate load balancing in heterogeneous networks where femtocells are deployed to improve capacity and coverage. In [17], a supervised learning strategy is introduced to estimate the needed CIO values to adjust the range of the femtocells through historical data that is generated using a network simulator. Hence, macrocells are relieved from increased user traffic. A similar approach is reported in [18], in which a combination of fuzzy logic control and RL algorithms is employed to determine the required femtocell range expansion, measured in both femtocells transmit power and CIO. The interplay between load balancing and network slicing using reinforcement learning is explored in [19], [20]. Furthermore, [21], [22] present two deep reinforcement learning frameworks to balance the load in the cellular networks. The state spaces of these works are equivalent and mainly derived from the RB utilization of the different cells and the fraction of the edge users. We refer the reader to [23], for a more comprehensive survey on SO techniques for load balancing.

It is worth mentioning that actor-critic techniques are becoming popular in wireless communications. For instance, different from the load balancing problem, actor-critic techniques are used in [24] for user scheduling and resource allocation in heterogeneous networks. In another context [25], actor-critic methods are employed joint optimization of content caching strategy, computation offloading policy, and radio resource allocation in an IoT setting. We refer the reader for the following detailed surveys about the applications of reinforcement learning in wireless communication [26]–[28].

2 System Model

In this section, we present our system model. We begin by describing the LTE network we are dealing with. Next, we discuss our load balancing action (control), namely, CIO. Finally, we describe the central agent, who is responsible for choosing the optimal CIOs.

2.1 Network Description

Consider the downlink (DL) of an LTE cellular network with N base stations (eNodeBs or eNBs) and K active user equipments (UEs). Initially, K_n UEs are associated with the nth base station, such that $\sum_{n=1}^{N} K_n = K$. The kth UE is associated with the nth base station if it results in the best-received signal at the kth UE. Specifically, the UE measures the reference signal receive power (RSRP) and selects the strongest cell based on the cell selection receive level value. The UE shall regularly search for a better cell according to the cell reselection criteria [29]. We assume that kth UE moves with a velocity v_k following an unknown mobility pattern to the eNBs. The kth UE periodically reports the channel quality indicator (CQI) ϕ_k to the associated base station. The CQI is a 4-bit binary code, that is based on the perceived SINR and Block Error Rate (BLER) measured by the UE in addition to the number of antennas and the type of receiver. Furthermore, the kth UE wishes to be served with a minimum data rate of ρ_k bits/second.

The associated base station assigns B_k physical resource blocks (PRBs) to the kth UE as:

$$B_k = \left\lceil \frac{\rho_k}{g(\phi_k, M_{n,k})\Delta} \right\rceil \tag{1}$$

where $\Delta = 180$KHz in LTE, and $g(\phi_k, M_{n,k})$ is the achievable spectrum efficiency based on the reported CQI ϕ_k and the antenna configuration $M_{n,k}$ using the default LTE scheduler. The function $g(\cdot)$ specifies the modulation and coding scheme (MCS) based on ϕ_k and $M_{n,k}$. The total required PRBs needed to serve all associated UEs in the nth base station is $T_n = \sum_{k=1}^{K_n} B_k$. The total offered PRBs by the nth base station is denoted by $\tilde{T}_n$.

2.2 Cell Individual Offset (CIO)

According to the 3GPP LTE specifications [30], a UE, initially served by some cell i, will commence a handover request to some neighbor cell j if the following condition holds

$$Z_j + \theta_{j\to i} > Hys + Z_i + \theta_{i\to j}, \tag{2}$$

where Z_i and Z_j are the measured values of RSRP from cells i and j, respectively, $\theta_{i\to j}$ is the CIO value of cell i with respect to cell j, $\theta_{j\to i}$ is the CIO value of cell j with respect to cell i, and Hys is a hysteresis value that minimizes the likelihood of ping-pong scenarios that arise due to fading fluctuations. One may interpret $\theta_{i\to j}$ as the offset value that makes the measured RSRP of cell i appear stronger (or weaker) when compared with the measured RSRP of cell j.

2.3 The Central Agent

The central agent[3] can monitor all KPIs at the network level, which are derived from the UEs and eNodeBs reports as we will specify in Section 3.1. Based on the observed KPIs, the central agent learns the dynamics of the presented LTE network and takes decision for each CIO value, $(\theta_{i\to j} : i \neq j \in \{1, \cdots, N\})$. The agent chooses a sequence of the decisions, i.e., a *policy* such that the UEs in the network enjoy better user experience in the long run according to some performance metric as we will formally describe next.

3 Reinforcement Learning Framework

In this section, we present the general RL framework, which is used throughout. The RL technique is a dynamic learning framework, where an agent learns to self-optimize its actions by interacting with an environment. Specifically, the agent seeks to specify the optimal policy to maximize the long term average of a certain reward function. RL is abstracted as a Markov decision process (MDP), where the agent observes the current state of the environment and applies an allowable action, which changes the state of the environment and earns a reward as a result of the agent's action [4]. RL techniques are used to find the optimal decisions for MDPs when the dynamics of the system can

3. We note that the 3GPP specifies an architecture for centralized self-optimizing functionality, which is intended for maintenance and optimization of network coverage and capacity by automating these functions [31]. In this work, we focus on this architecture for our agent. Decentralized agents are an interesting future extension to this work. The decentralized agents can be obtained by exploiting decentralized machine learning techniques such as multi-agent DRL [32] and federated learning [33], [34].

be learned from experience. Hence, in the sequel, we recast the load balancing problem as an MDP[4].

In MDPs [4], there is an *agent*, which is a decision-maker; and an *environment*, which corresponds to everything outside the agent. The agent interacts with the environment. At discrete time instants $t = 0, 1, 2, \cdots$, the agent observes the *state* of the environment $S(t)$, which is a pre-specified representation of the environment that belongs to a state space $\mathcal{S}$. Based on this perceived state, the agent chooses an *action*, $A(t)$, from an action set, $\mathcal{A}$. When the agent applies $A(t)$, the state of the environment is changed to $S(t+1)$. The result of applying this action is assessed by a *reward function*, $\mathcal{R}(t+1) \in \mathbb{R}$. The central agent aims at finding the optimal policy, i.e., the optimal action sequence that maximizes the long-term average reward.

For perfect-knowledge MDP, the dynamics of the system can be fully described via the probability distribution, $\mathbb{P}\left(S(t+1) = s', \mathcal{R}(t+1) = r | S(t) = s, A(t) = a\right)$, where $s', s \in \mathcal{S}$, and $a \in \mathcal{A}$. For RL, however, the agent is learning the dynamics of the system from experience in addition to its usual decision-making capabilities. The MDP formulation is suitable for goal-oriented tasks, where the agent knows what the end goal is but has little knowledge about how to reach that goal and does not have training data. In our problem, the LTE network corresponds to the environment, the state corresponds to a subset of the KPIs, the action corresponds to setting the CIOs, and the reward is some quantification of the user experience. To fully describe the load balancing problem in terms of an MDP, we need to specify $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}(t)$ as follows.

3.1 State Space ($\mathcal{S}$)

We start with the state $S(t)$. Since KPIs are regularly reported to the core network to monitor the LTE network, it is practical to adopt a subset of these KPIs as the state of the environment.

We propose four KPIs[5] to define $S(t)$ [36]. The first is the resource block utilization (RBU), $\mathbf{U}(t) \in [0,1]^N$, where $\mathbf{U}(t) = [U_1(t) \quad U_2(t) \quad \cdots \quad U_N(t)]$, where $U_n(t)$ corresponds to the fraction of the utilized RB in the nth eNB at time t. This reflects the congestion level of each cell. In practice, $U_n(t) > 0.7$ is an indicator that the nth cell is a congested cell [37].

Secondly, the total DL throughput vector, $\mathbf{R}(t) \in \mathbb{R}_+^N$, where $\mathbf{R}(t) = [R_1(t)\ R_2(t) \cdots R_N(t)]$ and $R_n(t)$ represents the total DL throughput in the nth cell at time t. A cell with a higher throughput for the same RBU and number of UEs admits better user experience.

Thirdly, the number of connected users per cell $\mathbf{K}(t) = [K_1(t) \quad \cdots \quad K_N(t)] \in \mathbb{N}^N$, where $K_n(t)$ enumerates the number of connected users to the nth eNodeB at time t. This serves a dual purpose, as it tracks the effect of handover procedures that result from changing the CIO levels, and it quantifies the *average* user's experience given the total throughput of the cell.

Finally, the modulation and coding scheme (MCS) penetration $\mathbf{M}(t) \in [0,1]^{N \times \tau}$, where τ is the total number of modulation and coding schemes, which are 29 active MCS in LTE [30]. The element (n, j) in the matrix $\mathbf{M}(t)$ is the ratio of users in the nth cell that employs MCS with index j. This is a metric to assess the relative channel qualities of the users over the cell.

Now, we are ready to write the state $S(t)$, which is a concatenation of all these KPIs, i.e.,

$$S(t) = [\mathbf{U}(t)^T \quad \mathbf{R}(t)^T \quad \mathbf{K}(t)^T \quad \text{vec}(\mathbf{M}(t))^T]^T \tag{3}$$

where $\text{vec}(\cdot)$ is the vectorization function. All four KPIs are necessary to describe the state. For example, $R_n(t)$ is not enough to describe the user experience in the nth cell as it needs to factor in $K_n(t)$ and $U_n(t)$. By the same token, $K_n(t)$ is not enough to characterize the congestion of the cell, e.g. if the traffic load is small because most users are using voice services, $\cdots$ etc.

Note that due to the different units of the state features, each feature needs to be normalized by its maximum to ensure that there is no dominant feature. Additionally, the size of the state vector is $3N + N\tau$. This could be prohibitively large for large number of cells. Even for moderate cell sizes, the convergence of the agent's policy may be challenging due to the large search space. To represent the state compactly, we replace the $\mathbf{M}(t)$ with only the ratio of users utilizing the MCS indexes from 0 to 9 at each cell. This is due to the fact that we perform load balancing by triggering the handover procedure. As the cell-edge users suffer from high levels of interference and low signal power, low SINR is expected near the cell edge [38]. This SINR is used to calculate the CQI value which is reported to the BS. Then, the MCS parameter is assigned based on the CQI value. So, there is a mapping between the SINR and MCS index of the UE. The lower the SINR, the lower the MCS index. According to table 7.1.7.1-1 in 3GPP specifications, [30], the users with MCS between [0:9] has the lowest modulation order which is equivalent to QPSK in the digital modulation scheme. Thus, when a user has an MCS between [0:9], he/she has low SINR, low channel quality, and low modulation order which indicates with high probability that this user is near the edge. A simulation run was performed and showed that the performance using the full and compact MCS sets is virtually the same.

3.2 Action Space ($\mathcal{A}$)

The central agent controls the CIOs of all eNBs. Specifically, the agent selects the action,

$$A(t) = (\theta_{i \to j}(t) : i \neq j, i, j = 1, \cdots, N) \tag{4}$$

The values of $\theta_{i \to j}(t)$ can come from:

4. We argue that the Markov property is suitable to assume in our load balancing problem as we control the cellular network by basically triggering the handover process. The handover condition in (2) depends only on the current state of the cellular network. Furthermore, we design our MDP state to be as descriptive as possible, hence, our problem does not suffer from the hidden state problems in [35]. Relaxing the Markov assumption is an interesting future direction that is outside the scope of this work.

5. The reference [36] presents a systematic method of characterizing the most relevant KPIs that are highly correlated with the average user throughput. The paper concludes that the presented KPIs in addition to the BLER and the CQI are enough to estimate the average user throughput. In this work, however, we do not include BLER and CQI as they require the eNodeB to report the KPIs of individual users, which may be challenging. Moreover, the MCS penetration is a proxy for representing CQI and BLER at the cell aggregation level.

1) *Finite countable set:* In this case, the possible action values for $\theta_{i\to j}(t)$ is a discrete subset of $[\theta_{\min}, \theta_{\max}]$ dB of size L, where $\theta_{\min}, \theta_{\max}$ are the minimum and maximum possible CIO values. Hence, the cardinality of the action space $\mathcal{A}$ is given by[6]:

$$|\mathcal{A}| = L^{N(N-1)/2} \tag{5}$$

2) *Uncountable set:* In this case, $\theta_{i\to j}(t)$ is picked from the continuum of $[\theta_{\min}, \theta_{\max}]$ dB.

3.3 Reward Function ($\mathcal{R}(t)$)

To formally investigate the user's experience and/or the overall system performance, we need to define a reward function. The agent aims at maximizing the total (discounted) reward.

Possible reward functions that are of practical importance include:

1) *Instantaneous sum throughput:* For this reward function, the agent is concerned only about the total throughput of the network, i.e.,

$$\mathcal{R}(t) = \sum_{n=1}^{N} \sum_{k_n=1}^{K_n} \hat{R}_{k_n}(t) \tag{6}$$

where $\hat{R}_{k_n}(t)$ is the actual measured throughput of the k_nth user in the nth cell at time t. This reward function assesses the overall performance of the LTE network and the average user experience in that network.

2) *Percentage of non-blocked UEs:* Here, the central agent needs to choose CIOs such that the fraction of users that cannot be served with a minimum data rate $\underline{\rho}$ is minimized, i.e.,

$$\mathcal{R}(t) = 1 - \frac{1}{K} \sum_{n=1}^{N} \sum_{k_n=1}^{K_n} E_{k_n}(t) \tag{7}$$

where $E_{k_n}(t) = 1$ if the k_nth is not served with the minimum required data rate $\underline{\rho}$. This reward function targets the individual user experience.

3) *Average deviation of the total number of offered PRBs:* The central agent aims at only balancing the traffic load over the cells irrespective of the throughput. The central agent aims at minimizing the deviation of the offered PRBs from the mean, i.e.

$$\mathcal{R}(t) = -\sum_{n=1}^{N} \left| \tilde{T}_n(t) - \frac{1}{N} \sum_{n=1}^{N} \tilde{T}_n(t) \right|. \tag{8}$$

The central agent is designed to work with one of these rewards or a weighted average of all of them. Generally, the agent in RL implements a stochastic policy, π, where $\pi(a|s)$ is the probability that the agent performs an action $A(t) = a$ given it was in a state $S(t) = s$. The central agent aims at maximizing the expected long-term sum discounted reward function, i.e.,

$$\max_{\pi} \lim_{L\to\infty} \mathbb{E}_{\pi} \left[\sum_{t=0}^{L} \lambda^t \mathcal{R}(t) \right] \tag{9}$$

where λ corresponds to the discount factor. The discount factor signifies how important future expected rewards are to the agent. A discount factor $\lambda = 0$ corresponds to a myopic agent that cares only about the immediate rewards and does not care about long-term planning and for $\lambda = 1$ the agent gives equal weight to the present and all future rewards.

3.4 Action-Value Function

To assess the quality of each state-action pair, (s, a), the agent needs to calculate a state-action value function, referred to as the Q-function, defined by

$$Q_{\pi}(s,a) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \lambda^k \mathcal{R}(t+k+1) | S(t) = s, A(t) = a \right]. \tag{10}$$

This function calculates the long-term expected sum rewards starting from the state s and applying the action a while following the policy π.

For a finite state space $\mathcal{S}$ and action space $\mathcal{A}$, the agent in any MDP formulation (and by extension RL) essentially needs to construct a *Q-table*, that enumerates all the possible states versus all possible actions. The agent calculates $Q_{\pi}(s,a)$ for all (s,a) pairs. The optimal policy π^* satisfies the Bellman optimality equation [4]:

$$Q_{\pi^*}(s,a) = \sum_{s',r} \mathbb{P}(s', r|s, a)[r + \lambda \max_{a'} Q_{\pi^*}(s', a')]. \tag{11}$$

It can be shown that deterministic policies suffice to attain the same optimal long-term reward function when the Q-function can be accurately estimated. Hence, in this work, we apply deterministic policies. The central agent then applies the optimal action corresponding to each state, which is the one with the highest state-action value, i.e., the agent applies the action $a^*(s)$ such that:

$$a^*(s) = \arg\max_{a' \in \mathcal{A}} Q_{\pi^*}(s, a'). \tag{12}$$

4 Classification-Based Deep Q-Learning Technique

In this section, we present our basic technique to tackle the load balancing problem, namely, the classification-based deep Q-learning technique. We use deep Q-learning technique [39], [40] to construct an agent, whose task is to maximize the long-term average of one of the reward functions in Section 3.3. We note that direct enumeration of the Q-table needed to solve the RL problem is infeasible. This is due to the fact that the state space $\mathcal{S}$ is a continuous space with an uncountable number of states. Thus, we rely on approximating the Q-table by means of deep neural networks (DNN). We utilize deep neural networks to directly predict the optimal *action* for each state even if this state was never observed during the training phase. More specifically,

6. The expression of the cardinality in (5) assumes that all cells are adjacent to each other. Although this may be impractical, we generally formulate the problem as such. However, for specific scenarios, we set the relational CIOs for all non-neighboring cells to be zeros.

the neural network aims at approximating the optimal Q-function, $Q_{\pi^*}(s,a)$, by

$$Q(s,a;\mathbf{w}) \approx Q_{\pi^*}(s,a) \tag{13}$$

where $Q(s,a;\mathbf{w})$ is the function corresponding to the neural network weights $\mathbf{w}$.

4.1 Deep Neural Network

We use the neural network as a *multi-class classifier*. The DNN aims at identifying the optimal action given the state of the environment. The structure of the DNN is as follows.

The input layer of the neural network comprises of L_s neurons, where $L_s = 3N + N\tau$ if the full state vector in (3) is used, and $L_s = 4N$, when compacted versions of the state (i.e., with 10 lower MCS indexes penetration) is used. In this scheme, the classifier classes are all the possible CIO vectors. Hence, the output layer of the neural networks consists of $|\mathcal{A}| = L^{N(N-1)/2}$ output neurons, with linear activation function. To learn the (possible) non-linear dependencies of the actions $A(t)$ on $S(t)$, we further add N_h hidden layers. Each hidden layer has the size of $n_i,\ i = 1, \cdots, N_h$ neurons and with rectified linear unit (ReLU) activation function.

4.2 Q-Learning

Now, our DNN is ready to learn a Q-function approximation by RL. The learning is done over N_e episodes. Each episode corresponds to a complete simulation of the environment (in our case, the NS-3 simulator) over T_{sim} time period with a time step Δ.

To balance the exploration and exploitation, we define $\epsilon(t)$ to be the probability of picking a random action in time t (exploration), where,

$$\epsilon(t) = (\epsilon_d)^{\ell(t)} \tag{14}$$

where $0 < \epsilon_d < 1$ is the decay factor of the exploration probability[7] and $\ell(t)$ is the index of the time step corresponding to t. At time $t = 0, 1, 2, \cdots$, the central agent picks a random action with probability of $\epsilon(t)$ and exploits the action that maximizes the approximate Q-function $Q(s,a;\mathbf{w}_t)$, with the probability $1-\epsilon(t)$, i.e.,

$$A(t) = \begin{cases} \arg\max_{a' \in \mathcal{A}} \quad Q(S(t), a'; \mathbf{w}_t) & \text{w.p.} \quad 1-\epsilon(t) \\ \bar{A} & \text{w.p.} \quad \epsilon(t) \end{cases} \tag{15}$$

where $\mathbf{w}_t$ are the weights of the DNN at time t, and $\bar{A}$ denotes a random action that is drawn uniformly (i.e., with probability $\frac{1}{|\mathcal{A}|}$) from the action space.

The central agent sends the CIO values corresponding to $A(t)$ to all eNodeBs. The reward $\mathcal{R}(t+1)$ is calculated and the state $S(t+1)$ is observed. In this case, the estimate of the Q-value of the $(S(t), A(t))$, denoted by $\hat{Q}(S(t), A(t))$, is updated using the Q-learning update equation[8]:

$$\hat{Q}(S(t), A(t)) = \mathcal{R}(t+1) + \lambda \max_{a'} Q(S(t+1), a'; \mathbf{w}_t). \tag{16}$$

Note that, $\max_{a'} Q(S(t+1), a'; \mathbf{w}_t)$ is the *predicted* optimal Q-value using DNN with weights $\mathbf{w}_t$. Intuitively, (16) means that the estimated Q-value is a sum of the collected reward due to applying $A(t)$ and the highest discounted future reward predicted by the DNN with weights $\mathbf{w}_t$.

Now, the updated Q-value should update the weights of the DNN, i.e., we *train* the DNN with the results of this time step using back propagation as in supervised learning. To do so, we construct a *target value* $y(t)$ (i.e., a label in supervised learning) in the following manner,

$$\begin{aligned} y(t) = [&Q(S(t), a_1; \mathbf{w}_t) \quad Q(S(t), a_2; \mathbf{w}_t) \ \cdots \\ &\underbrace{\hat{Q}(S(t), A(t))}_{\text{from the update equation}} \quad \cdots \ Q(S(t), a_{|\mathcal{A}|}; \mathbf{w}_t)] \end{aligned} \tag{17}$$

Hence, the DNN is trained by the new example $(S(t), y(t))$. This is performed by minimizing the training loss function $L(\mathbf{w})$, via a training optimizer (e.g., stochastic gradient descent (SGD)).

5 Enhanced Classification-Based Deep Q-Learning Technique

In this section, we enhance the learning technique (see Figure 1) in Section 4 to reach a stable learning and faster convergence. Although our vanilla Q-learning technique leads to satisfactory results for simplified scenarios (e.g., the 3-cell system in [1]), the convergence of the scheme is slow for practical cases. Here, we enumerate some shortcomings of the previous scheme and propose three extensions that significantly enhance the performance and/or the convergence rate.

5.1 Experience Replay (Mini-Batches)

The scheme in Section 4 inherently uses a supervised learning to update the weights of the DNN. This requires that the data set samples are *statistically independent*. The samples' independence ensure a low-variance classifier, i.e., ensuring that the classifier *generalizes* well and does not depend on the training data. More specifically correlated samples that are mislabeled introduce a systematic bias, whose effect remains irrespective of the size of the data set [41]. In our application, the independence assumption does not hold since the LTE network KPIs are naturally correlated in time in any cellular network, and they are functions of related quantities. Hence, we need to break the time correlation of states to have a generalizable classifier. This is the motivation of experienced replay (also known as mini-batches) investigated in [42], [5].

7. Note that we use random actions to explore the state-action value space. This is done only with the model learning phase. Once the model learning completes, i.e., the weights of the NN converges, the NN outputs a *deterministic* action value for every state vector.

8. We note that although the general update equation for Q-learning is given by $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$ (see [39]), we set $\alpha = 1$ in this work. This is because the term α corresponds to the step-size of the algorithm, e.g., the learning rate in a stochastic gradient descent. Since we are using a deep neural network, the step-size is adjusted in the optimizer of the neural network and not within the RL update.

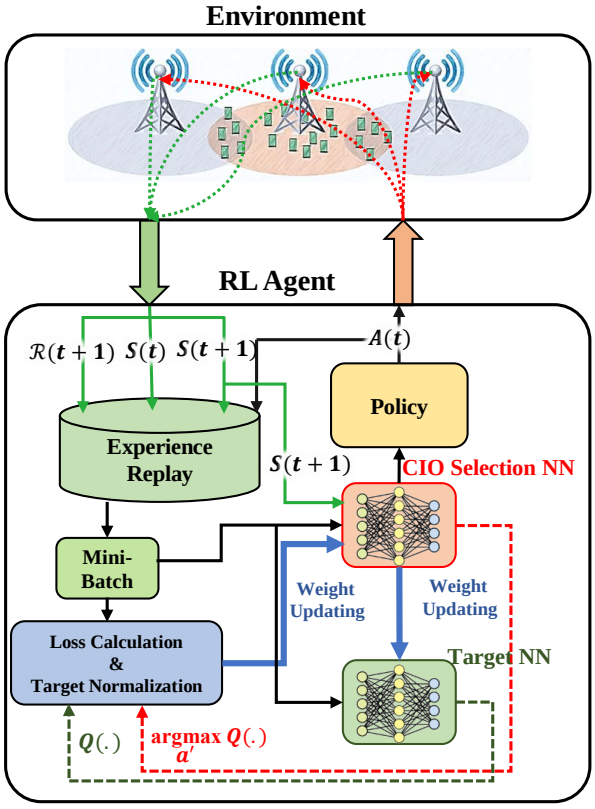


Fig. 1: Enhanced classification-based Q-learning model

In the experienced replay, we do not update the weights of the deep neural networks directly at each time step. Rather, we collect the experiences of the RL agent in a buffer of size B. By experiences, we mean the tuple $(S(t), A(t), S(t+1), \mathcal{R}(t+1))$ for all $t = 0, 1, 2, \cdots$. Hence, the buffer stores a data set of size B, in a set $\mathcal{D}_B$ as follows:

$$\begin{aligned}\mathcal{D}_B = &\{(S(0), A(0), S(1), \mathcal{R}(1)), \cdots, \\ &(S(B-1), A(B-1), S(B), \mathcal{R}(B))\}\end{aligned} \tag{18}$$

The data set is updated continuously by discarding the oldest experiences, when the buffer is full. At each step, a *random batch* of size $B_m < B$ is retrieved from the buffer to update the neural network weights (training). This randomization breaks the temporal correlation and smoothes learning over changes in the data distribution.

5.2 Double Deep Q-Learning (DDQN)

One problem that arises with our Q-learning technique in Section 4 is the overestimation of some of the Q-values. More specifically, due to the maximization step at each update of (16), positive bias exists for estimated Q-values. The overestimation may be non-uniform over the learned states and hence distorts the relative action preferences [6]. As pointed out by [6], the problem stems from using the same Q-function values to select the best action and to evaluate the value of the action. This adds a non-zero bias [6, Theorem 1]. Thus, to construct unbiased estimated Q-values, we need to *separate* the action selection from the target evaluation process.

To that end, we may construct two separate DNNs to implement double deep Q-learning (DDQN) [6]. In this work, we use the original DNN for choosing the best action, while we use a *delayed* version, by one episode, of this DNN for evaluation as in [6]. This delayed DNN is called the *target DNN*, whose weights are denoted by $\bar{\mathbf{w}}_t$. We refer to the original DNN as the CIO selection neural DNN. Thus, the Q-value of the $(S(t), A(t))$ state action pair is updated using the following update equation:

$$\begin{aligned}Q(S(t),A(t)) = \mathcal{R}(t+1) + \lambda \times \\ \underbrace{Q(S(t+1), \underbrace{\arg\max_{a'} Q(S(t+1), a'; \mathbf{w}_t)}_{\text{from CIO selection neural network}}; \tilde{\mathbf{w}}_t)}_{\text{from target evaluation neural network}}.\end{aligned} \tag{19}$$

The remaining steps are the same. Hence, the DDQN decreases the bias due to the maximization step. This facilitates faster training and more stable weight learning.

5.3 Target Normalization

Although our scheme in Section 4 implicitly uses supervised learning, there is a fundamental difference between our scheme and conventional supervised learning in terms of the target function. In RL, the target function (Q-function) is *non-stationary*, and in particular the variance increases with time. This may cause desensitization for weight updates if the target Q-values are quite high. Moreover, the target function in RL is not available prior to training as in supervised learning. This motivates normalizing the target function in addition to the commonly used normalization of the input (state) [7].

In this work, we propose a different normalization technique than [7]. Let $\bar{m}_{tn}(j) = \frac{1}{B_m}\sum_{i=1}^{B_m} Q_j(S_i, A_i)$, and $\sigma_{tn}(j)$ be the mean and the standard deviation of the jth mini-batch targets. Hence, the target $Q_j(S_i, A_i)$ is normalized as:

$$\bar{Q}_j(S_i, A_i) = \frac{Q_j(S_i, A_i) - \bar{m}_{tn}(j)}{\sigma_{tn}(j)}, \tag{20}$$

where $\bar{Q}_j(S_i, A_i)$ is the normalized target.[9,10]

6 Scaling Up the Deep Q-Learning Approach: Regression-Based Deep Q-Learning

Till now, we assumed that the action space is discrete. Hence, the agent picks an action for every given state in a similar fashion to multi-class classification problems. This technique works well for LTE networks with a small number of cells. If the number of cells is large and/or the cardinality of the CIO set is large, the classification technique becomes prohibitive as the dimensionality of the action space grows exponentially with the number of cells as in (5). Thus, successful training of such deep RL techniques

9. Note that because the current reward $\mathcal{R}_{i+1}$ is not normalized, the update predicted future Q-value term in (19), $Q_j(S_{i+1}, a'; \mathbf{w}_t); \tilde{\mathbf{w}}_t)$ needs to be *de-normalized* within the update equation.

10. It is worth noting that although our normalization technique can be regarded as a simplification to its counterpart in [7], our numerical results show that our simplified normalization technique outperforms [7] for our specific load balancing problem.

becomes exceedingly intractable. To tackle this problem, we propose using the regression-based deep Q-learning techniques. Thus, in this section, we assume that the CIO set is the entire continuous set $[\theta_{\min}, \theta_{\max}]$. We propose using two constructions for RL with continuous action space, namely, the Deep Deterministic Policy Gradient (DDPG) [8], and the Twin Delayed Deep Deterministic Policy Gradient (TD3) [9]. Next, we present the construction of both machines and how to adapt them to our problem.

6.1 Deep Deterministic Policy Gradient (DDPG)

DDPG combines the actor-critic [43] approach with deep Q-learning and its enhancements. The actor-critic algorithm produces continuous actions by optimizing a cost function, while it evaluates the current policy by means of Q-learning. DDPG uses experience replay and two separate target DNNs as in Section 5 as we will show next.

The DDPG algorithm uses two DNNs. First, the actor function $\mu(s; \mathbf{w}_t^a)$, which outputs a deterministic action for a state s based on the weights $\mathbf{w}_t^a$. The second neural network estimates the critic function, which is the normal Q-value $Q(s, a; \mathbf{w}_t^c)$ based on the weights $\mathbf{w}_t^c$. In addition, we have two target DNNs (similar to the DDQN technique), one corresponding to the actor with weights $\bar{\mathbf{w}}_t^a$ and the other corresponds to the critic with weights $\bar{\mathbf{w}}_t^c$.

Based on [8], our implementation for the DDPG uses experience replay as in Section 5, i.e., we construct a data set of the experiences in a buffer $\mathcal{D}_B$ with size B as in (18). The exploration is DDPG is based on randomizing the output of the actor function with a random noise, i.e.,

$$A(t) = \mu(s; \mathbf{w}_t^a) + \mathcal{N}(t) \tag{21}$$

where $\mathcal{N}(t)$ is a random noise that can be either random samples from an Ornstein–Uhlenbeck (OU) process with parameters (θ_n, σ_n), or uncorrelated zero-mean Gaussian noise samples with variance σ_n^2. The entire experience $(S(t), A(t), \mathcal{R}(t+1), S(t+1))$ is then stored in $\mathcal{D}_B$.

At each time step, we randomly and uniformly pick a batch of size B_m samples to calculate the target function based on the ith experience entry as: $(S_i, A_i, \mathcal{R}_{i+1}, S_{i+1})$:

$$y_i = \mathcal{R}_{i+1} + \lambda Q(S_{i+1}, \mu(S_{i+1}; \bar{\mathbf{w}}_t^a); \bar{\mathbf{w}}_t^c), \quad i = 1, \cdots, B_m \tag{22}$$

To get the new critic weights $\mathbf{w}_{t+1}^c$, we update the weights of the critic function by minimizing the mean square loss (MSE) along the mini-batch, $L(\mathbf{w}_t^c)$,

$$L(\mathbf{w}_t^c) = \frac{1}{B_m} \sum_{i=1}^{B_m} \left(y_i - Q(S_i, A_i; \mathbf{w}_t^c)\right)^2 \tag{23}$$

To get the new actor weights $\mathbf{w}_{t+1}^a$, we update the actor function by maximizing the expected reward function. Therefore, the scheme calculates the gradient ascent of the expected returned state-action value, which is given by $J(\mathbf{w}_t^a) = \mathbb{E}[Q(s, a; \mathbf{w}_t^c) | s = S_i, a = \mu(S_i; \mathbf{w}_t^a)]$, with respect to $\mathbf{w}_t^a$. Applying the chain rule, we thus calculate

$$\nabla_{\mathbf{w}_t^a} J(\mathbf{w}_t^a) \approx \frac{1}{B_m} \sum_{i=1}^{B_m} \nabla_a Q(s, a; \mathbf{w}_t^c)|_{s=S_i, a=\mu(S_i; \mathbf{w}_t^a)} \nabla_{\mathbf{w}_t^a} \mu(s; \mathbf{w}_t^a)|_{s=S_i} \tag{24}$$

Finally, DDPG uses soft target updates, i.e., the target DNNs are updated as a linear combination of new learned weights and old target weights,

$$\bar{\mathbf{w}}_{t+1}^c = \gamma \mathbf{w}_{t+1}^c + (1 - \gamma) \bar{\mathbf{w}}_t^c \tag{25}$$

$$\bar{\mathbf{w}}_{t+1}^a = \gamma \mathbf{w}_{t+1}^a + (1 - \gamma) \bar{\mathbf{w}}_t^a \tag{26}$$

where γ is the soft update coefficient, which is a hyper-parameter chosen from the interval $[0, 1]$. This constrains the target values to vary slowly; this, in effect, stabilizes the RL.

6.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is introduced in [9] to improve the performance of its predecessor, the DDPG. It is shown in [9] that DDPG suffers from the overestimation bias. This is due to the similarity between the policy and target functions as a result of slow updates of both functions. To remedy this, TD3 employs a pair of independently trained critic functions instead of one as in the case of DDPG. TD3 favors underestimating the Q-values by choosing the minimum of the two critics. Unlike overestimation errors, underestimation errors do not propagate through algorithm updates. This decreases the bias and the variance of the Q-value estimates. This provides a more stable approximation, thus improving the stability with respect to DDPG.

Additionally, TD3 uses delayed updates for the target actor and critic functions, i.e., the target and actor functions are updated once every T_u time steps, where T_u is a hyper-parameter of the scheme. This mitigates the problems of agent divergence due to poor policy estimation. Delaying the updates result in a more stable Q-value estimation, which in turn leads to a more stable policy estimation. Finally, TD3 uses a target smoothing regularization technique that adds clipped noise to the target policy prior to updating the weights. This ensures that the target fitting is still valid in the vicinity of the actual action stored in the replay buffer, leading to smooth value estimation.

Specifically, TD3 randomly initializes two critic networks $Q(s, a; \mathbf{w}_t^{c_1})$ and $Q(s, a; \mathbf{w}_t^{c_2})$ with weights $\mathbf{w}_t^{c_1}$, and $\mathbf{w}_t^{c_2}$, respectively. We randomly initialize the actor network $\mu(s; \mathbf{w}_t^a)$. We construct target networks with weights $\bar{\mathbf{w}}_t^{c_1}$, $\bar{\mathbf{w}}_t^{c_2}$, and $\bar{\mathbf{w}}_t^a$, respectively. Initially the target weights are set to their actor and critics values. Similar to DDPG, TD3 uses experience replay and randomizing of the actor output, hence the action $A(t)$ is obtained exactly as (21) and the entire experience $(S(t), A(t), \mathcal{R}(t+1), S(t+1))$ is stored in

the buffer as in DDPG[11].

One difference between DDPG and TD3 is in the mechanism of updating the DNNs. By randomly sampling a batch of size B_m from $\mathcal{D}_B$, we first use the the target actor DNN to compute the target action $A_{i+1} = \mu(S_{i+1}; \bar{\mathbf{w}}_t^a)$ that corresponds to an experience entry $(S_i, A_i, \mathcal{R}_{i+1}, S_{i+1})$. Then, the smoothed target action $\tilde{A}_{i+1}$ is calculated by adding the clipped noise,

$$\tilde{A}_{i+1} = \text{clip}(A_{i+1} + \tilde{\epsilon}, [\theta_{\min}, \theta_{\max}]), \quad i = 1, \cdots, B_m \quad (27)$$

where $\tilde{\epsilon} = \text{clip}(\mathcal{N}(0, \tilde{\sigma}^2), [-c, c])$ for some maximum value $c > 0$, where $\text{clip}(x, [a, b]) = b$ if $x > b$, is equal to a if $x < a$, and is equal to x if $a \leq x \leq b$.

Secondly, the target function is calculated using the minimum estimate of the Q-value from the two target critics for the perturbed input, i.e.,

$$y_i = \mathcal{R}_{i+1} + \lambda \min_{j=1,2} Q(S_{i+1}, \tilde{A}_{i+1}; \bar{\mathbf{w}}_t^{c_j}), \quad i = 1, \cdots, B_m \quad (28)$$

The weights of the two critics are updated as (23), hence, i.e.,

$$\mathbf{w}_{t+1}^{c_j} = \arg\min_{\mathbf{w}_t^{c_j}} \frac{1}{B_m} \sum_{i=1}^{B_m} (y_i - Q(S_i, A_i; \mathbf{w}_t^{c_j}))^2 \quad (29)$$

Finally, we update the actor function as in (24) every T_u time steps and not at every step as in DDPG and update the target functions as DDPG target update in (25). Figure 2 summarizes the structure of the TD3 algorithm.

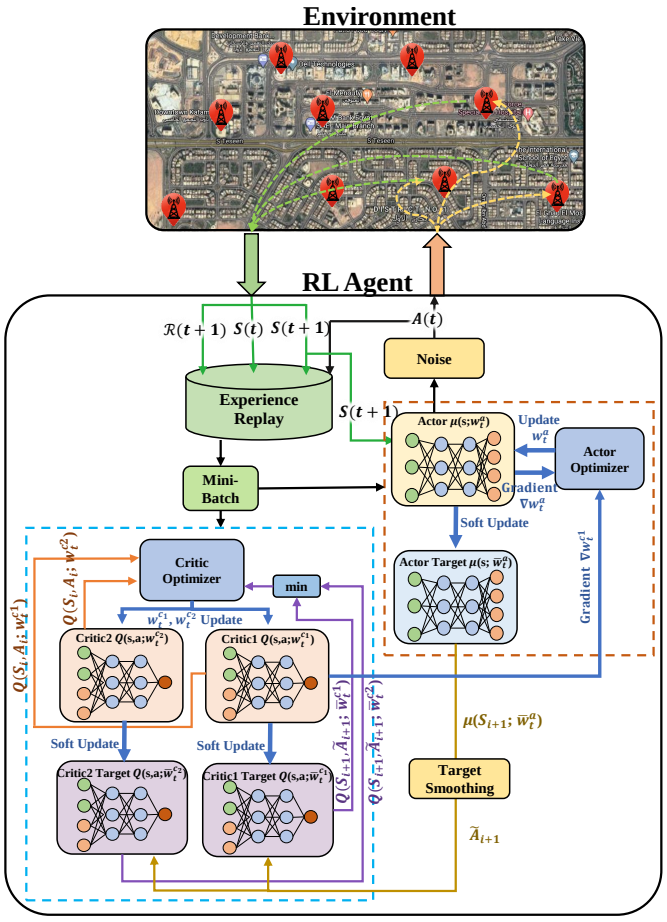


Fig. 2: TD3-based load balancing model

7 Environment Simulator and Case Studies

7.1 Environment: NS-3 LTE Simulator

The RL framework introduced in this paper assumes that online interaction between the central agent and the LTE network is possible. Unlike supervised learning, where the true label for each learning example exists prior to the training, the RL agent needs to learn a complete *policy* from experience with no prior knowledge of the true optimal policy. Hence, we cannot rely on historic records of cellular operators[12]. To that end, we use the NS-3 LTE simulator as our environment.

NS-3 simulator is a free discrete-event network simulator for Internet systems [44]. The NS-3 LTE module is a highly-accurate, open-source simulator that mimics the complete LTE system. The simulator provides the complete radio protocol stack for eNodeBs and the UEs, in addition to core network interfaces and protocols. The NS-3 LTE module allows testing new self-optimized network (SON) protocols [10]. Hence, NS-3 is well-suited for our purposes in this work.

Thus, we implement the eNodeBs and the UEs using the NS-3 LTE module. The module implements an FDD channel access. We modified the built-in protocols of NS-3 to allow different CIOs for every pair of cells. To interface the environment to the agent, we use the NS3gym interface [45], which is an interface between OpenAI Gym and NS-3 that allows for seamless integration of those frameworks. The interface takes care of delivering state and action information between the OpenAI Gym agent and the simulation environment. Next, we describe the two environment scenarios, which are simulated via NS-3.

7.2 Synthetic LTE Network Scenario

This synthetic scenario aims to show the gain that can be obtained by applying the proposed approach. In addition, this scenario is designed such that the optimal policy is predictable in order to assess the correct behavior of the agent. Three cells aligned on a line with 500m intersite distance are considered in this scenario, as shown in Figure 3. The UEs are randomly deployed such that most of the users are placed closer to the center of the middle cell. The remaining UEs are cell-edge users in the middle cell. Thus, all UEs are initially attached to the middle eNodeB. This deployment results in an overloaded cell that has some cell-edge users on the common boundaries of the

11. It is worth noting that the DDPG can be seen as a special case of the TD3 with $T_u = 1$, $\tilde{\sigma}^2 = 0$, and we fix the weights of one of the critic functions to be arbitrarily large positive values. Although DDPG can be seen as a special case of the TD3, we discuss DDPG in details for the following reasons: 1) DDPG is the building block of the TD3, hence, it is natural to build TD3 based on the shortcomings of the DDPG, 2) DDPG has less computational complexity with respect to TD3 (4 neural networks compared to 6 in TD3), thus, there is a tradeoff between computational complexity and performance.

12. By using historic data in supervised learning fashion, the agent would merely learn the most frequent action that the LTE network operator previously applied. This does not ensure converging on the optimal policy.

Parameter	Value
System Bandwidth	5 MHz
eNB antenna height	30 m
eNB antenna Pattern	Omni
eNB Tx Power	20 dBm
UE Traffic Model	CBR (1 Mb/s)
Blockage Threshold (ρ)	0.5 Mb/s
UE Mobility Model	Random walk (speed = 3 m/s)
UE Antenna Height	1.5~2 m
Number of UEs	41
Handover	Event A3 Hysteresis = 3dB Time to trigger = 40ms
Pathloss Model	COST Hata
Step Time (Δ)	200 ms

TABLE 1: Synthetic scenario simulation parameters

two underutilized cells. Such an extreme case is fitting to test the behavior of the load balancing technique. The UEs in this scenario have a constant bit rate (CBR) of 1 Mbps traffic model and walk randomly at a speed of 3 m/s. Other configuration parameters of this scenario are listed in Table 1.

In the synthetic scenario, the agent can use either DDQN or actor-critic algorithms that allow using continuous action space to learn the optimal values of relative CIO. When the DDQN is used the CIO values are selected from the discrete set {-6, -3, 0, 3, 6} dB. The configuration parameters of the DDQN RL agent[13] are given in Table 2. On the other hand, the configuration of the actor-critic algorithms used for this scenario are similar to those used for the next scenario.

Hyper-parameter	Value
Number of episodes (N_e)	200
Number of steps/episode (T_{sim})	50
Discount (λ)	0.95
Number of hidden layers (N_h)	1
Activation function	ReLU
Activation of the output layer	linear
Loss function	Huber loss
Optimizer	Adam(0.001)
Batch size	32

TABLE 2: Simulation parameters of the DDQN agent

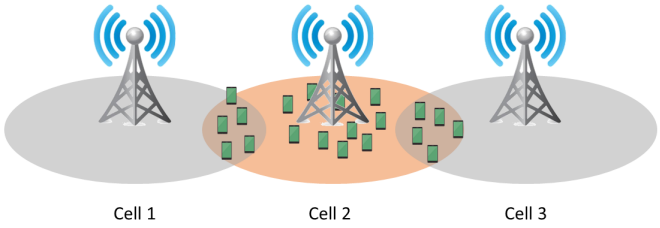


Fig. 3: The considered synthetic network

7.3 Realistic LTE Network Scenario

Although synthetic scenarios are usually used to test the efficiency of load balancing techniques in the literature, as in [1], [16], more realistic simulation scenarios should be considered before adopting any approach for real network implementation. For this purpose, an area of size 900m×1800m from the urban Fifth Settlement neighborhood in Egypt is selected to perform the simulation using a realistic placement of eNodeBs. The resulting network, illustrated in Figure 4a, represents a cluster of six eNBs whose locations are provided by a 4G network operator.

For more realistic users' mobility in our environment, we use Simulation of Urban Mobility (SUMO) [11], due to its simplicity and efficiency to produce realistic traffic behaviors. In addition, SUMO has the ability to import a realistic environment from real maps such as Open Street Map (OSM). This imported environment considers the real road structure, number of lanes, traffic light rules, buildings, etc. After extracting the map from OSM, the SUMO simulator is used to introduce realistic mobility models of the UEs. Figure 4b shows the exported region within the SUMO simulator. The mobile UEs in this scenario are either vehicles or pedestrians. The pedestrians walk with a speed ranging between 0-3 m/s. The vehicles mobility characteristics, i.e. acceleration, deceleration, speed factor, and speed deviation, are 1.85 m/s^2, 0.9 m/s^2, 1.13, and 0.11, respectively. These values are taken from [46] to emulate a realistic behavior of the vehicles. The UEs are randomly distributed on the available streets and pedestrian lanes at the beginning of the simulation. Afterward, each UE has a random trip from a source to a destination street during the simulation time. All of those UEs are assumed to have a full buffer traffic model, i.e., the users are always active. The unlimited demand of the users allows reaching a congestion state with a small number of UEs. The simulation has 40 UEs and eNBs with 32 dBm Tx power. The other simulation parameters of the realistic are common with the synthetic one, and are listed in Table 1.

Even though the number of cells in our realistic LTE network scenario is still not that large, using DDQN agent requires a DNN with a huge and prohibitive number of neurons in the output layer. Therefore, a continuous action space-based RL algorithm is used in the agent side to implement the load balancing approach suggested in this work. The agent determines the value of the relative CIOs within the continuous range $[-6, 6]$. As we will see later, the adopted scheme TD3 in this framework is compared to DDPG and Soft-Actor Critic (SAC)[14] algorithms. For the three algorithms, we used the stable implementation of the Open AI Baselines [52]. The simulation parameters of these algorithms are listed in Table 3.

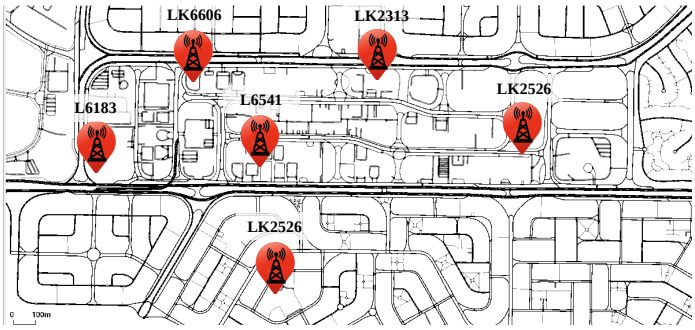
8 Numerical Results

In this section, we show the results of applying the proposed agents/approaches to the aforementioned LTE network sce-

13. Although Huber loss was originally proposed for regression problems, we choose to fix our loss function throughout this paper to always be Huber loss. The DDQN network computes estimates of Q-values which is similar to regression-based RL from the loss point of view. Furthermore, performance results using either of MSE or Huber loss are the same for the synthetic scenario.

14. For completeness, we compare our adopted techniques (DDPG and TD3) with the SAC [47]. SAC is the successor of Soft Q-Learning (SQL) [48]. At the same time, SAC shares the clipped double-Q feature with TD3. The key feature of SAC is learning the policy based on maximizing the both the expected return and the randomness in the policy, usually called entropy regularization [49]. This strikes a balance between learning acceleration and prematurely converging to a poor local optimum. We are not investigating its construction in detail as we will show that TD3 algorithm slightly outperforms SAC for our load balancing problem in terms of the average throughput. The SAC is a natural contender for the TD3 historically. Note that SAC optimizes a stochastic policy (in contrast to deterministic policies in the cases of DDPG and TD3). Moreover, the TD3 and the SAC are comparable in terms of the computational complexity and performance in the RL literature. For more information on the SAC algorithm, we refer the interested reader to the seminal work of [47], and some recent works such as [50] and [51].

(a) Google map view



(b) SUMO view

Fig. 4: Realistic network topology

Parameter	DDPG	TD3	SAC
Batch size (B_m)	128	128	64
Soft update coefficient(γ)	0.001	0.005	0.005
Policy delay (T_u)	-	2	-
Exploration noise	OU($\theta_n = 0.15, \sigma_n = 0.1$)	Normal($\sigma_n = 0.5$)	-
Entropy regularization coefficient	-	-	0.1
Hidden layers (N_h, n_i, Activation)	(2, (64,64), ReLU)		
Discount (λ)	0.99		
Number of episodes (N_e)	200		
Number of steps/episode (T_{sim})	250		

TABLE 3: Simulation parameters of different RL agents

narios. Note that in all simulations, the actions, generated by this agent, are applied to the environment at the beginning of each time step. However, the reward is estimated by averaging the instantaneous rewards of the sub-frames (1 ms) during the last quarter of the step time. The reason behind that is to ensure that the network has finalized the procedure of the triggered handovers and stabilized. We call this reward the step reward, versus the episode reward which is the average step reward of this episode. Due to the random exploration noise, the obtained episode reward during the learning process differs from one run to another. Thus, the results in this paper are averaged over 10 independent runs, with 95% confidence intervals represented by the shaded area around each curve[15]. The performance of the adopted agent is compared to the baseline system in which all cells have fixed relative CIOs of 0 dB.

15. The simulation runs are performed on a workstation with fourteen Intel Xeon CPU cores at 2.6 GHz and NVIDA GeForce RTX 2080 GPU. Despite these capabilities, the training of one agent model with 200 episodes takes about 21 hours, which is totally dominated by the NS-3 simulation time.

8.1 Synthetic Scenario Results

In the following, we discuss the results obtained for the synthetic scenario. First, the relative performance of the enhanced DDQN and an actor-critic RL algorithm, TD3, to solve the load management problem are compared for the synthetic environment. Then, we show the gain that can be obtained when several DDQN agents with different rewards to be optimized are used. Afterward, the channel quality variation because of applying the load management is debated.

8.1.1 Comparing DDQN and TD3 algorithms

We compare the enhanced DDQN, that uses discrete action space, to the adopted TD3 algorithm, with continuous action space, when the synthetic scenario is considered as an environment. Figure 5 shows the episode reward, namely the average throughput, that is achieved by TD3 and DDQN compared to the baseline and the optimal policy[16]. Each episode starts from a different seed and thus each episode has a different (random) distribution and mobility of the UEs. This forces the agent to experience more states and generalizes the obtained results. In Figure 5, we observe that the DDQN agent begins to converge and exploit its experience after about 75 episodes. Whereas, the TD3 agent converged earlier after 25 episodes. However, the DDQN agent, in all 10 runs, succeeded to reach almost the most reachable reward value which is 16.63 Mb/s, on average. On the other hand, due to the continuous action space that must be explored by TD3, the faster convergence of

16. The optimal policy for the synthetic scenario is to select the CIO values [6,-6] that offload all middle cell edge-users to the unutilized neighboring cells.

TD3 is not always to the best action. This causes TD3 to have wider confidence interval and worse average episode reward after convergence compared to DDQN, with this synthetic scenario. Nevertheless, this observation cannot be generalized. This is because the CIO values included in the discrete CIO list are sufficient for this synthetic scenario. More specifically, the optimal policy here includes using the two extreme CIO values [6, -6] of the discrete action set. Indeed, it is not the case with the optimal policy of a more realistic scenarios.

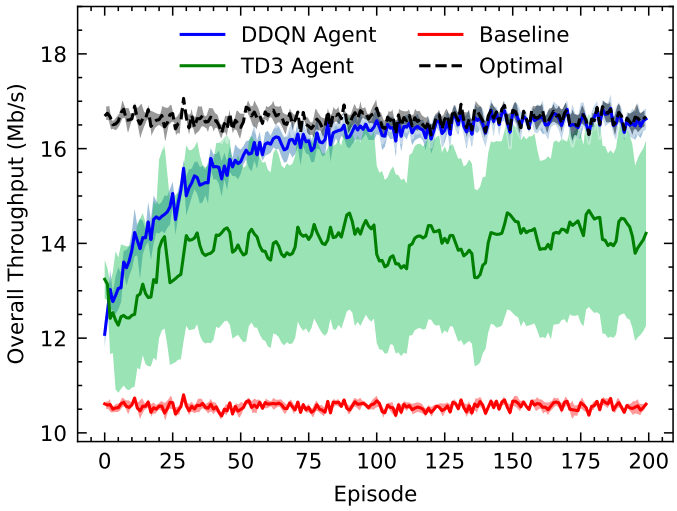


Fig. 5: Performance comparison of DDQN and TD3 agents

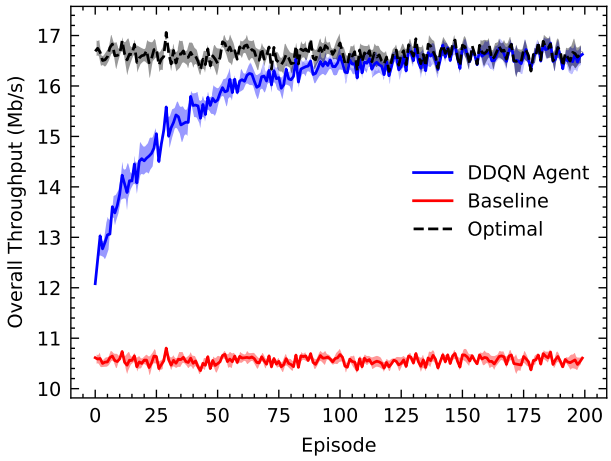
8.1.2 *Using Different Reward Functions*

The three suggested reward functions in Section 3.3 are investigated in this section. We aim to assess the gain achieved when each of the proposed rewards is considered by the DDQN agent.

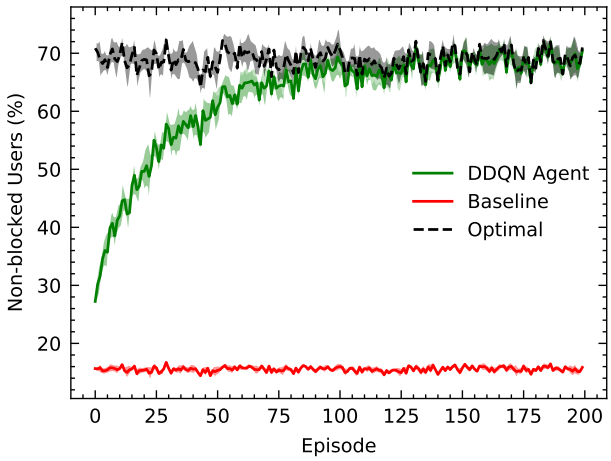
Figure 6a shows the results when the average overall DL throughput is considered by the DDQN agent to balance the load of the synthetic network. After convergence, we observe a significant enhancement in the overall DL throughput up to 57% compared to the baseline scheme. This is because the agent becomes capable of selecting the CIO values that match the optimal policy.

The performance of the load balancing framework when the objective of the agent is to maximize the percentage of non-blocked users is shown in Figure 6b. On its way to enhance the users' experience, the agent has to find the optimal policy that offloads enough users to the underutilized cells such that the number of users with a throughput higher than the blockage threshold is maximized. Although about 84.5% of the users were blocked in the baseline system, the percentage of satisfied users increased to be about 69% of the users after convergence in 6b. This is because moving cell-edge users to the underutilized cells allows the users of the congested cell to enjoy more PRBs, i.e. more throughput. At the same time, the throughput of the offloaded users is getting better as they have the chance to use more PRBs in the underutilized cells they moved to.

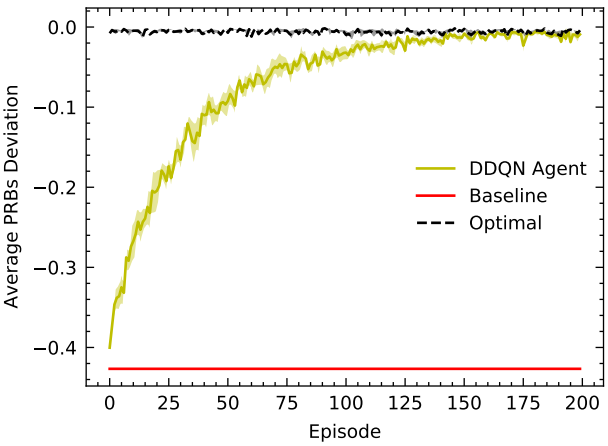
The RB utilization deviation reward is investigated in Figure 6c. The first observation here is that the baseline



(a) Overall DL throughput reward



(b) Percentage of non-blocked users reward



(c) Utilization deviation reward

Fig. 6: Performance using different reward functions

system has constant deviation since all UEs are attached to the middle eNB when the CIOs are zeros. The optimal policy should be able to achieve near 0 deviation which indicates that all base stations are equivalently loaded. After it converged, the agent succeeds in balancing the load and obtaining the optimal utilization deviation. It is worth mentioning that for such a simple synthetic scenario, choosing any of the three rewards would optimize the other two. This is because, for the synthetic scenario, the optimal policy, that optimizes any of the reward functions, is to offload the middle cell edge users to the cells on the sides. In other words, the impact of balancing the load on the network is equivalent to maximize the overall throughput or enhancing the individual experience. However, for the more complicated and realistic scenarios, there can be a conflict, or independence, between optimizing the previously mentioned reward functions, as we will see later.

8.1.3 Effect of Managing the Load on the Channel Quality

Finally, we investigate the impact of managing the load on the channel quality of the UEs in this synthetic scenario. Figure 7 illustrates the average CQI reported when a DDQN agent is trying to maximize the overall throughput of synthetic scenario. In this figure, the average channel quality of the users decreased and converged on less average CQI by 15.4% compared the baseline CQI. This is expected since changing the CIO in equation (2) counterfeits the actual RSRP to trigger the handover of a UE to an underutilized cell with lower channel quality. Nevertheless, despite this slight degradation in the radio channel quality of the offloaded users, the throughput significantly enhanced, as we noticed before in Figures 6a. The enhanced quality of experience despite the worsen quality of the channel is due to the fact that the decrease in the MCSs, due to the degraded channel quality, of the offloaded cell-edge users is compensated by providing more PRBs in the underutilized cells.

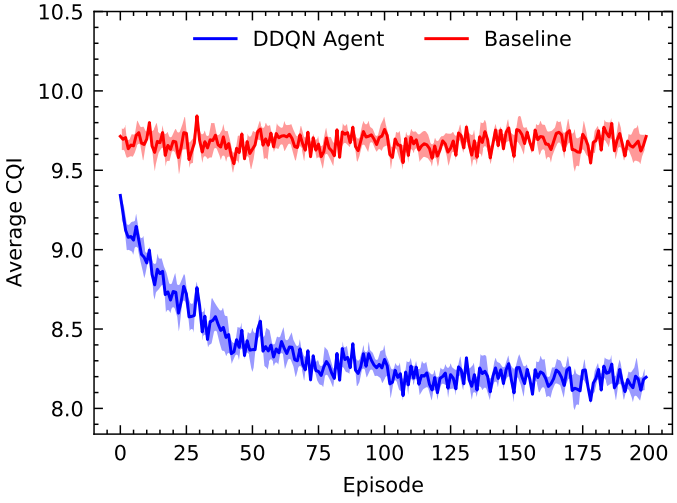


Fig. 7: Average channel quality indicator

8.2 Realistic Scenario Results

A more general view of the impact of managing the load on a realistic network is included in this section. Different actor-critic RL algorithms[17], that are hired to solve our load management problem, are compared. In addition, the importance of carefully designing the reward function, to be optimized in the realistic scenarios, is discussed.

8.2.1 Comparing Different RL Algorithms

In this section, the relative performance of Different previously discussed RL schemes is investigated for realistic scenario. Without loss of generality, the reward function used for different comparisons is the throughput reward function. The performance of the three RL algorithms during the learning process is depicted in Figure 8. In this figure, the episode rewards over 200 episodes (50k steps) are shown for the compared schemes. We observe from Figure 8 that TD3 and SAC are always better than DDPG in terms of the episode reward. Besides, TD3 and SAC have comparable performance. TD3 has wider confidence intervals due to the exploration technique used by TD3, which depends on adding random noise to the selected action at the start of the learning. In general, the exploration scheme and how much the algorithm should explore before the exploitation are considered open research issues that still need more improvement, especially in the case of continuous action spaces [53]. On the other hand, TD3 has a slightly better average reward, less fluctuation in the learning curves , i.e. smoother learning, against the number of episodes, and a faster rise in the learning curve, which results in better learning speeds, compared to SAC. Generally speaking, when the agent is intended to interact with and control an actual network, the learning speed, especially in the early stage, is of utmost importance to the operator to ensure short network instability periods.

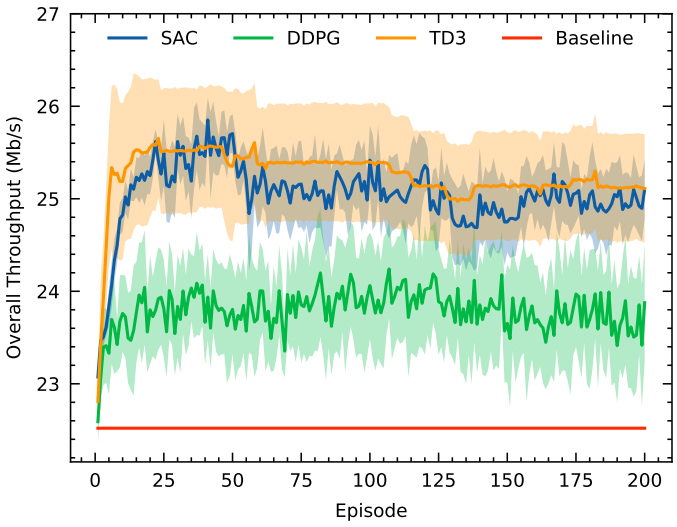


Fig. 8: Comparing the learning process of multiple RL agents

Back to the comparisons of the algorithms, as the learning process involves adding random noise to the policy, an

17. We note that using DDQN for this realistic scenario means that the cardinality of the action set will be $|\mathcal{A}| = 5^{15} = 30517578125$ actions. This large number of classes are computationally prohibitive to implement via state-of-the-art NN classifiers. Consequently, for the realistic scenario, we only show results for the policy-gradient methods and do not include any results using the DDQN algorithm.

additional evaluation phase after disabling the exploration and the learning should be performed. After performing the training phase over the 200 episodes, the resultant agent models of different runs are considered in the evaluation phase. This time we will focus our attention not only on the episode reward but also on the reward obtained at each time step. This allows us to better assess the quality of actions selected by the agents at every single step, i.e., for every single state. Over 250 time steps, the rewards for SAC, DDPG and TD3 are reported in Figures 10a, 10b, and 10c, respectively, averaged over the 10 models. By comparing the performance of these agents to the baseline system, the average improvement in the overall throughput is about 13.3% for the TD3 agent, 10.6% for the SAC agent and 7.1% for the DDPG agent. As expected, the TD3 scheme outperforms the DDPG due to delayed updates, avoidance of overestimation errors, and target smoothing. In Figure 10b, the selected actions by the DDPG agent result in worse performance than the baseline for some states. On the other hand, TD3 and SAC agents succeed in finding better policies compared to the the baseline system and they result in better overall throughput for almost 100% of the simulated states.

To show the effectiveness of the state space proposed in this paper, which contains a set of KPIs that are readily available to the network operator, we compare our proposed state-space with the one presented in the context of deep reinforcement learning mobility load balancing (DRL-MLB) technique [16]. The proposed states in [16], denoted as the DRL-based state space, include the RB utilization of the different cells derived from the averaged RB utilization and the fraction of the edge users, which can be considered as a subset of our state definition. For the sake of fair comparison, we use our proposed framework to simulate the performance of the two state spaces with the same parameters using TD3 and the overall throughput reward. Figure 9 illustrates the learning curves when the two state spaces are used compared to the baseline system. The results in this figure prove that our state space is more representative of the environment state, which leads to up to 11.5% of the average overall throughput improvement in the final episode, compared to 6.7% improvement when the DRL-based state space is used.

8.2.2 Effect of Changing the Reward Function

Next, an example to illustrate the relation between the different reward functions in presence of the realistic scenario. In the following example, one function is optimized by the agent and the other two functions are reported. Figure 11 reports the performance of the realistic LTE network during the training phase of the TD3 agent when the percentage of non-blocked users is adopted as the reward for the agents' action. From Figure 11a, and after 200 episodes, about 7.5% of the users become non-blocked compared to the baseline system. Figures 11b and 11c show how the average overall throughput and PRBs utilization deviation change during this training phase, i.e., while the agent is trying to maximize the number of non-blocked users. It is observed from Figure 11b that the decrease in the number of blocked users coincided with an improvement in the overall experience; indicated by the 6.6% increase in the average overall throughput. This improvement is less than

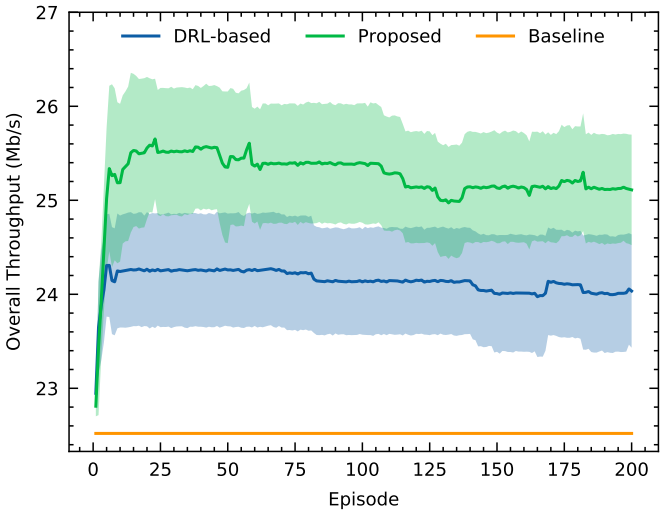


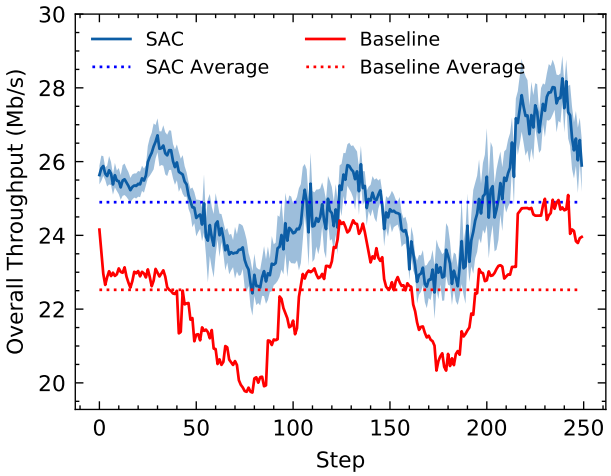
Fig. 9: Comparing the proposed state space to the state space presented in [16]

the 11.5% improvement obtained, see Figure 8, when the overall throughput reward function is used. Then, it is up to the service provider to choose either the number of non-blocked users or the overall throughput reward that best suits his objective, i.e, whether to optimize the individual or overall experience. The reason for this behavior is that for the sum throughput, the agent does not really care about fairness or blocked users. Maximizing the sum throughput, in many cases, cannot assign any resources to edge users or can even move the cell boundaries and provide no coverage for these users (i.e., it blocks these users). On the other hand, if we adopt the percentage of blocked users as our reward, this can clearly result in a reduction in the sum throughput as, in this case, we are committed to providing service even to the edge users with very bad channel conditions; this will clearly degrade the system's overall sum throughput.

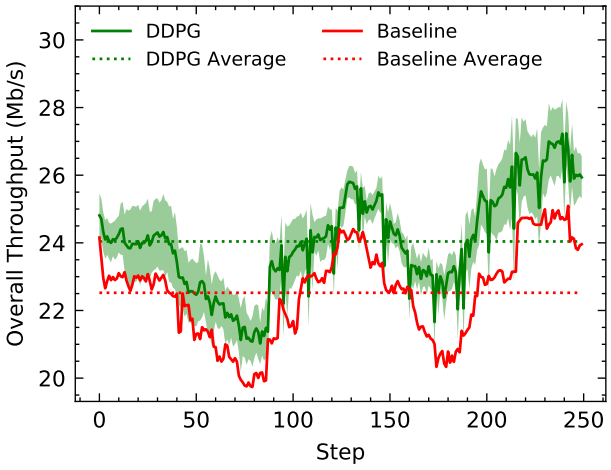
In Figure 11c, there is no clear trend in the utilization deviation that is correlated with the clear trend of increased users' throughput during the training phase. As a result, the enhancement in the user and overall experiences is not attributed to the agent attempt to *balancing* the load between the cells, but to intelligently manage the handover so that this experience is optimized. Even though the balancing is not an aim itself, most of the presented RL frameworks to solve the congestion problem use a reward that guarantees equal load distribution, for example see [16]. Here we emphasis on the importance of considering more *meaningful* rewards when trying to implement an RL agent to solve the congestion problem in the cellular networks, as a substitute of balancing the load between the cells.

9 Conclusions and Discussions

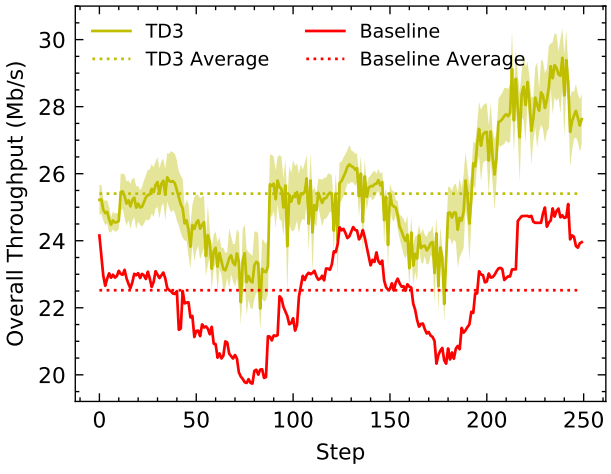
In this paper, we presented a family of reinforcement learning strategies for achieving load management in LTE cellular networks. We provided a comprehensive framework for RL with several algorithms and their applicable scenarios as well as enhancements. RL algorithms with discrete and continuous action spaces, i.e. DDQN, TD3, DDPG, and SAC, are compared to solve the load management problem. The



(a) SAC agent

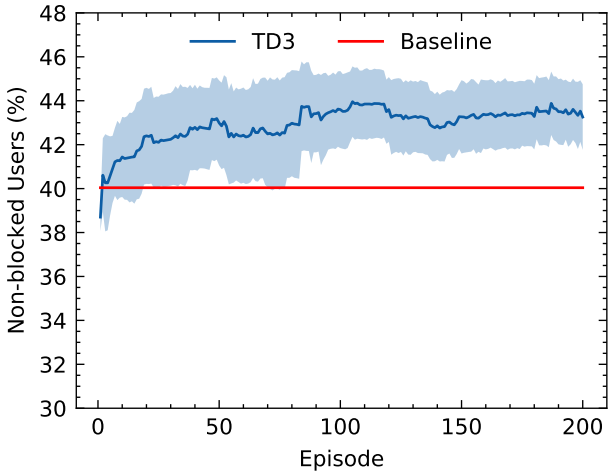


(b) DDPG agent

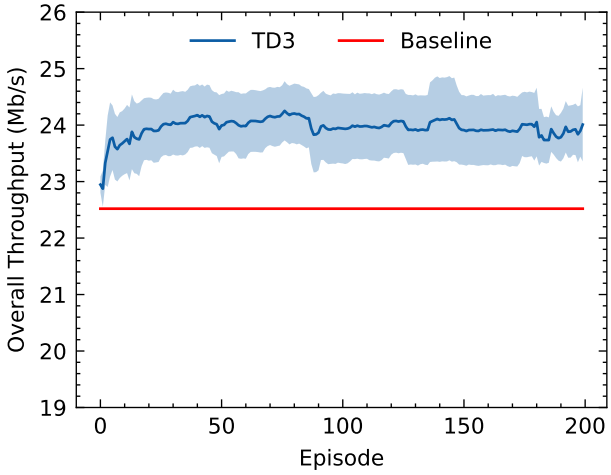


(c) TD3 agent

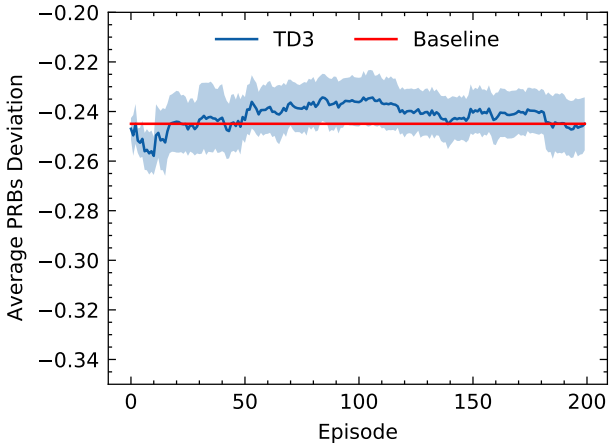
Fig. 10: Evaluating the learned models of multiple RL agents



(a) Percentage of non-blocked users



(b) Average overall throughput



(c) Utilization deviation

Fig. 11: Performance using percentage of non-blocked users reward function

states of the RL agent were described in terms of a subset of key performance indicators (KPIs), which are readily available to network operators, whereas different RL reward functions were explored including the network DL throughput, the ratio of non-blocked users, and the load deviation from the average. Two cellular network environments are used to assess the performance of the proposed framework, a small linear cell geometry and realistic LTE networks. In order to simulate the two environments, we used NS-3, a high-level simulator that allows the extraction of practical KPIs in addition to SUMO for simulating realistic mobility patterns and actual radio sites in Egypt. Our simulation results showed that, for simple and limited scenarios, DDQN is more reliable to end up with better policies than the actor-critic based algorithms. However, this technique is not scalable to solve the load management problem in realistic cellular networks. In addition, TD3 outperforms the other actor-critic based algorithms in terms of early-stage learning speed, the learning stability, and the average obtained reward after convergence. The performance improvement due to TD3 is demonstrated by a 13.3% gain in the average overall throughput over the baseline under the realistic scenario. The percentage of non-blocked users increased by about 8%, while the average PRB deviation showed only an insignificant improvement. For the synthetic scenario, much larger performance gains were achieved. Moreover, regarding selecting the reward function, we pointed out the importance of carefully designing the reward function that represents the actual objective of the operator since balancing the load does not always mean better quality of user experience, especially in realistic and complicated scenarios. Future extensions of this work include the design of a distributed, rather than a centralized, load management system for cellular networks. This can be achieved by modeling the load management problem as a multi-agent DRL with federated learning, such that a separate agent is hired to control the load at a base station level. The well-designed agents and the selected cooperation scheme between the agents are key factors of successfully managing the load of the overall cellular system. Despite the challenges facing such a design, it is expected to offer a cautious solution to the scalability problem, that obstructs applying the centralized approach to the real world cellular networks. Finally, it is worth noting that our proposed reinforcement learning techniques can be seamlessly applied to 5G networks.

REFERENCES

[1] K. M. Attiah, K. Banawan, A. Gaber, A. Elezabi, K. G. Seddik, Y. Gadallah, and K. Abdullah, "Load balancing in cellular networks: A reinforcement learning approach," in *IEEE CCNC*, 2020.

[2] Ericsson. Ericsson mobility report November 2019. [Online]. Available: https://www.ericsson.com/4acd7e/assets/local/mobility-report/documents/2019/emr-november-2019.pdf

[3] H. Holma and A. Toskala, *LTE advanced: 3GPP solution for IMT-Advanced*. John Wiley & Sons, 2012.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[5] R. Liu and J. Zou, "The effects of memory replay in reinforcement learning," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2018, pp. 478–485.

[6] H. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[7] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems*, 2016, pp. 4287–4295.

[8] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[9] S. Fujimoto, H. V. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[10] N. Baldo, M. Miozzo, M. Requena-Esteso, and J. Nin-Guerrero, "An open source product-oriented lte network simulator based on ns-3," in *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*. ACM, 2011, pp. 293–298.

[11] D. Krajzewicz and C. Rossel, "Simulation of urban mobility (sumo)," *Centre for Applied Informatics (ZAIK) and the Institute of Transport Research at the German Aerospace Centre*, 2007.

[12] G. Alsuhli, K. Banawan, K. Attiah, A. Elezabi, K. Seddik, A. Gaber, M. Zaki, and Y. Gadallah, "Paper source code," https://github.com/Ghada-sy/Load-Management, 2020.

[13] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What will 5g be?" *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, June 2014.

[14] A. Lobinger, S. Stefanski, T. Jansen, and I. Balan, "Load balancing in downlink lte self-optimizing networks," in *2010 IEEE 71st Vehicular Technology Conference*, May 2010, pp. 1–5.

[15] S. S. Mwanje and A. Mitschele-Thiel, "A q-learning strategy for lte mobility load balancing," in *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sep. 2013, pp. 2154–2158.

[16] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Load balancing for ultradense networks: A deep reinforcement learning-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9399–9412, 2019.

[17] C. A. S. Franco and J. R. B. de Marca, "Load balancing in self-organized heterogeneous lte networks: A statistical learning approach," in *2015 7th IEEE Latin-American Conference on Communications (LATINCOM)*, Nov 2015, pp. 1–5.

[18] P. Muñoz, R. Barco, J. M. Ruiz-Avilés, I. de la Bandera, and A. Aguilar, "Fuzzy rule-based reinforcement learning for load balancing techniques in enterprise lte femtocells," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 5, pp. 1962–1973, Jun 2013.

[19] 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 Communications Letters*, vol. 23, no. 8, pp. 1337–1341, 2019.

[20] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "Gan-powered deep distributional reinforcement learning for resource manage-

ment in network slicing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 334–349, 2020.

[21] S. S. Mwanje, L. C. Schmelz, and A. Mitschele-Thiel, "Cognitive cellular networks: A q-learning framework for self-organizing networks," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 85–98, 2016.

[22] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Deep reinforcement learning based mobility load balancing under multiple behavior policies," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[23] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2392–2431, Fourthquarter 2017.

[24] Y. Wei, F. R. Yu, M. Song, and Z. Han, "User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach," *IEEE Trans. on Wireless Communications*, vol. 17, no. 1, pp. 680–692, 2018.

[25] ——, "Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor–critic deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2061–2073, 2019.

[26] J. Jagannath, N. Polosky, A. Jagannath, F. Restuccia, and T. Melodia, "Machine learning for wireless communications in the internet of things: A comprehensive survey," *Ad Hoc Networks*, vol. 93, p. 101913, 2019.

[27] Y. Qian, J. Wu, R. Wang, F. Zhu, and W. Zhang, "Survey on reinforcement learning applications in communication networks," *Journal of Communications and Information Networks*, vol. 4, no. 2, pp. 30–39, 2019.

[28] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C.Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[29] 3GPP ETSI TS 36.304 V15.5.0 , *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA), User Equipment (UE) procedures in idle mode*, 2019.

[30] 3GPP ETSI TS 136.213 V14.2.0, *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA), Physical layer procedures*, 2017.

[31] 3GPP TR 32.836 V0.2.0, *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Telecommunication management; Study on NM Centralized Coverage and Capacity Optimization (CCO) SON Function (Release 12)*, 2012.

[32] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[33] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[34] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[35] S. Whitehead and L.-J. Lin, "Reinforcement learning of non-markov decision processes," *Artificial Intelligence*, vol. 73, no. 1-2, pp. 271–306, 1995.

[36] K. Abdullah, N. Korany, A. Khalafallah, A. Saeed, and A. Gaber, "Characterizing the effects of rapid lte deployment: A data-driven analysis," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2019, pp. 97–104.

[37] H.-H. Sung, M. G. Jacinto, K. K. Jat, and G. Dousson, "Determining network congestion based on target user throughput," Jan. 9 2018, uS Patent 9,867,080.

[38] M. S. Haroon, Z. H. Abbas, F. Muhammad, and G. Abbas, "Coverage analysis of cell-edge users in heterogeneous wireless networks using stienen's model and rfa scheme," *International Journal of Communication Systems*, vol. 33, no. 10, p. e4147, 2020.

[39] V. François-Lavet, P. Henderson, R. Islam, M. Bellemare, J. Pineau *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[40] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[41] M. Dundar, B. Krishnapuram, J. Bi, and R. Rao, "Learning classifiers when the training data is not IID." in *IJCAI*, 2007, pp. 756–761.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[43] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, November 2012.

[44] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[45] P. Gawlowicz and A. Zubow, "ns3-gym: Extending OpenAI Gym for Networking Research," *CoRR*, 2018. [Online]. Available: https://arxiv.org/abs/1810.03943

[46] A. Marella, A. Bonfanti, G. Bortolasor, and D. Herman, "Implementing innovative traffic simulation models with aerial traffic survey," *Transport Infrastructure and Systems*, pp. 571–577, 2017.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[48] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *arXiv preprint arXiv:1702.08165*, 2017.

[49] S. Han and Y. Sung, "Diversity actor-critic: Sample-aware entropy regularization for sample-efficient exploration," *arXiv preprint arXiv:2006.01419*, 2020.

[50] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[51] Z. Hou, K. Zhang, Y. Wan, D. Li, C. Fu, and H. Yu, "Off-policy maximum entropy reinforcement learning: soft actor-critic with advantage weighted mixture policy (sac-awmp)," *arXiv preprint arXiv:2002.02829*, 2020.

[52] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore,

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[53] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.