Energy-Efficient and Context-aware Trajectory Planning for Mobile Data Collection in IoT using Deep Reinforcement Learning

Sana Benhamaid, Hicham Lakhlef, Abdelmadjid Bouabdallah

Energy-Efficient and Context-aware Trajectory Planning for Mobile Data Collection in IoT using Deep Reinforcement Learning

1st Sana Benhamaid
Université de Technologie de Compiègne
Sorbonne Universités, CNRS
Compiègne, France
sana.benhamaid@hds.utc.fr

2nd Hicham Lakhlef
Université de Technologie de Compiègne
Sorbonne Universités, CNRS
Compiègne, France
hlakhlef@utc.fr,

3rd Abdelmadjid Bouabdallah
Université de Technologie de Compiègne
Sorbonne Universités, CNRS
Compiègne, France
madjid.bouabdallah@hds.utc.fr

***Abstract*—IoT networks are often composed of spatially distributed nodes. This is why mobile data collection (MDC) emerged as an efficient solution to gather data from IoT networks that tolerate delay. In this paper, we study the use of reinforcement learning (RL) to plan the data collection trajectory of a mobile node (MN) in cluster-based IoT networks. Most of the existing solutions use static methods. However, in a context where the MN has little information (no previous data set) about the environment and where the environment is subject to changes (cluster mobility, etc.), we want the MN to learn an energy-efficient trajectory and adapt the trajectory to the significant changes in the environment. For that purpose, we will train two reinforcement learning (RL) algorithms: Q-learning and state-action-reward-state-action (SARSA) combined with deep learning (DL). This solution will allow us to maximize the collected data while minimizing the energy consumption of the MN. These algorithms will also adapt the trajectory of the MN to the significant changes in the environment.**

***Index Terms*—Internet-of-Things, Mobile data collection, energy efficiency, Adaptive trajectory planning, Deep Q-learning, Deep SARSA**

I. Introduction

In the era of pervasive IoT, millions of IoT devices are deployed in intelligent applications. These devices generate huge amounts of data and consume large amounts of energy to communicate their data to the base stations or their neighbors. Since IoT devices usually have limited batteries, once emptied or depleted, those batteries are often difficult to replace especially if the devices are placed in an isolated or a difficult-to-access position. In the literature, various authors studied how to reduce the energy consumption of spatially distributed IoT nodes [1]. Researchers have studied cluster-based energy-efficient solutions which consist in regrouping IoT devices into clusters and electing a node depending on its energy capability. This node will serve as a relay node between the cluster devices and the gateways or receivers. The relay node will be responsible for sending data and will allow IoT devices to save their energy by only sending their data through the relay node [2]. To enhance the energy efficiency of cluster-based solutions, authors proposed energy-efficient routing protocols applied to these contexts. For example, in [3], the authors proposed an energy-efficient cluster-based routing protocol for wireless sensor networks that uses different parameters to balance the energy load and decrease the energy consumption of the network. However, using a routing protocol to reduce the energy consumption in an IoT network implies that these devices must be connected which is not always realistically possible if the nodes are placed in isolated or harsh conditions. One of the solutions proposed to tackle the limitation of routing-based energy-efficient solutions are the MDC approaches [4]. However, the most challenging part in mobile sink-based solutions is to determine and plan the trajectory of the mobile sink in order to efficiently gather data from the IoT nodes. Most existing approaches to MDC are static and only find a solution for scenarios with fixed parameters. These solutions suppose that the MN has a great knowledge of its environment and do not consider a change of context in the system such as a change in the clusters' data generated or position. Recently, artificial intelligence techniques such as machine learning (ML) or DL are used in order to propose intelligent solutions that allow devices to learn and take intelligent decisions. In contrast to static methods, intelligent schemes will be able to discover new clusters at any point and adapt the trajectory of the MN accordingly without the need for expensive re-computation

The paper is organized as follows: Section II discusses the works related to our problem, Section III presents background information on RL, Section IV presents the system model and Markov Decision Process. Sections V and VI describe our RL algorithms and present and discuss the simulation results. Finally, discussions and conclusions are presented in Section VII.

II. Related Works

In the literature, most of the existing data collection and trajectory planning solutions in the data collection context use non-learning based approaches. For example, in [9], the authors proposed a framework that optimizes the deployment and mobility of multiple unmanned aerial vehicles (UAV) in

order to collect data in the uplink from ground IoT devices and minimize the energy consumption of the MNs. In [10], the authors proposed a computing and networking framework for energy-efficient disaster management using UAVs where UAVs use deep learning algorithms for real-time route planning, obstacle avoidance and object detection. In [11], the authors proposed an algorithm that optimizes the UAV stops for data collection from neighboring sensors and the itinerary followed by the UAV to ensure efficient collection of all data with minimum energy consumption. However, in IoT networks, devices do not constantly collect and transmit data, their activity depends on the environment around them or the period of time the node is visited in. In the previous solutions, a change in the activity of the IoT network is not considered. Consequently, having a mobile sink periodically collecting data, following a static trajectory, may cause energy waste and does not constitute an optimal solution to achieve energy efficiency.

In recent years, researchers have considered learning based solutions and especially RL as a very promising trend in this field. In [12], the authors proposed a distributed RL approach for path planning and collision avoidance of UAVs. In this study, the MNs are given a visiting order and the algorithm designs an optimal path for data collection in IoT networks with devices having different communication radios. Compared to this study, our solution uses deep RL which is more suitable for complex environments and the MN can explore the environment and establish its own visiting order based on its experience. In [13], the authors proposed a deep RL scheme that plans the trajectory of an MN which is used as a data collector and charger in wireless powered IoT networks. The solution minimizes the average data buffer length, the residual battery level of the system and avoid devices data overflow. In [14], the authors proposed a multi-agent RL approach that allows the control of the trajectory of a team of cooperative UAVs in order to maximize the collected data under flying time and collision constraints. In [15], the authors proposed a double deep Q network to plan the trajectory of a UAV on an IoT data harvesting mission. The solution proposed aims to maximize the collected data under flying time and navigation constraints and allows the UAV to adapt to variations in the number of IoT devices. Contrary to our solution, the previously discussed solutions do not take into consideration the inconsistencies in the IoT devices activity (e.g., a device can collect more data on a certain period of the day more than another).

Our solution consists in planning an adaptive and energy-efficient trajectory for the MN using a deep RL algorithm. The MN will visit clusters and collect data from an elected node in the cluster called the relay node. The relay node is responsible for collecting the data from the IoT nodes in the cluster and transmitting it to the MN. The objective of our solution is to maximize the collected data while minimizing the energy consumption of the MN and IoT devices. In our solution, the MN has information on the position of all the relay nodes. It will build an energy-efficient data collection trajectory depending on the context of the network (e.g., the clusters' activity). Fundamentally, the MN will adapt its trajectory and save energy by not visiting a cluster with no data to transmit. The learning algorithm used to plan the trajectory is also computed on the MN. Other solutions proposed RL solutions to plan the trajectory of MNs in order to either optimize data collection or minimize the energy consumption and need to perform expensive computations in order to adapt to a change in the context. This solution doesn't require a previously recorded data set. The MN will build its own knowledge by exploring the environment. In our paper :

- We formally describe our environment and propose a system model of the environment.
- We define our problem as a Markov Decision Process.
- We propose a Deep SARSA (DS) and deep Q-learning (DQN) path planning algorithms.
- We carry out simulations on these algorithms and compare their performances in terms of data collection, energy consumption and adaptability.

III. Background on Reinforcement Learning

Reinforcement learning is a machine learning technique that consists in learning by interacting with the environment [5] [7]. It is a computational and goal-oriented approach to learning by interaction as the agent learns how to map situations to actions to maximize a numerical reward. The agent is not given the right actions to take but instead explores the environment and tries which actions lead to the most reward. RL problems are often described with Markov decision processes [8]. Reinforcement learning is a semi-supervised machine learning algorithm. It is different from supervised learning which consists in learning from a set of labeled data provided by a knowledgeable external supervisor [6]. The goal of supervised learning is to find the right answer while for RL it is to find how to carry out a certain task and when a data set is not available, the learner uses its previous experience. RL is also different from unsupervised learning. Unsupervised learning is a learning paradigm that aims to find correlations in sets of unlabeled data. This type of learning is not based on correct examples. Finding structure in an RL agent experience might be interesting. However, in our solution, the most important objective is to maximize the reward (e.g. maximize the collected data and minimize the energy consumption) in an environment the agent needs to explore and adapt the trajectory to the context's changes. covered by unsupervised learning. Consequently, RL is the most suitable technique to our problem.

IV. System Model and Problem Formulation

In this section, we will formally describe the environment, present our system model and formulate our problem as a Markov decision process.

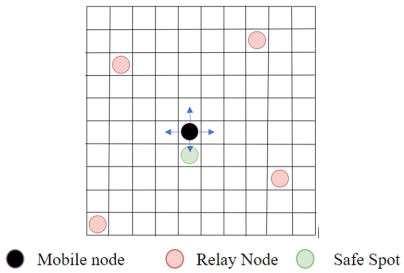

Fig. 1. System representation

TABLE I
REWARDS DEFINITION

Reward	Type	Environment Conditions
r_{ed}	Positive	if $cd_k > 0$ and $w_{c,i} = w_{m,i}$
r_{sd}	Positive	if $cd_k > 0$ and $w_{c,i} > w_{m,i}$
r_{ee}	Positive	if $cd_k = 0$ and $w_{c,i} = w_{m,i}$
r_{ne}	Negative	if $cd_k = 0$ and $w_{c,i} > w_{m,i}$
r_{move}	Negative	if the MN moves without collecting data
r_{ss}	Positive	if the MN collects data from the clusters and ends the episode on a SS
r_{finish}	Negative	if the MN energy reaches 0 without being in a safe spot

A. System Model

We consider an IoT network composed of a number of clusters randomly distributed in a square grid world of size $M \times M \in \mathbb{N}^2$ with an MN collecting data through a relay node from a number K of clusters of IoT devices. We suppose that the relay node is already chosen depending on its energy capability (i.e., the node with the biggest energy capacity). The k-th relay node is permanently located at $[x_k(t), y_k(t)]$ in the grid world with $k \in [1, K]$ and $t = 1, 2, ..., T$, represents the number of time slots. We consider the velocity $v(t)$ of the MN constant and enough for the MN to move from one square of the grid to the one next to it. We consider the velocity as constant since we want our energy-efficient solution to be adapted to any type of MN. At time step *t*, the MN has an energy capacity $u(t)$ and $w_{c,i}(t)$ the amount of energy spent to visit a cluster or a safe spot at time step t. We also consider $w_f(t)$ the minimum level of energy needed for the MN to move to a safe stop spot z with z in$[1, Z]$. A safe spot is an area in the square grid where the MN will go when its energy level is not sufficient to visit another cluster in order to recharge its energy. We calculate the minimum energy needed for the MN to reach a cluster or a safe spot by the following

$$w_{m,i}(n) = |x_m(t) - x_i(t)| + |y_m(t) - y_i(t)| \quad (1)$$

Where $[x_m(t), y_m(t)]$ corresponds to the coordinates of the MN and $[x_i(t), y_i(t)]$ are the coordinates of a cluster or a safe spot on the grid. We also suppose that the only information possessed by the MN about the clusters is their initial position. The amount of collected data by the MN from a relay node is given by cd_k. Figure 1 represents a system with an MN, four relay nodes and one safe spot.

B. Markov Decision Process

We will formulate our problem as a Markov decision process problem. In this section, we define the state space, action space and reward function. The RL algorithms will interact with the environment (the MDP) in order to find an optimal data collection policy that maximizes the data collection and minimizes the energy consumption of our MN guided by the rewards' feedback. The MDP is defined by the tuple (S, A, R, P) with state-space S, action-space A and reward function R.

The state in time t, is given by $s_t = p_t, e_t, C$ where:

- $p_t \in \mathbb{R}^2$ is the MN position on the grid;
- $e_t \in \mathbb{N}$ is the remaining energy of the MN in time t;
- $C \in \mathbb{R}^{2\times K}$ represents the coordinates of the K cluster heads.

The MN can move to one of the four adjacent grids from its current grid in each time slot. The action-space is defined as

$$A = north, east, south, west \quad (2)$$

The movement of our MN from a position p_t is expressed as

$$p_{t+1} = \begin{cases} p_t + (-X, 0) & \text{if } a_t = west \\ p_t + (X, 0) & \text{if } a_t = east \\ p_t + (0, X) & \text{if } a_t = north \\ p_t + (0, -X) & \text{otherwise} \end{cases} \quad (3)$$

where *X* is length of a square in the grid. The reward function is a function that maps state-action pairs to a real-valued reward, i.e., $R : S \times A \to \mathbb{R}$. It consists of the positive rewards and negative rewards (penalties) summarized in Table I

V. OUR SOLUTION

The main aim of our solution is to plan an adaptive and energy-efficient trajectory that maximizes the quantity of collected data and minimizes the energy consumption of the MN. For that, we want our MN to explore the environment, find an optimal trajectory and modify it in case a cluster stops its activity or in case a new cluster appears. To achieve this goal, we chose to use two RL algorithms. In this section, we will present Q-learning and SARSA and discuss why allying DL with RL is more efficient in our case. Finally, we will present our DS algorithm specified in Algorithm 1 and our DQN algorithm specified in Algorithm 2.

A. Q-learning And SARSA

The standard architecture of an RL algorithm is given in figure 2. Formally, the MN observes a state s of its environment at time step t $s_t \in S$ where S is the set of states. It executes an action $a_t \in A$, where A is the set of possible actions and interacts with its environment. This action changes the state of the environment to a new state s_{t+1} and the agent will receive a reward r_t accordingly. The agent's goal is to find a policy π that maps a state s_t to a probability of choosing action a_t. This policy is represented as following

$$\pi : S \to P(A) \quad (4)$$

The reward is the sum of discounted future rewards. To calculate the reward we use γ, where $\gamma \in [0, 1]$ is a discount

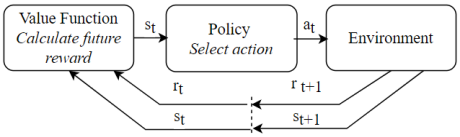

Fig. 2. Reinforcement learning standard architecture

factor that determines the effect of the future rewards to the current one and T is the time when the process terminates. The future reward is defined as follows

$$R_t = \sum_{i=0}^{T} \gamma^k r_{t+k+1} \tag{5}$$

We also define the state-action value function which represents an agent at state s taking action a under policy π

$$Q^{\pi}(s,a) = E_{\pi}[R(t)|s_t = s, a_t = a] \tag{6}$$

where $E_{\pi}[R(t)|s_t = s, a_t = a]$ is the expected return. The learning goal is to find the optimal state-action function which is based on the Bellman equation.

Q-learning is the most famous RL algorithm. It is an off-policy algorithm which means the agent will sometimes apply a policy even if it is sub-optimal in order to improve the policy. To update and estimate the current state-action value function, we use the next state-action value. It is important to note that even if the next state s_{t+1} is given, the next action a_{t+1} is still unknown. The update equation of the state-action value function in Q-learning is

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max Q(s',a') - Q(s,a)] \tag{7}$$

where α represents the learning rate which is a hyper-parameter that controls the model change in response to the estimated error. As we can see, in Q-learning the next action is chosen greedily to maximize the next state-action value $Q(s_{t+1}, a_{t+1})$. Contrary to Q-learning, SARSA is an on-policy RL algorithm. It means if the agent finds a good policy, it will be more likely to repeat it. In contrast to Q-learning, in SARSA, when updating the current state-action value, the action a_{t+1} will be taken instead of chosen greedily. Consequently, the update equation of the state-action value function in SARSA is

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{8}$$

As its name suggests, in SARSA the training quintuple is (s, a, r, a', s') which means in every update, the quintuple will be derived while $a^{'}$ in Q-learning is just for estimation.

In a complex and sophisticated environment with large state and action spaces like our environment problem, computing and updating table values for each state-action pair is not efficient. In this case, it is more interesting to find an approximation of the Q-value rather than directly computing it. In order to do that, we will use DL combined with Q-Learning and SARSA to approximate the optimal Q-function $Q_{\pi}(s,a)$. DL is a machine learning technique based on neural networks (NNs) with a powerful generalizing ability. This approach

initialize action-value with random weights θ and θ', replay memory size *H*, number of episodes *F*
initialize ϵ for exploration and ϵ_{expt}
for *episode* :$1, ..., F$ **do**
  Initialize the environment and receive initial state s_t;
  Choose a_t randomly from action space;
  Set $t = 0$;
  while $u > w_{i,ss}$ **do**
    Execute action a_t, compute r_t and observe the next state s_{t+1};
    Choose action a_{t+1} from the s_{t+1} ;
    Store experience (s_t, a_t, r_t, s_{t+1}) in the replay memory with a random placement policy;
    Sample a random mini-batch of G experiences (s_i, a_i, r_i, s_{i+1}) from replay memory;
    Calculate the target value;
    Update the weights $\theta' = \theta$ every J time step;
    Decrease u;
    Set $t = t + 1$;
  end
  Decrease ϵ;
end

Algorithm 1: DS energy-efficient Mobile Data Collection

initialize action-value with random weights θ and θ', replay memory size *H*, number of episodes *F*
initialize ϵ for exploration and ϵ_{expt}
for *episode* :$1, ..., F$ **do**
  Initialize the environment and receive initial state s_t;
  Set $t = 0$;
  while $u > w_{i,ss}$ **do**
    if $\epsilon > \epsilon_{expt}$ **then**
      Choose a_t randomly from action space;
    else
      Select $a_t = argmax Q(s_t, a_t, \theta)$;
    end
    Execute action a_t, compute r_t and observe the next state s_{t+1};
    Store experience (s_t, a_t, r_t, s_{t+1}) in the replay memory with a random placement policy;
    Sample a random mini-batch of G experiences (s_i, a_i, r_i, s_{i+1}) from replay memory;
    Calculate the target value;
    Update the weights $\theta' = \theta$ every J time step;
    Decrease u;
    Set $t = t + 1$;
  end
  Decrease ϵ;
end

Algorithm 2: DQN-based energy-efficient Mobile Data Collection

can retrieve highly abstract structures or features from the real environment and then precisely depict the complicated dependencies between raw data. However, it can not directly select a policy for decision-making problems. Consequently, allying Q-learning and SARSA with DL constitutes the most appropriate solution for our problem.

B. The proposed algorithms

We propose a DS and DQN algorithm in order to plan the trajectory of our MN. The trajectory will maximize the quantity of collected data and minimize the energy consumed by the MN. The proposed algorithm will also adapt the

Fig. 3. The training curves of the MN's energy consumption for DS and DQN

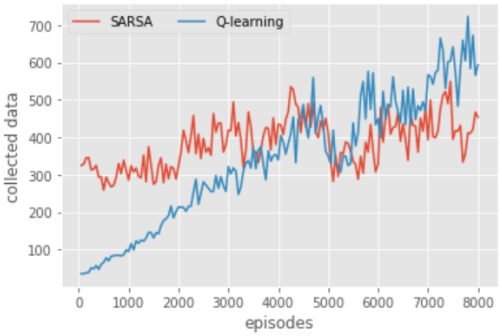

Fig. 4. The training curves of the MN's collected data for DS and DQN

MN trajectory depending on the clusters' activity. In addition to previously described specifications, DS and DQN uses experience replay which is a technique that allows us to store the agent's experiences at each time step in a data set called replay memory [17]. At time step *t*, the agent's experience is defined as following

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1}) \tag{9}$$

The main reason for using experience replay is to break the correlation of consecutive samples [18]. In our solution, we use two NNs. The policy network θ approximates the optimal policy by finding the optimal Q-function. It accepts the current state s_n and finds the evaluation of the value $Q(s_n, a_n, \theta)$. We also use a second network called target network θ' to improve the stability of learning. The target network weights are frozen with the original policy network and are updated periodically. It accepts the next state s_{n+1} and outputs the Q-value $Q(s_{n+1}, a_{n+1}, \theta')$. These values are optimized to minimize the loss function defined by

$$L(\theta) = \mathbb{E}[(T_t - Q(s_t, a_t))^2] \tag{10}$$

where T_n is the target value in the DQN algorithm. It is defined as following

$$T_t = r_t + \gamma^{t-1} maxQ(s_{t+1}, a_{t+1}, \theta') \tag{11}$$

For DS T_t is described as

$$T_t = r_t + \gamma^{t-1} Q(s_{t+1}, a_{t+1}, \theta') \tag{12}$$

where the Q-value for the next state s_{t+1} is passed to the target NN θ' for more stability in learning. ϵ is the exploration rate which represents the probability that our MN will explore the environment and ϵ_{expt} is the threshold from which our MN will stop exploring the environment and will exploit the experience acquired through the policy network.

VI. Simulation and Performance Analysis

In this section, we will present the simulations carried out in order to assess the learning performance of using DQN and SARSA. Both of our algorithms are trained for a total of 8000 episodes. We assume that four clusters are randomly distributed in a 10 x 10 units area. The battery capacity of the MN is set to 100 units. The MN consumes 1 unit of energy to move from one area to another. The MN consumes 1 unit of energy to collect data when the relay node is in its range. The cluster head is considered in the range of the MN if they are both on the same position $[x_t, y_t]$ at time step t. An episode ends if the MN collects all the cluster data and is in a safe spot or if its energy is equal to 0. The maximum of possible data to collect for the MN for each episode is between 600 and 800 units. The MN will start by exploring the environment since ϵ is initialized to 1 and will exploit the data when ϵ is inferior to 0.001 (i.e., corresponding to ϵ_{expt} in algorithms 1 and 2). The neural network is composed of an input layer of 256 neurons (i.e., the grid is fed to the NN as an image and is divided into 256 pixels). The output layer is a dense layer with the size of the action space. The configuration of the algorithms is given in Table II.

TABLE II
Parameters' values for DS and DQN

Parameters	Values
Number of training episodes	8000
Learning rate	0,01
Discount factor	0,99
Exploration rate	1 (will be decayed)
Replay memory size	50000
Minibatch size	64

We carried out this simulation with the goal of maximizing the amount of collected data while minimizing the energy consumption as well as adapting the MN's trajectory to the clusters' activity. We want to determine which approach is more suitable for our problem. It is important to note, that using learning approaches to plan a trajectory is more appropriate for networks with no critical data that need to be collected during a single training episode (e.g., all the clusters data should be collected during the maximum time where the MN can move). In order to observe the behavior of our algorithms in an environment where the context could change, we programmed a decrease in the activity for two clusters between episode 5000 and episode 6000 (e.g., no data to transmit). The clusters resume their activity after episode 6000. The rewards for this policy

$r_{ed}, r_{sd}, r_{ss}, r_{ee}, r_{nc}, r_{ne}, r_{finish}, r_{move}$ are set to 500, 50, 1000, 10, -100, -300, -500, -1 respectively.

Before episode 5000, all the clusters are active. As shown in Figure 4, for the DQN algorithm, the amount of collected data is very low at the beginning of the training and then increases gradually. This means that our MN explores the environment, visits the cluster positions and builds a trajectory to efficiently collect the cluster data. While for DS, the agent quickly learns a trajectory and visits multiple clusters. We can notice that before episode 5000, the MN under the DS algorithm collects more data while consuming less energy than under the DQN algorithm. However, the MN is not visiting all the clusters as the maximum of possible data to collect for each episode is between 600 and 799 units.

Between episode 5000 and episode 6000, we programmed a decrease in activity of two clusters. When the clusters' activity drops, we notice that the amount of collected data drops for the two algorithms. We also remark that the energy consumption of the MN drops drastically for the DQN algorithm while it remains in the same range for the MN training under the DS algorithm (as shown in Figure 3. For the DQN algorithms, these results mean that the MN does not visit the clusters which stopped their activity and stops the data collection to save energy. The clusters resume their activity after episode 6000. We can see through Figure 4 that the amount of collected data increases lightly for the DS algorithm and drastically for the DQN algorithm. These results show that the MN training under the DQN algorithm finds a trajectory that not only collects all the clusters' data during one episode while consuming less energy compared to before episode 5000. While under the DS algorithm, the MN certainly consumes less energy but does not visit all the clusters.

We can conclude through the results we obtained that in order to plan an energy-efficient and adaptive trajectory, DQN is more appropriate than DS. The DQN tries to learn the best trajectory to collect the data from all the clusters while consuming the minimum energy. The algorithm also showed great adaptability between episodes 5000 and 6000 and quickly re-learned the trajectory after episode 6000 without doing expensive re-computation and learning while enhancing the performances. We can also conclude that the DS algorithm took safer decisions (e.g., choosing a path is a more appropriate solution for problems that require less risk-taking). These results also show one of the drawbacks of SARSA as the algorithm is easily stuck at a local optimum.

VII. Conclusion

In this paper, we have introduced two energy-efficient deep RL algorithms with experience replay for trajectory planning of an MN in a MDC and cluster-based IoT networks scenario. The results we obtained in our paper show that the MN, knowing only the cluster positions, uses information about the environment to learn and find an energy-efficient data collection trajectory with both of the algorithms. However, under the same parameters, the DQN algorithm shows more adaptability to the significant changes in the context scenario (e.g., the amount of collectible data) without the need for expensive retraining or recollection of data while DS shows a less risky and safer trajectory.

References

[1] Abbasian Dehkordi, Soroush and Farajzadeh, Kamran and Rezazadeh, Javad and Farahbakhsh, Reza and Sandrasegaran, Kumbesan and Abbasian Dehkordi, Masih: A survey on data aggregation techniques in IoT sensor networks. Wireless Networks, 26(2), 1243-1263 (2020)

[2] Song, Liumeng and Chai, Kok Keong and Chen, Yue and Schormans, John and Loo, Jonathan and Vinel, Alexey: QoS-aware energy-efficient cooperative scheme for cluster-based IoT systems. IEEE Systems Journal, v. 11, pp 1447–1455 (2017)

[3] Toor, Amanjot Singh and Jain, AK: Energy aware cluster based multi-hop energy efficient routing protocol using multiple mobile nodes (MEACBM) in wireless sensor networks. AEU-Inter. Journal of Electronics and Communications, v. 102, pp 41–53 (2019)

[4] Wala, Tanuj, Narottam Chand, and Ajay K. Sharma. "Energy efficient data collection in smart cities using iot." Handbook of Wireless Sensor Networks: Issues and Challenges in Current Scenario's. Springer, Cham, 2020. 632-654.

[5] Ray, Susmita : A quick review of machine learning algorithms. 2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon) pp 35–39 (2019)

[6] Sutton, Richard S and Barto, Andrew G: Reinforcement learning: An introduction. MIT press, pp 1139–1159 (2018)

[7] Wiering, Marco A., and Martijn Van Otterlo. "Reinforcement learning." Adaptation, learning, and optimization 12.3: 729 (2012)

[8] Otterlo, Martijn van, and Marco Wiering. "Reinforcement learning and markov decision processes." Reinforcement learning. Springer, Berlin, Heidelberg, pp 3–42 (2012)

[9] Mozaffari, Mohammad and Saad, Walid and Bennis, Mehdi and Debbah, Mérouane: Mobile unmanned aerial vehicles (UAVs) for energy-efficient Internet of Things communications. IEEE Trans. on Wireless Communications, v. 16, pp 7574–7589 (2017)

[10] Lorincz, Josip and Tahirović, Adnan and Stojkoska, Biljana Risteska : A Novel Real-Time Unmanned Aerial Vehicles-based Disaster Management Framework, 2021 29th Telecommunications Forum (TELFOR), pp. 1-4 (2021)

[11] Ghorbel, Mahdi Ben and Rodríguez-Duarte, David and Ghazzai, Hakim and Hossain, Md Jahangir and Menouar, Hamid: Joint position and travel path optimization for energy efficient wireless data gathering using unmanned aerial vehicles. IEEE Trans. on Vehicular Technology, v. 68, pp 2165–2175 (2019)

[12] Hsu, Yu-Hsin and Gau, Rung-Hung: Reinforcement Learning-based Collision Avoidance and Optimal Trajectory Planning in UAV Communication Networks. IEEE Trans. on Mobile Computing, v. 16, pp 306–320 (2020)

[13] Zhang, Jidong and Yu, Yu and Wang, Zhigang and Ao, Shaopeng and Tang, Jie and Zhang, Xiuyin and Wong, Kai-Kit: Trajectory Planning of UAV in Wireless Powered IoT System Based on Deep Reinforcement Learning. 2020 IEEE/CIC Inter. Conf. on Communications in China (ICCC), v. 16, pp 645–650 (2020)

[14] Bayerlein, Harald and Theile, Mirco and Caccamo, Marco and Gesbert, David: Multi-UAV Path Planning for Wireless Data Harvesting with Deep Reinforcement Learning. arXiv preprint arXiv:2010.12461, pp 1447–1455 (2020)

[15] Bayerlein, Harald and Theile, Mirco and Caccamo, Marco and Gesbert, David: UAV path planning for wireless data harvesting: A deep reinforcement learning approach. arXiv preprint arXiv:2007.00544, pp 1447–1455 (2020)

[16] Jang, Beakcheol and Kim, Myeonghwi and Harerimana, Gaspard and Kim, Jong Wook : Q-learning algorithms: A comprehensive classification and applications. IEEE Access 7 : 133653-133667 (2019)

[17] Fan, Jianqing and Wang, Zhaoran and Xie, Yuchen and Yang, Zhuoran : A theoretical analysis of DQN. Learning for Dynamics and Control. PMLR (2020)

[18] De Bruin, Tim and Kober, Jens and Tuyls, Karl and Babuška, Robert : The importance of experience replay database composition in deep reinforcement learning. Deep reinforcement learning workshop, NIPS (2015)