Multi-Agent Reinforcement Learning Methods for Congestion Problems

A Thesis

submitted to the designated
by the General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Christos Spatharis

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WITH SPECIALIZATION
IN TECHNOLOGIES - APPLICATIONS

University of Ioannina

June 2018

Examining Committee:

- **Konstantinos Blekas**, Associate Professor, Department of Computer Science and Engineering, University of Ioannina (Supervisor)
- **Isaac Lagaris**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina

Dedication

Dedicated to my family and friends.

Acknowledgements

Before anything else, I would like to thank my advisor, Prof. Konstantinos Blekas for the guidance, support and valuable advice he provided me during the completion of my master thesis. Furthermore, I would like to acknowledge Prof. George Vouros, Prof. Georgios Chalkiadakis and PhD Student Theocharis Kravaris for their excellent collaboration and help throughout this project. Finally, I would like to acknowledge the DART Project, that supported this thesis.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Glossary

MDP - Markov Decision Processes

MA-MDP - Multi-Agent MDP

RL - Reinforcement Learning

ATM - Air Traffic Management

DCB - Demand-Capacity Balancing

ATC - Air Traffic Control

MARL - Multi-Agent Reinforcement Learning

IRL - Independent Reinforcement Learners

Ed-MARL - Edge-Based Multi-Agent Reinforcement Learners

Ag-MARL - Agent-Based Multi-Agent Reinforcement Learners

Abstract

Christos Spatharis, M.Sc. in Computer Science, Department of Computer Science and Engineering, University of Ioannina, Greece, June 2018.
Multi-Agent Reinforcement Learning Methods for Congestion Problems.
Advisor: Konstantinos Blekas, Assistant Professor.

Multi-agent systems can be used to address problems in a variety of domains, including robotics, telecommunications, congestion avoidance and distributed control. Reinforcement learning framework can provide a robust and natural way for agents to learn how to coordinate their action options in multi-agent systems. The objective of this thesis is to propose and investigate the use of Collaborative Multi-Agent Reinforcement Learning methods for autonomous agents for resolving congestion problems. Such problems require the investigation of a joint policy in order to maximize a pay-off function. Agents have limited information about others payoffs and preferences, and need to coordinate their action to achieve their tasks while adhering to operational constraints. We study three different Multi-Agent Reinforcement learning methodologies: the independent case, the edge-based case and the agent-based case. We have applied these schemes to an interesting traffic application: solving the demand-capacity imbalances during pre-tactical phase in Air Traffic domain. Several experiments have been made based on real-world data and the results obtained confirm the effectiveness of our methods in resolving the demand-capacity problem.

Εκτεταμενη Περιληψη

Χρήστος Σπαθάρης, Μ.Δ.Ε. στην Πληροφορική, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούνιος 2018.
Multi-Agent Reinforcement Learning Methods for Congestion Problems.
Επιβλέπων: Κωνσταντινος Μπλέκας, Αναπληρωτής Καθηγητής.

Η παρούσα εργασία πραγματεύεται τον τρόπο επίλυσης προβλημάτων συμφόρης στο εναέριο δίκτυο με χρήση μεθόδων πολυπρακτορικής ενισχυτικής μάθησης. Πιο συγκεκριμένα, οι πράκτορες του δικτύου μας ανταποκρίνονται σε αεροσκάφη που πραγματοποιούν προκαθορισμένες διαδρομές και σκοπός τους είναι να εκτελέσουν ομαλά τη διαδρομή τους δίχως να δημιουργήσουν πρόβλημα στον εναέριο χώρο.

Ο εναέριος χώρος, χωρίζεται σε εναέρια μπλοκ ή τομείς και κάθε τομέας έχει μια προκαθορισμένη τιμή ***Χωρητικότητας*** την οποία δεν πρέπει να υπερβεί σε καμία χρονική στιγμή. Αυτό το πρόβλημα, είναι γνωστό και ως ***Ανισορροπία μεταξύ Ζήτησης-Χωρητικότητας*** και σκοπός μας σε αυτή την εργασία είναι η εύρεση της βέλτιστης κοινής πολιτικής των πρακτόρων ώστε να αποφευχθούν οι συμφορήσεις στον εναέριο χώρο. Η ***Ζήτηση*** είναι η ποσότητα που μετράει πόσα αεροσκάφη διανύουν ή πρόκειται να διανύσουν έναν συγκεκριμένο εναέριο τομέα σε μια συγκεκριμένη χρονική στιγμή. Συνεπώς, όταν η τιμή της ***Ζήτησης*** ξεπεράσει την τιμή της ***Χωρητικότητας***, τότε πλέον υπάρχει ανισορροπία μεταξύ των δύο ποσοτήτων και δημιουργείται ένα ***σημείο συμφόρησης*** στον τομέα.

Η λύση του προβλήματος εντοπίζεται στην υπαγωγή των αεροσκαφών που προκαλούν τη συμφόρηση σε κάποιους κανονισμούς λειτουργίας. Στην περίπτωση μας αυτοι οι κανονισμοί λειτουργίας μεταφράζονται σε λεπτά ***καθυστέρησης***. Στην εργασία μας, μελετάμε την επιβολή καθυστερήσεων στα αεροσκάφη κατα τη διάρκεια της ”προ-τακτικής” φασής. Η συγκεγκριμένη φάση λαμβάνει χώρα αρκετές μέρες

πριν την πτήση των αεροσκαφων και περιλαμβανει την επιβολή καθυστερήσεων σε αεροσκάφη που επρόκειτο να δημιουργήσουν πρόβλημα στον εναέριο χώρο κατά τη μέρα αναχώρησής τους. Παράλληλα, κατά την επιβολή καθυστερήσεων χρήζει προσοχής το γεγονός ότι κάθε λεπτό καθυστέρησης κοστίζει στην αντίστοιχη εταιρεία κάποιο χρηματικό ποσό για τον δεδομένο τύπο αεροσκάφους.

Στην προσέγγισή μας, χρησιμοποιήσαμε μεθόδους πολυπρακτορικής ενισχυτικής μάθησης στην οποία οι πράκτορες συνεργάζονται μεταξύ τους για την επίλυση του κοινου προβλήματος. Υλοποιήσαμε τρεις διαφορετικές μεθόδους οι οποίες επιτυγχάνουν διαφορετικές λύσεις του προβλήματος. Επιπλέον, αξίζει να σημιεωθεί ότι τα πειράματα έγιναν πάνω σε πραγματικά δεδομένα χιλιάδων πτήσεων που προσέφερε το πρόγραμμα DART.

Οι μέθοδοί μας βασίζονται στην εύρεση βέλτισης κοινής πολιτικής η οποία ορίζει για τον κάθε πράκτορα αν πρέπει να καθυστερήσει την αναχώρησή του από το αεροδρόμιο ή αν είναι ελεύθερος να πετάξει με ασφάλεια. Στόχος της εργασίας είναι να βρούμε αυτή την πολιτική η οποία να εξαλοίφει πλήρως τους τομείς συμφόρησης ενώ παράλληλα να μειώνει τα λεπτά καθυστέρησης και το συνολικό κόσος, όσο το δυνατόν περισσότερο.

Chapter 1

Introduction

1.1 Introduction

A multi-agent system is a group of autonomous, interacting entities sharing a common environment, which they perceive with sensors and upon which they act with actuators. They are used in a variety of domains such as robotics, distributed control, resource management, collaborative decision making, congestion avoidance systems, and many more. The agents participating in the multi-agent system, often need to learn new behaviours online, in order to gradually improve their performance as well as the performance of the whole multi-agent system. This is caused by the complexity of the environment that makes the a-priori design of the agent behaviors difficult or even impossible, meaning that it is infeasible to hardcode the behaviour of all the agents.

A Reinforcement Learning agent constantly learns by interacting with its environment. At each time step, the agent using its sensors perceives the state of the environment and takes an action, which causes the environment to transit into a new state. The quality of each transition is being evaluated by a scalar reward, and the aim of the agent is to maximize the cumulative reward.

Moving towards congestion problems, there are many situations where resources of a limited capacity have to be shared by multiple agents simultaneously. These are ever present in the modern world, plaguing various aspects of our business, activities, and daily lives. Most notably, congestion problems appear regularly in various traffic domains. The complexity of these problems makes them difficult to solve with pre-programmed agent behaviours. Multi-agent Reinforcement Learning (MARL) has proven to be a suitable framework for such problems, as it allows autonomous agents to learn by interacting within a common environment. In these kind of environments, agents do not have fully knowledge of the state, but only partial and they have to work together in order to come up with a solution.

More specifically, in the ***Air-Traffic Management (ATM)*** domain, congestion problems arise whenever demand of airspace use exceeds capacity, resulting to ***hotspots***. This is known as the ***Demand and Capacity Balance (DCB)*** problem. The current ATM system worldwide is based on a time-based operations paradigm that leads to DCB issues. These further impose limitations to the ATM system, that are resolved via airspace management or flow management solutions, including regulations that generate delays and costs for the entire system. These demand-capacity imbalances are difficult to be predicted in pre-tactical phase (prior to operation) as the existing ATM information is not accurate enough during this phase.

Against this background, this thesis formalises a generic problem where agents aim to coordinate their joint actions towards the simultaneous performance of tasks, with respect to operational constraints on the use of resources. The generic problem is formulated as a ***multi-agent MDP (MA-MDP)*** and it is instantiated to the demand-capacity problem at the pre-tactical stage of air traffic management operations where resources correspond to air sectors with limited capacity, and where the issue at hand is to minimize the scheduled flight delays, and thus delay costs, while ensuring efficient utilisation of airspace. As part of this formulation, we devise a reward function that takes into account agents' participation in congestions and penalties in terms of implied cost when agents deviate from their schedule to perform tasks. We then proceed to describe multiagent reinforcement learning techniques for learning joint policies, and explore their efficacy using real-world data from the air traffic management domain. Considering operational constraints for the joint performance of the tasks, the proposed methods support agents to reconcile conflicting options and

jointly decide about individual policies, while possessing no information about the preferences and payoffs of others.

The structure of this thesis is organized as follows. Chapter 2 introduces the basic background in single-agent Reinforcement Learning and Markov Decision Processes (MDP). Chapter 3 describes the Demand-Capacity Problem along with the Problem Specifications and presents the related work. In Chapter 4, we begin with the modelling of MDP for the multi-agent case and continue with the collaborative multi-agent Reinforcement Learning algorithms that we formulated in our research. Finally, Chapter 5 discusses the results of our research, and Chapter 6 concludes the thesis.

Chapter 2

Reinforcement Learning

2.1 Markov Decision Processes

This section gives an overview of ***Markov Decision Processes (MDP)***. A Markov Decision Process provides a framework suitable for sequential decision problems that depend on the action taken by an agent at each time step. The agent receives a reward, which depends on the action and the state and the goal is to find a function, called a policy, which specifies which action to take in each state, in order to maximize some function of the sequence of rewards.

MDPs are useful for studying a wide range of optimization problems solved via dynamic programming and Reinforcement Learning. The problem of calculating a complete mapping from states to actions in a stochastic environment with a known transition model, is called a Markov decision problem.

An MDP is usually defined as a tuple (S, A, T, R, γ) where:

- $\boldsymbol{S}$ is a set of states, which represents every possible state the agent can be in, which can be discrete or continuous.

- $\boldsymbol{A}$ is a set of actions, which are the available actions the agents can take in a particular state.
- $\mathbf{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [\mathbf{0}, \mathbf{1}]$ is the transition function; which is a function of the current state **(s)**, the action taken **(a)** and the state where we will end up **(s')**. This transition produces a probability of ending up in state **s'**, starting from the state **s** and taking the action **a**. The transition function denotes a probability distribution over next states given the current state and action such that:

$$\sum_{s' \in S} T(s, a, s') = 1 \quad \forall s \in S, \forall a \in A$$

- $\mathbf{R_i} : \mathbf{S} \times \mathbf{A} \rightarrow \Re$ is a reward function , which is a value that tell us how good it is to enter the new state. This value can be either positive or negative, but must be bounded.
- $\gamma \in [\mathbf{0}, \mathbf{1}]$ is a discount factor.

There are two basic properties on which the Markov Processes are based on:

- Only the present matters; which means that the transition function only depends on the current state **s** and not any of the previous states.
- Things are stationary, therefore rules do not change over time.

A solution must specify to the agent what action to take at any state. A solution of this kind is called a policy. We usually denote a policy by π, and $\pi(s)$ is the action recommended by the policy π for state **s**. If the agent has a policy, then no matter what the outcome of any action, the agent will always know what to do next.

The ultimate goal of the MDP is to find a policy that can tell us, for any state, which action to take. The optimal policy is the one that maximizes the long-term expected reward. In order to evaluate an agent's policy, we have the following concept of the state-value function: The value of a state s ,or the state-value function, under a policy π is defined as the expected return when the agent starts at state **s** and follows a policy π thereafter. Then the state-value function $V^{\pi}(s)$ becomes

$$V^{\pi}(s) = E^{\pi}\left[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s\right]$$

where T is the final time step, t is the current time step, r_t is the received immediate reward at the time step t.

In the previous equation, $T \to \infty$ if the task is an infinite-horizon task such that the task will run over an infinite period. If the task is episodic, T is defined as the terminal time and each episode is terminated when time step T is reached. So, we call the state where each episode ends as the terminal state s_T . In a terminal state, the state-value function is always zero such that $V_{(s_T)} = 0, \quad \forall s_T \in S$. An optimal policy π^* will maximize the agent's discounted future reward for all states such that

$$V^*(s) \geq V^\pi(s), \quad \forall \pi, \forall s \in S$$

The state-value function under a policy -the reward function and the transition distribution are known- can be rewritten as a Bellman equation as follows:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \cdot V^\pi(s')$$

where $T(s, \pi(s), s') = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ is the probability of the next state being $s_{t+1} = s'$ given the current state $s_t = s$ and action $a_t = a$ at time step t, and $R(s, a, s') = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$ is the expected immediate reward received at state s' given the current state s and action a. If the agent follows the optimal policy π^* starting at state s, we have the optimal state-value function denoted by $V^*(s)$. The optimal state-value function $V^*(s)$ is also called the Bellman optimality equation, where

$$V^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V^*(s') \right]$$

Similar to the state-value function, we denote as action-value function the expected return of choosing a particular action a at state s and then following a policy π thereafter. The action-value function $Q^\pi(s, a)$ is given as

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V^\pi(s')$$

If the agent chooses action a at state s and follows the optimal policy π^* thereafter, the action-value function becomes the optimal action-value function $Q^*(s, a)$, where

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^*(s')$$

2.2 Reinforcement Learning

Machine learning is a subset of artificial intelligence and its applications are typically classified into three broad categories; supervised earning, unsupervised learning and reinforcement learning. Reinforcement Learning is different from supervised learning, which is the kind of learning studied the most in the field of machine learning research. Supervised learning is learning from a training set of labelled examples provided by an external supervisor. Each example is a description of a situation together with the label of the correct action the system should take to that situation, which is often to identify a category to which the situation belongs. The objective of supervised learning is for the system to generalize its responses so that it acts correctly when new situations arise that are not contained in the training set. In interactive problems, as the one we deal with in this thesis, it is often difficult to obtain examples of desired behaviour that are both correct and representative of all the situations in which the agent has to act. In uncharted territory, where it is infeasible to hardcode a behaviour, the agent must be able to learn from its own experience.

Reinforcement learning is also different from unsupervised learning, which is typically about finding structure hidden in collections of unlabelled data. Although one might be tempted to think that reinforcement learning is similar to unsupervised learning because it does not rely on examples of correct behaviour, reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure.

The key idea behind reinforcement learning, is for an agent to learn to take decisions by interacting with the environment. In real life, when an infant play, wave its arms, or look around, it has no explicit teacher, but it does have a direct connection -through its senses- to the environment. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals. Throughout our lives, such interactions are undoubtedly a major source of knowledge about the world and ourselves. Whether

we are learning how to drive a car or how to communicate with other people, we are acutely aware of how our environment responds to what we do.

Reinforcement learning is about learning how to map states of the world to actions in order to maximize a numerical reward signal. The learner is not told which actions to take beforehand, but instead must discover by its own which actions yield the most reward by trying them. Most of the time, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Trial-and-error search along with delayed reward, are two of the most important characteristics of Reinforcement Learning.

We formalize the problem of reinforcement learning as a Markov decision process. As we discussed in the previous section, a learning agent must be able to sense the state of its environment to some extent through its sensors and must be able to take actions that affect the state. The agent must also have goals relating to the state of the environment. Any method that is well suited to solve such problems, we consider it to be a reinforcement learning method. The task of Reinforcement Learning is to find an optimal policy that maximizes the expected total reward and to achieve that, the agent uses the observed rewards to learn this optimal policy.

One of the challenges that arise in reinforcement learning, is the trade-off between exploration and exploitation. To obtain higher reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing a high reward. Although, in order to discover such actions, it has to try out actions that they have not been selected in the past. The agent has to exploit what it has already learned to obtain higher reward, but it also has to explore new actions in order to make better action selections in the future. The agent must try a variety of actions and progressively favor those that appear to be the best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. The exploration-exploitation dilemma has been intensively studied by mathematicians for many decades, yet remains unresolved.

Summing up the above, a reinforcement learning setup is composed of two basic components, an agent and an environment. The environment is where the agent is acting upon, while the agent represents the reinforcement learning algorithm. The environment starts by sending a state to the agent. The agent then, based on its

knowledge, takes an action in response to that state. After that, the environment send a pair of next state and reward back to the agent. The agent will update its knowledge with the reward returned by the environment to evaluate its last action. The loop keeps going on until the environment sends a terminal state.

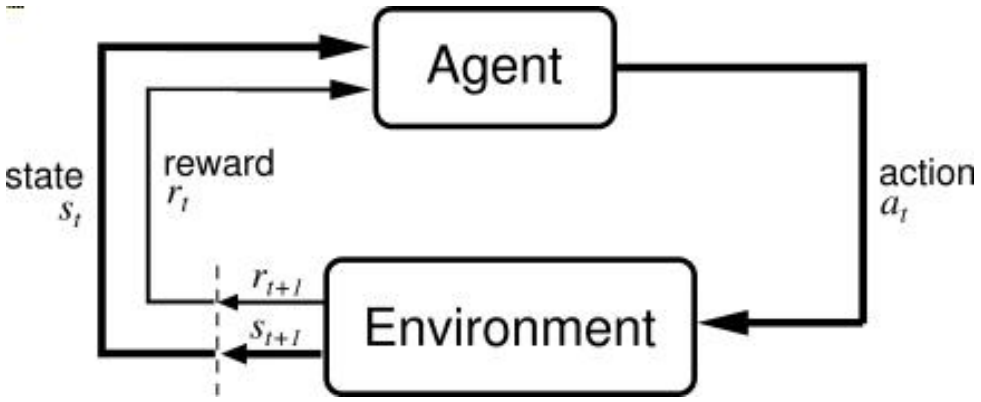


Figure 2.1: Reinforcement Learning Illustration

In many complex domains, reinforcement learning is the only feasible way to train a program to perform at high levels. For example:

- **Gaming**. Recently, DeepMind created AlphaGo, a Deep Reinforcement Learning Go player that beat the World Champion. We can easily understand that the agents' behaviour in this game could not be hardcoded, if we think that Go has over 10^{170} possible states. Similar algorithms used to train Neural Networks along with Reinforcement Learning in order to beat tons of games that were impossible in the past.

- **Manufacturing**. Robots in factories use Reinforcement Learning to pick a device from one box and putting it in a container. Whether they succeed or fail, they memorize the object and gain knowledge and train to do this job with great speed and precision.

- **Power Systems**. Reinforcement Learning and optimization techniques are utilized to assess the security of the electric power systems and to enhance Microgrid performance.

- **Healthcare**. Many Reinforcement Learning applications in health care mostly pertain to finding optimal treatment policies.

- **Advertising**. Learning to rank using one-shot learning for emerging items and new users will bring more money.

There are two ways to divide reinforcement learning algorithms. The first separation is between Model-Based and Model-Free methods. In the first approach, the agent learns the Markov Decision Process model or an approximation of it. This means that the agents have to learn the transition function as well as the reward function. Having done that, they use it to find the optimal policy. Contrary to the first approach, Model-Free methods derive the optimal policy without explicitly learning the model.

The second possible way of dividing the reinforcement learning algorithms is into Passive Learning and Active Learning algorithms.

In Passive Learning, the agent's policy is fixed and the agent tries to learn how good the policy is by observing the world. The three basic approaches of this division are the following:

- **Direct Utility Estimation**. A simple method for this was invented in the late 1950s in the area of adaptive control theory by Widrow and Hoff. This is a model-free method and the idea is that the utility of a state is the average of the total reward from that state onward, and each trial provides a sample of this value for each state visited. Thus, at the end of each sequence, the algorithm calculates the observed reward of the state and updates the estimated utility for that state accordingly, just by keeping a running average for each state in a table.

- **Adaptive Dynamic Programming**. This method does not take advantage of the constraints in the Bellman equation. It is a Model-Based method, as it learns the transition function T and the reward function R. So, based on the underlying MDP model, it can perform policy evaluation.

- **Temporal Difference Learning**. It combines the best of both previous approaches. The key is to use the observed transitions to adjust the values of the observed states so that they agree with the constraint equations. The basic idea of all temporal-difference methods is, first to define the conditions that

hold locally when the utility estimates are correct, and then, to write an update equation that moves the estimates toward this ideal ”equilibrium” equation.

On the other hand, Active Learning is the approach in which the agent must also learn what to do. As opposed to a Passive Learning agent, who has a fixed policy that determines its behaviour, the principal issue here is exploration. The agent must gain as much as possible experience from its environment in order to learn how to behave in it. An agent of Active Reinforcement Learning must make a trade-off between exploitation and exploration to maximize its reward, as we described earlier.

2.2.1 Q Learning

This section will go through one of the most important Reinforcement Learning algorithms, known as Q-Learning [Watkins, 1989]. Q-Learning is an Active Temporal Difference method which learns an action-value function instead of learning utilities. As a model-free algorithm, the action-value function is approximated instead of the state-value function, in order to discover an optimal policy. We will use the notation $Q(s, a)$ to denote the value of being in state s and taking the action a.

The learning procedure of the algorithm goes as follows: At each time step t, the agent observes the state s_t and takes the action a_t according to a followed policy. As a consequence of its action, it transits into a new state s_{t+1} , and receives an immediate reward r_{t+1}. The goal is to maximize the Q-value. The update equation for Temporal Difference Q-learning, which is calculated whenever action a_t is executed in state s_t leading to state s_{t+1}, is given below:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Here α refers to the learning rate and determines how fast are we approaching the goal. A value near zero will make the agent not acquire any new knowledge, while a value of 1 would make the agent consider only the most recent information. The discount factor γ determines the importance of future rewards. A value of 0 will make the agent short-sighted by only considering current rewards, while a value approaching 1 will make it strive for a long-term high reward.

Since Q-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs. High initial values, also known as "optimistic initial conditions", can encourage exploration. Below, we present the basic Q-Learning algorithm.

Algorithm 2.1 Q-learning algorithm

Require:

States $\mathcal{S} = \{1, \ldots, n_s\}$

Actions $\mathcal{A} = \{1, \ldots, n_a\}$

Reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

Transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$

Learning rate $\alpha \in [0, 1]$

Discounting factor $\gamma \in [0, 1]$

Initialize $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ arbitrarily

while Q is not converged **do**

  Start in state $s \in \mathcal{S}$

  while s is not terminal **do**

    Calculate π according to Q and exploration strategy (e.g. $\pi(s) \leftarrow \arg\max_a Q(s, a)$)

    $a \leftarrow \pi(s)$

    $r \leftarrow R(s, a)$ {Receive the reward}

    $s' \leftarrow T(s, a)$ {Receive the new state}

    $Q(s', a) \leftarrow Q(s, a) + \alpha * (r + \gamma * \max_{a'} Q(s', a') - Q(s, a))$

    $s \leftarrow s'$

  end while

end while

return Q

Chapter 3

Problem Specification

3.1 The Demand-Capacity Problem

Congestion problems are omnipresent in our every day life and are we face them in a wide variety of domains such as traffic control, air traffic management and data routing. In our thesis, we deal with problems that arise in the Air Traffic domain. Specifically, we use real-world data to create real scenarios and use our multi-agent reinforcement learning methods to resolve them. The scenarios used contain thousands of flights during heavily congested days and we are challenged to resolve the demand-capacity problem, while simultaneously providing better solutions than the Network Manager.

Air Traffic Management (ATM) is a combination of procedures and techniques that make possible for an aircraft to fly from an origin point to a destination. It is formed by many services supporting this purpose. The demand-capacity problem considers two important types of objects in the ATM system: aircraft trajectories and airspace sectors.

Aircraft trajectories are series of spatio-temporal points containing the longitude, latitude and altitude of the aircraft at a specific time point t_i. At the same time, flight plans are intended trajectories, which consist of events of flights crossing air blocks and sectors, and flying over specific waypoints. Each event specifies the sector that is crossed, the entry and exit locations (coordinates and flight levels), and the entry and exit times, or the exact time that the flight will fly over a specific sector (duration). Other information such as estimated take-off time are specified, and, in case of delay, the take-off time can be calculated by adding the delay to the original take-off time.

Sectors are air volumes segregating the airspace, each defined as a group of air-blocks. These are specified by a geometry (the perimeter of their projection on earth) and their lowest and highest altitudes. As an example, Figure 3.1 depicts projections of airblocks above Europe. Airspace sectorization may be done in different ways, depending on sector configuration. Such a configuration determines the number of active (open) sectors and only one sector configuration can be active at a time. Airspace sectorization changes frequently during the day, given different operational conditions and needs. This happens transparently for the flights.

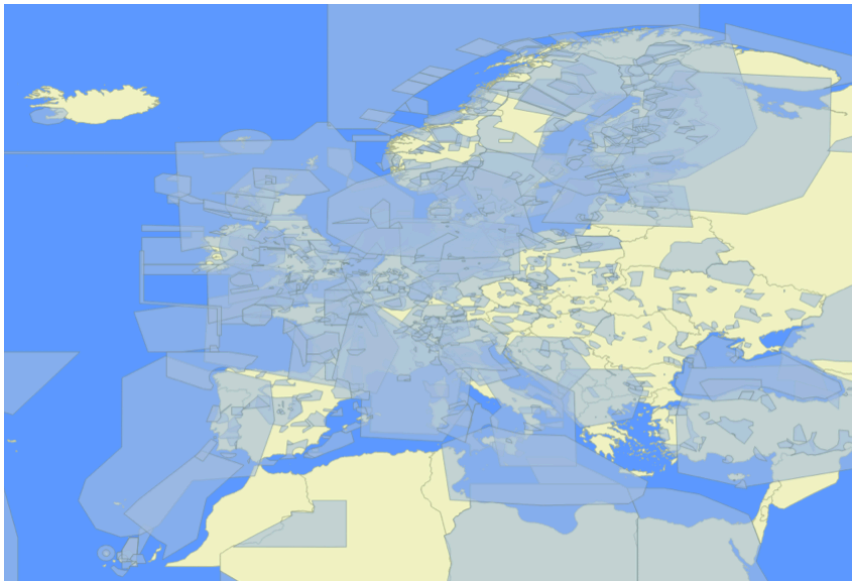


Figure 3.1: Airlocks in 2D: Sectors are groups of adjacent airblocks.

The capacity of sectors is of utmost importance. That is because this quantity determines the maximum number of flights allowed to fly within a sector during a specific time interval. On the other hand, the demand for each sector is the quantity that specifies the number of flights that co-occur (or predicted to occur) during a

specific interval within a sector. Demand must not exceed sector capacity for any time interval, as this could result into conflict on the air. There are different types of measures to monitor the demand evolution, with the most common ones being Entry Rate and Occupancy Count. In this work, we consider Occupancy Count.

The Occupancy of a given sector is defined as the number of flights inside the sector during a selected period, referred as Occupancy Counting Period. In turn, this Occupancy Counting Period is defined as a picture of the sector occupancy taken every time step along an interval of fixed duration. The Step value defines the time difference between two consecutive Occupancy Counting Periods. The Duration value defines the time difference between start and end times of each Occupancy Counting Period. For instance, considering the example in Figure 3.2 for a specific sector, the occupancy counts corresponding to the set of flights at different moments P with duration of 1min and step of 1min are: (a) At P: 1,2,3; (b) at P+1: 1,3,4,5; (c) at P+2: 3,4,6; and (d) at P+3: 4,6,7,8.

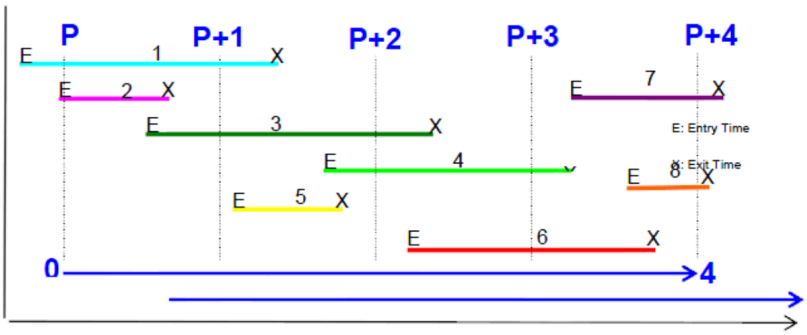


Figure 3.2: Occupancy Step = 1min., Duration = 1min.

In this thesis, we consider the demand-capacity process during the pre-tactical phase. Pre-tactical flow management is applied at least six days prior to the day of operations, and consists of planning and coordination activities. This phase aims to compute the demand for the operations day, compare it with the predicted airspace capacities on that day, and make any necessary adjustments to the flight plans. At this phase, trajectories are sent to the Network Manager who takes into account sector capacities to detect problematic areas. The main objective of this phase is to optimize efficiency and to balance demand and capacity through an effective distribution of resources.

Summing up, our objective is to demonstrate how reinforcement learning methods can help in trajectory forecasting when planned demand exceeds sectors capacity. In this case, regulations in form of delays are applied to the trajectories. Finally, we have to remember that the planning is happening in the pre-tactical phase and after a flight departs we cannot impose any delay or change its route.

3.2 Problem Specification

Let there be n trajectories denoted as $Traj$ that must be executed over the airspace in a total time period of duration H hours. The airspace consists of a set of sectors, denoted by $Sectors$. Time can be divided in intervals of duration Δt, equal to that of the Occupancy Counting Period.

Each agent represents a flight and each flight has a trajectory that is a sequence of timed positions in the airspace. From this sequence, we can derive the series of sectors that each flight crosses along with the entry and exit time for each of these sectors. For the first (last) sector of the flight, i.e. where the departure (resp. arrival) airport resides, the entry (resp. exit) time is the departure (resp. arrival) time. However, there may exist flights that cross the airspace but do not depart nor arrive in any of the sectors within our airspace. In this case, we consider the entry and exit time of sectors within the airspace of our interest.

Thus, a trajectory T in $Traj$ is a time series of elements of the form:

$$T = \{(s_1, entry_{t_1}, exit_{t_1}), ..., (s_m, entry_{t_m}, exit_{t_m})\}$$

where $s_i \in Sectors$, $i = 1, ...m$.

For instance, considering the trajectories T_1 and T_2 in Figure 3.3, these are specified as follows:

$$T_1 = \{(sector_5, 10:00, 10:20), (sector_2, 10:20, 10:45)\}$$

$$T_2 = \{(sector_1, 10:00, 10:05), (sector_2, 10:05, 10:15),$$

$$(sector_7, 10:15, 10:25), (sector_{12}, 10:25, 10:35)\}$$

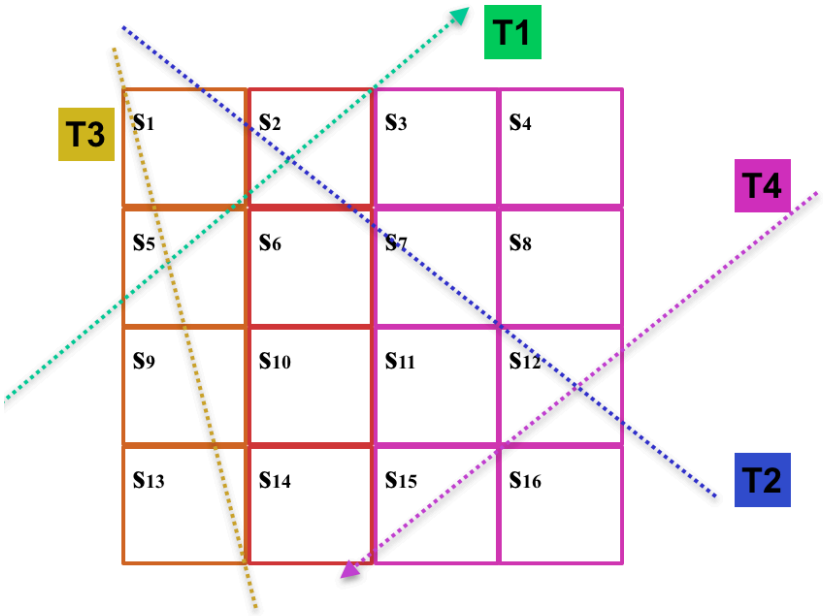


Figure 3.3: Example of trajectories crossing sectors

This information per trajectory suffices to measure the demand $D_{s_i,p}$ for each of the sectors $s_i \in Sectors$ in the airspace in any Occupancy Counting Period p of duration Δt. This measurement of the demand will eventually lead to determine whether the sector is under violation or not.

Specifically, $D_{s_i,p} = |T_{s_i,p}|$, i.e. the number of trajectories in $T_{s_i,p}$, where $T_{s_i,p} = \{T \in Traj | T = (..., (s_i, entry_{t_i}, exit_{t_i}), ...)$, and the temporal interval $[entry_{t_i}, exit_{t_i}]$ overlaps with period $p\}$. In Figure 3.3, considering the trajectories T_1 and T_2 crossing the sector s_2, we have $T_{s_2,p} = \{T_1, T_2\}$, with $p = [10:10, 10:15]$.

Each sector $s_i \in Sectors$ has a specific capacity C_{s_i} . The aim is to resolve imbalances of sectors' demand and capacity. These are cases where $D_{s_i,p} > C_{s_i}$, for any period p of duration Δt in H, in any $s_i \in Sectors$. Δt equals to the Occupancy Counting Period duration. We refer to these cases as capacity violation or Demand-Capacity imbalance cases, resulting to hotspots. In case of capacity violation, the interacting trajectories are defined as hotspot-consulting trajectories which means that one or more of these trajectories needs to be delayed in order to resolve the occuring imbalance. In order to solve the Demand-Capacity problem in these sectors, the flights that participate in them are being given delays in their take-off time.

However, by imposing delays to interacting trajectories often leads to new hotspots in a subsequent time period for the same and/or other sectors crossed by that trajectory. Moreover, it should be stated again that the trajectory or the sequence of sectors crossed cannot be changed for any flight. That means, changing the route of the flight plan is not possible. Instead, by imposing the right delays, along with the restrictions, we are challenged to solve the problem.

In our approach, we consider an agent-based formulation of the problem. That is, every agent A_i is an aircraft performing a specific trajectory in a specific date and time. As it will be specified in the next chapter, these agents have their own interests and preferences, but they collaborate in order to solve the common problem. It must be stated that agents do not have any kind of communication and monitoring constraints given that imbalances are resolved at the planning phase, rather than during operation.

Therefore agents have to learn the joint delays to be imposed to their trajectories with respect to the operational constraints concerning the capacity of sectors crossed by these trajectories. It must be also noted that agents have conflicting preferences since they prefer to impose the smallest delay possible - as it will eventually lead to lower their cost- to their own trajectory, while also executing their planned trajectories safely and efficiently.

Furthermore, each aircraft might has its own type (e.g. Airbus, Boeing, etc.), and each type has its own cost per minute of delay. Another important aspect of the considered flights is whether a flight is commercial or not. Commercial flight describes an aircraft operation to transport passengers, cargo or mail for remuneration or other valuable consideration. On the other hand, non-commercial flights include flight training, military missions and similar operations. The non-commercial flights cannot be delayed opposed to the commercial ones, but we must take them into consideration while solving the Demand-Capacity problem as they fly through the sectors.

Agents with interacting trajectories are considered to be ”neighbours” given that they have to jointly decide on their delays. The actions that every agent takes into a neighbourhood affect immediately all the other neighbouring agents, but not the

other agents outside the neighbourhood. This implies that agents form "neighbourhoods", taking also advantage of the inherent sparsity of the problem. However, these neighbourhoods have to updated when delays are imposed to the agents' trajectories and as a result a dynamic update of the neighbours is necessary.

The problem can also be visualised as a graph. We define a society of agents $S = (Traj, Agents, E)$, with one node per agent $A_i \in Agents$ and any edge $(A_i, A_j) \in E$ connecting agents with interacting trajectories in $Traj$. Next, we define as $N(A_i)$ the neighbourhood of agent A_i, i.e. the set of agents connected to agent $A_i \in Agents$ including itself. It is obvious, that the set of edges are dynamically updated by adding new edges when new interacting trajectories appear.

As presented earlier, in order to solve the Demand-Capacity problem, every agent should be assigned with an appropriate delay. This delay value is bounded. So, for every agent A_i, $D_i \subseteq \{0, 1, 2, ..., MaxDelay_i\}$ denotes the range of delays this agent can take. Each agent A_i has a $MaxDelay_i$ value, which is the maximum delay we can impose to this agent. Moreover, there is also a Maximum Delay value that none of the agents can surpass in a given day.

Considering two neighbours A_i and $A_j \in N(i)-\{A_i\}$, agents must select among the sets of available delay options D_i and D_j respectively, so as to increase their expected payoff with respect to their preferences on delays, and resolve the demand-capacity problem. This problem specification emphasises on the following problem aspects:

- Agents need to coordinate their strategies to execute their trajectories jointly with others with respect to their preferences and operational constraints
- Agents need to explore and discover how different combinations of delays affect the joint performance of their trajectories, given that the way different trajectories do interact is not known beforehand
- Agents' preferences on the options available may vary depending on the trajectory performed, and are kept private

3.3 Related Work

The field of research concerning congestion problems has been heavily studied in the past. In game theory, Rosenthal[14] introduced and studied the congestion games along with the existence of Nash equilibria. Many more classes of congestion games have been introduced since then, and their algorithmic and complexity aspects have been extensively studied in the literature (see, e.g., [15, 16, 17]).

Bazzan [8] et al. were among the first to frame congestion problems as a multiagent coordination problem, while Dresner and Stone [9] proposed a multiagent reservations-based traffic intersections control mechanism.

More recently, Agogino and Tumer [10] proposed a multiagent framework for air-traffic control, and have evaluated their methods using real world airports' data. However, their agents do not correspond to aircrafts but are assigned to specific ground locations throughout the airspace. Agents can perform one of three actions: they can set separations between airplanes; they can order ground delays; or they can order airplane reroutes. This approach is the closest to ours, however it does not guarantee to resolve the DCB problem, can handle only up to two interdependent congestion instances and, moreover the agents are essentially individual action learners.

In the approach of Malialis et al. [11], who use Q-learning to address multiagent congestion problems, they evaluate their approach in road traffic simulation settings using up to 1000 agents, but do not tackle multiple interdependent congestion instances, and do not use real-world data. They devise reward functions that take into account the existence of abstract groups of congested resources, and provide different reward(increased punishment) to agents when they are about to consume a congested resource.

There have also been a few papers using agent-based modelling to address the tactical phase of the ATM problem — i.e., they focus on avoiding aircraft collisions. Two such recent works use decentralized negotiation [12] or iterative peer-to-peer and multiparty collision avoidance methods [13]. None of these works uses machine learning or stochastic decision making methods, however.

As far as we are aware, there are no works that actually deal with the same

prolem in the way that we propose, i.e. using collaborative multiagent RL methods to resolve the DCB problem in pre-tactical phase. Only in T. Kravaris et al. in [2] show encouraging preliminary results on medium size problems in simulated settings.

Chapter 4

Collaborative Reinforcement Learning

In human society, as well as any other society, learning is an essential component of intelligent behaviour. However, each individual agent need not learn everything from scratch by its own discovery. Instead, they may exchange information and knowledge with each other and learn from their peers or teachers. When a task is too big for a single agent to handle they may cooperate in order to accomplish the task.

4.1 Multi-Agent Reinforcement Learning

In this section, we will generalize the Markov Decision Process model to the multi-agent case. Specifically, in this thesis we present the case where the agents act collaboratively trying to maximize their payoff. This can be considered as a sequential decision-making problem. In this category of problems, the agents select a joint action which provides them with a reward and a transition to a new state. As in the case

of single agent, we have no prior knowledge about the effect of the actions that the agents will choose. As a consequence, the agents have to interact with the environment and learn the optimal joint actions based on the received rewards.

Before proceeding further, we should define the basic naming notations to distinguish the multi-agent case from the single-agent. In the multi-agent case, we use the payoff term instead of reward. Also, we denote as utility what we used to call value in the single-agent case and as strategy, the actions that the agents will perform.

There are many models that can describe a multi-agent system interacting with its environment. In our work, we chose the collaborative multi-agent MDP framework [Guestrin, 2003] which is an extension of the MDP framework we discussed in Chapter 2. In this model each agent selects an individual action in a particular state and based on the resulting joint action the system transits into a new state and the agents receive an individual reward. The global reward is the sum of all individual rewards. This approach differs from other multi-agent models, for example, multi-agent MDPs [Boutilier, 1996] or decentralized MDPs [Bernstein et al., 2000], in which all agents observe the global reward. In a collaborative MDP, the goal of the agents is still to maximize the global reward, but the individually received rewards allow for solution techniques that take advantage of the problem structure.

Continuing with the description of MDPs, a multi-agent MDP can be regarded as one large single agent in which the joint action is represented as a single action. It is then possible to learn the optimal Q–values for the joint actions using standard single-agent Q–learning. In this MDP, either a central controller models the complete MDP and communicates to each agent its individual action, or each agent models the complete MDP separately and selects the individual action that corresponds to its own identity. In the latter case, the agents do not need to communicate but they have to be able to observe the executed joint action and the received individual rewards. The problem of exploration is solved by using the ϵ-greedy exploration-exploitation strategy for all agents [3].

In spite of the fact that this approach leads to the optimal solution, it is infeasible for problems with many agents. In the first place, it is intractable to model the complete joint action space, which is exponential in the number of agents. For example, a problem with 20 agents, each able to perform 2 actions (add delay or not) results

in more than one million Q–values per state. Secondly, the agents might not have access to the needed information for the update because they are not able to observe the state, action, and reward of all other agents. Finally, it will need many time steps to explore all joint actions resulting in very slow convergence.

So, in order to exploit its various advantages, we use the model of collaborative multi-agent MDP framework [4][5] which assumes:

- A society of agents $S = (Traj, Agents, E)$.
- A time step $t = t_0, t_1, t_2, ..., t_{max}$, where $(t_{max} - t_0) = H$
- A local state per agent A_i at time t, that correspond to the delay imposed to the trajectory T_i , ranging to the sets of delay options assumed by A_i . Such a state is denoted s_i^t . The joint state $s_{i,j}^t$ of agents A_i and A_j at time t is the tuple of the state variables for both agents. A global state s_t at time t is the tuple of all agents' local states.
- A local strategy for agent A_i at time t, denoted by str_i^t is the action that A_i performs at that specific time point. An action for any agent at any time point, in case the agent is still on ground, may be, either impose a delay or not. Thus, at each time point the agent has to take a binary decision. When the agent flies, then it just follows the trajectory and neither can take additional delay nor change its route. The joint strategy of a subset of agents A of $Agents$ executing their trajectories (for instance of $N(A_i)$) at time t, is a tuple of local strategies, denoted by str_A^t (e.g. $str_{N(A_i)}^t$). The set of all joint strategies for $A \subset Agents$ is denoted as $Strategy_A$. The joint strategy for all agents $Agents$ at time t is denoted str^t .
- The state transition function gives the transition to the joint state s^{t+1} based on the joint strategy str^t taken in joint state s^t .
  Formally $Tr : State \times Strategy \rightarrow State$. It must be noted that although this transition function may be deterministic in settings with perfect knowledge about society dynamics, the state transition per agent is stochastic, given that no agent has a global view of the society, of the decisions of others, while its neighbourhood gets updated. As a result no agent can predict how the joint state can be affected in the next time step. Thus, for agent A_i this transition function

is actually $Tr : State_i \times Strategy_{A_i} \times State_i \rightarrow [0, 1]$, denoting the transition probability $p(s_i^{t+1}|s_i^t, str_i^t)$.

- The local reward function of agent A_i, denoted R_i , is the reward that the agent gets by executing its own trajectory in a specific joint state of its neighbours in $N(A_i)$, and the joint strategy of agents in $N(A_i)$. The joint reward, denoted by R_A , for a set of neighbours A specifies the reward received by agents in A by executing their actions in their joint state, according to their joint strategy. The joint reward R_A for $A \subseteq Agents$ depends on many variables. More specifically, it depends on (a) the number of hotspots occurring while the agents execute their trajectories according to their joint strategy $Strategy_A^t$ in their joint state s_A^t , (b) the chosen delay, (c) the duration that the flight stays within the violated sector and (d) the cost per minute of delay for the respective type of aircraft.

  The reward R_A for a set $A \in Agents$ with join state s_A depends on the participation (contribution) of agents in hotspots according to their joint strategy str_A^t. In our work we have used the following reward function:

$$R_A(s_A^t, str_A^t) = \begin{cases} TDC \times 81 - \lambda \times Delay \times Cost(Aircraft), & \text{if } TDC > 0 \\ C - \lambda \times Delay \times Cost(Aircraft), & \text{if } TDC = 0 \end{cases}$$

  where,

  TDC denotes the duration of agents' participation in congestion instances, i.e. their contribution in hotspots. This is multiplied by the factor 81 which is the average strategic cost (in Euros) per minute in Europe when 92% of the flights do not have delays [18]. This term evaluates the cost of delaying by means of agents' participation in hotspots.

  Delay is the total delay (i.e. the summation of individual delays in minutes) imposed to the agents in A.

  Cost(aircraft) is the summation of strategic delay cost corresponding to the aircraft types executing the trajectories, as specified in [18].

  The parameter **[λ > 0]** represent the importance of minimizing delays and it provides the weight of measuring strategic delay costs.

C is a positive constant representing the total payoff of agents in case where no hotspots (TDC = 0) occur.

It must be noted that alternative schemes of the reward function can be also used by taking into account broader airline-specific strategic policies and considerations regarding flight delays.

- A local policy of an agent A_i is a function $\pi_i : State_i \rightarrow Strategy_{A_i}$ that returns local strategies for any given local state, for A_i to execute its trajectory. The objective for any agent in the society is to find an optimal policy π^* that maximizes the expected discounted future return.

This model assumes the Markov property, assuming also that rewards and transition probabilities are independent of time. Thus, the state next to state s is denoted by s' and it is independent of time.

4.2 Advantages and Disadvantages of Multi-Agent Reinforcement Learning

The distributed nature of multi-agent RL has many benefits. In addition to the speedup made possible by the parallel computation, multiple RL agents can gain a lot from sharing experience, i.e. by communication, teaching or coordination. On the other hand, multi-agent RL carries the burdens of single-agent RL, such as curse of dimensionality and the exploration-exploitation trade-off.

Experience sharing can help agents that perform similar tasks learn faster and produce better solutions. As an example, agents can exchange information during the training; better trained agents can serve as teachers to the poor trained or new agents can learn by imitating the more experienced. Moreover, parallel computation is possible exploiting the decentralized nature of the problem.

Conversely, the curse of dimensionality is caused by the exponential growth of the state-action space in the number of state and action variables. For example, Q-Learning algorithm estimate values for each possible state-action pair, and as a result

a growth in terms of agents and dependencies among them, will leads to an exponential increase in the computational complexity. Furthermore, in multi-agent RL, complications in the efficiency of the algorithms arise by increasing the number of the agents as the exploration-exploitation strategy becomes harder to handle. Agents need not only to obtain information about the environment, but also have to communicate with a subset of the other agents in order to coordinate their actions. Too much exploration may destabilize the other agents, thereby making the learning process more difficult for the exploring agent.

4.3 Coordination Graphs and Max-Plus Algorithm

The most important and challenging part of collaborative Reinforcement Learning is for the agents to cooperate and find an optimal joint policy to follow.

Let there be n agents and each agent A_i follows its individual strategy str_i. As a reminder, strategy refers to the action the agent selects in a particular time step. The joint vector containing all the agents' strategies is $str = (str_1, ..., str_n)$ and generates a payoff for the group of agents, denoted as $u(str)$. The coordination problem is to find the optimal joint strategy vector str^* that maximizes the payoff, that is, $str^* = \max_{str} u(str)$.

The optimal joint strategy can be computed by producing all possible joint strategy vectors and choosing the one that maximizes $u(str)$. It is easily understood that this approach is infeasible as the size of the joint strategy space grows exponentially with the number of the agents. Fortunatelly, as we mentioned earlier, the nature of our problem helps us, as the agents do not depend on the actions of all other agents, but only on a subset of them. This subset of agents, has been denoted as the neighbourhood of an agent, and it is formed when the agent participates in a hotspot.

So, we exploit the Coordination Graphs (CG) framework which assumes that the strategy an agent A_i follows, depends only on a subset of the systems' agents, $j \in \Gamma(i)$, that is the neighbourhood of this agent. The global payoff can be written as a linear combination of all local payoff functions:

$$u(str) = \sum_{i=1}^{n} f_i(str_i)$$

Each local payoff function f_i depends on the joint strategy of agent A_i and its neighbours in $N(A_i)$. This can be visualised as an undirected graph G = (V, E) with $|V|$ nodes (agents) and $|E|$ edges and each edge $(i, j) \in E$ shows that the agents A_i and A_j have to coordinate their actions.

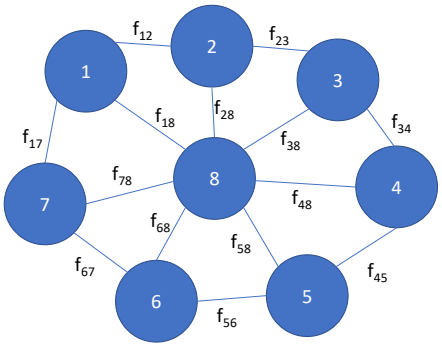


Figure 4.1: Example of a Coordination Graph.

Figure 4.1 shows an example of a Coordination Graph containing 8 nodes and between the nodes the corresponding edges. The payoff contribution of a pair of actions (str_i, str_j) of two neighbouring agents A_i and A_j is denoted as f_{ij}. In our case, every CG we examine is a complete graph, that is, every agent connects and shares an edge with every other agent within its neighbourhood. As a result, every neighbourhood is formalized as an individual Coordination Graph. Each agent A_i may belong to many Coordinated Graphs, as it can take part to more than one hotspots.

To solve the coordination problem, we have to find the optimal joint strategy $str^* = max_{str} u(str)$. In order to do that, we can either apply Variable Elimination (VE) algorithm or the approximate Max-Plus algorithm. Briefly, VE algorithm eliminates the agents one by one and before the elimination the agent collects the payoff produced by its edges. It guarantees that it will always find the optimal joint strategy

and it does not depend on the elimination order. Contrary, the execution time of the algorithm heavily depends on the order we choose to eliminate the agents. To find the optimal elimination order it is considered to be an NP-complete problem, but there exist some heuristics.

Nevertheless, an approximate alternative to the VE algorithm exists. Max-Plus algorithm [7] is a popular method for computing the maximum a-posteriori (MAP) configuration in an undirected graphical model. It operates by iteratively sending locally optimized messages $\mu_{ij}(str_j)$ between node i and j over the corresponding edge in the graph. After convergence, each node then computes the MAP assignment based on its local incoming messages only [1]. In order to find the optimal joint strategy str^*, each agent A_i repeatedly sends messages μ_{ij} to its neighbours:

$$\mu_{ij}(str_j) = \max_{str_i}\{f_i(str_i) + f_{ij}(str_i, str_j) + \sum_{k\in\Gamma(i)-j} \mu_{ki}(str_i)\}$$

where $\Gamma(i) - j$ is the neighbours of A_i except A_j.

The produced message is an approximation of the maximum payoff that the agent A_i is able to achieve given an action of agent A_j, and it is computed by maximizing over the sum of the payoffs and all the incoming messages to A_i. These messages are being exchanged until they converged to a fixed point.

The main difference between VE and Max-Plus algorithm, lies to the fact that in the latter an agent only has to sum over the received messages of its neighbours instead of enumerating over all possible action combinations. Each agent A_i selects the optimal strategy str_i^* based on a function that combines its payoff and the summation of the incoming messages from its neighbours. We denote this function as $g_i(a_i)$:

$$g_i(a_i) = f_i(str_i) + \sum_{j\in\Gamma(i)} \mu_{ji}(str_i)$$

$$str_i^* = argmax_{str_i} g_i(str_i)$$

The optimal joint action str^* is computed by local optimizations, as every node individually maximizes its own $g_i(str_i)$. The main disadvantage of the method is that it does not guarantee convergence in graphs with cycles. Below, we present the Max-Plus Algorithm for a Coordination Graph $CG = (V, E)$.

Algorithm 4.1 Max-Plus Algorithm

Input: $CG = (V, E)$

Initialize messages $\mu_{ij} = \mu_{ji} = 0$ for every $(i, j) \in E$

Initialize $g_i = 0$ for every $i \in V$

while $fixed\_point = false$ **do**

$fixed\_point = true$

for `every agent` i **do**

for `all neighbours` $j = \Gamma(i)$ **do**

send j message

$$\mu_{ij}(a_j) = \max_{a_i}\{f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i)-j} \mu_{ki}(a_i)\} - c_{ij}$$

if $\mu_{ij}(a_j)$ differs from previous message by a small threshold **then**

$fixed\_point = false$

end if

determine $g_i(a_i) = f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)$

$a_i^* = argmax_{a_i} g_i(a_i))$

end for

end for

end while

return a^*

The following sections will describe three collaborative Reinforcement Learning methods used in this thesis, considering that the agents do not know the transition and reward model (model-free methods). The three methods that will be discussed are: Independent Reinforcement Learners, Agent-Based Collaborative Reinforcement Learners and Edge-Based Collaborative Reinforcement Learners.

4.4 Independent Reinforcement Learners

This method implements the Q-Learning algorithm independently for every agent participating in the multi-agent system.

In this approach (Claus and Boutilier, 1998), the agents ignore the actions and rewards of the other agents, and learn their strategies independently, that is each agent learns its own policy independently and treats other agents as part of the environment. Each agent A_i stores and updates an individual local table Q_i and the global $Q - function$ is defined as a linear combination of all individual contributions, $Q(s, str) = \sum_{i=1}^{n} Q_i(s, str_i)$.

A local table Q_i is updated using the global temporal-difference error, the difference between the current global Q-value and the expected future discounted return for the experienced state transition, using

$$Q_i(s_i, str_i) := Qi(s_i, str_i) + \alpha[R(s_{N(A_i)}, str_{N(A_i)}) + \gamma \underset{str_i}{max} Q_i(s_i', str_i) - Q_i(s_i, str_i)]$$

It must be noted that instead of the global reward used in [6], we use the reward received by the agent, taking into account only the joint state and joint strategy of its neighbourhood.Furthermore, this method considers only local states and strategies and it is in contrast to the approach of Coordinated Reinforcement Learning model proposed in [6], since that model needs to know the maximising joint action in the next state, the associated maximal expected future return, and needs to estimate the global Q-value in the global state. As already pointed out, agents' neighbourhoods are not stable and they need to be updated occasionally during the learning process.

4.5 Sparse Cooperative Q-Learning

In this section, we describe the two other methods using Collaborative Reinforcement Learning. These methods are based on Sparse Cooperative Q-Learning, or $SparseQ$, methods which also approximate the global Q-function into a linear combination of local Q-functions. The decomposition is based on the structure of a Coordinated Graph which is chosen beforehand. This decomposition can either be in terms of the edges or nodes(agents). In our approach, we construct the Coordination Graph assuming the following:

- Every agent (flight) is a node.

- An edge between two graph nodes exists if and only if the two corresponding nodes (flights) are involved in the same hotspot.

Two connected flights on that graph are called neighbours. A flight can be a neighbour of another flight only if they both can be found at the same sector, at the same occupancy period. It has to be noted here, that given a maximum delay for each flight, the set of possible interacting flights at a time slot is finite.

In the agent-based decomposition the local function of an agent is based on its own action and those of its neighbouring agents. In the edge-based decomposition each local function is based on the actions of the two agents it is connected to. In order to update a local function, the idea is to base the update not on the difference between the current global Q–value and the experienced global discounted return, but rather on the current local Q–value and the local contribution of this agent to the global return. In this thesis we utilize the edge-based decomposition.

In the edge-based decomposition, we have our coordination graph $G = (V, E)$ with $|V|$ nodes (agents) and $|E|$ edges. Each edge $(i, j) \in E$ corresponds to a local Q-function Q_{ij} and the sum of all Q-functions defines the global Q-function:

$$Q(s, a) = \sum_{(i,j)\in E} Q_{ij}(s_{ij}, str_{ij})$$

where $s_{i,j} \subseteq s_i \subseteq S_j$ is the subset of the state variables related to A_i and A_j. Figure 4.2 shows an example of an edge-based decomposition for a 4-agent problem.

Furthermore, in order to update a local Q-function, we have to propagate back the reward received by the individual agents. This is complicated, as the rewards are received per agent, while the local Q-functions are defined over the edges. As a result, for an agent in a neighbourhood we cannot directly derive the dependency generated the reward. So, we need to associate every agent with a local Q-function Q_i that is directly computed from the edge-based Q-functions Q_{ij}. In order to compute Q_i, we assume that each edge-based Q-function contributed equally to the two agents that form the edge. Then, the local Q-function Q_i of agent A_i is defined as the summation of half the values of all local Q-functions Q_{ij} of A_i and its neighbours $j \in \Gamma(i)$:

$$Q_i(s_i, str_i) = \frac{1}{2} \sum_{j\in\Gamma(i)} Q_{ij}(s_{ij}, str_{ij})$$

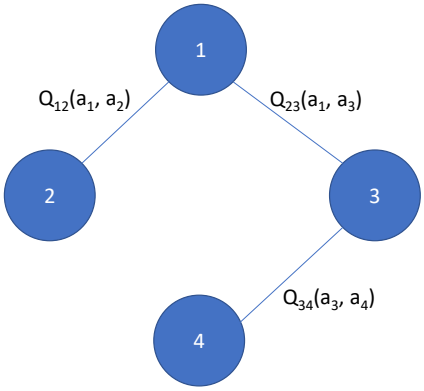


Figure 4.2: An edge-based decomposition of the global Q-function for a 4-agent problem.

Finally, to compute Q we have to sum all local Q-functions Q_i. The following subsections, describe two update methods for the edge-based decomposition.

4.5.1 Edge-Based Update

This is a variant of the edge-based update of the sparse cooperative edge-based decomposition Q-learning method proposed in [8]. Given two peer agents, A_i and A_j, the Q-function is denoted succinctly $Q_{ij}(s_{i,j}, str_{i,j}$. For every agent A_i we made the assumption that the reward R_i is divided proportionally over its neighbours $\Gamma(i)$. We use the following to update:

$$\frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(s_{ij}, str_{ij}) :=$$

$$\frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(s_{ij}, str_{ij}) + \alpha[\sum_{j\in\Gamma(i)} \frac{R_i(s_i, str_i)}{|\Gamma(i)|} + \gamma\frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(s'_{ij}, str^*_{ij}) - \frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(s_{ij}, str_{ij})]$$

In order to update an individual local Q-function Q_{ij}, we remove the sums, because one half of every local Q-function Q_{ij} is updated by the agent A_i and the other half

by the agent A_j. Adding the two of them gives us the following update rule for a single local Q-function Q_{ij} :

$$Q_{ij}(s_{ij}, str_{ij}) :=$$
$$Q_{ij}(s_{ij}, str_{ij}) + \alpha[\frac{R_i(s_i, str_i)}{|\Gamma(i)]} + \frac{R_j(s_j, str_j)}{|\Gamma(j)]} + \gamma Q_{ij}(s'_{ij}, str^*_{ij}) - Q_{ij}(s_{ij}, str_{ij})]$$

Each local Q-function Q_{ij} is updated with a proportional part of the received reward of the two agents forming the edge and with the contribution of this edge to the maximization of the joint strategy str^*_{ij} in state s'_{ij}. The latter is computed by either applying the approximate max-plus algorithm we described earlier or by using our variant that uses each agent's max value for the next state. In our method the strategy str^*_i is the best known strategy for that agent and it is depicted directly from the agent's value function, $Q_i(s_i, str_i)$, which has been calculated as the summation of local Q_{ij} values in its neighbourhood:

$$str^*_i = argmax_{str_i} Q_i(s_i, str_i)$$

$$Q_i(s_i, str_i) = \frac{1}{2} \sum_{j \in \Gamma(i)} Q_{ij}(s_{ij}, str_{ij})$$

The above method for the selection of str^*_i provides an approximation that, as our experimental study has shown, it offers qualitative solutions in a significantly efficient way than a message-passing approach.

4.5.2 Agent-Based Update

As we previously discussed, the edge-based update method divides the reward proportionally over the different edges of an agent. A different approach from this, is the agent-based update, in which we first compute the temporal-difference error per agent and then divide this value over the edges. For this, we use:

$$\frac{1}{2} \sum_{j \in \Gamma(i)} Q_{ij}(s_{ij}, str_{ij}) :=$$
$$\frac{1}{2} \sum_{j \in \Gamma(i)} [Q_{ij}(s_{ij}, str_{ij})] + \alpha[R_i(s_i, str_i) + \gamma Q_i(s'_i, str^*_i) - Q_i(s_i, str_i)]$$

In order to transform into a local update function, we re-write the temporal difference error as a summation of the neighbours of agent A_i :

$$R_i(s_i, str_i) + \gamma Q_i(s'_i, str^*_i) - Q_i(s_i, str_i) = \sum_{j \in \Gamma(i)} \frac{R_i(s_i, str_i) + \gamma Q_i(s'_i, str^*_i) - Q_i(s_i, str_i)}{|\Gamma(i)|}$$

Just as the edge-based update, there are two agents which update the same local Q-function Q_{ij}. So, we can decompose by removing all sums. When we add the contributions of the two involved agents -A_i and A_j-, we get the following local update equation:

$$Q_{ij}(s_{ij}, str_{ij}) :=$$
$$Q_{ij}(s_{ij}, str_{ij}) + \alpha \sum_{k \in (i,j)} \frac{R_k(s_k, str_k) + \gamma Q_k(s'_k, str^*_k) - Q_k(s_k, str_k)}{|\Gamma(k)|}$$

This update rule propagates back the temporal-difference error from the two agents which are involved in the local Q-function of the edge that is updated. The difference of agent-based update from the edge-based update method is that the latter directly propagates back the temporal-difference error related to the edge that is updated.

This is also a variant of the original agent-based update method proposed in [1], as it computes the maximizing joint action str^* using the maximum next value for every agent instead of max-plus or VE algorithm.

Chapter 5

Experimental Results

In this chapter, we will present the experimental results of our proposed methods upon real-world scenarios. Initially, we will describe the experimental procedure along with the environment settings and present the results of every method. Finally, we will compare our results to the solutions given by the Network Managers.

5.1 Experimental set-up

In this thesis, we have performed a series of experiments -both in artificial and real-world data- in order to test and compare the efficiency of the three collaborative Q-learning methods. The efficiency is measured in terms of the resulting number of hotspots, the average delay achieved and the distribution of delays to interacting flights. Moreover, we tested our methods in real-world scenarios with thousands of flights.

Concerning the free parameters in the reward function, λ and C, we experimented with plenty of values. A typical value for $lambda$ is $lambda = 20$, after experimental

study. Using this value, the proposed methods provide solutions with reduced average delays per flight, compared to cases where $\lambda < 20$. For the constant term C, we experimented with values resulting to high positive feedback when the hotspots that the agent participated were eliminated. We used the value 500.000 for our experiments, as lower values tested did not solve the problems and higher values would not reach the performance of this value.

For the real world data, all methods have been executed for 10000 episodes following an ϵ-greedy exploration-exploitation strategy starting from probability 0.9, which every 80 episodes is diminished by the value of 0.01. At episode 7200 the probability becomes 0.001 and is henceforth considered zero. Then, a pure exploitation phase starts. The learning procedure we followed for the artificial data will be presented on the next section as the scenarios were significantly smaller than the real ones, but their contribution was crucial as they helped us understand that the methods can be successfully applied to our problems and motivated us to continue our research.

In addition to the above, and in order to enhance the efficiency and the quality of results received by any of the proposed methods, we have considered a deterministic rule for the flights that do not participate in any congestion. These flights are set to have delay equal to 0, that is they are free to depart from the airport at their specified take-off time, unless they participate in any hotspot in the future before their departure time. It must be pointed out that any of these flights may participate in congestions in any future state, as the delays imposed to other flights may lead to the creation of new hotspots in subsequent times. In this case the rule does not apply for these flights and the corresponding agents participate in the multi-agent RL process. The rationale for that rule is to prune the search space by having the delay of flights that do not contribute to congestions, and thus do not have neighbours in the agents' society, set to zero.

Furthermore, as mentioned before, there are many flights that considered to be non-commercial. We cannot deal with those flights, and as a result these flights have delay equal to 0. However, these flights take part in the multi-agent RL process as they increase the Occupancy Count of the sectors they cross.

Parameters	Value
Number of planes, N	100
Max delay	10
Number of sectors	16 (4 × 4)
Capacity of sectors, Cap	∈ [4, 10]
Total time period duration, H	180

Table 5.1: Parameter values used during the simulated experiments

5.2 Experimental Results with Artificial Data

We have performed a series of experiments with artificial data in order to test and compare the efficiency of the three collaborative Q-learning methods to resolving the DCB problem in ATM. To this purpose, we create specific simulation scenarios of trajectories crossing an airspace. The scenarios are artificial, but correspond to typical and difficult cases in the real world, found in datasets provided by CRIDA, the Spanish Reference Centre for Research, Development, and Innovation in ATM. They have been used during the initial phase of our research in order to control the experimental settings and explore the potential of the proposed methods.

For the simulation we consider that the airspace comprises a grid of sectors, all having a specific capacity value (that could possibly differ from sector to sector). Table 5.1 presents the data used in producing the experimental cases and the parameter values used in all simulated runs.

All three approaches follow an ϵ-greedy exploration strategy starting from probability 0.8, which is gradually reduced in subsequent rounds. However the Ind-Colab-RL differs from the other methods in that it initiates an ϵ-greedy exploitation phase for 1000 rounds with high probability, while in a subsequent phase of 1000 rounds, it does pure exploration. To evaluate the three approaches in cases of varying difficulty, we modify the capacity of sectors (Cap), and the number m of sectors that each flight crosses. Herein we report results only for the most hard cases in the grid considered, where $m \in [3, 4]$. For every capacity value $Cap \in [4, 10]$, we generated 50 random experimental cases. Figure 5.1 shows the mean value and the standard deviation of the final (after learning) number of hotspots, as well as the mean delay for all flights

and for all experiments performed. According to the results and as shown in Fig. 5.1 (a), all methods demonstrated very similar behaviour with respect to hotspots' eradication, with Ed–Colab–RL being slightly more effective compared to others. The x-axis in Fig. 5.1 (a) shows the capacity of each sector, while y-axis shows the number of hotspots when agents' strategies converge.

When the capacity of sectors was greater than or equal to 7 all methods reached the optimum policy for the hotspot criterion. However, an improvement in the 'mean delay' criterion is shown in Fig. 5.1 (b) concerning the edge-based and the agent-based collaborative RL approaches: x-axis in this figure shows the varying capacity of each sector, and the y-axis shows the mean delay achieved by each method. Ind–Colab–RL shows the worst performance, while the performance of Ed–Colab–RL is similar to that of Ag–Colab–RL, although the later is more consistent while the capacity of sectors increases. This confirms that the proposed multi-agent formulation provides a promising framework for tackling the DCB problem.

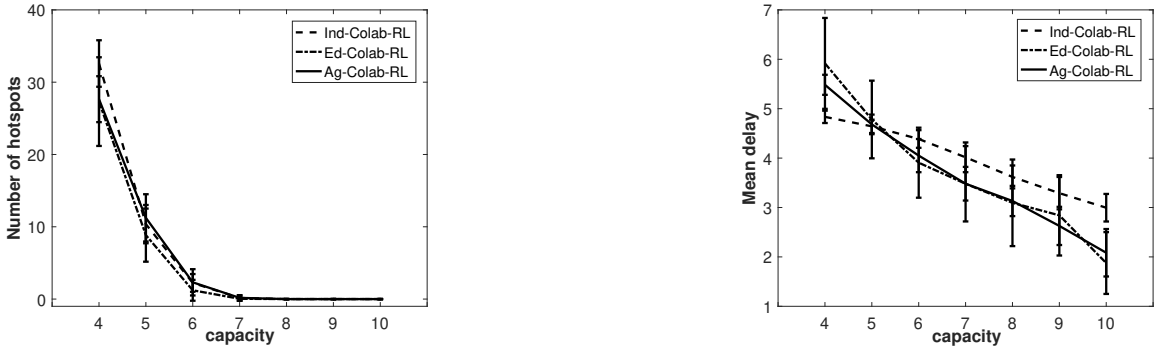
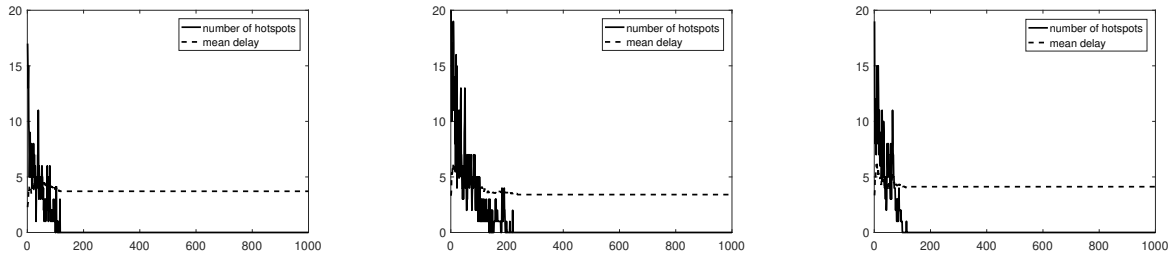


Figure 5.1: Comparative results: Plots illustrate (a) the number of hotspots and (b) the mean delay estimated by each method in terms of various values of sectors' capacity (x-axis)

Figure 5.2 illustrates an example of the received learning curves by each method, i.e. the number of hotspots and mean delay as estimated for 1000 episodes during learning (we set sector's capacity as C = 7 to all cases). For the Ind–Collab–RL method, these episodes are from the pure exploration phase.

All methods were able to converge rapidly, achieving strategies with zero hotspots to any sector, and with flights' delay much less than the maximum acceptable delay.



(a) Ind-Colab-RL (b) Ed-Colab-RL (c) Ag-Colab-RL

Figure 5.2: Learning curves received by three methods in a setting considering sectors' capacity equal to 7. The x-axis shows the number of the learning episode, while the y-axis shows the number of hotspots and mean delay achieved in each episode.

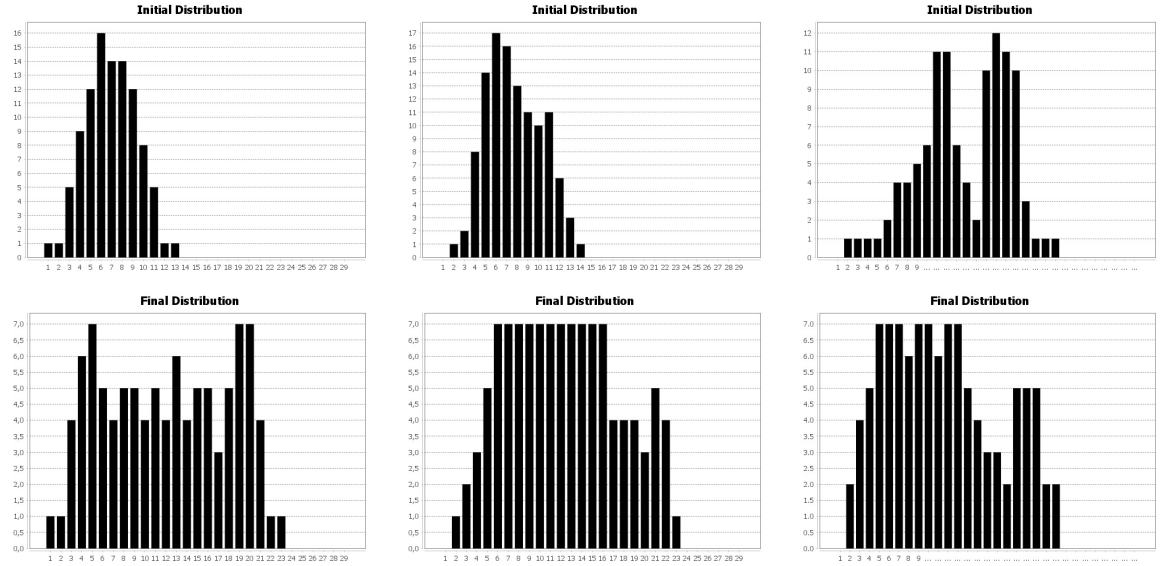


Figure 5.3: An example of the distribution of interacting flights in Occupancy Counting Periods (upper) initially and (lower) as produced by three methods

Finally, in Figure 5.3 we present an example of the distribution of hotspots (y-axis) in terms of Occupancy Counting Periods in a number of 29 non-overlapping occupancy periods, each of duration equal to 6 time instants (e.g. 6 minutes). This was obtained by measuring the interacting flights to a specific sector in different periods: (a) at the beginning and (b) at the end of learning. As can be seen, our schemes manage to offer strategies with significantly reduced hotspots (zero in these cases, given that demand in any occupancy period is not greater than capacity).

5.3 Experimental Results with Real-World Data

In this section, we present three different real-world scenarios used in our study to test our multi-agent Reinforcement Learning methods. All the real-world scenarios concern the flights above Spain in a specific day, i.e. in a period of 24 hours, H = 1440 mins. We have selected among the given days the most challenging ones in order to test our algorithms. Moreover, the learning procedure we followed for the real-world scenarios is the one we described in section 5.1. That is because the real-world scenarios include thousands of flights and as a result the methods need much more time to learn. Some characteristics of the scenarios used are:

- We deal with the DCB problem at the pre-tactical stage, we consider flight plans, which are intended, scheduled trajectories, that specify departure and arrival airports, as well as events of flights crossing sectors and flying over specific waypoints, with the time schedule for these events to happen.
- There may be multiple sector configurations applied in a single day. However, at each time instant one such configuration is considered; specifying the sectors that are open in a specific time period and their capacities.
- We have used reference values for the cost of delay to European airlines as provided in [18], which are also used by SESAR 2020 Industrial Research.
- Our scenarios also include the non-commercial flights that cross Spain in the selected days. We do not deal with them (i.e., we cannot impose any delay to them) but consider them part of the problem.

Table 5.2 contains the parameters used in the real-world scenarios. Specifically, it contains the number of flights, the max delay value and the number of sectors for each scenario respectively.

Parameters	Scenario 1	Scenario 2	Scenario 3
Number of planes, N	3676	3529	5544
Max delay	71	71	66
Number of sectors	144 (12 × 12)	144 (12 × 12)	169 (13 × 13)

Table 5.2: Parameters used in our study with the three real-world scenarios

Problem	Number of Delayed Flights	Avg Delay (mins) (all flights)	Avg Delay (mins) (delayed flights)
Ed-MARL	63	0.19	0.89
Ag-MARL	65	0.21	0.91
IRL	65	0.25	0.91
Ed-MARL	276	0.99	3.88
Ag-MARL	249	1.07	3.51
IRL	291	1.24	4.10
Ed-MARL	618	0.75	9.36
Ag-MARL	536	0.93	8.12
IRL	704	1.06	9.92

Table 5.3: Comparative results in three scenarios

Table 5.3 shows the experimental results. Specifically, they give the number of delayed flights as a result of applying the methods indicated in the first column; the average delay to the delayed flights, as well as the average delay to all flights, in the second and third columns, respectively. The number of delayed flights circumscribes all the flights that have been given at least one minute of delay during the pre-tactical phase. All the results indicate the average values of results produced in 10 independent experiments. Each row of the table represents a scenario resolved by our three proposed methods.

To be more accurate, the results shown in Table 5.3 correspond to solutions given by our methods. This means that our algorithms solved the DCB problem resulting to 0 hotspots at the end of the day.

Additionally, in Figure 5.4 we show the learning curves of the three multi-agent RL

methods for 10000 learning episodes. In each case, the x-axis indicates the number of episodes, while the y-axis indicates the average delay of all flights achieved at each episode. Actually, this is the average of the delays reported in all independent experiments per problem.

Furthermore, the histograms show the distribution of delays to the flights. The x-axis shows the delay imposed (in minutes), while the y-axis the percentage of flights to which this delay has been imposed. For constructing the histograms and for better visualization of the results according to operational concerns, we have used two kind of bins: a more detailed scheme with 6 or 7 bins (second column) and a one with two bins (third column) which shows the percentage of delays in less (respectively, more) than half an hour.

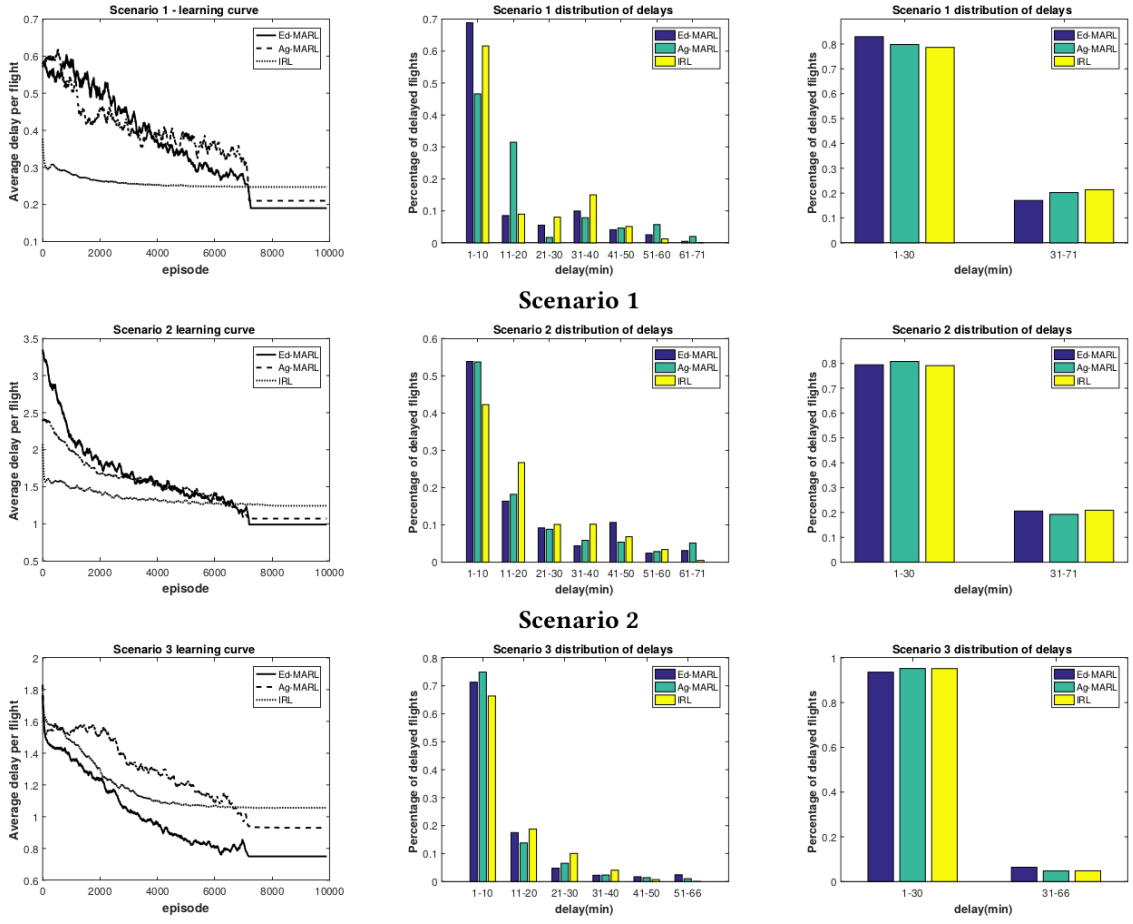


Figure 5.4: The learning curves and the distribution of delays of the three comparative multiagent RL methods.

An initial observation from the application of the multi-agent RL methods is that

all three methods manage to provide solutions to the congestion problems, imposing delays that result to zero hotspots. Furthermore, the quality of the policy found by all methods is quite significant since, as shown in the histograms of Figure 5.4, less than 20% of the delayed flights have delay more than half an hour. Actually, this percentage becomes very low in the case of scenario 3 which is the most complex one. This is valuable by considering that the aim is to reduce the delays imposed to flights, even if the number of flights delayed increase.

Comparing the three methods, as presented in Table 5.3, we can observe that the collaborative multi-agent RL methods (Ed-MARL & Ag-MARL) seem to perform better in terms of delayed flights and average delay to the flights. More specifically, Ed-MARL provides results with significantly reduced average delay compared to the other two, while Ag-MARL provides results with significantly reduced delayed flights. Also, from the learning curves it is obvious that each one of the three methods converges a little after getting into the exploitation phase (after 7200 episodes). Independent RL method seems to converge much faster - especially in the first two scenarios-, however it converges to a local maximum as far as the delays imposed to the concerned flights.

Last but not least, histograms in Figure 5.4 show that both collaborative multiagent RL methods provide a distribution with very high percentage (almost $70 - 80$ %) of delayed flights with delays less than 10 mins. The Ag-MARL method is more compact and performs slightly better since it achieves a distribution with fewer delays in large values of delay that the other methods.

5.3.1 Comparison with CFMU

CFMU stands for Central Flow Management Unit. CFMU tracks all flight plans and is responsible for air traffic control and devising solutions that optimize the handling of traffic flows according to the available capacity. The Network Manager carries out air traffic management network functions for the European Commission. These units decide the delay that needs to be given to the flights in order to reach their destination safely.

For the comparison, we have selected one of the available days in our database. The parameters of the scenario produced by this day are depicted in Table 5.4.

Parameters	Scenario
Number of planes, N	5408
Max delay	95
Number of sectors	144 (12 × 12)

Table 5.4: Parameters of the selected scenario.

Problem	Number of Delayed Flights	Avg Delay (mins) (all flights)
Ed-MARL	577	2.89
Ag-MARL	573	3.17
IRL	618	3.73
CFMU	281	18.55

Table 5.5: Comparative results to CFMU

Table 5.5, presents the results of our methods along with the results provided by the CFMU. It must be pointed out that according to CFMU data, their regulated flights do not resolve the DCB problem, i.e. even if we impose regulations to the CFMU regulated flights we still have cases within the day where demand exceeds capacity. In contrary, our three methods always converge to 0 hotspots. It is also clear, that our approach leads to better results in average delay.

Comparing the three methods, we can observe that Ed-MARL provides again the lowest average delay and Ag-MARL the lowest number of flights even though the three methods are close. IRL solves the problem as well and provide better solution than CFMU, but in the specific scenario cannot out-perform the two collaborative methods.

Figure 5.5 depicts the distribution of flights in a congested sector in the initial problem. On the other hand, Figure 5.6 and Figure 5.7 show the picture of the sector after imposing the regulations decided by two of our methods (Ed-MARL and IRL respectively).

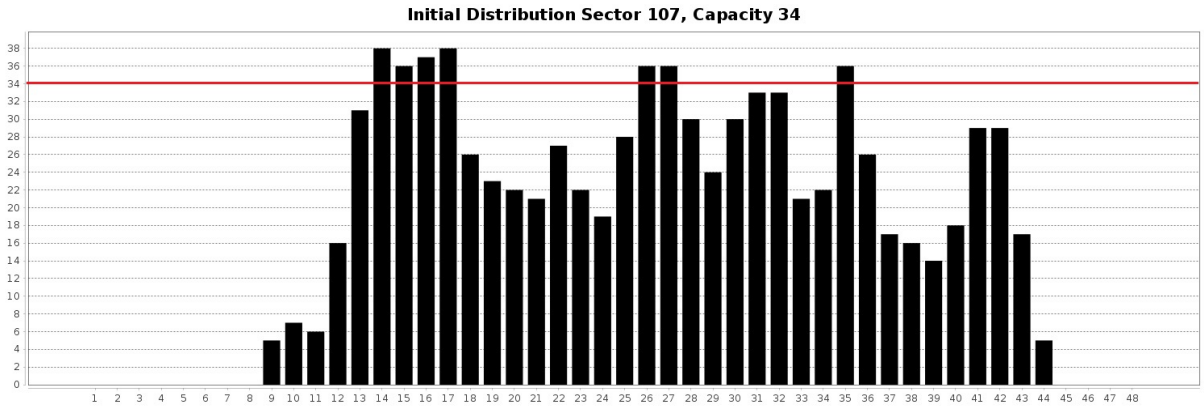


Figure 5.5: The initial problem state in a congested sector.

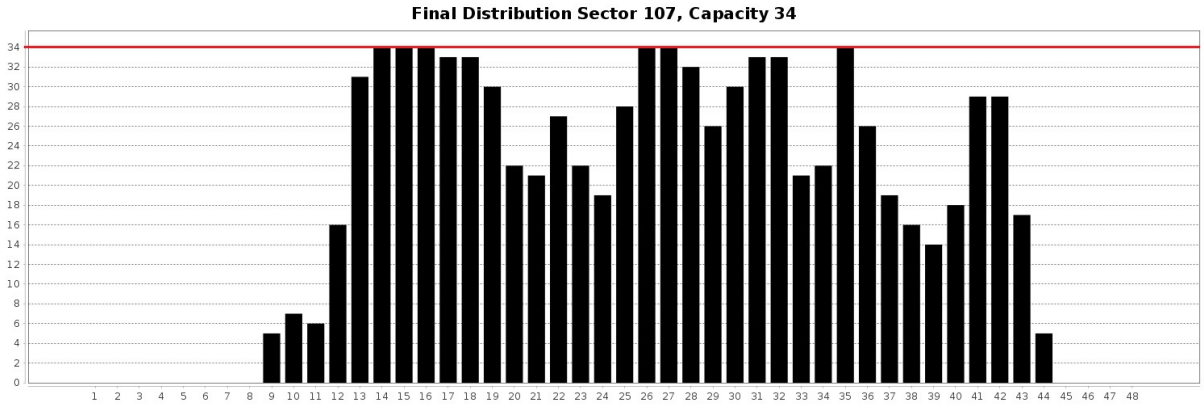


Figure 5.6: The final state after the regulations. (Ed-MARL)

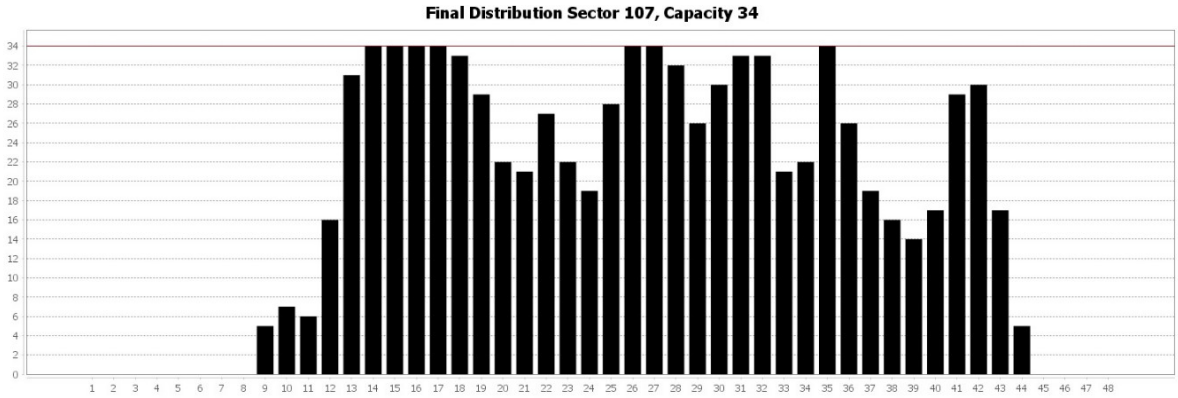


Figure 5.7: The final state after the regulations. (IRL)

In the previous Figures, we observe the configuration of the sector before and after the regulations imposed to the flights that participate in the sector. The x axis shows the periods for measuring demand according to the hourly counting period metric (60' window with a step of 30'), while the y axis shows the demand. The red line indicates the capacity for the sector. Thus, any bar above that line indicates excess in capacity (hotspot). It is easy to observe that both our methods solve the problem of demand-capacity imbalance presented in the examined sector by imposing delays to the flights.

As results show, methods do "push" excess of capacity in subsequent periods within the same sector, or in other sectors (not shown here). This happens in small scale, i.e. solutions affect the demand for only 2 or 3 subsequent periods within the sector. This shows that delays imposed do not increase the workload per sector considerably, leaving much space for increasing further the demand, if this is also the case in the initial problem.

Summing up, the results reported from our methods show the following:

- They manage to find solutions at every given problem – i.e. the do manage to regulate flights crossing an operational space in a day so as to resolve all hotspots
- They manage to find solutions effectively. They do converge to solutions quite fast, few rounds after exploration, in most of the cases.
- They manage to reduce the average delay for the regulated flights considerably, compared to the average delay for the regulated flights reported by CFMU. The same holds for the average delay considering all flights.

Chapter 6

Conclusions and Future work

In this thesis, we tackled the problem of demand-capacity balancing in the Air Traffic domain. We used collaborative Reinforcement Learning methods during the pre-tactical planning phase in order to eliminate the hotspots by imposing delays to the flights composing them. Also, while solving the problem, we tried to minimize the mean delay of the flights along with the expected cost. The most crucial point of our work, is the formulation of our problem as a coordination problem using the collaborative multi-agent MDP framework where every aircraft is an agent. Moreover, we provided a novel reward function that takes into consideration the imposed delays, the expected cost for every minute of delay and the agents' quota of participation in the congested sectors they cross.

We presented three different multi-agent Reinforcement Learning algorithms: the Independent Learners, the Edge-Based Collaborative Reinforcement Learners and the Agent-Based Collaborative Reinforcement Learners. All of them, solved the problems and eliminated the hotspots at each case, with results -in terms of mean delay, cost and delayed flights- being significantly promising. Most importantly, our methods were used on real-world scenarios with thousands of flights in a complex environment and managed to efficiently solve the problem.

Experimental results in real-world problems show the potential of the proposed methods, in terms of efficiency (i.e. speed of convergence) and efficacy (in terms of quality of solutions achieved). In few words, collaborative multiagent RL methods

are promising to resolving real-world complex DCB problems in ATM, compared to independent RL learners used in most of works related to congestions resolution.

Concerning the future work, we have in mind a lot of improvements and our foal is to extend our current work in even more challenging problems. Some interesting extension might be:

- Devising new reward functions. As we already concluded, the reward function is of utmost importance and the free parameters that it involves must be chosen carefully after extensive experimentation. To this end, we will also try out new reward schemes from the literature.
- Resolving problems occurring at the tactical stage. An interesting alternative to our problem would involve solving the problem while the aircrafts have already departed. This might allow us to use re-routing algorithms in order to solve the problem before congestions occurred in the air.
- Extend our methods using Deep Learning. Dealing with continuous action spaces, as the one discussed in the previous extension, we need to ensure the scalability of our methods.
- Test our methods in other traffic fields. We aim to generalize our approach in other traffic domains that involve situations where limited resources need to be used by multiple agents.

Appendix A

References

[1] Jelle R. Kok and Nikos Vlassis. *Collaborative multiagent reinforcement learning by payoff propagation*. J. Mach. Learn. Res., 7:1789–1828, December 2006.

[2] Theocharis Kravaris, George A. Vouros, Christos Spatharis, Konstantinos Blekas, Georgios Chalkiadakis, and Jose Manuel Cordero Garcia. *Learning policies for resolving demand-capacity imbalances during pre-tactical air traffic management*. MATES 2017.

[3] N. Vlassis. *A concise introduction to multiagent systems and distributed AI*, 2003.

[4] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[5] Carlos Ernesto Guestrin. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford, CA, USA, 2003. AAI3104233.

[6] Carlos Guestrin Guestrin, Michail Lagoudakis, and Ronald Parr. *Coordinated reinforcement learning*. In Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning, pages 227–234, 2002.

[7] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[8] Ana L. C. Bazzan, Joachim Wahle, and Franziska Klügl. 1999. *Agents in Traffic Modelling - From Reactive to Social Behaviour*. In KI-99: Advances in Artificial Intelligence, 23rd Annual German Conference on Artificial Intelligence, Bonn, Germany, September 13-15, 1999, Proceedings. 303–306.

[9] K. Dresner and P. Stone. 2004. *Multiagent traffic management: A reservation-based intersection control mechanism*. In Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS '04). 530–537.

[10] Adrian K Agogino and Kagan Tumer. 2012.*A multiagent approach to managing air traffic flow*. Autonomous Agents and Multi-Agent Systems 24, 1 (2012), 1–25.

[11] Kleanthis Malialis, Sam Devlin, and Daniel Kudenko. 2016. *Resource Abstraction for Reinforcement Learning in Multiagent Congestion Problems*. In Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '16). 503–511.

[12] Baraa Munqith Albaker and Nasrudin Abd Rahim. 2010. *Unmanned aircraft collision avoidance system using cooperative agent-based negotiation approach*.Int. J. Simulation, Syst. Sci. Technol 11, 4 (2010), 1–8.

[13] David Sislak, Přemysl Volf, and Michal Pechoucek. 2011. *Agent-based cooperative decentralized airplane-collision avoidance*. IEEE Transactions on Intelligent Transportation Systems 12, 1 (2011), 36–46.

[14] R. W. Rosenthal. 1973. *A Class of Games Possessing Pure-Strategy Nash Equilibria*. International Journal of Game Theory 2 (1973), 65–67.

[15] C. Meyers. 2006. *Network flow problems and congestion games: complexity approximation approximation results*. Ph.D. Dissertation. Cambridge, MA, USA.

[16] I. Milchtaich. 2004. *Social Optimality and Cooperation in Nonatomic Congestion Games*. Journal of Economic Theory 114 (2004), 56–87.

[17] Michal Penn, Maria Polukarov, and Moshe Tennenholtz. 2005. *Congestion games with failures*. In Proceedings 6th ACM Conference on Electronic Commerce (EC-2005), Vancouver, BC, Canada, June 5-8, 2005. 259–268.

[18] Andrew J Cook and Graham Tanner. 2015. European airline delay cost values. (2015). http://www.eurocontrol.int/publications/european-airline-delay-cost-reference-values.

Author's Publications

[1] Theocharis Kravaris, George A. Vouros, Christos Spatharis, Konstantinos Blekas, Georgios Chalkiadakis, Jose Manuel Cordero Garcia: **Learning Policies for Resolving Demand-Capacity Imbalances During Pre-tactical Air Traffic Management**. MATES 2017: 238-255

[2] Christos Spatharis, Theocharis Kravaris, George A. Vouros, Konstantinos Blekas, Georgios Chalkiadakis, Jose Manuel Cordero Garcia, Esther Calvo Fernandez: **Multiagent Reinforcement Learning Methods to Resolve Demand Capacity Balance Problems**. SETN 2018

[3] C. Spatharis, T. Kravaris, K. Blekas and G. A. Vouros. **Multiagent Reinforcement Learning Methods for Resolving Demand - Capacity Imbalances**. 37th AIAA/IEEE Digital Avionics Systems Conference (DASC), Sept. 2018.

Short Biography

Christos Spatharis was born in Ioannina, Greece in 1992. He received his BSc degree from the Department of Computer Science & Engineering of University of Ioannina in 2016. In 2016 he became an MSc student at the same institution under the supervision of prof. Konstantinos Blekas. His academic interests are in the area of Machine Learning, Robotics and Computer Vision.