TÉCNICO LISBOA

Optimization of UAV message delivery trajectories in partitioned Mobile Ad-hoc Networks using Deep Queue Learning

Ana Catarina Fernandes Rodrigues

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. António Manuel Raminhos Cordeiro Grilo

Examination Committee

Chairperson: Prof. Pedro Filipe Zeferino Aidos Tomás
Supervisor: Prof. António Manuel Raminhos Cordeiro Grilo
Member of the Committee: Prof.João Paulo Baptista de Carvalho

June 2023

Declaração

Declaro que o presente documento e um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to thank everyone that supported and encouraged me throughout this thesis.

First, I want to thank my supervisor Professor António Grilo. This work was definitely a challenge, that was only possible with my professor guiding me, being available and ready to help and to discuss certain topics.

To my parents and sister, for all the love and patience while dealing with me in times of pressure, and for always being available with caring advice and for always being present.

To the rest of my family that was always interested in motivating and cheering me up.

To my best friends, Carolina, Catarina, Beatriz, Joanas e Inesês, thank you for always being supportive, encouraging, for distracting me in difficult times, and cheering me up. You inspire me to be better and pursue what I want.

To all my friends from college, especially Carolina with your company everything was so much easier and less stressful.

Last but not least, I would like to thank my boyfriend, that motivated and helped me in a crucial part of this work, when everything seemed to be going in the wrong path. Thank you for the help, love, and patience.

You all gave me the motivation and strength to write this work, and for that, I will always be grateful.

Abstract

In our increasingly connected world, wireless and mobile communications have become indispensable. However, these essential connections are often limited to metropolitan cities with established network infrastructure. Rural and unmanaged areas, as well as disaster regions, face frequent disruptions or a lack of infrastructure. Unmanned aerial vehicles (UAVs) present a promising solution to address these challenges. This work explores the application of automatic learning techniques, specifically reinforcement learning algorithms, for establishing optimal communication paths between network nodes.

Building upon previous research, we propose the development of an algorithm that already exists: a reinforcement learning algorithm and its subsequent optimization by integrating neural networks. This enhancement aims to extend the algorithm's applicability to more complex network scenarios. To evaluate the proposed approach, it was used openAIGym to simulate, and networks comprising *N* nodes and a UAV.

The results reveal that the reinforcement learning approach proved to be the most efficient, obtaining results comparable to the previous work upon which our study was based. However, when applied to more complex networks, improvements are necessary to achieve better outcomes. While it was expected that the introduction of neural networks would enhance the performance of reinforcement learning, challenges were encountered that must be addressed in future research. Future work will focus on resolving the identified challenges in the last approach.

Keywords

Communications; Machine Learning; Reinforcement Learning; Neural Networks ;Unmanned Aerial Vehicles;

Resumo

Neste mundo cada vez mais conectado, as comunicações wi-fi e móveis tornaram-se indispensáveis. No entanto, estas conexões essenciais estão geralmente limitadas a cidades metropolitanas com infraestrutura de rede estabelecidas. Áreas rurais bem como regiões de desastres, enfrentam interrupções frequentes ou falta de infraestrutura. Os veículos aéreos não tripulados (UAVs) apresentam uma solução promissora para enfrentar estes desafios. Este trabalho explora a aplicação de técnicas de aprendizagem automática, especificamente algoritmos de aprendizagem por reenforço, para estabelecer caminhos de comunicação ótimos entre os nós da rede.

Com base em pesquisas anteriores, propomos o desenvolvimento de um algoritmo já realizado num trabalho anterior: aprendizagem por reenforço e de seguida a otimização por integração de redes neuronais. Esta modificação visa estender a aplicação do algoritmo a cenários de rede mais complexos. Para avaliar a abordagem proposta, foi utilizado o openAIGym para simular, e redes compostas por N nós e um UAV.

Os resultados revelam que a abordagem através da aprendizagem por reenforço provou ser a mais eficiente, obtendo resultados comparáveis ao trabalho anterior em que se baseou o nosso estudo. No entanto, quando aplicado a redes mais complexas, são necessárias melhorias para alcançar melhores resultados. Embora fosse expectável que a introdução de redes neurais melhorasse o desempenho do aprendizagem por reenforço, foram encontrados desafios que devem ser abordados em pesquisas futuras. O trabalho futuro incidirá na resolução dos desafios identificados na última abordagem.

Palavras Chave

Comunicações; Aprendizagem automática; Aprendizagem por reforço; Redes neuronais; Drones;

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ANN	Artificial Neural Network
AD	Average Delay
FRA	Ferry Relaying Algorithm
DQN	Deep Q-Network
DR	Deliver Ratio
DRL	Deep Reinforcement Learning
DRL-SA	Deep Reinforcement Learning Scheduling Algorithm
DQL	Deep Q-Learning
DTN	Delay Tolerant Network
DTNs	Delay Tolerant Networks
GA	Genetic Algorithm
HoP-DTN	Homing Pigeon based DTN
LoS	Line of Sight
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MF	Multiple Ferries
ML	Machine Leaning
MPT	Microwave Power Transfer
MRT	Multiple Route

MURA	Multiple Route Algorithm
NRA	Node Relaying Algorithm
RG	Relative Gain
RL	Reinforcement Learning
RR	Round Robin
SGD	Stochastic Gradient Descent
SIRA	Single Route Algorithm
SRT	Single Route
TD	Temporal-Difference Learning
TSP	Travelling Salesmen Problem
UAV	Unmanned Aerial Vehical
UAVs	Unmanned Aerial Vehicals

Nomenclature

$V_\pi(s)$	Optimal Value Function for Policy π
α	Learning Rate
β	Decay Rate for the reward
ϵ	Exploration Rate
γ	Discount Factor
π	Policy
a'	Possible Next Action
$A(s)$	Set of Actions that can be taken from state s
A	Agent
a	Action
A_t	Action at Time t
E	Environment
E_π	Expected Value for Policy π
e_t	Agent's Experience
F	Sum of Flow Rates
$feature$	Vector of weights used to find the optimal set of weights
G_t	The Sum of Future Rewards at Time t
N_0	Number of Packets in All Nodes at the Beginning of a Action

$p(s, rs, a)$ The Probability of Transitioning to state s' with Reward r from taking Action a in State s

P Policy P

$Q(s, a)$ Q-Value for each State-action Pair

q_π Q-value Function for Policy π

$R(S_t)$ Reward of State at Time t

R_t Reward at Time t

$rate$ Decay Rate for Exploration vs. Exploitation Decision

s' Possible Next State

s State

S_t State at Time t

t Temporal Instance

t_a Duration of a Action

$V(S_t)$ Value Function

v Velocity

v_π State-value Function for Policy π

$V_t(s)$ Optimal Value Function

w Vector of Weights

1

Introduction

Contents

1.1 Context

Due to their great mobility and low cost, Unmanned Aerial Vehicals (UAVs), often known as drones or remotely piloted aircraft, have found a wide range of applications in recent decades, and have become available to the general public. UAVs were first used in the military to deliver material objects, but as they became less expensive and smaller, also became more useful for other applications such as weather monitoring, traffic control, freight delivery, emergency search and rescue, forest fire detection, communication relay, and nowadays, the most popular use is in photography and video purposes.

A drone can follow a path controlled by a human on the ground or guided by a computer program that calculates the optimal path for the Unmanned Aerial Vehical (UAV) to take. In this thesis, the UAV will be used in the communication relaying theme, where a drone can be used: as a Base Station in order to cover a certain area to allow coverage for the users, depending on the area it may need a large number of UAVs, or as a ferrying node, where UAVs are used to send and receive messages between users.

This study will be done using a Delay Tolerant Network (DTN), a network in which nodes are not connected continuously. To forward data, these networks can have two different approaches, reactive one: where the delivery is achieved through the inherent mobility of existing mobile nodes or, proactive one: where some mobile nodes are introduced and their mobility is controlled, by a human or a computer. The proactive approach will be investigated because the DTN network will only contain static nodes. The extra mobile node will be an UAV.

In this report the drone will communicate with all of the nodes allowing them to send messages between all of them, in an All-to-All scenario. To optimize and choose the best route, the drone may follow, in order to deliver the messages between the nodes there are several approaches: in this work, a learning technique will be used, that learns the optimal and best path through Deep Reinforcement Learning (DRL), a subfield of machine learning that combines Neural Networks and Reinforcement Learning (RL), will be investigated in this study in order to culminate in a solution and have conclusions to the routing problem. Figure 1.1 is an illustration of the scenario explained before. This strategy is still limited as explained in the article [5], and no additional works have been generated using this methodology, necessitating further exploration.

1.2 Motivations

In the twenty-first-century wireless communication and connection are ubiquitous, particularly in metropolitan areas with mobile and network coverage, allowing users to stay connected at all times and in all locations. Some places face other reality, where network infrastructures do not exist, either because they are too expensive to build, or the place is located on a non-developed rural area, or it's a war zone,

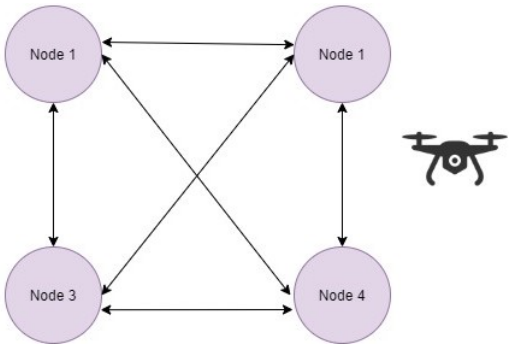


Figure 1.1: All-to-All situation, each node can receive and send data for and from each other node, with the help of a mobile node, the UAV

or even in a small island in the middle of nowhere in the ocean.

However, there is a need to communicate in these situations, so UAVs can be a mechanism that allows this to still happen, without the need to build expensive infrastructure. For example, in an underdeveloped area of the world, in a rural environment, with small populated villages dispersed in a country where there is no network and communication infrastructure, is still necessary to be able to communicate between villages, the UAVs can allow this to happen.

For example if occurs an earthquake/tsunami, communication infrastructures can get compromised, and that is a need to communicate with the emergency services and inform the population, an UAV can do that job, becoming a base station for telecommunications. Another example is in a rural village, where there is no money to afford to build an infrastructure for a base station, the UAV can be used for telecommunication purposes allowing the rural village to communicate with other places.

The optimal path is a special need for this way of communicating using UAVs, these devices have a battery that needs to be recharged sometimes, and the messages they carry can have a certain temporal duration, that expires. Even though there are already some drones that use combustion engines. Another factor is the size of the drone, a bigger drone can have engines with more power and efficiency, in order to deliver faster and safeguard the battery.

Whit this brief explanation, we are not trying to consider the UAVs as the best alternative to the lack of communication/network infrastructures, it is necessary to take into account other solutions such as satellites, balloons, and also the price of a UAV is a factor to consider compared to other solutions in the market. This work based in the use of UAVs is only to help in the search of alternatives in the message delivery scenario.

1.3 Objectives

The goal is to determine the best and optimal path through an algorithm for a DTN with *N* static nodes that will interact via a message carrier, which will be a single UAV. The requirements are to deliver the largest number of messages in the shortest period of time possible.

The path optimization algorithms considered in this thesis belong to the class of Reinforcement Learning algorithms, namely Deep Reinforcement Learning.

In this work the intention is to propose new algorithms and evaluate them through simulation.

1.4 Outline

In order to have the objectives, theoretical approach, and articles this work relied on, as the practical part, this study is organized into five chapters.

The first chapter introduces this theme, its main purpose and motivations, the use of UAVs to deliver and send messages from/to nodes, and a brief context of the UAVs in solving network issues.

The second chapter is the State of the Art, in which the story of UAVs throughout the years is described, what they are used and their use cases in UAV-aided wireless communications. The basis of structuring problems solved with RL is Markov Decision Process (MDP), so there is a subsection that highlights this concept. At the end of this chapter a theoretical explanation of RL and DRL is performed, and a pseudo-code of the algorithms can be found. A brief explanation of the tool used to perform the algorithms, Open AI Gym, and how it works. At the end of this chapter, all of the related work is commented on, this will be the main basis for the decisions in the course of this work, with a special focus on the one written by [4], that performed an All-to-All scenario where each node can send and receive messages to each other network node.

In the third chapter, the practical part of this work is explained; the proposed algorithms are explored and discussed further.

The fourth chapter, is where the obtained results are presented. In this chapter, the differences between the Reinforcement Learning RL and Deep Reinforcement Learning DRL are presented and discussed. A comparison with the All-to-All Scenario performed in [4] is also analyzed.

Lastly, in the fifth chapter, conclusions and comments on this work will be made.

2

State of the Art

Contents

The concerns of this work are to find an optimal path that a UAV should take in a DTN with *N* static nodes, in order to send messages, and see how much time the UAV took to deliver them.To find the optimal path, two algorithms will be developed. The first one using RL and the other with DRL.

This chapter will provide an overview of UAVs, their usage, and requirements. RL and DRL theory will also be investigated further. The used software to create and simulate the environment is presented. This software, Open AI Gym is a toolkit for Reinforcement Learning algorithms development and simulation. It supports Python, which was the used programming language to write the code used to simulate the environment of the UAV and the nodes.

In order to facilitate the understanding of this topic an analysis of related works will be performed.

2.1 Unmanned Aerial Vehicles

2.1.1 History of the UAVs

Unmanned Aerial Vehicles, known as drones, have had a huge crescent impact in the 21st Century. They are becoming less expensive and highly mobile, allowing them to find a large field of applications where their use is beneficial. This range of applications goes from military purposes to delivery of material objects, weather monitoring, traffic control, freight delivery, emergency search and rescue, forest fire detection to communication relaying, recently they have been used for photography and video purposes, this last use is the most popular one. Nowadays, being small and less expensive, drones are available to the general public.

UAVs are not only used because of their size and cheap prices, but also, for communication purposes, on-demand UAVs can be more swiftly deployed, which makes them more adequate for unexpected and time-limited missions. UAVs fly in a low-altitude and have a short-range Line of Sight (LoS), so the communication connections can be easily made in most scenarios, this will lead to a significant improvement in the performance compared to communications between source and destination. The fact that UAVs move easily makes them more adaptable to the type of communication environment, allowing for better performance.

As highlighted before, a drone can follow a path controlled by a human on the ground or guided by a computer program that calculates the optimal path for the UAV to take. There are three typical use cases of UAV-aided wireless communications [6]:

- UAV-aided ubiquitous coverage: when a UAV is employed to help communication infrastructure by providing wireless coverage in a specific area. For example, if the infrastructure has been damaged, either partially or totally, as a result of natural disasters, or if the base station is in an offloaded crowded area;

- UAV- aided relaying: UAV is used to facilitate communication between individuals or groups of users who haven't access to a reliable direct communication link. If one intermediate node fails and disconnects the link, per example, this UAV would be used as a temporary replacement for the faulty one;
- UAV- aided information dissemination and data collection: where UAVs are used to provide (or gather) delay-tolerant information to or from a large number of dispersed wireless devices, such as wireless sensors in precision agriculture;

2.2 Markov Decision Processes

MDP allows to formalize sequential decision-making. This is the basis of structuring problems solved with RL. The main objective is to estimate the value for each element of a function $Q_*(s, a)$, that is the value for each state-action pair or else, for $V_*(s)$, which is the associated value for each state. These values are given by following an optimal policy. The associated value of each state-action pair is the discounted sum of the rewards received in each state and future ones. MDP has a decision maker, the *agent*, that interacts with the *environment* it's placed in. These interactions occur sequentially over time. At each time step, the agent will get some representation of the environment's *state*, and perform a selection of the action to take, the environment will then transition to a new state, and a *reward* will be given to the agent as a consequent of the action the agent chose. The goal is to maximize the actual and cumulative reward. The components of a MDP are a tuple (A, E, S, A, P):

- **Agent**- is the entity which is being trained to make correct decisions.
- **Environment**- it is the world where the agent moves, and responds to the agent. The environment takes the agent's current state and action as input, and as output returns the agent's reward and next state.
- **State**- is a concrete and immediate situation in which the agent finds itself.
- **Action**- set of all possible moves the agent can make at the current time step.
- **Policy**- is the process behind picking an action. In practice is the probability distribution assigned to the set of actions.

Another important definition of MDPs is the Markov Property, that given the current state, the future state does not depend on the past ones. The future state only depends on its immediate previous state.

At each time step t = 0,1,2,.. the agent receives some presentation of the agent state S_t, and based on this state the agent selects an action A_t, this gives us (A_t,S_t) pair. Time is incremented to the next

time step $t+1$, environment transitions to S_{t+1} and the agent receives a numerical reward, R_{t+1} for the action A_t taken at S_t. At each time t, it has $f(A_t,S_t) = R_{t+1}$.

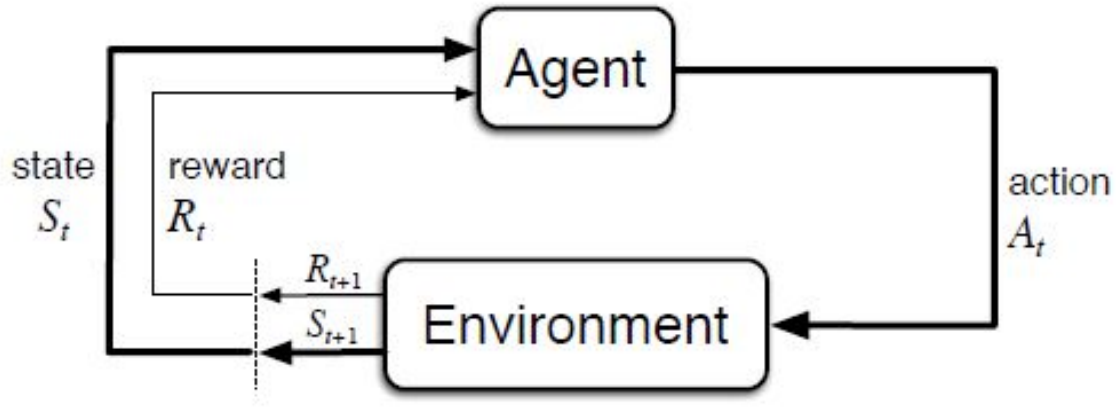


Figure 2.1: The agent-environment interaction in RL [1]

In Figure 2.1, a state transition is represented. At temporal instant t the environment is at state S_t, the agent observes the environment and selects the action A_t. Then, the environment is shifted to state S_{t+1} and grants the agent R_{t+1} as a reward.

The reward R_t and state S have well-defined probability distributions. These distributions depend on the preceding state and action that occurred in the previous time $t-1$. For example, $s' \epsilon S$ and $r \epsilon R$, there is some probability $S_t = s'$ and $R_t = r$. This probability is determined by the particular values of the preceding state $s \epsilon S$ and action $a \epsilon A(s)$. $A(s)$ is a set of actions that can be taken from state s. The probability of transitioning to state s' with reward r from taking action a in state s can be defined with for all $s' \epsilon S$, $r \epsilon R$, $a \epsilon A$:

$$p(s', r|s, a) = P_r\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

The goal of an agent in an MDP is to maximize its cumulative rewards. The concept of *expected return* of the rewards at a given time step corresponds to the sum of future rewards, defined as G at time t (G_t). This concept is crucial because the agent's objective is to maximize the expected return, it is what drives the agent to make the decisions it makes. Another important concept is the discounted return, now the agent's goal is to maximize the expected discounted return of rewards, he will choose action A_t at each time t to maximize the expected discounted reward. There is a discount factor, $\gamma\ \varepsilon[0,1]$, it will be the rate for which we discount future rewards and will determine the present value of future rewards. Our agent will care more about immediate reward over future rewards since future rewards will be more heavily discounted. After presenting the concept of an MDP, it is necessary to explain policy and value function. The probability of an agent selecting a specific action from a specific state is defined by the

policy. In terms of rewards, selecting one action over another in a given state may increase or decrease the agent's reward, the value function will help the agent decide which action to take in order to maximize its reward.

A policy is a function that maps a given state to probabilities of selecting each action from that state and can be denoted as π. An agent follows policy π at time t, so $\pi(a, s)$ is the probability that $A_t = a$ if $S_t = s$. For each state $s \epsilon S$, π is a probability distribution over $a \epsilon A(s)$.

Value functions are functions of state-action pairs, that can estimate how good it is for the agent to perform an action in a given state. The notion of how good a state-action pair is depends on the expected return. Consequently, value functions are defined with respect to specific ways of acting, because the way an agent acts is influenced by the policy it's following. The *state-value function* for policy π is v_π and tells us how good any given state is for an agent following policy π (gives us the value of a state under π). Mathematically, the value of state s under policy π is the expected return from starting from state s at time t and following π:

$$v_\pi(s) = E_\pi[G_t|S_t = s].$$

Another function is the *action-value function*, q_π, which can estimate how good it is for an agent to take any action from a state while following policy π (gives us the value of an action under π). Mathematically, the value of action a in state s under π is the expected return from starting at state s at time t, taking action a, and following π.

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a].$$

The action-value function is commonly known as Q-function and the output for any given state-action pair is called Q-value. The letter Q is used to represent the quality of taking a given action in a given state.

An RL algorithm learns optimal policies, the main goal is to find a policy that will give a lot of rewards to the agent, if the agent follows that policy. A policy π is considered to be better or equal to policy π' if the expected return of π is greater than or equal to the expected return of π' for all states.

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v'_\pi(s) \forall s \epsilon S.$$

$v_\pi(s)$ gives the expected return for starting in state s and following π. An optimal policy is a policy that is better than or at least the same as all other policies. An optimal policy has an associated *optimal state-value* function, that gives the largest expect return achievable by any policy π for each state, defined

as:

$$v_*(s) = \max_{\pi} v_\pi(s). \tag{2.1}$$

Of course, there is an *optimal action-value function* associated too, (optimal Q-function), that gives the largest expected return achievable by any policy π for each possible state-action pair, defined as:

$$q_*(s, a) = \max_{\pi} q_\pi(s, a). \tag{2.2}$$

The optimal *Q-function* (q_*) has one fundamental property: it must satisfy the *Bellman optimal equation*. This equation states that for any state-action pair (s, a) at time instant t, the expected return from starting in state s and choosing action a, by following the optimal policy, the expected return will be equal to the expected reward which is R_{t+1}. This reward is obtained by selecting action a in state s plus the maximum of the expected discounted return. The maximum of the expected discounted return is achievable of any possible next state-action pair (s', a') where state s' and a' is the potential next possible state-action pair.

The Bellman equation is defined as:

$$q_*(s, a) = E[R_{t+1} + \gamma \max_{a'} q_*(s', a')]. \tag{2.3}$$

The use of the Bellman equation is common to find q_*. The optimal policy can be determined because with q_*, for any state s, with a reinforcement learning algorithm, it is possible to find the action a that maximizes $q_*(s, a)$

2.2.1 Exploration vs. Exploitation

There are two important concepts that are presented in learning algorithms. Exploration is the act of exploring an environment to find information about it. Exploitation is the act of exploiting information that is already known about the environment, in order to maximize the expected return. The main question is how the agent learns and decides which type of actions to take. His main goal is to maximize the expected return, but only using exploitation is not right.

For example, after lunch, a person needs to work and be concentrated, but on that day is suffering from afternoon fatigue and needs to be more productive. Usually, that person drinks a coffee and knows it will make her more awake and ready to work, this is exploitation, or that person can try different approaches, like taking a nap, doing some other activity, and then start working again, trying something new and observe what will contradict the afternoon fatigue: this is exploring. There needs to be a balance between trying something usual and trying new things. The person will observe how productive is in each day and will understand better what strategies are more effective, eventually consistently choosing the most reliable one.

In order to get the balance between exploiting and exploring, there are some techniques, but we will enhance the epsilon greedy strategy. An exploration rate ε will be initially set to 1, and it is the probability that the agent will explore the environment, rather than exploit it. In the beginning, it is one hundred percent sure that the agent will explore. As the agent learns more about the environment, at the beginning of each episode, ε will decay, so the probability of the agent exploring becomes smaller, while the agent is learning and knowing more about the environment, it will become greedy in terms of exploiting. In each time step, to determine if the agent will exploit or explore, a random number is generated, between 0 and 1. If it is greater than epsilon, will choose via exploitation, chooses the action with the highest Q-value in that state. On the contrary case, if the random value is smaller than epsilon, the next action will be chosen via exploration, choosing a random action and seeing what happens in the environment.

2.3 Reinforcement Learning (RL)

Since the second half of the $20th$ century RL has been an exciting and significant topic in Machine Leaning (ML), one main reason for this popularity is due to computer algorithms, that have been able to achieve human-level performance on games. RL is a field of ML, next to Supervised and Unsupervised Learning. It is a way of solving MDP and consists of an agent, placed in an environment that learns to perform actions that are rewarded, leading to the maximum reward, using exploitation and exploration of information.

Being one of ML pillars, RL in contrast to Supervised learning, that is continuous value prediction and class/label prediction, it has brought interest because it is an algorithm that learns by himself.

There are two groups of RL, model-free and model-based algorithms. The first one does not learn the model of their environment's transition function to make predictions of future states and rewards. These methods are Q-Learning and Deep Q-Networks or Policy Gradient methods, they don't create a model of the environment's transition function. [1]

2.3.1 TD-Learning

Temporal-Difference Learning (TD), learns from raw experiences without knowing the dynamics of the environment, it does not have to wait for the outcome of the full simulation to update the value function - bootstrapping. There are two types of TD:

- TD Prediction - there is a constant policy π, experiences to update the $V_\pi(s)$, which represents the expected future discounted reward when the agent starts in state s and follows policy π. Tries to see how good a policy is by searching the optimal value function $V_\pi(s)$ for that policy;

- TD Control - The policy is not defined. The objective is to learn the optimal policy through experience, which maximizes the expected total reward.

A very well-known algorithm of TD-Learning is TD(0) which will be explained.

TD(0)

TD(0) is the simplest form of TD-Learning and it is mainly known as one step TD, a prediction algorithm and model-based RL. TD(0) must calculate the next state for each possible action in the current state, and the value function must be updated after each step with the value of the next state, as well as the obtained reward along the way. This reward is the factor that keeps the learning grounded, and the algorithm converges after a sufficient number of episodes. The main objective is to obtain the optimal value function when following a known policy, as the name indicates, "one step TD", TD(0) only needs to wait until the next time step to update the value function $V(S_t)$. TD(0) can be written as

$$V(S_t) = V(S_t) + \alpha * [R_{t+1} + \gamma * V(S_{t+1}) - V(S_t)]; \tag{2.4}$$

$$V(S_t) = (1 - \alpha) * V(S_t) + \alpha * [R_{t+1} + \gamma * V(S_{t+1})]. \tag{2.5}$$

where $V(S_t)$ is the value of the existing state value, and $R_{t+1} + \gamma * V(S_{t+1}) - V(S_t)$ is the new state value or more commonly known TD-Target. It is the received reward when transiting to a new state, plus the value of the next state times the discount rate factor $\gamma \epsilon [0, 1]$, which is used to control the importance given to future rewards. The value of the current state at time t is weighted with $1 - \alpha$ of the previously stored one and α times the TD-target. The learning rate, $\alpha \ \epsilon [0, 1]$ and states the importance of new information over the information that was already learned.

Algorithm 1 Tabular TD(0) for estimating v_π

Initialize policy π to be evaluated;
Initialize Learning rate $\alpha \ \epsilon \ [0, 1]$;
Initialize V(s) for all s $\epsilon \ S^+$, arbitrarily, with $V(terminal) = 0$;
for each episode **do**
  Initialize s;
  for each step of episode **do**
    $s \leftarrow$ action given by π for s;
    Take action a, observe reward, r and next state s';
    $V(s) \leftarrow V(s) + \alpha[r + (s') - V(s)]$;
    $s \leftarrow s'$;
    until s is terminal;
  end for
end for

2.3.2 Q-Learning

Q-learning is an off-policy reinforcement learning algorithm that is used for learning the optimal policy in a MDP, the objective is to find the best action to take given the current state. The expected value of the total reward over all successive steps is the maximum achievable, the goal of Q-learning is to find the optimal policy by learning the optimal Q-values for each state-action pair. It is off-policy because the algorithm learns from actions that are outside the current policy. This algorithm uses value iteration, which consists in iteratively updating the Q-values for each (s,a) pair, using the Bellman equation until the Q-function converges to the optimal Q-function q_*. The update of the state-action pair values is done by:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha * [R_{t+1} + \gamma * \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{2.6}$$

where, γ is the discounted factor $\epsilon\ [0, 1]$ and sets the importance of future rewards.

Algorithm 2 Q-Learning: An off-policy TD control algorithm for estimating optimal policy

Initialize Learning rate $\alpha\ \epsilon\ [0, 1]$;
Initialize Q(s,a) for all s $\epsilon\ S^+$, arbitrarily, with $Q(terminal) = 0$;
for each episode **do**
  Initialize s;
  for each step of episode **do**
    choose a from s using policy derived from Q(e.g.$\epsilon - greedy$);
    Take action a, observe reward, r and next state s';
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + (s') - Q(s)]$;
    $s \leftarrow s'$;
    until s is terminal;
  end for
end for

The information in sections 2.3.1 and 2.3.2 is presented on the basis of [1].

2.3.3 Implementation of Reinforcement Learning in DTNs

In 2005, Henkel et al. [5] had a scenario where in a DTN, there are *N* task nodes distributed on a survey area. These task nodes need to forward data to a monitoring station. Each of them has a buffer that generates data flow, and messages to be sent in the network. For that, a mobile helper node is introduced, a ferry, that travels at a constant velocity and has a large buffer. It gathers the messages from the nodes to the hub in order to retrieve the collected messages. The goal of the ferry corresponds to choose routes that minimize the average packet delay through the network. The main idea is to exploit cycles that visit a subset of nodes before returning to the hub for data delivery. The ferry learns by observing a model of the system, and updates its flight plan. To explore this problem, an MDP is formulated and the RL algorithm is applied, TD(0). In order to achieve its goal, the ferry visits the task

nodes following an optimal sequence, the policy. The ferry relies on observing the nodes buffer state and learns through the received reward when returning to the hub to deliver the traffic. The reward is given by:

$$r(s,a) = -\int_0^{ta} (Ft + N_0)e^{-\beta t}\, dt, \tag{2.7}$$

where, *s* is the state, *a* is the action, t_a is the duration of the action, *F* is the sum of all flow rates from each node, N_0 is the number of packets in all nodes at the beginning of the action and β is a decay for the reward to compensate nodes that are further away. The agent has an incentive to visit nodes which have accumulated a large buffer of undelivered traffic. However, it also tries to penalize flying to a far-away node more than visiting a near-by node.

Each state has a value that represents the benefit of being in that state. This value is the expected value of the sum of the discounted future rewards obtained when following a fixed strategy and starting in that state. The value function maps states to state values, once the optimal value function is found, the optimal policy will be easily extracted.

$$\pi(s) = arg\max_{\pi}(r(s,a) + e^{-\beta t_a} V_t(s_{t+1})). \tag{2.8}$$

where, t_a is the duration of an action and s_{t+1} is the following state of s. Following this, the optimal value function is:

$$V_{t+1}(s) = V_t(s) + \alpha(r(s,a) + e^{-\beta t_a} V_t(s') - V_t(s)). \tag{2.9}$$

where α is the learning rate. In situations which is applied the action selected by policy π on state s, the resulting state is, s'. In order to explore rather than exploit, the agent will sometimes choose a random action with probability ϵ as explained above in Section 2.2.1.

In [5] it was tested an approach using four algorithms, Round Robin (RR), an algorithm based in the Travelling Salesmen Problem (TSP), Stochastic model and RL. In order to compare the performance of the algorithms, the relative gain was defined, as the percent reduction in average delay compared to the worst-performing algorithm. The average delay is reduced when learning agents were introduced. The computational complexity of the problem rises exponentially with the increase of network nodes. Finding a route for 3 nodes takes 1 hour, 6 hours for 4 nodes, and for 5 nodes takes 20 hours. The state representation comprises all combinations of traffic present in each node, and the action space is all the combinations of all possible paths from going from one node to all of the others. The fact that it is a hub-to-node scenario is very restrictive for real-life situations.

2.4 Deep Reinforcement Learning (DRL)

Adding Artificial Neural Network (ANN) to RL, introduces a new concept, DRL. The main difference between RL vs.DRL, represented in Figure 2.2, is that instead of having a Q-value table for each state-action pair with the optimal Q-value for a single action, there is a ANN that will train the state-action pair and compute the Q-value for each possible action for that state. With this the algorithm doesn't need to run the network for every action, this will increase speed and performance.

ANN produces a great job in approximating functions. In this case, instead of computing Q-values and finding the optimal Q-function, an approximation function is used to estimate it. Q-learning explained earlier is a learning method. Combining these two concepts brings a new learning process Deep Q-Learning (DQL). This approximation function is obtained via ANN and is called Deep Q-Network (DQN). Figure 2.3 illustrates the combination of Neural Network and Reinforcement Learning algorithms.

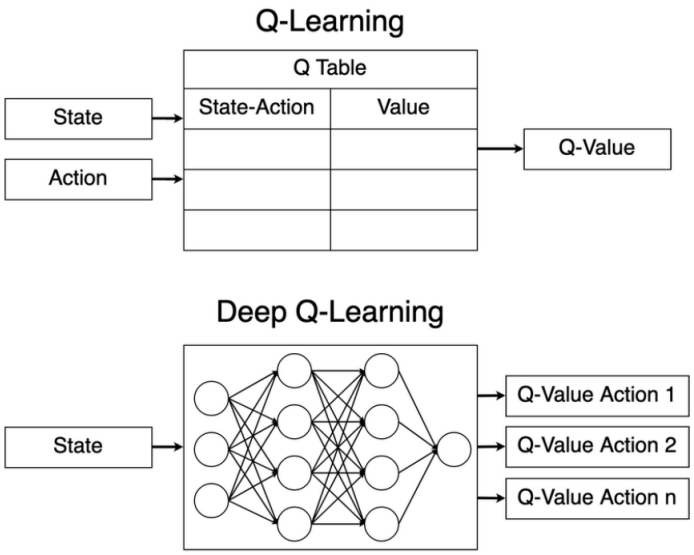


Figure 2.2: Illustrated Difference between Q-Learning and Deep Q-learning

2.4.1 Deep Q-Learning

There is an arbitrary ANN that, as an input, accepts states from a given environment. For each state, the network computes estimated Q-values for each action that can be taken from that state. This network's goal is to approximate the optimal Q-function, that will satisfy the Bellman equation presented in Equation 2.3. The loss of the network is calculated by comparing the outputted Q-values to the targeted Q-values from the right-hand side of the Bellman equation. The objective corresponds to minimizing this loss. After the loss calculation, the network weights are updated via Stochastic Gradient Descent (SGD). This

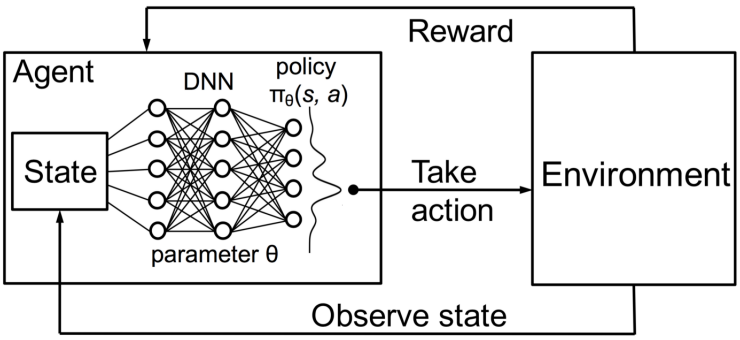


Figure 2.3: Combination of Neural Networks with Reinforcement Learning [2]

update is performed iteratively until the loss is minimal and an approximate optimal Q-function is found. With the DQN, it is possible the usage of the Bellman equation to estimate the Q-values with the goal to find the optimal Q-function. The layers of a DQN, are just input layers, followed by a non-linear activation function, just like a normal neural network. The output layer is a fully connected layer, that produces the Q-value for each action that can be taken from the given state passed as an input. During the training process of a DQN, a technique called experience replay is used, where the agent's experiences at each time step are stored in a data set - replay memory. At temporal instant t, the agent's experience e_t is defined in a tuple:

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1}). \tag{2.10}$$

This produces an insight of the agent's experiences at temporal instant t, containing the state of the environment s_t, the action a_t taken from this state, the reward r_{t+1} given to the agent at time $t+1$ as a result of the previous (s, a) pair, and the next state of the environment s_{t+1}. So, the replay memory stores the agent's experiences at each time step over all episodes played by the agent. The stored experiences will be used to sample batches for network training. Experience replay is the act of gaining experience and sampling from replay memory. The key reason to use replay memory instead of providing the network sequential experiences as they occur in the environment is to break the correlation between consecutive samples, leading to inefficient learning. In the beginning, the replay memory is initialized with capacity N in order to hold N experiences. The weights in the network are initialized. For each episode, it is also initialized the starting state. In order to gain experience, the agent either exploits and chooses a greedy action (highest Q-value) or explores, and selects a random action. The chosen action is executed, the reward and next state are observed and this experience of the agent is stored in replay memory. After being stored, random experiences are sampled from relay memory, and a state is extracted from those samples and passed to the network as input. The process occurs as explained earlier in this topic, the model outputs a Q-value for each possible action and the loss is calculated with

the Bellman equation. From the Bellman equation, we have this term:

$$\max_{a'} q_*(s', a'). \tag{2.11}$$

where s' and a' are the state and action of the following time step. So, s' is passed to the policy network that will output Q-values for each (s, a) pair using s' as the state and all the possible actions as a', it is then possible to obtain the max Q-value. Now this term is calculated for the original state input passed to the policy network.

$$E[R_{t+1} + \gamma \max_{a'} q_*(s', a')] \tag{2.12}$$

The loss is able to be computed between the Q-value given by the policy network for the original experience (s, a) pair tuple and the target optimal Q-value for this same (s, a) pair. In order to update the weights in the network, SGD is performed with the purpose of loss minimization, bringing the policy network output Q-values more closer to the target Q-values given by the Bellman equation. This process is done iteratively for each time step, until the end of the episode.

The following pseudo-code represents what was explained above:

Algorithm 3 Deep Q-Learning with Experience Replay

```
Initialize network Q;
Initialize target network Q';
Initialize experience replay memory D;
Initialize the Agent to interact with the Environment;
for each episode do
    /* Sample phase
    ε ← setting new epsilon with ε − decay
    Choose an action a from state s using ε − greedy(Q) policy
    Agent takes action a, observe reward r, and next state s'
    Store transition (s, a, r, s', done) in the experience replay memory D;
    if enough experiences in D then
        /* Learn phase
        Sample a random minibatch of N transitions from D
        for every transition (s_i, a_i, r_i, s'_i), done_i in minibatch do
            if done_i then
                y_i = r_i
            else
                y_i = r_i + γ * max'_{a ε A} Q'(s'_i.a')
            end if
        end for
        Calculate the loss L = 1/N Σ_{i=0}^{N−1} (Q(s'_i, a_i) − y_i)^2
        Update Q using SGD algorithm by minimizing the loss L
        Every C steps, copy weights from Q to Q'
    end if
end for
```

2.5 Open AI Gym

Open AI Gym is a tool/environment used for developing and testing learning algorithms. It has a library for *Python* language, and it allows the agent to interact in the environment. Gym has a set of custom environments that can be used, a class named *Env* implements the simulator where the environment and the agent will be trained. For environment description, we need the *observation_space* and *action_state*, the first one defines the structure and the values for the observation of the environment's state, and the other defines the actions that can be applied to that state in the environment. For the agent-environment iteration, there are a set of functions in the class *Env*:

- reset: resets the environment to its initial state. The observation space corresponds to the initial state;
- step: This function takes an action as input and applies it to the environment, which will transit to a new state. This function returns observation (observation of the new state), a reward (the reward the agent will receive for executing such action), done (if the episode is terminated, in order to reset the environment), and info (information about the environment).

In this simulation, the environment is designed *NodesEnv*. This environment is a map with nodes and a UAV, that is the agent, that will send messages to every node in the map. The *action_space* is the number of Nodes that the UAV can visit and the *observation_space* is the number of messages each node and the UAV have to send in its buffer.

Accordingly to [4], the state will be represented through:

- Messages in the UAV;
- Messages in each node;
- Number of generated messages;
- Number of expired messages.

2.6 Related Work

As explained before, Delay Tolerant Networks (DTNs) are sparse networks where complete direct end-to-end paths between source and destination do not always exist. Routing mechanisms in DTN rely on the nodes' mobility to connect disconnected nodes, by carrying messages around the network to overcome path disconnection. UAVs can be used as DTN relays, due to their maneuverability and accessibility. These devices are only beneficial when there is no mobile communications infrastructure or it's damaged and it's not possible to store, carry and forward data and information.

Routing in DTNs with dedicated nodes is a problem that's been looked into since the beginning of the 21st Century. First, it was considered a single ferry, using a single messenger as a ferry for messages forwarding in DTNs. A single ferry was not scalable in terms of traffic load, network size or geographic coverage, leading to solutions based on Multiple Ferries (MF).

In 2005, Zhao et al. [7] , introduced the MF concept in networks with stationary nodes, studying if the MF should take the same route or multiple routes. The author formulated four algorithms:

- Single Route Algorithm Single Route Algorithm (SIRA): all available ferries follow one route.
- Multiple Route Algorithm Multiple Route Algorithm (MURA): where ferries follow multiple routes to relay data, but don't exchange data between themselves.
- Node Relaying Algorithm Node Relaying Algorithm (NRA): nodes are used as relays between ferry routes.
- Ferry Relaying Algorithm Ferry Relaying Algorithm (FRA): ferries program their routes to find each other and exchange data.

The best-known algorithm is SIRA, where all nodes are equally spaced from each other. In order to compute the ferry route, an adapted solution of the Travelling Salesman Problem TSP is performed with the goal to visit all the nodes. Instead of optimizing the length of the route as in TSP, the delay is optimized. The TSP, was created in the 19^{th} century and corresponds to calculating the shortest and most efficient path for a person to follow, given a list of specific destinations and distance between each location. This route must pass once and only once by each city and end at the city, it started. TSP is one of the most intensively studied problems in optimization. Although giving realistic outcomes and reasonable results, several other things are not taken into account, such as message generation rates between nodes, making the TSP inefficient in real situations.

In 2007, Zhang et. al [8], proposed another two algorithms for multi-ferrying, Single Route (SRT), where all ferries serve the whole network and follow a single route that passes through all nodes and Multiple Route (MRT), where each ferry can serve a different cluster of nodes and follow a different route. Some ferries may serve one or more nodes in common, so the network keeps connected.

Hui Guo et. al, in 2010 [9] Homing Pigeon based DTN (HoP-DTN) was created. Every node has an associated messenger, called pigeon. On each node, the messenger has a buffer. A route to deliver the messages is chosen when the buffer is full. This approach considers all the nodes that the buffer needs to deliver information. A Genetic Algorithm (GA) is applied to TSP between those nodes to find the best path. The pigeon will make a round trip, starting in its home, going to the nodes where it needs to deliver the messages, disseminating the information, and coming back home again. This proposal has issues, like lack of flexibility, since it needs to have a very well-defined number of messengers, one for each cluster. Also, each message needs to wait for the buffer to be full, which causes delays in

the delivering and reaching the destination. When considering deadlines on messages, some loss of information occurs because messages will expire, and will never be delivered.

More recently, in 2017, Barroca et. al [10], proposes DTN-TSP-D, in which UAVs are used to deliver messages with deadlines in a DTN. The UAVs are smart enough to not load messages unable to reach the destination in time. The ground nodes are small clusters in the network created by proximity between nodes, using the *k-means* algorithm, the number of created clusters is identical to the number of UAVs in the network and are assigned to each ground node (created cluster). Inside each cluster, the UAV will deliver messages, in a ferry mode, following a ferry route. This is based on a GA altered TSP algorithm that takes into account the deadlines of the messages. In some situations, the UAV will have the need to leave its home cluster, entering pigeon mode, following HoP-DTN. Instead of using the full capacity of the buffer as in its original algorithm, deadline triggering is used to convert the ferry into a pigeon. The UAV will follow the best route, accordingly to a GA altered TSP. With this, the algorithm takes into account the deadlines of all messages in the buffer to compute the best route to deliver them. This route maximizes the number of timely delivered messages, in the minimum travelling distance. This algorithm produces consistent and better results than the previous algorithms.The simulations were made in a balanced network, where all nodes have the same generation rate, in a non-balanced network it will present non-consistent results.

The usage of Machine Learning techniques, was also a theme explored in 2017 by Smith et. al. In [11], where was proposed an adaptive robotic swarm of Unmanned Aerial Vehicles enabling communications between separated non-swarm devices. The swarm nodes uses ML and hyper-heuristic policy evolution to provide agility within the swarm, enabling each swarm member to select the most appropriate mobility policy given the swarm's abilities. Each agent is aware of its neighbours; each neighbour advertises its location,and if they are currently holding a package or not. Whit this information, the agent is able to plan his movements to others outside their communication range. This algorithm is supposed to deliver packages as efficient as possible. To select the best action that the UAV should take at each time step, Q-learning is used. The reward for the taken action is called packet score and it's the difference between the distance from the target node at time *t-1* and the distance at time *t*. If the packets get closer to the target node in that interval, the reward will have positive values. Drones are allowed by this algorithm, to learn if they should act as a relay or as a ferry.

Later in 2019, Li et. al [12], developed an algorithm of UAVs with Microwave Power Transfer Microwave Power Transfer (MPT), where the UAV can charge the sensing devices remotely and harvest their data. A Markov Decision ProcessMDP is formulated: the states are position of the UAV, battery level and data queue of the ground devices, channel conditions, and waypoints given the trajectory of the UAV; The actions are next positions of the UAV and selection of ground devices. This problem can be solved with Reinforcement Learning approach, but Deep Q-Learning is used as algorithm and it is preferable over

Q-learning, because the second one suffers from the well-known curse of dimensionality, which is not practical for the resource allocation in the UAV-assisted online MPT and data collection due to a large number of states and actions. The on board Deep-Q-Network is developed to minimize the overall data packet loss of the sensing devices, by optimal finding the device to be charged and interrogated for data collection. Deep Reinforcement Scheduling Algorithm Deep Reinforcement Learning Scheduling Algorithm (DRL-SA), derives from the optimal solution online by taking current network state and action as input and delivering a corresponding action-value function. DRL-SA learns an optimal resource allocation strategy asymptotically through online training at the on board deep-Q-Network. The algorithm uses a ε-greedy policy to balance the cost minimization of the network with respect to the already known knowledge by training new actions to get unknown information. Also, it carries out experience replay, to reduce expansion of the state space in which the algorithm's scheduling experiences at each time step are stored in a data set. The reward is given by cost function and is inversely proportional to the number of lost packets from the last state to the current one. This algorithm does not consider direct paths between two nodes, instead tries to optimize the path of the UAV in order to being close enough from the nodes to collect the data and to occur an efficient MPT. This approach reduces packet loss by at least 69.2 % compared to non learning ε-greedy algorithms. These studies are different from our work: they assume several nodes can transmit simultaneously to the UAV and distances are smaller between UAV and nodes. In this case, only one node can transmit at a certain time step to the UAV and distances might be bigger.

Saraiva M.C. in 2019, [4] in his thesis, Optimization of Message Ferrying UAV Paths using Monte Carlo Tree Search and Reinforcement Learning, tries to find the optimal path that the UAV should take in an unbalanced DTN with a reinforcement learning approach. Firstly it is produced an implementation based in [5], that implements TD(0), adapting the reward system and the used policy. The implementation will be done in a Hub Node Scenario and All-to-All scenario:

- **Hub Node Scenario** - Hub node that receives messages from all other nodes, with the help of the UAV.
- **All-to-All Scenario** - Every node can send and receive messages from every other node, with the help of the UAV.

All nodes have empty buffers and a set of sequential actions are performed. At the end of the cycle, the buffers are empty and will start from the initial state, the state-value function is already filled. The algorithm will repeat states, instead of going further and not having exploit a state enough times. The used algorithm is TD(0) and the action is chosen by the policy. TD(0) needs to be able to calculate all the possible states, because it uses state-action value table, it is necessary to know the next state and the associated value to calculate the reward. After the action is chosen, the environment will simulate the

next state, the value function is updated and the algorithm goes to the next iteration starting in the next state returned by the environment. When all episodes are finished, the value function will be filled with the values that indicate the best states that are better to be at. After this, the best sequence of actions will be extracted, starting in the initial state and following those actions. In the All-to-All scenario, the state space will be larger, since every node can send to every other node, and the state values will not be able to be kept in a table. A linear function approximation is used. The author also used a Monte Carlo Tree Search (MCTS) approach. Each state of the formulated problem is represented by the number of messages in the UAV, messages in each node, the number of generated messages, and the number of expired messages. In order to create a more complete solution to the problem, the deadline of the messages was also implemented. Messages have a time limit before they have to be delivered. The reward calculation was modified, in order to include the Deliver Ratio (DR). This new concept measures the ratio between the messages actually delivered to their recipient and generated messages. In the simulation, there are *N* nodes and N-1 possible actions, each node can generate messages at a different rate for each of the other nodes. The UAV, will have *N* buffers, with the messages it needs to deliver to each node. In the simulations, four algorithms were tested:

- RL - Total DR that ensures a total DR network;
- RL - Lowest DR, where DR was calculated separately for each node, the lowest one was chosen and used to calculate the reward;
- MCTS search of the best path was done constructing a tree;
- TSP path;
- Round Robin;

For the Hub Node scenario, it was used 0.3 for the ϵ (exploration rate), $alpha$ (learning rate), and 0.01 for β (discount in the reward) as initial values. Without deadlines, the RL-based algorithm had the best average delay, followed by the MCTS in comparison to TSP or RR. With the increase of the number of nodes, large networks will require more time and memory to compute correctly the paths that the UAV should follow. The relative gain, known in [5] was used to compare the algorithms. The relative gain, matches the results of the average delay, considering RL and MCTS the best approach, the gains are 30$\%$ compared to the RR and TSP. In the same scenario, but using message deadlines, the RL approach does not match its expectations from the previous simulations without deadlines, but the approach with the reward that considers DR and Lowest DR, shows better results. Only the Lowest DR was able to visit all the network nodes. In conclusion in the Hub Node Scenario with deadlines, the algorithm needs to guarantee it will visit all the nodes in the network, in order to not prejudice its DR. The reward not only depends on the state but also on the DR, which is an issue. Finally, with the

All-to-All scenario, only RL-Lowest DR will be tested, compared to TSP. For the parameters was used 0.3 for the ϵ (exploration rate), 10^{-9} for α (learning rate) and 0.01 for β (discount in the reward) as initial values. As expected, using RL-Lower DR is slightly better than using TSP, according to the average delay. Introducing deadlines also with this scenario, the author notices that execution time increases linearly with the number of nodes, because the action space grows too. RL-Lower DR showed similar results as the first scenario. The author, later concluded that the results between both scenarios are similar, proving that the All-to-All scenario is a generalization of the Hub Node Scenario. In the end, it was proposed for future work, using Deep Neural Networks, or an approximation of ANN instead of a linear function approach.

2.7 Summary

In the State of the Art of this thesis, a brief introduction of the UAV was performed, such has their history, characteristics and usages. The use of UAVs in DTNs can be beneficial and this topic was commented. The theory of Markov Decisions Processes was explained. A common field of Machine Learning is presented, Reinforcement Learning and two examples of algorithms of ML are explored, TD-Learning (TD(0)) and Q-Learning. The usage of acMDPs and RL in DTNs is illustrated through the interpretation of an article published by Henkel et. al. Since the main focus of this work is to implement DRL algorithms to be used in DTNs, a theoretical explanation of DRL is performed and Deep Q-Learning algorithm is explored. In the last section of this chapter, several related works are explained and comprised, in order to find similarities and motivations to our work.

3

Proposed Algorithms

Contents

The recreation of [4], only for the All-to-All scenario, using TD(0) with linear approximation, was explained in the previous chapter. The implementation of this algorithm is detailed in this chapter. The second approach is the DRL algorithm that will be further explained and the decisions made to optimize are highlighted.

After this brief presentation of the two approaches, it is proceeded the explanation of the algorithms performed to train the agent in the environment.

3.1 Reinforcement Learning

A similar implementation of the proposed algorithm and adaptation of RL in [4], is performed in order to compare the results obtained in that work, with the one developed in this thesis with DRL approach. The development and comprehension of the pratical side of this work had the help of [13] and [14].

3.1.1 Implementation of RL

As explained before, a solution equal to the one presented in [4], was implemented. This consists in the implementation of TD(0) algorithm, Section 2.3.1.

The algorithm TD(0) is mainly known as one-step TD. TD(0) is the simplest form of Temporal Difference Learning. At the current state, it must calculate the next state for each possible action and update the value function with the value of the next state, as well as the reward. In practical terms, in order to evaluate each possible next state, the drone will visit each state multiple times. The reward will assign better values to states that have a smaller number for the sum of the messages in all nodes, and will also value smaller routes taken by the drone.

The reward will be equal to the one Henkel et.al [5], used (see Equation 2.7). It is calculated using an integral done over ta, that is the duration of the action N_0, is the sum of the messages from all nodes in the beginning of the action. The negative integral will give better state-values to states which have less messages in all nodes.

The policy will be used to choose the action in a certain state, Equation 2.8. The value function is Equation 2.9 and updates the state-value each time the state is visited. The algorithm proposed by [4] is presented with the following pseudo-code, Algorithm 4. To extract the final result, it is implemented in the pseudo-code of Algorithm 5 . The state and action space will be initialize for N episodes, that will be the number of times the code will be running in order to update the value function. At this stage, every node has its buffer empty (vector with the messages to send for each node). Each episode will run for i iterations. When the cycle is finished, will proceed for the next episode, the buffers will be emptied, and the algorithm will start in the initial state but the value function is already filled with the calculated state-values. The author of [4] concluded that this process of repeating states would be useful, instead

of going further without having exploited that state enough times. This will also help if the algorithm gets stuck with a set of odd states, giving it the opportunity to start from the beginning again. As presented in Section 2.2.1, to get the balance between exploiting and exploring, the epsilon greedy algorithm will be used. The exploration rate ε is set to 1 and decaying in each episode, this will be the probability that the agent will exploit. A random number is chosen between 0 and 1. If this random number is smaller than the exploration rate, the agent will explore. If the opposite occurs - random number bigger than ε, the agent will exploit.

In each i iteration, an action will be chosen:

- randomly, with probability of ε; (Exploration)
- according to the policy (Equation 2.8), with probability of $1 - \varepsilon$; (Exploitation)

Algorithm 4 RL-implementation

for N episodes **do**
  Initialize State;
  Initialize Action Space;
  if current episode % (N/5) == 0 **then**
    /* This means every 20% of the episodes
    $\alpha = \alpha * 0.5$;
    $\epsilon = \epsilon * 0.5$;
  end if
  for i iterations **do**
    Choose a random number $0 \leq rnd \leq 1$;
    if $rnd < \epsilon$ **then**
      Select an action randomly;
    else
      Select the best action according to the Policy;
    end if
    $new\_state \leftarrow$ Generate state transition according to the action selected and current state ;
    Update state-value function accordingly;
    $State \leftarrow new\_state$
  end for
end for

The policy will choose the action with the best state value. TD(0) needs to be able to calculate all the possibly next states when choosing that action, a state-value table is used in this algorithm. To calculate the policy, it is necessary to know the next state and its associated value. The parameters, α and ϵ are decayed 20% in every iteration. This is an important aspect, since the author of the previous work, tried several approaches, but concluded this one was the best option, so much for α and ϵ parameters. He opted for the one in Equation 3.1, because in the first iterations the decay was slower but would later stabilize to a lower value in the last iterations.

$$\alpha = \alpha * rate \tag{3.1}$$

When the action is chosen it will simulate in the environment, this way it is possible to know every possible next state. The value state is updated and the next iteration begins in the new state returned by the environment.

Algorithm 5 RL extracting final result

Initialize State; Initialize Action Space;
  for i iterations **do**
    Choose best action according to the Policy;
    $new\_state \leftarrow transition(old\_state, action)$;
    $results.append(action)$
    $old\_state \leftarrow new\_state$
  end for

In algorithm 5, after iterating through the N episodes, the state-value table is already filled with values that will indicate which states are better to be at. It is now easy to extract the best sequence of actions, starting in the initial state and choosing iteratively according to policy, the best 20 actions to choose. The author of [4], chose 20 actions in the learning phase (algorithm 4). A sequence can be extracted by searching for a repetitive pattern in the results of taken actions.

Since, the first scheme of [4] is done for a different scenario than the one focused in this thesis, first it does a Hub-to-Node scenario, where each node can only send messages to the hub, the state space is smaller than the All-to-All scenario, where every node can receive and send to/from each other node, an adaptation to the value function is done. A linear approximation is used. In this algorithm TD(0) is used a linear approximation that can be performed using a vector of weight. This vector will help to estimate the value of each state. In this case, instead of trying to learn a value for $V(s)$, the value learned is given by $V(s, w)$, where w is a vector of weights.

$$V(s, w) = \sum_{i=0}^{n\_features} w_i * feature_i \tag{3.2}$$

The value function $V(s)$, instead of iterating until reaching optimal values $V^*(s)$, will iterate with the goal of finding the optimal weight vector. The optimal weight vector is the one that best represents the value of each state according to the features that represent it and are updated after each step:

$$w_i = w_i + \alpha * (r(s, a) + \gamma * V(s', w) - V(s, w)) * feature_i \tag{3.3}$$

3.2 Deep Reinforcement Learning

In this work in order to explore and continue the problem already formulated in the previous related work, explained in Section 2.6, a Deep Q-Learning algorithm will be developed and simulated in a

scenario where all nodes in the network will send and receive data to and from each other node, with the help of an UAV. This work aims to find an approach that will possible achieve more efficient results than the previous (RL implementation) and studied related work. The use of ANN instead of a linear approximation was proposed by the author of [4] and this will be our aim, to introduce ANN and RL combined, which are commonly known asDRL.

3.2.1 Deep Reinforcement Learning Implementation

In order to use a Deep Reinforcement approach, the chosen algorithm is Deep Q-Learning with Experience Replay. The theory for this part was explained in Section 2.4.1 a pseudo-code for this algorithm is Algorithm 3.

The unique change in this algorithm corresponds to the introduction of ANN and a Replay Memory (where the agent's experiences at each time step are stored). The calculation of the reward (Equation 2.7), the policy (Equation 2.8), and value function (Equation 2.9) used in the RL implementation will be maintained. Another change is instead of using the Linear Approximation function (Equation 3.2) and the vector of weights, this will not be necessary, because ANN (which is a non linear function) allows for a more complex environment without having the need to do a linear approximation.

In this algorithm it is necessary a training network and a target network (these two networks are Artificial Neural Networks). The second network is not trained and it is a way to see if the model was well trained, by using the outputs of the training network in a not trained target network. In each episode, to train this environment and learn the optimal path to deliver the messages, first it needs to fill the experience replay. Using the states as an input, the agent (UAV) will start to choose actions by exploring the environment (network) and getting rewards according to the chosen action. This reward will be calculated by Equation 2.7 as mentioned above. The tuple $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ that is the experience replay is filled, with this process.

After this, the agent will exploit, in the current state is select an action with probability $1-\varepsilon$ or an action predicted by the training model. The training model predicts Q-Values for each possible action from that state, the calculation of the Q-Values is computed with the value function from the RL implementation. The agent will choose the best action according to policy Equation 2.8. In the RL algorithm, the reward granted better values to the states that had a smaller number for the sum of all the messages in the nodes and also to smaller routes taken by the UAV. This pattern maintains is this DRL approach. In this phase what was stored in the experience replay is not being trained, it is only being used to compute Q-Values.

To train the network, a random batch of the experience replay is sampled in order to predict Q-Values. Each experience (tuple with state, action, reward and next state) represents the actual state s_t, the reward r_{t+1} the agent got by choosing the action a_t, and next state s_{t+1}. The Q-Values predicted

are used to calculate the Loss. The chosen states used to predict the Q-Values are the input for the target network, that is not trained. With this, the targeted network will predict the best rewarded action to take from each state. This would be the expected action that the trained network should have chosen (when was trained for the first time, with the data in the experience replay).

It is possible to calculate the Loss, with the Bellman Equation:

$$Q(S, A) = R + \gamma \max_i [Q(S', a_i] \tag{3.4}$$

Where $Q(S', a)$ is the $Q - value$ for the state where the action A' was chosen by the the target network model, and $Q(S, a)$ is the $Q - value$ for the state where action A was chosen in the training network. The expected $Q - value$ is given by the expected reward (R) and the discount factor ($gamma$), it is an hyperparameter that can be regulated in order to give more or less weight to the target $Q - value$. After this, the SGD algorithm is performed to minimize the loss – each iteration and episode trying to approximate the $Q - value$ ($Q(S', A)$) to the expected $Q - value$ ($Q(S, A)$).

To create the Artificial Neural Network that will receive the states as an input, were created layers – creates a kernel with the input layer (states) to produce an output. This will be followed by one fully-connected hidden layer and one fully-connected linear output that computes the Q-Values for each action from the input state, the structure chosen for the ANN is a simple one in order to be less complex in the formulated problem.

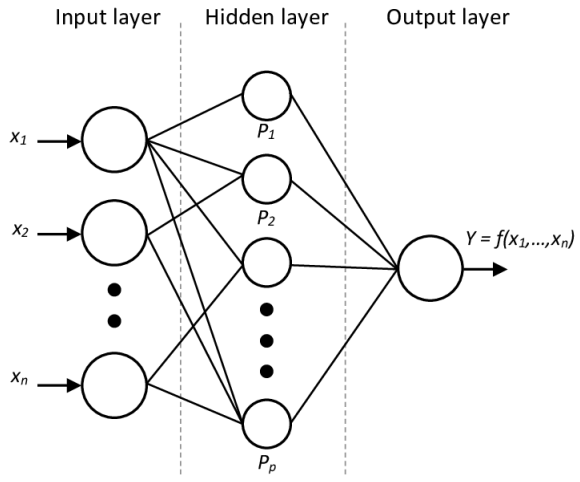


Figure 3.1: Representation of a Neural Network with the input layer, one fully-connected hidden layer and one-fully connected output layer, image is from [3].

In this algorithm a set of hyperparameteres are configured in order to maximize the training of the model and adapt the model to the data being trained. Such as in the RL implementation, the learning rate, the ε for the greedy algorithm, the batch size and the discount factor need to be defined. The learning rate, speeds the process of learning in the beginning and will help the model to converge. The

variable ε starts in a value close to 1 and will decay its value in order for the environment to explore in the beginning of the simulation and to exploit during the rest of the iterations. The batch size of the stored experiences can't be too small or too big, to few samples won't allow the model to see lots of data and train it.The contrary where too much date is seen, can have a set of wrong data. For last, the discount factor ($gamma$) used to calculate the policy that will choose the best action according to the $Q - Value$ of such action is used to define the weight given to the calculated reward in comparison to the computed Q-Values. The values chosen for this hyperparameteres are described in Chapter4.1.

For this implementation, using Python and Open AI Gym, were consulted examples of repositories on GitHub, such as [15].

4

Evaluation and Comparison

Contents

In this chapter, the results and performance obtained will be evaluated and compared with the results of [4]. The two algorithms (RL implementation and DRL implementation) will be compared with each other.

4.1 Simulation of the Presented Algorithms

For this work, the environment consists in N nodes that represent the nodes that can generate and receive messages and one UAV. In practice the UAV, is considered as a node that will start in a random position of the network. Figure 4.1 is an example of how the UAV travels in the environment.

This agent has N actions to choose in the first iteration in which the UAV is in a random position. It will after travel to the first node the algorithm chooses. After this, there are only $N-1$ available actions, because in the next step it can only travel to the rest of the nodes except the initial position of the UAV. For this algorithm, the key factor corresponds to the number of messages each node has to send to each other node, and the different generation rates of messages, we used buffers to store this information:

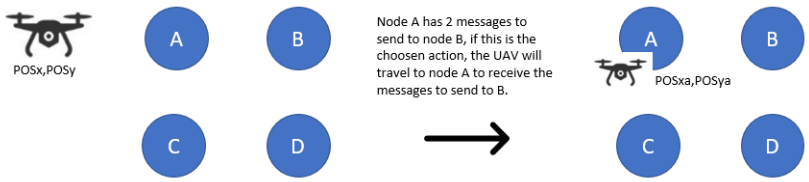


Figure 4.1: Explaining how the UAV helps in the receiving and sending process between all nodes

- N buffers to store the messages that each node has to send to each other node;
- one buffer of the UAV, that has the sum of messages each node has to receive in that step;
- N buffers with the generation rate of the messages of each node to the other nodes in the network.

As explained before, the state space is larger, compared to the [4] studied in the first place. This happens because in this scenario, where every node can send messages to every other node in the network, there is a need to have into account the number of messages in the buffers of each node, and in the UAV. A linear adaptation is done to follow the needs of a larger state space. Instead of finding the optimal value of the value function, the algorithm will iterate in order to find the optimal weight vector. In this work, the algorithm uses the same approach as [4]. There is a vector called $feature$, which contains the number of messages of each node to another, followed by the number of messages in the UAV buffer and one-hot encoding of the node the UAV is at. Per example, for one environment with $N = 4$, the feature vector is:

$$feature : [0, 2, 7, 2, 2, 0, 2, 2, 3, 1, 0, 8, 2, 2, 2, 0, 7, 5, 11, 12, 100, 0, 0, 0]. \quad (4.1)$$

Explaining this vector 4.1:

- the first 4 elements are the number of messages that node A has to send to node A, B, C, D (0 for itself, 2 to B, 7 to C and 2 to D);
- the next four elements are the messages that B as to send to each other node (2 for A and 0 for itself, 2 for C and D);
- next four are the ones C needs to send (3 to A and 0 for itself and 1 to B and 8 to D);
- next four are the number of messages D has to deliver (2 for A and B and C and 0 for itself);
- the other four elements are the sum of the messages that the UAV needs to deliver to each node in the environment (there is in total 7 messages for A, 5 for B, 11 for C and 12 for D);
- the last four elements is the one-hot encoding, 100 for the node the UAV is at and 0 for the others.

One-hot encoding is a process to convert categorical data variables into binary values, they are provided to the algorithm to help in the predictions. Instead of using 0 and 100 to convert the categorical data, it was tried to change this parameter to 1 and 0, and then to 10 and 0, but it was encountered the same issue [4] referred in his work. The features were being updated too slowly, so in order for the features to be all in same unit scale (hundreds) , we concluded that it would be better to keep one-hot encoding with the value of 0 and 100. So for this example with $N = 4$ nodes, the one-hot encoding corresponds:

Table 4.1: One-hot encoding representation for a network with 4 nodes

	Node A	Node B	Node C	Node D
Node A	100	0	0	0
Node B	0	100	0	0
Node C	0	0	100	0
Node D	0	0	0	100

The feature explained above is only used in the RL implementation, because the ANN utilization is a solution that allows more complexity. In this implementation, there is no need to perform a linear approximation.

To simplify the reward calculation and in order to prioritize routes that are smaller for the UAV to take from each node to another, it is used a vector that stores in its data the time it takes for the UAV to travel to some node. Per example, considering the UAV is in the initial random position in the network, the $routes\_time$ vector is:

$$routestime : [0.2, 0.1118034, 0.25, 0.15811388] \quad (4.2)$$

for node A, the UAV from the initial position will take 0.2 [t.u.], for node B 0.1118[t.u.], for node C 0,25[t.u.] and for node D 0.1581[t.u.] this time is calculated by:

$$distance = \sqrt[2]{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.3}$$

$$t = distance/v \tag{4.4}$$

$$v = 20 \tag{4.5}$$

where (x_1,y_1) are the positions of the target node and (x_2,y_2) are the positions where the UAV is.

To test this algorithms, the number of episodes, the max steps per episode and the hyperparameters, such has: learning rate (α); exploration rate (ε); decay rate for the reward (β); discount factor (γ); velocity for the UAV, v, need to be defined.

Table 4.2: Parameters used in the RL simulation

Parameters	Initial Value
Number of Episodes	10000
Maximum Steps	200
ε - Exploration Rate	0.3
α - Learning Rate	0.3
β - Decay Rate	0.01
γ - Discount Factor	$e^{-\beta t}$
v - Velocity	20

Table 4.3: Parameters used in the DRL simulation

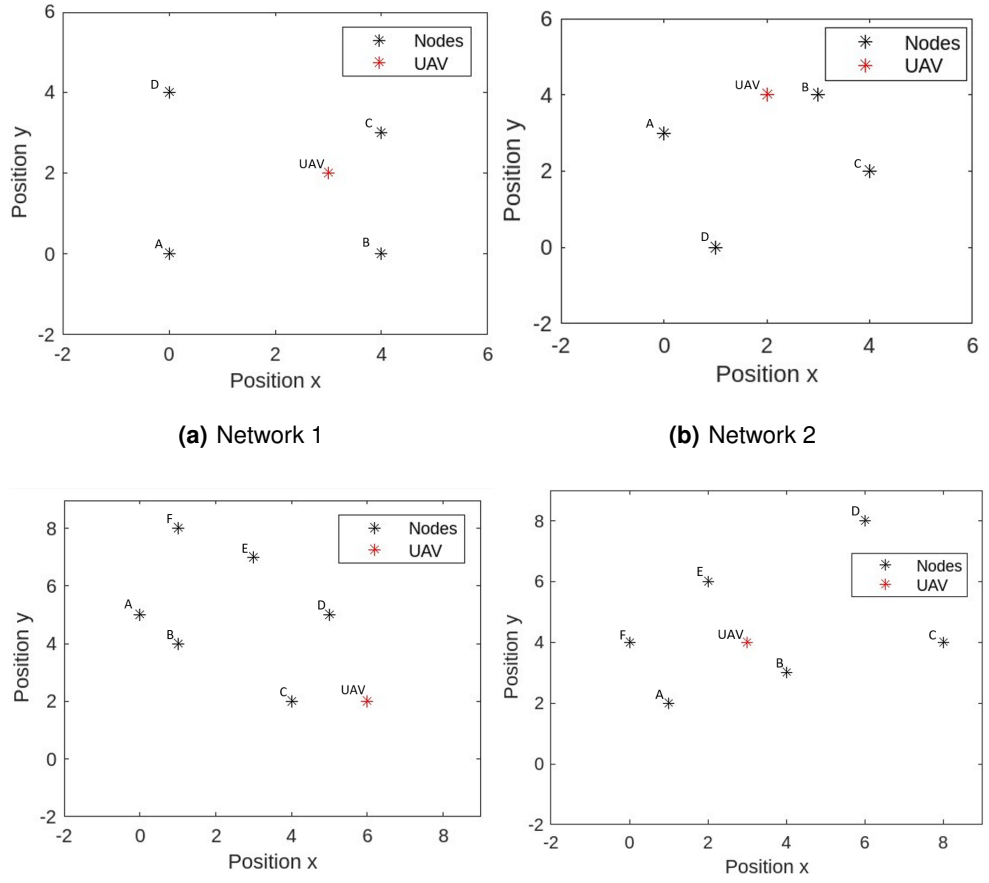
Parameters	Initial Value
Number of Episodes	10000
Maximum Steps	50000
Batch Size	200
ε - Exploration Rate	0.3
α - Learning Rate	0.01
β - Decay Rate	0.01
γ - Discount Factor	$e^{-\beta t}$
v - Velocity	20

For both algorithms, the CPU and Memory resources used where : $3GHz$ and $< 300MB$. The simulations had long execution times, mostly because of our computer's CPU. Even though, this algorithms were developed in Open AI Gym, that is a possibility that using Python language made them more slow.

In the DRL implementation, a very simple model was created using the $Keras$ package of Python. This allows the creation of a $Sequential$ model that will take as an input an array with the size of the batch-size and will output an array with the size of all the $Q - values$ for each of the possible actions from the inputed states. To learn more about the use of $Keras$ and how to understand the code behind the creation of ANN we consulted [16] documentation.

In this work, it was simulated four different networks. For two of them, with 4 nodes and the other

two with 6 nodes, each of these networks has an UAV that initiates the simulation in a random position in the network. Figure 4.2 represents the simulated networks. The first two networks (Fig. 4.2 a) and b)) are small networks with 5x5 dimensions. The third and fourth networks (Fig. 4.2 c) and d)) are biggest networks with 9x9 dimensions. Even thought, the position of the UAV is chosen randomly in the beginning of the simulation, it was decided to include the UAV position in the networks representation. This will facilitate the visualization of the path that the UAV takes when explaining the results in Section 4.3.1.



(a) Network 1 **(b)** Network 2

(c) Network 3 **(d)** Network 4

Figure 4.2: Representation of the simulated networks for both implementations(RL and DRL)

The messages generation in the network, from each node to another is presented next. This values were the same used by [4] in order facilitate the results comparison.

Table 4.4: Generation rates for Network 1, where A and C have the highest generation rates

	Node A	Node B	Node C	Node D
Node A	0	2	7	2
Node B	2	0	2	2
Node C	3	1	0	8
Node D	2	2	2	0

Table 4.5: Generation rates for Network 2, where A and B have the highest generation rates

	Node A	Node B	Node C	Node D
Node A	0	9	9	9
Node B	9	0	9	9
Node C	2	2	0	2
Node D	2	2	2	0

Table 4.6: Generation rates for Network 3, where B, C, D, E and F have the highest generation rates for each other

	Node A	Node B	Node C	Node D	Node E	Node F
Node A	0	2	2	2	2	2
Node B	2	0	9	9	9	9
Node C	2	9	0	9	9	9
Node D	2	9	9	0	9	9
Node E	2	9	9	9	0	9
Node F	2	9	9	0	9	0

Table 4.7: Generation rates for Network 4, where A and B have the highest generation rates

	Node A	Node B	Node C	Node D	Node E	Node F
Node A	0	9	9	2	2	2
Node B	2	0	9	2	2	2
Node C	2	9	0	2	2	2
Node D	2	9	9	0	2	2
Node E	2	9	9	2	0	2
Node F	2	9	9	2	2	0

In order to be a fair and reasonable comparison, the same four networks (Figure 4.2) for RL and DRL approaches were used, such as the same generations rates for both algorithms.

4.2 Performance Metrics

For this section, graphics and tables are presented, followed by comments about the obtained results.

To evaluate the performance of the RL algorithm, Average Delay (AD) will be used as a metric. The AD consists in the time it took for the UAV to deliver the messages to the destination node since the moment they were generated by the sending node. The message needs to be successfully delivered. There were always encountered cases in the simulations where the UAV buffer (vector used in the simulations to store the number of messages for each node) still had data, which means that not every generated message was delivered in the end of the simulation. These messages are not considered for the Average Delay.

In order to compare with the algorithm that motivated this work (RL implementation done in [4]), the Relative Gain (RG) is used. This performance metric is computed using the difference of the AD results in [4] and the AD results of this work.

$$RG = \frac{baseline - target}{baseline} * 100 \tag{4.6}$$

where $baseline$ is the AD obtained in [4] and $target$ is the AD computed in this work.

4.3 Results of RL and DRL implementations

In this section the results of each implementation are presented separately. It is easier to comprehend the differences between each algorithm and conclude which one had the best performance. This is measured in terms of optimal path computation for the UAV to take in order to visit every node in the network and delivering the messages. A comparison to evaluate both of the implementations of this work is computed in terms of average delay, execution time and memory usage.

4.3.1 Results of the RL implementation

For the Reinforcement Learning Algorithm, implemented with TD(0) and linear approximation to the optimal weight value computation (which is the one that best represents the value of each state), were simulated various combinations of networks, hyperparameters and generation rates of messages, in order to obtain the optimal path and with better average delay.

In Figure 4.2 the networks are represent. It can be seen that networks 1 and 2 are smaller and have less nodes. Networks 3 and 4 are bigger and with more nodes.

In the course of the simulations, it was observed that the increase of the network size increases the execution time of the algorithm and the time to send a message from a node to another increases as well. In Tables 4.4, 4.5, 4.6 and 4.7, the generation rate of messages in each network is presented. The parameters used to simulate the algorithm are described in table 4.2. One of the main objectives of this work is finding a path for the UAV to take, in order to deliver all the messages in it's buffer. For the path identification, we analyzed the actions that the agent took in the environment, and when a set of the nodes being visited was repeated several times in the running of the simulation, we considered that this was the path taken by the agent (UAV). In the following tables is present the average delay, the execution time, memory used and the path taken, for each network:

Table 4.8: Results for Network 1

Network 1	Average Delay[t.u.]	Average Memory Usage[MB]	Execution time[s]	Path taken
RL algorithm	86,8	277	25200	C,B,A,B,C,D

The UAV starts in a random position (3,2) in the network and starts it's journey by visiting node C. This node is the closest to the position of the UAV and in the beginning of the simulation. The UAV buffer already has 7 messages for node A and 2 for B and D. As it was studied the reward values smaller routes taken by the drone, as well as a smaller number for the sum of messages to deliver in that node. In the next step UAV chooses B , because the sum of messages to deliver is smaller then D. Even though the distance between C and D and the distance between C and B is similar. The execution time of the algorithm in this network takes longer compared to the one developed in [4]. This scenario is observed for each network simulation.

Table 4.9: Results for Network 2

Network 2	Average Delay[t.u.]	Average Memory Usage[MB]	Execution time[s]	Path taken
RL algorithm	75,6	289	23400	B,C,B,A,D,C

In this network, the UAV starts in position (2,4) and chooses node B as the next step to take, because similarly to network 1, it is the closest node, followed by node C also because of proximity. Throughout the simulation its is noticed that the chosen actions are being chosen, because in the reward calculation it is given more importance to smaller routes over the number of messages to be sent. This network has a similar behaviour to network 1. The main difference corresponds to the distance between the nodes (smaller), making the average delay compared to network 1 4.8 smaller as well.

Table 4.10: Results for Network 3

Network 3	Average Delay[t.u.]	Average Memory Usage[MB]	Execution time[s]	Path taken
RL algorithm	109,7	328	40520	C,D,E,F,E,D,C,B,A,F

The 3rd network is different than the two previous. Instead of having 4 nodes, it has 6 nodes and the dimensions of the network are larger. So, the execution time will be more elevated as it is confirmed, is almost the double of the execution time of the previous networks (25200 s and 23400 s). The generation rate of messages for this network is presented in table 4.6. Node A is a low generation node and the rest of the nodes B, C, D , E and F are the highest generation nodes. The sum of messages in each node's buffer is 38 (B, C, D , E and F) and for node A is 10 in the beginning of the first iteration. Analyzing the path taken by the UAV, it will randomly start in position (6,2) and choose the closest node that is C. When it first reached node F to deliver messages it went back to node E, it prioritized the smaller path.

Table 4.11: Results for Network 4

Network 4	Average Delay[t.u.]	Average Memory Usage[MB]	Execution time[s]	Path taken
RL algorithm	203,4	376	493900	B,A,F,E,B,C,B,D,E,F,A,B,C,D

For the last network, the average delay in this network is higher compared to the other 3 networks, mainly because in this network the arrangement of the nodes is more dispersed. Node B and node C are high generation nodes. These nodes generate 9 messages per temporal instant for the other nodes

in the networks. In the beginning of the simulation, the buffer for the number of messages of B and C has 45 messages to be delivered to the receiving nodes. Whereas nodes A, D, E and F are lower generation nodes only with 10 messages to be delivered in their buffer. Firstly, it starts in B, that is the closest node from the UAV, next one being visited is A because of the closeness to B and the number of messages in the buffer being smaller then the one in C. After following the path B,F,E, it will come back to B instead of going to D, again prioritizing the smaller route. After this, it will traverse the entire network one more time and finishes the path at node D. This situation didn't happened in any of the other networks, it can probably be because of the generation rates being equal for A, D, E and F (2 for each node) and equal for B and D (9 for each node).

It is possible to denote that when the dimensions of the network are larger and with the increase of the number of nodes, the average delay increases, as well as the execution time. The path taken in almost every case prioritizes the smaller route instead of the number of messages to deliver in each nodes buffer.

Following this, it was considered for comparison the results obtained in [4]. In table 4.12 the results are shown:

Table 4.12: Results for RL implementation in [4]

Network	Average Delay[t.u.]	Average Memory Usage[MB]	Execution time[s]
Network 1	79,9	< 200	822
Network 2	67,1	< 200	1451
Network 3	84,0	< 200	1445
Network 4	127,6	< 200	2012

Analyzing the results presented in Table 4.12 with the ones in Tables 4.8, 4.9, 4.10 and 4.11, it is clearly possible to understand that the algorithm performed in this work had a larger execution time, it used more memory and the average delay is bigger, which means that this algorithm was slower and more memory consumer. Besides this, it managed to find paths to deliver the messages without leaving any node of the network unvisited. The average delay - measures the time it took for a message to be delivered since it was generated - showed that this algorithm takes a longer time to deliver a message to its destination node. This can be due to the fact that the network nodes are organized in a more dispersed way than the ones used in [4] or because the UAV starts in a random position and is not considered a regular node. The first action in the environment is to choose where the path should start.

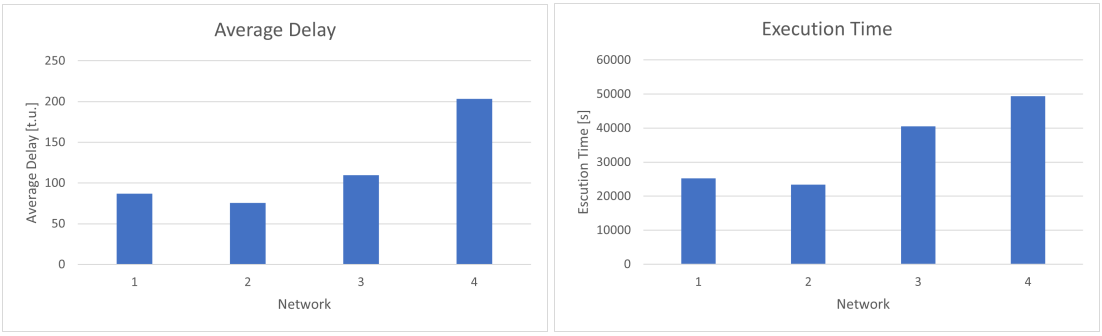
To do the comparison between the developed algorithm with the one studied in [4], the Relative Gain is calculated with Equation 4.6 but instead of subtracting the AD of the results obtained in [4] by the AD of the results of this work the opposite is done. Saraiva et. al, [4] obtained better results for the average delay, it is clear it will have a higher relative gain compared with the implementation performed in this work. In Table 4.13 the RG is presented.

It is seen in Figure 4.3 the comparison of the networks in terms of average delay and execution

Table 4.13: Relative Gain of [4] with respect to our work, considering the average delay as metric

	Network 1	Network 2	Network 3	Network 4
Relative Gain [%]	7,95	11,24	23,42	37,27

time of the algorithms. As mentioned above in the first raw commentary, it was already concluded that with larger and more dispersed networks the average delay tends to grow as well, because the UAV needs to travel longer distances between nodes, the messages will wait more time to be delivered on the opposite of a network where the nodes are more closer to each other. The execution time increases with the number of nodes, and the dimensions of the network, being similar for both networks with $N = 4$ and similar for both networks with $N = 6$.



(a) Average delay for each network

(b) Execution time for each network

Figure 4.3: Comparison of the average delay for each network as well as the comparison of the execution time of the simulation for each network

In order to evaluate better the performance of the algorithm, some simulations were performed using networks with $N = 8$, $N = 9$ and $N = 10$ and with bigger dimensions. The results for networks with this many nodes make the algorithm perform worse than for the simulations with less nodes. This is expected because the average delay for $N = 4$ is smaller than the average delay of a network with $N = 6$.

One important aspect of this simulation is to take a look and notice how many times the agent chose exploitation or exploration. In 10000 iterations of the algorithm, the agent chose 1112 times to go through exploration and 8888 times to go through exploitation. It is an expected observation because this choice is based in the $\varepsilon - greedy$ algorithm, where ε is the probability of the agent to explore the environment and $1 - \varepsilon$ the probability of exploiting. The act of exploring consists in visiting states that the agent does not have information about and exploiting is choosing states that were already visited. During the simulation the value of ε will decay, so the probability that the agent will explore becomes smaller. This can be proved in the graphic presented in figure 4.4 where exploitation was chosen a lot more times then exploring the environment.

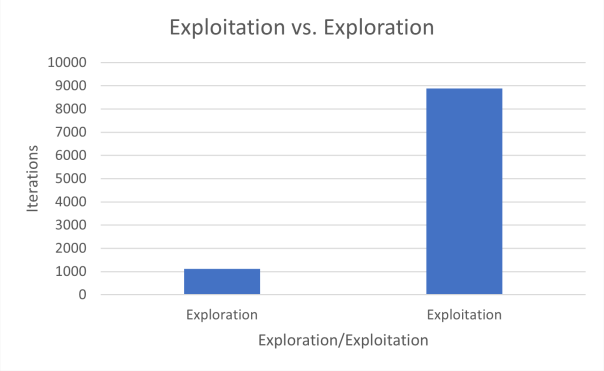


Figure 4.4: Difference between the number of iterations the agent chose exploitation instead of exploration based on the $\varepsilon - greedy$ algorithm

4.3.2 Results of the DRL implementation

For the DRL implementation with experience replay the same networks were maintained (Figure 4.2) and the same generation rates (Tables 4.4, 4.5, 4.6 and 4.7). This was decided in order to be useful for the comparison of results between the two implementations in this work.

The hyperparameters for this algorithm are defined in Table 4.3. The values were chosen after some simulations to understand what would work best in the environment and in the training of the environment.

Both of the networks, training network and target network, are Neural Networks that are trained to find the optimal $Q - Value$ for some possible action from a state that is the input on the networks. The ANN allows for more complex scenarios but also brings more complexity to solve this problem of the optimal path. In this simulation of Deep Reinforcement Learning using experience replay more obstacles were presented with a not so good expected result.

In Figure 4.5 it is possible to observe in the axis-y the value of the rewards and in the axis-x the training episodes. With only a quick observation it is noticed that several times the value of the reward is 0. This happens because the reward is calculated over the integral of 0 to the time of completing an action, Equation 2.7 ($r(s,a) = -\int_0^{ta}(Ft + N_0)e^{-\beta t}\,dt$). Sometimes, this algorithm wrongly considers that the UAV is in a position of the network where is considering that the time of completing an action is infinite ($t = \infty$), $e^{-\beta t}$ will be equal to zero. The policy that is used to choose the action corresponding to the one that minimizes the Loss between the Q-Values computed in the training network and the Q-Values computed in the target network is calculated with Equation 2.8. The calculated policy for values of $t = \infty$ will also be equal to zero or equal to a value that the model will approximate to zero. This value will lead for the algorithm to always be choosing a certain action. Per example, a network with $N = 4$ has $N - 1$ possible actions when the UAV is at a certain node (like the developed simulation consider in this work):

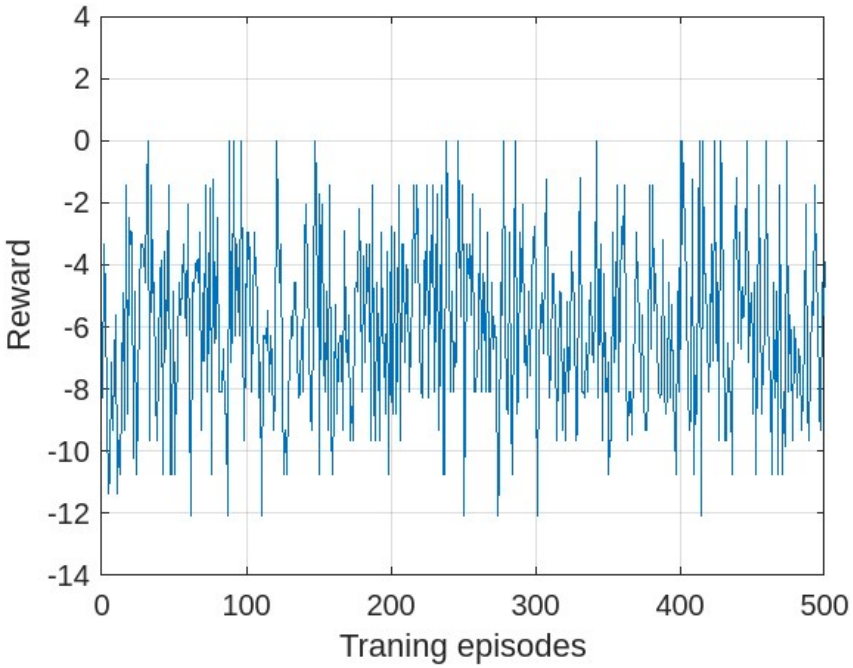


Figure 4.5: Evolution of the reward in a training simulation of the DRL algorithm

- Action = 0 : chooses node A;
- Action = 1: chooses node B;
- Action = 2: chooses node C;
- Action = 3: chooses node D;

The value of the policy will choose the action, so for the case where the policy is equal to zero, it will always chose the Node A, without taking into account if it is the closest one or the buffer of messages of node A has less or more messages then the buffer of the other nodes. This will increase the average delay of the network, leading to a higher value than the ones computed in the RL implementation. Also, the path does not go to every node in the network.

In this evaluation it is not considered the execution time [s] of the algorithm since it grows to much considerable values caused by the problem of the $t = \infty$ presented above and will not be very well representative of the performed simulations.

In table 4.14 it is possible to observe that the average delay is higher than the ones observed in the RL Implementation (Tables 4.8, 4.9, 4.10, 4.8) of this work and the ones in [4].

Table 4.14: Results for DRL implementation for the 4 networks analyzed

Network	Average Delay[t.u.]	Average Memory Usage[MB]
Network 1	306,4	< 456
Network 2	300,2	< 395
Network 3	408,7	< 541
Network 4	432,6	< 575

This happens because of the problem explained above, when the agent takes t $=\infty$ to complete an action. Since it will always choose the same action when this happens, the nodes in the network will keep generating messages for the other nodes. This generated messages will be waiting a long time to be delivered. The average memory used for this simulations is also higher compared to the other solutions because this algorithm is more complex due to the introduction of the neural network. It is necessary to take into account for the training of the proposed model there are two networks with the same sizes and characteristics.

Even though the behaviour of this algorithm was not the expected one the average delay followed the same pattern as in the RL implementation, it increases with the size/dimensions of the network and with the number of nodes.

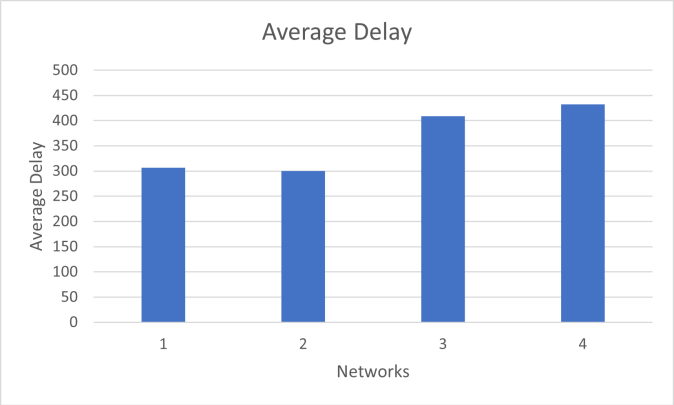


Figure 4.6: Average delay for each network in the Deep Q-Learning with experience replay algorithm

In order to compare this algorithm the RG was calculated. According to Equation 4.6 the baseline is the AD of the RL implementation and the target is the AD of the DRL implementation.

Table 4.15: Relative Gain of RL implementation with respect to DRL implementation, considering the average delay as metric

	Network 1	Network 2	Network 3	Network 4
Relative Gain [%]	71,7	74,8	73,1	52,9

In table 4.15, the Relative Gain of the first algorithm developed in this work - TD(0) with linear approximation in relation to the DRL implementation - Deep Q-Learning with experience replay is computed. It is possible to conclude that the first algorithm had much better results then the later one. This was mainly because of the problems enunciated above.

5

Conclusion

Contents

5.1 Achievements

The proposal for this work was to find the optimal path that the UAV should take in a set of nodes that generate messages to each other. In order for the UAV message deliver and in less duration time, using small routes between the nodes, a certain path is chosen.

Two approaches were developed and performed: using a Reinforcement Learning algorithm and the other one using a Deep Reinforcement Learning algorithm.

The first one consists in TD(0) with a linear approximation and it was a simulation based on one of the related works studied in the State of the Art. It corresponds to the work of Saraiva et. al in [4]. The second studied case in [4], where every node in the network can send and receive messages to every other node (All-to-All Scenario) is the base of the presented work. The linear function approximation approach was necessary because in this scenario the state space is larger. There is a need to take into account and store the number of messages each node has to deliver to the other nodes, the messages the UAV's buffer has stored. This could not be computed by using a table to store the $Q - values$ for each state.

The results for this algorithm were compared with the ones in [4], using four networks, two of them with $N = 4$ nodes and the other two with $N = 6$ nodes. These networks are used to simulate the algorithm. Each node has a different generation rate of messages. The generation rate will influence the computation of the reward that is used to decide which action the UAV should take. After results computing of this simulation, the average delay that is the time elapsed since the message is generated by a node until it is delivered to it's receiving node is observed. In every simulated network of this implementation the average delay was higher compared to the similar networks of [4]. This means that the previous algorithm, on which this algorithm was based, had a relative gain of 7% for the first network, 11% for the second, 23% for the third and 37% for the last network. This implementation, even though being based and using the same hyperparameters of [4] was less well performed. This proves that even though it was recreated the same algorithm other factors can be taken into account, such as CPU of the computer where the simulations were performed , the algorithm was created using a different programming language and using Open AI Gym. Another factor to have in consideration for the difference in the average delay is that in [4], the UAV was considered to be a normal node in the network. In this work it is considered the travelling node that goes to all nodes to receive and deliver messages (in a network with four nodes and a UAV, the UAV allows node A to send messages to B, C and D), and the UAV starts in a random position of the network in every simulation, may be more or less far away from the nodes in each simulation, this can vary the average delay.

Although the results of this algorithm are considered reasonable, when simulating the algorithm with more nodes ($N = 8, 9$ and 10) and with bigger dimensions the performance of the algorithm deteriorates. This can be explained because of the linear approximation used to calculate the values for each state.

Considering it probably is not computing the best approximation and will not outcome with the best value for the state, it will influence the calculation of the policy that is the parameter that chooses an action. It culminates in a not so optimal chosen path.

In Saraiva et. al work, [4], the same conclusions about the approximation function to calculate the value for each state were obtained, so as a future possible work the author of [4] suggested an approach that does not use the linear approximation, instead of this uses Neural Networks that are non-linear functions.

A new form of reinforcement learning is introduced, it's called Deep Reinforcement Learning and it is the combination of Neural Networks and reinforcement learning. Instead of having a $Q - table$ with the $Q - values$ for each state-action pair, a Neural Network is used, where the model is trained (the linear approximation of the RL algorithm was replaced with a Neural network). The model receives as an input a state, and will output the estimated $Q - values$ for each possible action from the given input state. The use of Neural Networks allows for much complexity to be introduced.

For the Deep Q-Learning algorithm with experience replay, the same four networks used in the RL approach were simulated and with the same generation rates. In this part of our work, the results were not as successful as expected. Besides using a non-linear approximation for the calculation of the $Q - values$ that is a a favorable point in relation to the RL implementation, the training of the model in this work, was not executed perfectly. This resulted in several times where the time it took for an action to be completed being a very large value of [t.u.], in this action the value of t was infinite leading our agent to consider that for that chosen action the value of t was infinite, not being able to compute the $Q - value$ in the correct way. This way the policy was not well calculated and the model begins to choose (according to the policy), the same action several times, not visiting every node in the network.

In result of this problem, the average delay of this approach was even worse than the ones obtained in the RL algorithm, when the opposite should have happened. To analyze it, the first approach had 52%-74% relative gain of average delay in relation to the values of average delay observed in the DRL implementation. The average delay is larger because the agent will continuously choose some action to go for the certain node with $t = \infty$ whilst the other nodes are still generating messages that will be waiting more time to be delivered. With this problem, the UAV does not even visit every node in the network, not being able to form a path where all the nodes of the network are visited.

Despite the advantage of using a non-linear approach in this type of formulated problems, where the state space easily grows to a more complex and larger state space, the improvement in relation of the use of a linear approximation function was not obtained. This happened because of a poorly trained model in the DRL approach. There are still not many developed works for this field of studies, in which drones are used for communication between nodes and where Deep Reinforcement Learning is used. Also, the chosen algorithm of this group of learning algorithm is the most basic one. For future work, to

improve this problem, we propose the use of Policy Gradient with Actor-Critic Methods, later explored in Section 5.2.

Nevertheless, in this work it was achieved to simulate the RL Implementation and able to verify the results computed previously and draw the same conclusions. Even though the DRL Implementation did not manage to perform as well as it was expected it was a good starting point in introducing Neural Networks and a way to understand the differences and the similarities between these two different types of learning, Reinforcement and Deep Reinforcement Learning.

5.2 Future Work

This work is a starting point in the introduction of Neural networks in reinforcement learning. As concluded in the previous section, the algorithm performed: (Deep Q-Learning with Experience Replay) did not achieve the expected results.

In order to find a better approach for this problem, the use of Policy Gradient, Actor Critic Algorithms or the combination of both (Policy Gradient - Actor Critic algorithm) can be explored. The Policy Gradient - Actor Critic algorithm is a policy based algorithm, learns the policy function relaying on experience. This algorithm uses two neural networks, one to learn the policy and the other one to learn the value function. In this case, the policy and the value function don't need to be the same as those used in previous work. This functions will be learned with the interaction of the neural networks with the environment.

This type of algorithms can handle larger actions spaces, that grow over the course of the simulations, and converges faster than Deep-Q learning with experience replay.

If the main goal is to correct the Deep Q-learning with experience replay, the used hyperparameters to train the algorithm can be varied and more studied, this way it is possible to analyze why the algorithm considers the duration of an action to be equal to ∞. The used neural networks, training and target networks can have another hidden layer, resulting in more complexity and larger state space.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018. ISBN 9780262039246.

[2] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016, pp. 50–56.

[3] Analytics India. (2022, Nov) Representation of a neural network with one hidden layer. Accessed 10th Nov 2022. [Online]. Available: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks

[4] M. Saraiva, "Optimization of Message Ferrying UAV Paths using Monte CarloTree Search and Reinforcement Learning," Master's thesis, Instituto Superior Técnico, May 2019.

[5] D. Henkel and T. X. Brown, "Towards autonomous data ferry route design through reinforcement learning," in *2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks*. IEEE, 2008, pp. 1–6.

[6] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: Opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 36–42, 2016.

[7] W. Zhao, M. Ammar, and E. Zegura, "Controlling the mobility of multiple data transport ferries in a delay-tolerant network," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 2. IEEE, 2005, pp. 1407–1418.

[8] Z. Zhang and Z. Fei, "Route design for multiple ferries in delay tolerant networks," in *2007 IEEE Wireless Communications and Networking Conference*. IEEE, 2007, pp. 3460–3465.

[9] H. Guo, J. Li, and Y. Qian, "Modeling and evaluation of homing-pigeon based delay tolerant networks with periodic scheduling," in *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–5.

[10] C. Barroca, A. Grilo, and P. R. Pereira, “Improving message delivery in uav-based delay tolerant networks,” in *2018 16th International Conference on Intelligent Transportation Systems Telecommunications (ITST)*. IEEE, 2018, pp. 1–7.

[11] P. Smith, R. Hunjet, A. Aleti, and J. C. Barca, “Adaptive data transfer methods via policy evolution for uav swarms,” in *2017 27th international telecommunication networks and applications conference (ITNAC)*. IEEE, 2017, pp. 1–8.

[12] K. Li, W. Ni, E. Tovar, and A. Jamalipour, “On-board deep q-network for uav-assisted online power transfer and data collection,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12 215–12 226, 2019.

[13] DeepLizard. (2017, May) Machine Learning & Deep Learning Fundamentals. Accessed 08-March-2021. [Online]. Available: https://deeplizard.com/learn/playlist/PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU

[14] DeepLizard, Maddy. (2018, May) Reinforcement Learning - Goal Oriented Intelligence. Accessed 20-March-2021. [Online]. Available: https://deeplizard.com/learn/playlist/PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv

[15] GitHub. (2022, May) Example of an implementation of DQN with experience replay. Accessed 07th Nov 2022. [Online]. Available: https://github.com/tensorneko/keras-rl2/blob/master/rl/agents/dqn.py

[16] Keras. (2021, May) Representation of a neural network with one hidden layer. Accessed 10th May 2021. [Online]. Available: https://keras.io/api/layers/core_layers/dense/