NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS

GRADUATE PROGRAM
INFORMATION AND DATA MANAGEMENT

POSTGRADUATE THESIS

Temporal Abstraction for Hierarchical Reinforcement Learning in Air Traffic Management

Alevizos A. Bastas

Supervisors: **Panagiotis Stamatopoulos**, Assistant Professor, NKUA
George Vouros, Professor, UPRC

ATHENS

FEBRUARY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΔΙΑΧΕΙΡΙΣΗ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΔΕΔΟΜΕΝΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρονική Αφαίρεση για Ιεραρχική Ενισχυτική Μάθηση στη Διαχείριση Εναέριας Κυκλοφορίας

Αλεβίζος Α. Μπάστας

Επιβλέποντες: **Παναγιώτης Σταματόπουλος**, Επίκουρος Καθηγητής, ΕΚΠΑ
Γεώργιος Βούρος, Καθηγητής, ΠΑΠΕΙ

ΑΘΗΝΑ

Φεβρουάριος 2019

POSTGRADUATE THESIS

Temporal Abstraction for Hierarchical Reinforcement Learning in Air Traffic Management

Alevizos A. Bastas
A.M: M1601

Supervisors: Panagiotis Stamatopoulos, Assistant Professor, NKUA
George Vouros, Professor, UPRC

February 2019

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρονική Αφαίρεση για Ιεραρχική Ενισχυτική Μάθηση στη Διαχείριση Εναέριας Κυκλοφορίας

Αλεβίζος Α. Μπάστας

Α.Μ: M1601

Επιβλέποντες: Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής, ΕΚΠΑ
Γεώργιος Βούρος, Καθηγητής, ΠΑΠΕΙ

Φεβρουάριος 2019

ABSTRACT

In this thesis, we report on the efficiency and effectiveness of hierarchical multiagent reinforcement learning methods (H-MARL), where the decisions are formed at various abstraction levels, for the computation of joint policies to resolve congestion problems in the Air Traffic Management (ATM) domain extending the work presented in [1]. Specifically, we aim to resolve cases where demand of airspace use exceeds capacity (demand-capacity problems), via imposing ground delays to flights at the pre-tactical stage of operations. Agents, representing flights, have limited information about others' payoffs and preferences, and need to coordinate among themselves to achieve their tasks while adhering to operational constraints. Specifically, we formalize the problem as a hierarchical multiagent Markov Decision Process (MA-MDP) and we present hierarchical multiagent reinforcement learning methods that allow agents to form own policies in coordination with others. We explore the effectiveness of alternative hierarchical multiagent reinforcement learning methods and alternative schemes of transferring experience between levels, towards reducing delays and number of delayed flights. Extensive experimental study on a real-world case, shows the potential of the proposed approaches in increasing the computational efficiency and the quality of solutions in resolving the demand-capacity balance problems also in comparison to other state of the art flat models and methods [1].

SUBJECT AREA: Artificial Intelligence, Machine Learning, Reinforcement Learning

KEYWORDS: Air Traffic Management, Demand Capacity Balance, Collaborative Multiagent Reinforcement Learning, Q-Learning, Hierarchical Reinforcement Learning, State Abstraction, Action Abstraction

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία, παρουσιάζουμε την αποδοτικότητα και αποτελεσματικότητα ιεραρχικών πολυπρακτορικών μεθόδων ενισχυτικής μάθησης, όπου οι αποφάσεις διαμορφώνονται σε διάφορα επίπεδα αφαίρεσης, για τη σύνθεση κοινών πολιτικών με σκοπό την επίλυση προβλημάτων συμφόρησης στη διαχείριση εναέριας κυκλοφορίας, επεκτείνοντας τη δουλειά που παρουσιάζεται στο [1]. Συγκεκριμένα, στοχεύουμε στην επίλυση περιπτώσεων όπου η ζήτηση για χρήση του εναέριου χώρου υπερβαίνει την προσφορά (Demand - Capacity problems), μέσω της επιβολής καθυστερήσεων στις πτήσεις πριν το τακτικό επίπεδο των διαδικασιών. Οι πράκτορες, οι οποίοι αντιπροσωπεύουν πτήσεις, έχουν περιορισμένη πληροφόρηση σχετικά με τις αμοιβές και τις προτιμήσεις των υπόλοιπων πρακτόρων και χρειάζεται να συντονιστούν μεταξύ τους για να επιτύχουν τα καθήκοντά τους ενώ τηρούν λειτουργικούς περιορισμούς. Συγκεκριμένα μοντελοποιούμε το πρόβλημα σαν μια ιεραρχική πολυπρακτορική Μαρκοβιανή Διαδικασία Αποφάσεων και παρουσιάζουμε ιεραρχικές πολυπρακτορικές μεθόδους ενισχυτικής μάθησης που επιτρέπουν στους πράκτορες να διομορφώνουν τις δικές τους πολιτικές σε συντονισμό με τους υπόλοιπους. Εξερευνούμε την αποτελεσματικότητα εναλλακτικών ιεραρχικών μεθόδων ενισχυτικής μάθησης και εναλλακτικών σχημάτων μεταφοράς της εμπειρίας μεταξύ επιπέδων με σκοπό τη μείωση των καθυστερήσεων και του αριθμού των πτήσεων στις οποίες έχει επιβληθεί καθυστέρηση. Εκτενής πειραματική μελέτη σε μία πραγματική περίπτωση, δείχνει τη δυνατότητα των προτεινόμενων προσεγγίσεων να βελτιώσουν την υπολογιστική αποτελεσματικότητα και την ποιότητα των λύσεων για την επίλυση προβλημάτων εξισορρόπησης ζήτησης-προσφοράς (Demand - Capacity problems) σε σύγκριση και με άλλα state of the art μοντέλα και μεθόδους [1].

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη, Μηχανική Μάθηση, Ενισχυτική Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Διαχείριση Εναέριας Κυκλοφορίας, Ισορροπία Ζήτησης Χωρητικότητας, Συνεργατική Πολυπρακτορική Ενισχυτική Μάθηση, Μάθηση Q, Ιεραρχική Ενισχυτική Μάθηση, Αφαίρεση Καταστάσεων, Αφαίρεση Ενεργειών

ACKNOWLEDGEMENTS

I would like to acknowledge the supervisor Assistant Professor Panagiotis Stamatopoulos of the Department of Informatics and Telecommunications at National and Kapodistrian University of Athens. His undergraduate and graduate courses were a great influence to me. I would also like to acknowledge Professor George Vouros of the Department of Digital Systems at University of Piraeus for his invaluable guidance and contribution. I am also thankful to the Phd student Theocharis Kravaris for his assistant and cooperation. Finally, I would like to acknowledge the DART Project[1], that supported this thesis.

[1]DART has received funding from the SESAR Joint Undertaking under grant agreement No 699299 under European Unions Horizon 2020 research and innovation programme. DART project's website: http://www.dart-research.eu

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Complex problems with large state spaces and hierarchical structure are common in the fields of AI and Reinforcement Learning. An effective way to combat the combinatorial explosion that arises with such problems has been proven to be state and action abstraction (or aggregation). Such techniques treat groups of states as a unit or complex actions as primitive ones and thus effectively reduce the search space. By doing so we can speed up planning and while producing qualitative solutions in limited time.

Specifically, for problems with complex hierarchical structure, hierarchical reinforcement learning methods have been proposed that exploit temporal abstractions by invoking the execution of temporally extended subtasks. In many cases these subtasks follow a policy which is different from the one that the main task follows. Additionally, such methods may exploit state abstraction in order to ignore information that is irrelevant to the currently operating subtask, while preserving all the important aspects, necessary to still being able to solve the problem.

On the other hand, congestion problems, modelling situations where resources of a limited capacity have to be shared by multiple agents simultaneously, are ever present in the modern world. Most notably, congestion problems appear regularly in various traffic domains. It is of no surprise that they have drawn much attention in the AI and autonomous agents research for at least two decades now [3] and have been the focus of game theoretic models for much longer [4].

In the air-traffic management (ATM) domain, in which this thesis focuses, congestion problems are complex problems that arise naturally whenever demand of airspace use exceeds capacity, resulting to hotspots or to Demand - Capacity Balance (DCB) problem(s).

Against this background, this thesis formalises the DCB problem as a hierarchical multiagent Markov Decision Process (MA-MDP) at various abstraction levels, where agents, representing flights, aim to coordinate their joint actions with respect to own preferences and operational constraints on the use of airspace. We consider planning air traffic management operations at the pre-tactical phase: Given air sectors' limited capacity, one of the major issues is to minimise ground delay costs, while ensuring efficient utilisation of airspace. As part of the formulation, we use the reward function devised in [1] that considers agents' contribution to hotspots, ground delays, and implied cost when agents deviate from their schedule.

The multiagent problem specification and the proposed multiagent reinforcement learning (H-MARL) methods allow to impose ground delays to individual flights, always considering the joint effects of imposed delays to the evolution of airspace demand. While agents represent flights, their environment comprises the airspace and other flights aggregated into "traffic". This is in contrast to regulating in a first-come-first-regulated basis - as it is the case today in ATM.

Specifically, the current ATM system worldwide is based on a time-based operations paradigm. As the system deals with an increasingly large number of flights, aiming to making efficient use of resources, this paradigm implies some limitations to the ATM system, often leading to DCB issues. These limitations are resolved via airspace management or flow management solutions, including regulations imposed to airspace compartments (sectors), which are transferred to flights crossing these compartments in a first-come-first-regulated basis, generating delays (and costs) for the entire system. These demand-capacity imbalances are difficult to be predicted in "pre-tactical" phase of operations (i.e. from several days to few hours before operations), as the existing ATM information is not accurate enough during this phase: Providing methods for assessment of delays at the pre-tactical phase, and contributing to decision-making processes towards the minimization of delays is crucial for operational and economic reasons.

Indeed, considering operational constraints for the joint performance of the trajectories, the proposed hierarchical multiagent methods support each individual agent to reconcile conflicting options (e.g. resolving hotspots without any delay) jointly with others and to jointly decide about individual policies on delays, while possessing no information about the preferences and payoffs of others.

Our goal is to resolve DCB problems, while reducing the average ground delay per flight w.r.t. the number of flights, compared to state of the art approaches where no hierarchical structures are being used. In doing so, we aim to distribute ground delays among flights without penalizing a small number of them and utilize efficiently the airspace so as to have an even distribution of demand to sectors in different periods.

Therefore, we consider only ground delays and subsequently we succinctly call these "delays". The proposed H-MARL methods are evaluated in real-world DCB problem cases, each one comprising flight plans for a specific day above Spain. The data sources used to produce those cases include real-world operational data regarding flight plans per day of operation, data regarding sector configurations at any given time, and reference values for the cost of strategic delay to European airlines, currently used by SESAR 2020 Industrial Research [5].
An initial observation from the application of the H-MARL methods is that they, quite effectively, manage to provide solutions to DCB problems, imposing delays that result to zero hotspots. By utilizing a variety of different metrics (such as the number of flights with delay, average delay per flight, and delay distributions) we provide evidence on the potential of the proposed methods to produce qualitative solutions: Indeed, results are quite significant since, the average delay per flight (i.e. the ratio of summing all delays to the total number of flights) is reduced considerably compared to the solutions provided by state of the art approaches where no hierarchical structures are being used, while a small percentage of flights have delay more than half an hour, and only a small percentage of flights get delay.

We envisage the work laid out in [1] and in this thesis to be seen as a first step towards devising multiagent methods for deciding on delay policies for correlated aircraft trajectories at the pre-tactical phase, answering the call of ATM domain for a transition to a Trajectory Based Operations (TBO) paradigm (SESAR in Europe[1] and Next Gen in the US[2]).

[1] https://ec.europa.eu/transport/modes/air/sesar_en
[2] https://www.faa.gov/nextgen/

The contributions made in this work are as follows:

- The Demand-Capacity Balance problem is formulated as a hierarchical multiagent Markov Decision Process (MA-MDP) at multiple levels of abstraction, resulting to hierarchical models.
- A generic hierarchical multiagent reinforcement learning (H-MARL) method is devised, able to operate at multiple levels of abstraction: This is instantiated to alternative H-MARL methods exploiting different levels of abstraction of the action-state space regarding the DCB problem.
- Hierarchical multiagent reinforcement learning methods are evaluated in relation to other state of the art flat models and methods [1]. Because the hierarchical multiagent reinforcement learning methods operate on abstract levels, increased computational efficiency and production of more qualitative solutions in limited time w.r.t. the average delay and the number of delayed flights is expected, compared to the flat methods.
- All methods are evaluated in a real-world case comprising large number of flights in busy days above Spain.

The structure of this thesis is as follows: Section 2 provides a specification for the DCB problem and introduces terminology from the ATM domain. Section 3 presents related work, regarding state aggregation and hierarchical reinforcement learning. Section 4 presents a generic hierarchical framework for multiagent reinforcement learning at different levels of state-action space abstractions. Section 5 specifies the reward function used. Section 6 presents Independent Q-Learners, Independent Hierarchical Q-Learners and the proposed methods instantiating the generic hierarchical framework. Section 7 presents evaluation cases and results. Finally, section 8 concludes the thesis outlining future research directions.

2. PROBLEM SPECIFICATION

As already pointed out in the introductory part of this thesis, the current Air Traffic Management (ATM) system leads to demand-capacity imbalances.

With the aim of overcoming ATM system drawbacks, different initiatives, notably SESAR in Europe and Next Gen in the US have promoted the transformation of the current ATM paradigm towards a new, trajectory-based operations (TBO) one: In the future ATM system, the trajectory becomes the cornerstone upon which all the ATM capabilities will rely on. The trajectory life cycle describes the different stages from the trajectory planning, negotiation and agreement, to the trajectory execution, amendment and modification.

This life cycle requires collaborative planning processes, before operations: The envisioned advanced decision support tools will exploit trajectory information to provide optimised services to all ATM stakeholders. The proposed transformation requires high-fidelity aircraft trajectory prediction capabilities and/or adequate and sufficient information regarding operations at the pre-tactical stage, supporting the trajectory life cycle at all stages efficiently.

In addition to these capabilities, considering flight trajectories in isolation from the overall ATM system may lead to inefficiencies to trajectory planning (due for instance to conflict resolution) and huge inaccuracies to assessing trajectory execution. Accounting for network effects and their implications on the joint execution of individual flights requires considering interactions among trajectories, in conjunction to considering operational conditions that influence any flight. Being able to devise methods that capture aspects of that complexity and take the relevant information into account, would greatly improve our planning and decision-making abilities. Towards this goal, our specific aim is to assess ground delays to be imposed to trajectories towards resolving DCB problems, i.e. cases where imbalances regarding the demand of airspace use and the provided airspace capacity do occur.

More specifically, the DCB problem (or DCB process) considers two important types of objects in the ATM system: aircraft trajectories and airspace sectors.

Sectors are air volumes segregating the airspace, each defined as a group of airblocks. Airblocks are specified by a geometry (the perimeter of their projection on earth) and their lowest and highest altitudes. Airspace sectorization may be done in different ways, depending on sectors' configuration, determining the active (open) sectors. 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 flights.

The capacity of sectors is of utmost importance: this quantity determines the maximum number of flights flying within a sector during any time period of specific duration (typically,

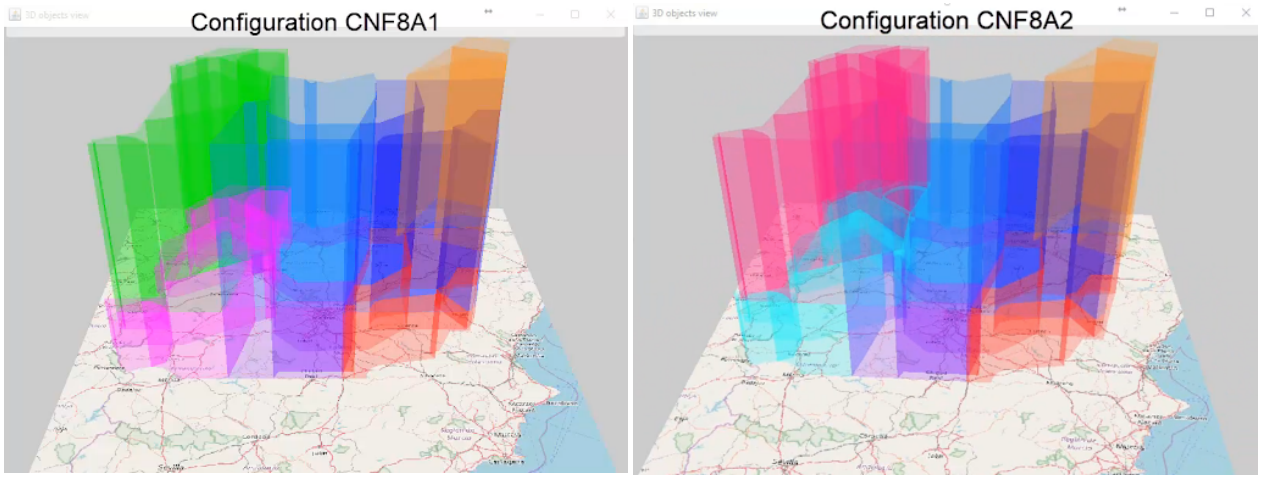


Figure 2.1: Configurations of sectors in the Spanish airspace. Colours are for distinguishing between sectors. Illustrations have been created using the V-Analytics platform [2].

in 60' periods).

The demand for each sector is the quantity that specifies the number of flights that co-occur during a time period within a sector. The duration of any such period is equal to the duration of the period used for defining capacity. Demand must not exceed sector capacity for any time period. There are different types of measures to monitor the demand evolution, with the most common ones being the Hourly Entry Count and the Occupancy Count. In this work we consider Hourly Entry Count, as this is the one used by the Network Manager (NM) at the pre-tactical stage.

The Hourly Entry Count (HEC) for a given sector is defined as the number of flights entering the sector during a time period, referred to as an Entry Counting Period (or simply, counting period). HEC is defined to give a "picture" of the entry traffic, taken at every time "step" value along a period of fixed duration: The step value defines the time difference between two consecutive counting periods. For example, for a 20 minutes step value and a 60 minutes duration value, entry counts correspond to pictures taken every 20 minutes, over a total duration of 60 minutes. Aircraft trajectories are series of spatio-temporal points of the generic form $(long_i, lat_i, alt_i, t_i)$, denoting the longitude, latitude and altitude, respectively, of the aircraft at a specific time point ti . Casting them into a DCB resolution setting, trajectories may be seen as time series of events specifying the entry and exit locations (coordinates + flight levels) and the entry and exit times for the sectors crossed, or the time that the flight will fly over specific sectors. Thus, given that each trajectory is a sequence of timed positions in airspace, this sequence can be exploited to compute the series of sectors that each flight crosses, together with the entry and exit time for each of these sectors. For the first (last) sector of the flight, where the departure (resp. arrival) airport resides, the entry (resp. exit) time is the departure (resp. arrival) time. For flights that cross the airspace but do not depart and/or arrive in any of the sectors of the airspace of interest, we only consider the entry and exit time for sectors within that airspace.

Therefore, we consider a finite set of discrete air sectors $\mathbf{R} = \{R1, R2, ...\}$ segregating the airspace.

As already pointed out, sectors are related to a set of operational constraints associated to

their capacity, whose violation results to demand-capacity imbalances (congestion problems): These are cases where $D_{R,p} > C_R$, where *p* is a counting period of pre-defined duration *d*, $D_{R,p}$ is the demand for sector *R* during counting period *p*, and C_R is the capacity of the sector for any period of duration *d* equal to the counting period duration. These cases are capacity violation or demand-capacity imbalance cases, resulting to congestion events, or capacity excess events, mentioned as hotspots.

Thus, a trajectory *T* in **T** is a time series of elements of the form:
$T = \{(R_1, entry_1, exit_1)....(R_m, entry_m, exit_m)\}$, where R_l , $l = 1, ..., m$ is a sector in the airspace. This information per trajectory suffices to measure the demand for each of the sectors *R*∈**R** in the airspace, in any counting period *p*.

Specifically, the demand in sector *R* in period *p* is $D_{R,p} = |\mathbf{T}_{R,p}|$, i.e. the number of trajectories in $\mathbf{T}_{R,p}$, where $\mathbf{T}_{R,p} = \{T \in \mathbf{T} | T = (..., (R, entry_t, exit_t), ...)$, and the temporal interval $[entry_t, exit_t]$ overlaps with period $p\}$.
In case of hotspots, trajectories requiring the use of the sector *R* at the same period *p*, i.e. trajectories in $\mathbf{T}_{R,p}$, are defined to be interacting trajectories for *p* and *R*.

The overall objective of the DCB process at any phase of operations (Strategic, Planning and Tactical Phase) is to optimise traffic flows according to air traffic control capacity while enabling airlines to operate safe and efficient flights. Planning operations start as early as possible - sometimes more than one year in advance. Given that the objective is to protect air traffic control service of overload [6], stakeholders involved in the process always looking for optimum traffic flow through a correct use of the capacity, guaranteed: safety, better use of capacity, equity/fairness among flights and airlines, information sharing among stakeholders and fluency.

In this work we consider the demand-capacity process during the pre-tactical phase, assuming a trajectory-based operations environment with appropriate processes in place, enabling an enhanced accuracy of pre-tactical flight information, which is crucial to detect imbalances and impose regulations[1] . Pre-tactical flow management is applied at least six days prior to the day of operations and consists of planning and coordination activities.

In this operational context we consider an agent A_i to be the aircraft performing a specific flight trajectory (simply, trajectory), in a specific date and time. Thus, we consider that agents and trajectories coincide, and we may interchangeably speak of agents A_i , trajectories T_i, flights, or agents A_i executing trajectories T_i . Agents, as it will be specified, have own interests and preferences, although they are assumed collaborative, and take autonomous decisions on resolving hotspots: It must be noted that agents do not have communication and monitoring restrictions, given that hotspots are resolved at the pre-tactical phase, rather than during operation.

To resolve hotspots, agents have several degrees of freedom: They may either change their trajectory, cross sectors other than the congested ones, or change the schedule of crossing sectors in terms of changing the entry and exit time for each of the crossed sectors. In this thesis we consider only changing the schedule of crossing sectors by impos-

[1] Today, accuracy of information is limited, and the predictability of the complex ATM system is low: This explains also why many demand-capacity imbalances today are left to be solved at the operational (tactical) phase.

ing ground delays: i.e., shifting the whole trajectory by a specific amount of time.

Now, the problem is about agents to execute their trajectories jointly, in an efficient and safe way, w.r.t. resources operational constraints.

Specifically, in the DCB problem the goal is to

- Resolve all demand-capacity imbalances, providing a solution with zero hotspots, in conjunction to
- minimizing the average delay per flight (ratio of total delay to the number of flights); so as to
- distribute delays among flights without penalising a small number of them

To resolve a hotspot occurring in period p and sector R, a subset of interacting trajectories in $\mathbf{T}_{R,p}$ must be delayed. It must be noted that agents have conflicting preferences towards resolving hotspots, since they prefer to impose the smallest delay possible (preferably none) to their own trajectory, or they may have different requirements on the maximum delay to be imposed to their flights (reducing costs), while also they do need to jointly execute their planned trajectories safely and efficiently.

Clearly, imposing delays to trajectories may propagate hotspots to a subsequent time period for the same and/or other sectors crossed. Also, the sets of interacting trajectories in different periods and sectors may change. This can be done in many different ways when imposing delays to flights, resulting to a dynamic setting for any of the agents, where the sets of interacting trajectories do change according to agents' decisions.

Agents executing interacting trajectories and contributing to hotspots are considered to be "peers" given that they may decide jointly on their delays: The decision of one of them directly affects the others. This implies that agents form "neighbourhoods" of peers. Such neighbourhoods provide a way to take advantage of the spatial and temporal sparsity of the problem: For instance, a flight crossing the northwest part of Spain in the morning, will never interact in any direct manner with a flight crossing the southeast part of the Iberian Peninsula at any time, or with an evening flight that crosses the northwest part of Spain. However, as mentioned above, these neighbourhoods have to be dynamically updated when delays are imposed to flights, given that trajectories that did not interact prior to any delay may result to be interacting when delays are imposed, and vice-versa.

The *society of agents* **(A, E)** is modelled as a *coordination graph* [7] [8] with one vertex per agent A_i in **A** and any edge (A_i, A_j) in **E** connecting agents with interacting trajectories in **T**. The set of interacting agents, and thus edges, are dynamically updated when new interacting pairs of trajectories appear. $\mathbf{N}(A_i)$ denotes the *neighbourhood* of agent A_i, i.e. the set of agents interacting with agent A_i in any period *p* and sector *R*, including also itself. These are the peers of A_i [1].

The options available in the inventory of any agent A_i for contributing to the resolution of hotspots may differ between agents: These, for agent A_i are in $\mathbf{D}_i = 0, 1, 2, ..., MaxDelay_i$.

[1] It must be pointed out that neighbourhoods have a temporal dimension, since they dynamically change as agents get delays: We ignore this important detail for simplifying the presentation, but this dynamic adjustment is something that it is considered by all proposed H-MARL methods.

We consider that these may be ordered by the preference of agent A_i to any such option, according to the function $\gamma(i) : \mathbf{D}_i \rightarrow \mathbf{R}$. We do not assume that agents in $\mathbf{A} - \{A_i\}$ have any information about $\gamma(i)$. This represents the situation where airlines set own options and preferences for delays even in different individual own flights, depending on operational circumstances, goals and constraints. We expect that the order of preferences should be decreasing from 0 to $MaxDelay_i$, although, with a different pace/degree for different agents.

Problem statement: Considering any two peers A_i and A_j in the society **(A, E)**, with A_j in $\mathbf{N}(A_i) - A_i$, these agents must select among the sets of available options D_i and D_j respectively, so as to increase their expected payoff w.r.t. their preferences on options $\gamma(i)$ and $\gamma(j)$ to resolve the DCB problem: A solution consists of assignment of delays to flights, such that all imbalances are resolved, resulting to zero hotspots, minimizing flights' delays.

This problem specification emphasises on the following problem aspects:

- Agents (i.e. individual flights) need to coordinate their strategies (i.e. chosen options to delays) to execute their trajectories jointly with others, considering traffic, w.r.t. their preferences and operational constraints;
- Agents (i.e. individual flights) need to jointly 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 do not know in advance the interacting trajectories that emerge due to own decisions and decisions of others, and of course they do not know whether these interactions result to new hotspots);
- Agents' preferences on the options available may vary depending on the trajectory performed, and are kept private;
- There are multiple and interdependent hotspots that occur in the total period **H** and agents have to resolve them jointly;
- The setting is highly dynamic given that hotspots change while agents choose their delay strategies, in ways that are unpredictable for agents.

It must be pointed out that this thesis does not consider agents' preferences on delays, although their incorporation in the proposed methods is straightforward, as also shown in [1].

2.1. Data Sources

This subsection presents the data sources used for the experimental evaluation of the algorithms. I would like to acknowledge the DART project for providing the data sources and also the Phd student Theocharis Kravaris for processing these data sources and producing scenario files [9] covering the flight plans per day of the following format:

- First line: Comma delimited values of the following quantities in the presented order *number of sectors, number of flights,size of period, period step, global maximum delay, number of periods, alpha parameter (regarding the reward function), gamma parameter (regarding the reward function)*
- Second line: Comma delimited capacities of sectors

- The rest of the file contains for each flight the flight id and for each possible delay the flight plan, consisting of the time of the take off, the sectors crossed during the flight and the duration the aircraft needs to cross each sector. The last three values of each line consist of the model of the aircraft, a boolean value indicating if the flight is commercial and the local maximum delay of the flight. Therefore the format of the predescribed lines is the following:

$$
\begin{aligned}
&flight\ id\\
&, d_0, take\ off_0\\
&, sector_0\ sector_1\ ...\ sectorn_0\\
&, duration_0\ duration_1\ ...\ duration_{n1}\\
&, d_1, take\ off_1\\
&, sector_0\ sector_1\ ...\ sector_n\\
&, duration_0\ duration_1\ ...\ duration_n\\
&, ...\\
&, d_m, take\ off_m\\
&, sector_0\ sector_1\ ...\ sector_n\\
&, duration_0\ duration_1\ ...\ duration_{nm}\\
&, model, commercial, local\ maximum\ delay.
\end{aligned}
$$

  Where n_i denotes the number of sectors of the corresponding flightplan and m denotes the number of flight plans for the corresponding flight and for each possible delay.

3. RELATED WORK

In order to increase the effectiveness of reinforcement learning methods in complex problems, abstraction operators are often applied in states, actions or in both. Through abstraction, a deeper exploration of state / action space could be achieved, as well as a qualitative improvement of the decision process with much better generalization capabilities.

State abstraction (or state aggregation) has been extensively studied in the fields of artificial intelligence and operations research. The idea behind state abstraction is that, instead of working at the ground (original) state space, the decision maker usually finds solutions in the abstract state space much faster by treating groups of states as a unit, ignoring irrelevant state information. This manages to reduce the complexity of a problem by filtering out properties or by aggregating features, while preserving all the important aspects, necessary to still being able to solve the problem. There are two forms of abstraction within the reinforcement learning framework:

a) State abstraction, which groups together states with similar environmental configurations and associated behavior [2] [10]: These approaches may assume human supervision (e.g. [11], [12]), or they may support discovering appropriate state abstractions automatically (e.g. as in [13] [14] [15]).

b) Temporal abstraction approaches [16] [17] which mainly add to the ground (original) action space abstraction layers of temporally extended actions: Representing actions flexibly at multiple levels of temporal abstraction has the potential to speed planning and optimization effort. A significant work in temporal abstraction can be found in [16], where the options framework was proposed, involving abstractions over the space of actions. At each step, the agent chooses either a one-step action or a “multi-step” action policy (option). Each option defines a policy over actions and can be terminated according to a stochastic function. This defines a hierarchical structure of high-level policies that invoke options solving sub-tasks. The MAXQ framework [18], decomposes the value function of an MDP into combinations of value functions of smaller constituent MDPs, while the approach proposed in [19] combines hierarchies with short-term memory to handle partial observations. Automatically discovering subtasks and hierarchies, as for example in [14], [20], adds further challenges and possibilities. Also, other methods have been proposed to learn options in real-time by using various reward functions [20], or by composing existing options [21]. Value functions have also been generalized to consider goals along with states [22]. Another approach towards temporal abstraction can be found in [17] that involves simultaneously learning options and a control policy to compose options in a deep reinforcement learning setting. This approach does not use separate Q-functions for each option, but instead treats options as part of the input. A recent hierarchical neural network framework has been proposed in [23] that supports temporally extended action sequences with different levels of abstraction.

Although our methods are hierarchical imposing a temporal abstraction, they differ from approaches like the MAXQ and options frameworks which exploit a hierarchy of temporally extended subtasks. We rather aggregate states' and actions' temporal dimension in a direct way, e.g. by treating temporal instants as an aggregated temporal "point". With the term "hierarchical" we refer to the hierarchy of such temporal abstractions that our methods impose to both states and actions.

Specifically, while we apply state aggregation at the temporal dimension of states, one of the proposed methods (described at subsection 6.2.1) also uses abstraction at the temporal dimension of actions. Thus our work is closer to the first of the aforementioned classes of state abstraction methods that group together states with similar environmental configurations and associated behavior. Also, our methods do not discover state abstractions automatically, but these, together with the hierarchical structure imposed, are specified by humans.

4. HIERARCHICAL MULTIAGENT REINFORCEMENT LEARNING

In this section we propose a hierarchical reinforcement learning framework for learning at different state-action space abstractions: the abstract levels abstract the action-state space in its temporal dimension, and the “ground” level comprises the original state/action space. As said in the previous section, although the abstraction used is in the temporal dimension of actions and states, the approach is quite different from a temporal abstraction approach where temporally extended actions are incorporated at abstraction levels.

Next we formulate the generic hierarchical MDP framework with multiple levels of abstraction: The agent should learn a policy at both abstract and ground levels, leveraging knowledge acquired at the abstract levels to improve performance while moving towards the ground (original) level of abstraction. The proposed framework is described below in detail, after formulating the hierarchical MDP.

4.1. The Hierarchical MDP Framework

According to the problem specification, and using the model of collaborative multiagent MDP framework [24], we formulate the problem as a Hierarchical MDP comprising the following constituents:

- The *society of agents* **(A, E)**, as specified above.
- A set of *abstraction levels* $\{1, ...h\}$.
- The *time step* at level *L*, denoted ts_L, where *L* is in $\{1, ...h\}$: $t_L = t_0^L, t_1^L, t_2^L, t_3^L, ..., t_{max(L)}^L$, where $t_{max(L)}^L - t_0^L =$ **H** and $t_{i+1}^L - t_i^L = ts_L, i = 0, ...max(L) - 1$. Notice that the number of time instants per level depends on the time granularity for that level, given that **H** is constant at all levels. The size of the timestep determines the amount of time instances that pass until an action completes. Thus the agents make decisions every ts_L time points. When we present the local strategy per agent A_i, we additionally define action abstraction per level *L* based on the timestep ts_L.
- The *abstraction step* at level *L*, denoted as_L, where *L* is in $\{1, ...h\}$, defines the amount of time instants that correspond to the same abstract state. We utilize the abstraction step at level *L* to impose aggregation on states. We will explore methods that use unitary timestep (i.e $ts_L = 1$) and thus do not impose action abstraction but they exploit state aggregation by using an abstraction step greater than 1. We consider that it holds that $as_L \geq ts_L$ so that all states observed by the agent ,corresponding to actions taken every ts_L time points, that belong in the same as_L interval are aggregated together.

- A *local state* per agent A_i at time *t* and level of abstraction *L*, comprising state variables that correspond to (a) the delay imposed to the trajectory T_i executed by A_i, ranging to the sets of options assumed by A_i, i.e. in $\mathbf{D}_i^L = 0, \ldots MaxDelay_i$, and (b) the number of hotspots in which A_i is involved in (for any of the sectors). Such a local state is denoted by $s_i^{L,t}$. It must be noticed that the set of options for any agent depends on the time step considered at each level. This prunes the states the agent will explore at the abstract level *L*, as states with delay that is not a multiple of the timestep are unreachable. Additionally, we aggregate original (ground) states with delays between $i * as_L$ and $(i+1) * as_L$, $i \in N$, to the same abstract state.
  The *joint state* $\mathbf{s}_{A_g}^{L,t}$ of a set of agents A_g at time *t* at abstraction level *L* is the tuple of the state variables for all agents in A_g. A *global (joint) state* $\mathbf{s}^{L,t}$ at time *t* at level *L* is the tuple of all agents' local states.
  The set of all joint states with respect to abstraction level *L* for any subset A_g of *A* is denoted $\mathbf{State}_{A_g}^L$, and the set of joint society states with respect to abstraction level *L* is denoted by $\mathbf{State}^L$.

- The *local strategy* for agent A_i at time t, and at level *L* in $\{1, \ldots h\}$ - denoted by $\mathbf{str}_i^{L,t}$ – results from the sequence of actions decided by A_i up to that specific time instant. This is translated to the total delay the agent imposes at its trajectory. Such an action, in case the agent at a time point *t* is still on ground, may be either to add further delay to its total delay until the next time instant $t + ts_L$, or not. Therefore, while agents take a binary decision, the amount of delay to be added at each time point depends on the abstraction considered: This is equal to ts_L, where *L* is the level of abstraction considered, imposing an abstraction on actions. When the agent has taken off, then its strategy is considered fixed and it follows the intended/predicted trajectory. The location (i.e. sector) of that agent at any time point can be calculated by consulting its original trajectory, given that no rerouting has been applied.
  The joint strategy of a subset of agents A_g of *A* executing their trajectories (for instance of $\mathbf{N}(A_i)$) at time *t* at abstraction level **L**, is a tuple of local strategies, denoted by $\mathbf{str}_{A_g}^{L,t}$ (e.g. $\mathbf{str}_{\mathbf{N}(A_i)}^{L,t}$). The joint strategy for all agents *A* at any time instant *t* at abstraction level *L* is denoted $\mathbf{str}^{L,t}$.
  The set of all joint strategies for any subset A_g of *A* at abstraction level *L* is denoted $\mathbf{Strategy}_{A_g}^L$, and the set of joint society strategies is denoted by $\mathbf{Strategy}^L$.

- The state transition function Tr at abstraction level *L*, denoted Tr_L, gives the transition to the joint state $\mathbf{s}^{L,t+1}$ based on the joint strategy $\mathbf{str}^{L,t}$ taken at abstraction level *L*, in joint state $\mathbf{s}^{L,t}$.
  Formally $Tr_L : \mathbf{State}^L \times \mathbf{Strategy}^L \rightarrow \mathbf{State}^L$.
  Given that no agent can predict how the joint state can be affected in the next time step, for agent A_i this transition function is actually
  $Tr_L : \mathbf{State}_{\{A_i\}}^L \times \mathbf{Strategy}_{A_i}^L \times \mathbf{State}_{A_i}^L \rightarrow [0,1]$,
  denoting the transition probability $p(\mathbf{s}_i^{L,t+1} | \mathbf{s}_i^{L,t}, \mathbf{str}_i^{L,t})$.

- The local reward of an agent A_i, denoted Rwd_i, is the reward that the agent gets by executing its own strategy in a specific joint state and hierarchy level *L*, i.e. with any agent executing a trajectory in $\mathbf{Traffic}(A_i)$, according to the sectors' capacities, and the joint strategy of agents in $\mathbf{N}(A_i)$ at level *L*. Although the agent's strategy depends on the level *L* of abstraction, the reward function is level independent.

- A (local) policy of an agent A_i at level *L* is a function $\pi_i^L : \mathbf{State}_{A_i}^L \rightarrow \mathbf{Strategy}_{A_i}^L$ that returns local strategies for any given local state, for agent A_i to execute its trajectory.

5. REWARD FUNCTION

For the DCB problem we have formulated the individual delay reward Rwd_i : For an agent A_i in **A** this reward depends on the participation (contribution) of that agent in hotspots occurring while executing its trajectory according to its strategy $\mathbf{str}_i^t$ in state $\mathbf{s}_i^t$, i.e. according to its decided delay at any level of abstraction. Specifically, as already specified, the reward function applies to any level and it is level-agnostic. Thus, we do not refer to states/strategies' levels here.

Formally:

$$Rwd_i(s_i^t, str_i^t) = C(s_i^t, str_i^t) - \lambda * DC(str_i^t)$$

where,

- $C(s_i^t, str_i^t)$ is a function that depends on the participation of A_i in hotspots while executing its trajectory according to the strategy $strt_i$, and
- $DC(strt_i)$ is a function of agent's delay cost.

The (user-defined) parameter λ is used for balancing between the cost of participating in hotspots and the cost of the ground delay imposed towards achieving the goal: Zero hotspots and the minimum possible delay.

In the DCB problem, both functions $C(s_i^t, str_i^t)$ and $DC(str_i^t)$ represent delay costs at the strategic phase of operations. We have chosen $C(s_i^t, str_i^t)$ to depend on the total duration of the period in which agents fly over congested sectors. This is multiplied by 81 which is the average strategic delay cost per minute (in Euros) in Europe when 92% of the flights do not have delays [5]. If there is not any congestion, then this is a large positive constant that represents the reward agents get by not participating in any hotspot. Overall, the actual form of $C(s_i^t, str_i^t)$ is as follows:

$$C(s_i^t, str_i^t) = \begin{cases} -TDC * 81 \text{ if } TDC > 0 \\ PositiveReward \text{ if } TDC = 0 \end{cases}$$

where, *TDC* is the total duration in congestions (i.e. hotspots in our problem case) for agent A_i. The first case holds when there are hotspots in which agents participate (thus, the total duration in hotspots, *TDC*, is above 0), while the second case holds when agents do not participate in hotspots. The $DC(str_i^t)$ component of the reward function corresponds to the strategic delay cost when flights delay at gate. In our implementation, this depends solely on the minutes of delay and the aircraft type, as specified in [5]. As such, the actual form of this function is as follows:

$$DC(str_i^t) = StrategicDelayCost(str_i^t, AircraftType(A_i))$$

where str_i^t is the delay imposed to the agent A_i and $StrategicDelayCost$ is a function that returns the strategic delay cost given the aircraft type of agent A_i and its delay. Notice however that in the general case the function $DC(str_i^t)$ could be taking into account additional airline-specific strategic policies and considerations regarding flight delays.

6. INDEPENDENT Q-LEARNERS AND INDEPENDENT HIERARCHICAL Q-LEARNERS

6.1. Independent Q-Learners

In the Independent Reinforcement Learners (IRL) framework, each agent learns its own policy independently from the others and treats other agents as part of the environment.

The independent Q-learning variant proposed in [7] decomposes the global Q-function into a linear combination of local agent-dependent Q-functions.

$Q(\mathbf{s}, \mathbf{str}) = \sum_i^N Q_i(s_i, str_i)$

Although the original proposal considers a "flat" method, i.e. at a single level of abstraction – the ground one, considering the Hierachical MDP framework, each local value, Q_i^L w.r.t. abstraction level *L*, for agent A_i is calculated according to the local state, s_i^L , and the local strategy, $\mathbf{str}_i^L$, at abstraction level *L*. Whenever an agent is in a state *s*, it abstracts this state to the corresponding abstract state s^L and updates the Q-value of this abstract state based on the temporal-difference error. More specifically, the local value Q_i^L, w.r.t. abstraction level *L*, is updated according to the temporal-difference error, as follows:

$Q_i^L(s^L, str^L) = Q_i^L(s^L, str^L) + \alpha[Rwd_i(s, str^L) + \delta * max_{str} Q_i(s'^L, str^L) - Q_i(s^L, str^L)].$

It must be noted that even though we update Q-values corresponding to abstract states s^L, we compute the reward based on the ground state s *L*.

Also it must be pointed out that instead of the global reward $Rwd(\mathbf{s}, \mathbf{str})$ used in [7], we use the reward Rwd_i received by the agent A_i , taking into account only the local state and local strategy of that agent. Thus, this method is in contrast to the approach of Coordinated Reinforcement Learning model proposed in [7], since that model needs agents to know the maximizing joint action in the next state, the associated maximal expected future return, and needs to estimate the Q-value in the global state.

6.2. Independent Hierarchical Q-Learners

We explore a Hierarchical Independent Learners approach in order to increase the computational efficiency and produce more qualitative solutions in terms of reducing delays and number of delayed flights. We create multiple abstractions of the state space, denoted by $\mathbf{State}^L$ at level L, and simulate a number of episodes at any abstract level.

The Q values learned by the agent at any abstract level are used to increase the computational efficiency (in terms of the number of episodes agents run until convergence) and efficacy (in terms of quality of solutions to the DCB problem) of learning at the next level,

until reaching the ground level.

In the following paragraphs we present in more detail the generic hierarchical RL framework at multiple abstraction levels. The presentation focuses only on delays of states which is the only state variable that is the subject of abstraction.

The proposed generic hierarchical RL method comprises the following phases:

1. **Consider the ground (original) state space, State.** Considering the state space **State**, the local state per agent A_i at time *t* includes (a) the delay imposed to the trajectory T_i executed by A_i, and (b) the number of hotspots in which A_i is involved in (for any of the sectors) having the corresponding delay being imposed to it. At this "ground" level the distance between consecutive time points is one time instant (i.e. one minute).

2. **Map State to multiple abstract spaces State^L, where |State^L| « |State|.**This includes the abstraction of the ground (original) state space. Various alternative schemes can be applied for creating the mapping of states in **State** to states in **State**^{L}. The simplest approach is to divide time into a number of *K* equidistant intervals at level *L*, where $K = MaxDelay/as_L$ and as_L is the abstraction step at level *L*. Thus, as shown in Fig. 6.1 , the same state in the abstract space **State**^{L} abstracts all states with delays between consecutive time points t and $t + as_L$. It must be noticed that in this step all abstract spaces **State**^{L} are formed for every *L* in $\{1, ...h\}$, based on the mapping of states in **State** to states in **State**^{L}.

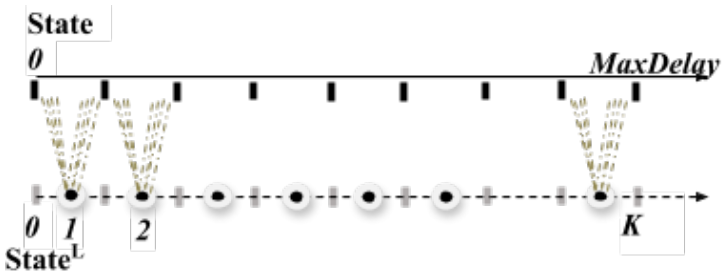


Figure 6.1: Construction of the abstract space: Delay is partitioning into a number of K equidistant intervals and delays between consecutive time points t and $t + as_L$ are mapped to the same state in the abstract space.

3. **Solve MDP in State^L space.** At any level of abstraction, according to the hierarchical MDP formulated above, the agent can either add to its total delay a number of time instants equal to ts_L, or not. When the decision is at the original state level, it may add one time instant. If the decision is at an abstract level the agent may add to its delay a number of time instants up to the next time point $t + ts_L$, until the *MaxDelay* is reached.

4. **Map solution from abstract space State^L to the next abstract State^{L+1} space.** After learning an abstract policy to solve the DCB problem at level *L*[1], the goal is to further refine it to a policy applied to the next abstract state space **State**$^{L+1}$. To do that, we have to first map states between abstraction levels. We consider any of the following alternative cases for mapping states between abstraction levels and

[1] It must be noticed that it is not necessary this estimation to provide a solution to the DCB problem, i.e. impose delays that when applied result to zero hotspots.

estimating the Q-values at the abstract state-action space of level *L+1* in terms of the values learned at the previous abstract level *L*:

a) let us consider the abstract states s_i^L, s_i^{L+1}, the set of states of the (original) ground space **State** that s_i^L abstracts, denoted by $S_{i,L}$ and the set of states of the original ground space **State** that s_i^{L+1} abstracts, denoted by $S_{i,L+1}$. We map s_i^{L+1} to the abstract state s_i^L that maximizes $|S_{i,L+1} \cap S_{i,L}|$. Any state s_i^{L+1} in the set of states **State**$^{L+1}$ that have been mapped to the same abstract state s_i^L in **State**^{L} have the same Q* values per agent and strategy, equal to the Q* value computed for the state s_i^L, w.r.t the agent strategy.

   More formally:

   $$Q_i(s_i^{L+1}, str_i^{L+1}) = Q_i^*(s_i^L, str_i^L)$$

   for any states $s_i^{L+1} \in$ **State**$^{L+1}, s_i^L \in$ **State**^{L}, such that $s_i^{L+1} \rightarrow s_i^L$ and $str_i^{L+1} = str_i^L$, where the arrows specify mappings between states at the two different levels of abstraction.

   It should be also noticed that each pair of corresponding (mapped) abstract s_i^{L+1} and s_i^L states, in conjunction to corresponding delays (i.e. strategies), should have the same number of hotspots.

b) The Q values for any state s_i^{L+1} in the set of states **State**$^{L+1}$ are updated concurrently to any state corresponding (mapped) s_i^L in **State**^{L} at every *L* in $\{1, ...h\}$. More formally whenever an agent observes a state s_i, it abstracts this state to the corresponding abstract state s_i^L and updates the Q-value of this abstract state based on the temporal-difference error. More specifically the local value Q_i^L of the current level *L* is updated, as follows:

   $$Q_i^L(s^L, str^L) = Q_i^L(s^L, str^L) + \alpha[Rwd_i(s, str^L) + \delta * max_{str} Q_i^L(s'^L, str^L) - Q_i^L(s^L, str^L)]$$

   This method also updates concurrently the local values $Q_i^{L_j}$ of the succeeding levels L_j for $L_j > L$ that correspond to state s_i w.r.t. the strategy of level L according to the temporal-difference error, as follows:

   $$Q_i^{L_j}(s^{L_j}, str^L) = Q_i^{L_j}(s^{L_j}, str^L) + \alpha[Rwd_i(s, str^L) + \delta * max_{str} Q_i^{L_j}(s'^{L_j}, str'^{L_j} - Q_i^{L_j}(s^{L_j}, str^L)]$$

c) The Q values for any state s_i^{L+1} in the set of states **State**$^{L+1}$ are updated independently to any state corresponding (mapped) s_i^L in **State**^{L} at every *L* in $\{1, ...h\}$. Thus all Q values of level *L+1* are initialized to 0.

   More formally:

   $Q_i^{L+1}(s_i^{L+1}, str_i^{L+1}) = 0$ and updated as specified in the flat model.

   for any state $s_i^{L+1} \in$ **State**$^{L+1}$ and strategy str_i^{L+1}.

5. **Solve MDP in the next State^{L+1} space.** Given the Q-values at the abstract state-action space of level *L+1*, a refinement phase in the abstract space **State**$^{L+1}$ follows, towards computing a refined policy per agent, i.e. a policy with delays at the abstract state-action space of level *L+1*.

We repeat phases 3 and 4 for *L=L+1* until we reach the ground level and solve the MDP in the ground space **State**.

Given this generic *Independent Hierarchical Q-Learners* framework in the following paragraphs we present each method in more detail, based on the formalization provided above.

6.2.1. Descending Timesteps (DT)

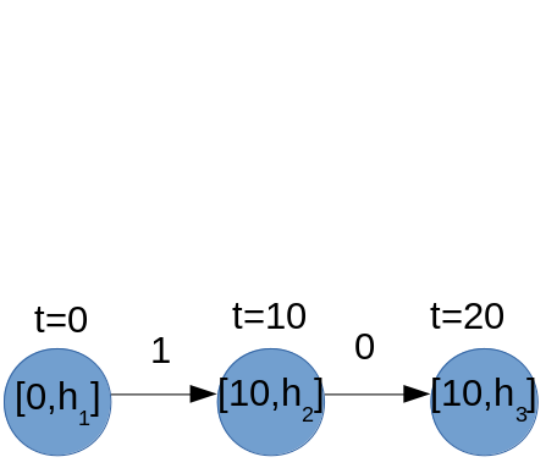
The method presented in this subsection realizes all of the predefined phases. At subsequent abstract spaces $State^L$ and $State^{L+1}$, we use time steps and abstraction steps greater than 1 time point and for any two abstract levels *L1* and *L2*, with $L_1 < L_2$, it holds that $ts_{L_1} > ts_{L_2}$ and also $as_{L_1} > as_{L_2}$. Additionally, for any abstract level *L*, it holds that $ts_L = as_L$. By using timesteps greater than one time point this method imposes abstraction on actions. The abstraction step being also non-unitary abstracts states of the ground (original) space **State** to abstract states at any $State^L$.

A key difference that distinguishes this method from other methods is that the set of options assumed by agent A_i at level L, $D_i^L = \{0, ...MaxDelay_i\}$, depends only at timestep ts_L, whereas other methods limit the set of options further, based on the solution provided at level *L-1*. Next, we present the hierarchical RL phases in more detail based on the aforementioned analysis.

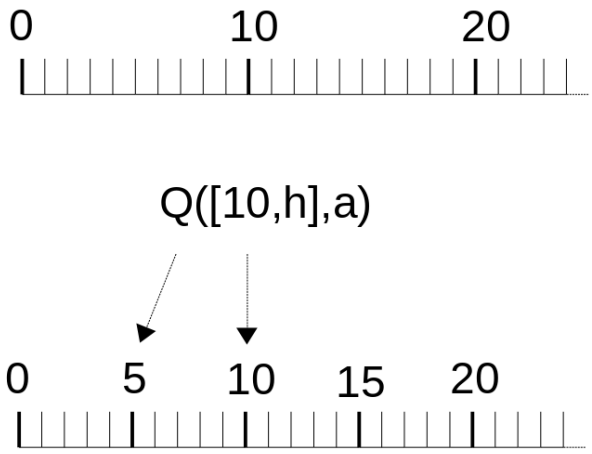
1. **Consider the original state space, State.**
2. **Map State to multiple abstract spaces State^L, where |State^L| « |State|.** In our approach each state s_i in **State** with delay between time points t and $t+as_L$ abstracts to the same abstract state s_i^L.
3. **Solve MDP in State^L space.**
4. **Map Q values from abstract space State^L to the next abstract State^{L+1} space.** The presented method uses option (a) of generic framework phase 4 to initialize Q-values in the state space **State**$^{L+1}$.
5. **Solve MDP in the next State^{L+1} space.**

Let us consider an example of 3 levels where $ts_1 = as_1 = 10$,$ts_2 = as_2 = 5$,$ts_3 = as_3 = 1$. As seen in figure 6.2a at level L=1, at time $t = 0$ the agent is in state [0,h1]. The agent chooses action 1 and after the corresponding transition it observes state [10,h2], after ts_L time points, at time $t = 10$. At this timestep the agent chooses action 0 resulting to state [10,h3], after ts_L time points, at time $t = 20$. This process continues until the end of the episode. The Q values from level 1 are mapped to corresponding states at level 2. As depicted in 6.2b the Q values of the first ten minute interval are mapped to the Q values of the first and second five minute intervals of level 2 w.r.t. the corresponding hotspots and action. This process is repeated until we solve the problem at level 3.

According to experimental results, also shown in the evaluation section, this method cannot achieve significant improvement regarding the computational efficiency compared to



(a) Example of local state transitions of the Descending Timesteps (DT) method at abstract level *L*, where $ts_L = 10$



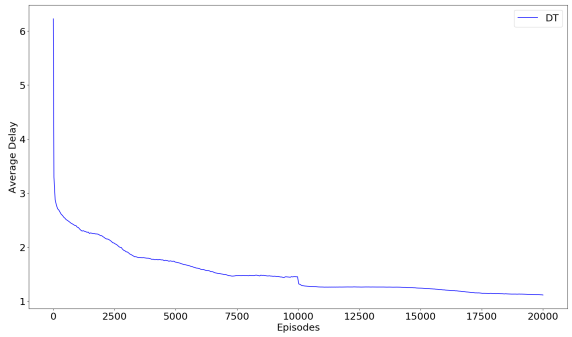
(b) Example of "transferring" the Q values of an abstract level *L* with $as_L = 10$ to the next abstract level *L+1* with $as_{L+1} = 5$

Figure 6.2: Example of the Descending Timesteps. Case (a) shows the transitions of the local state of an agent regarding the action at time *t* for an abstract level L with $ts_L = 10$, whereas case (b) shows the "transferring of the Q values of an abstract level *L* with $as_L = 10$ to the next abstract level *L+1* with $as_{L+1} = 5$

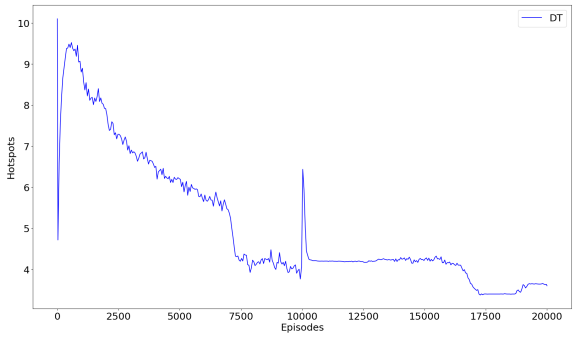
the flat method, nor it can improve significantly the quality of the produced solutions compared to the case where no hierarchical structures are being used [1].

In doing so we need to exploit the solution that level *L* produces in order to limit the state space explored and exploited at level *L+1*. This will allow more thorough exploration at the part of the state space that includes states of interest (i.e. states towards solving the problem) regarding level *L*. Hence, we expect level *L+1* to provide better estimation of Q values in less time and thus converge faster.

Specifically, after solving the MDP at the abstract space $State^L$, an estimation of the delay $delay_i$ decided by each agent A_i, is available. We exploit this estimation to limit the agents options D_i^{L+1}. For the Descending Timesteps (DT) method we restrict the agent's options in the corresponding interval $(k * asL, (k+1) * asL], k \in N$, that $delay_i$ belongs.



(a) Average Delay



(b) Hotspots

Figure 6.3: Result of 10 independent experiments. The learning curve of the descending time step method , given two levels and limitation of the state space, showing how agents manage to learn joint policies towards resolving DCB problems, while reducing the average delays for all flights. The x axis corresponds to the episodes, while the y axis to the average delay per flight case (a) or the hotspots case (b).

As Fig. 6.3 depicts, although the descending time step method manages to decrease the

average delay, this method does not result in 0 hotspots when the state space is limited at the second level. This is because, as mentioned previously, states with delay that is not a multiple of the timestep, being unreachable, have not been explored. This pruning of the state space could make states with 0 hotspots unreachable and thus the solution provided by the abstract level(s) could be in a completely different part of the state space in relation to solutions resulting to 0 hotspots.

In the next subsections we study methods that use unitary timesteps at all levels, so that all states are indeed reachable and can be explored. We also exploit non unitary abstraction steps to treat states that correspond to the same abstract state as equivalent. Additionally, these methods effectively limit the state space based on the estimation of the delay provided at the previous level. We exploit this estimation to further limit the agent's options D_i^{L+1}: The set of options assumed by A_i in the next **State**$^{L+1}$ space is in $D_i^{L+1} = \{max\{0, delay_i - d^{L+1}\}, ..., delay_i\}$, where d^{L+1} is the amount of time instants we subtract from the delay estimation $delay_i$ to the state space **State**$^{L+1}$, allowing the agent to search and refine the solution prescribed at the abstract level L in a wider set of options than those defined by the $delay_i$.

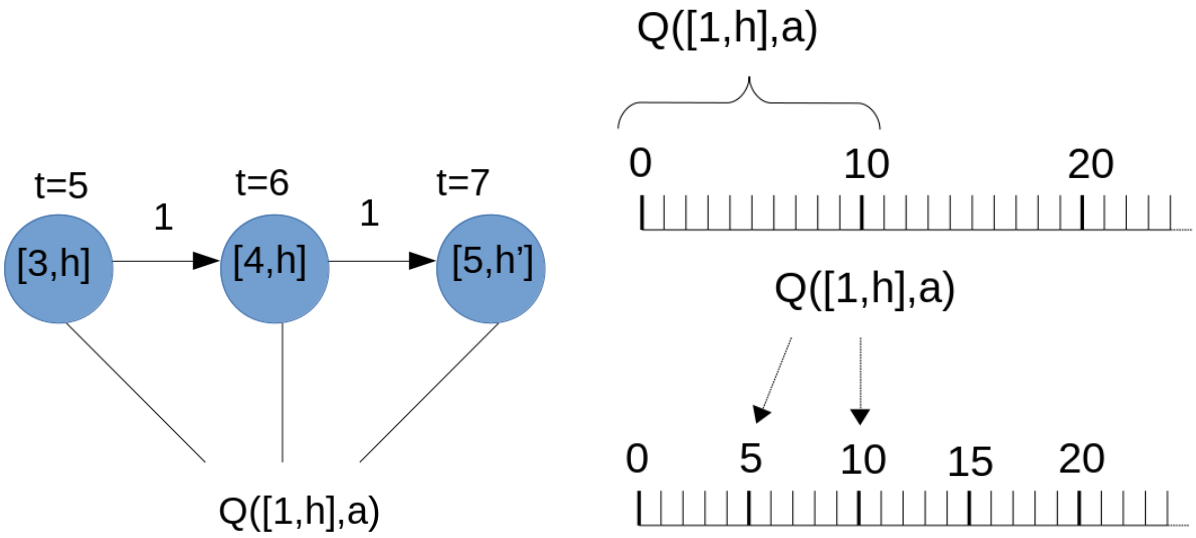
We also explore alternative schemes for "transferring" Q values between levels. More specifically, the method presented at subsection 6.2.2 maps solution from abstract space **State**^{L} to the next abstract **State**$^{L+1}$ space by inheriting the Q-values of level *L* to level *L+1* as previously according to option (a) of the generic framework phase 4. The method presented at subsection 6.2.3 when operating at level *L* updates concurrently the Q-values of all levels *L+k*, up to *h*, according to option (b) of the generic framework step 5. Finally, the method presented at subsection 6.2.4 does not transfer values between levels, not it allows updating Q values based on estimations at other levels: It follows option (c) and allows levels to operate independently.

6.2.2. Unitary Timestep, Limit State Space (UTLSS)

The following method uses timesteps of size 1 at all levels, and descending abstraction step as the level of abstraction increases. More formally for any two abstraction levels L_1 and L_2, where $L_1 < L_2$, it holds that $ts_{L_1} = ts_{L_2} = 1$ and $as_{L_1} > as_{L_2}$. Additionally, we limit the state space at any abstraction level, based on the delay estimation provided by the previous abstraction level. As a result the set of options assumed by agent A_i at level *L*, denoted by D_{L_i} is a subset of $\{0, ...MaxDelay_i\}$ and differs at each level. Phases 1-5 are described below.

1. **Consider the ground (original) state space, State.**
2. **Map State to an abstract space State^{L}, where |State^{L}| « |State|.** This method abstracts each state s_i in **State** with delay between consecutive time points t and $t + as_L$ to the same abstract state s_i^L.
3. **Solve MDP in State^{L} space.** Solve MDP in **State**^{L} space using unitary timestep.
4. **Map solution from abstract space State^{L} to the next abstract State^{L+1} space.** The presented method uses option (b) of generic framework phase 4 to initialize Q-values in the state space **State**$^{L+1}$.

5. **Solve MDP in the next State^{L+1} space.** Solve MDP in the next **State**$^{L+1}$ space using a smaller abstraction step. This method also exploits the estimation provided by the abstract **State**^{L} space and effectively limits the state space to be explored in **State**$^{L+1}$. After solving the MDP at the abstract space **State**^{L}, an estimation of the delay $delay_i$ decided by the agent, is available. We exploit this estimation to limit the agent's options D_i^{L+1} considered at the next level of abstraction: The set of options assumed by A_i in the next **State**$^{L+1}$ space is in $D_i^{L+1} = \{max\{0, delay_i - d^{L+1}\}, ..., delay_i\}$, where d^{L+1} is the amount of time instants we subtract from the delay estimation $delay_i$ to provide more option for the agent to consider while solving the problem in state space **State**$^{L+1}$.



(a) Example of local state transitions and the corresponding updated Q value of the Unitary Timestep, Limit State Space (UTLSS) method at abstract level *L*, where $as_L = 10$

(b) Example of "transferring" the Q values of an abstract level *L* with $as_L = 10$ to the next abstract level *L+1* with $as_{L+1} = 5$

Figure 6.4: Example of the Unitary Timestep, Limit State Space (UTLSS). Case (a) shows the transitions of the local state of an agent regarding the action at time *t* and the corresponding updated Q value for an abstract level L with $as_L = 10$, whereas case (b) shows the "transferring of the Q values of an abstract level *L* with $as_L = 10$ to the next abstract level *L+1* with $as_{L+1} = 5$

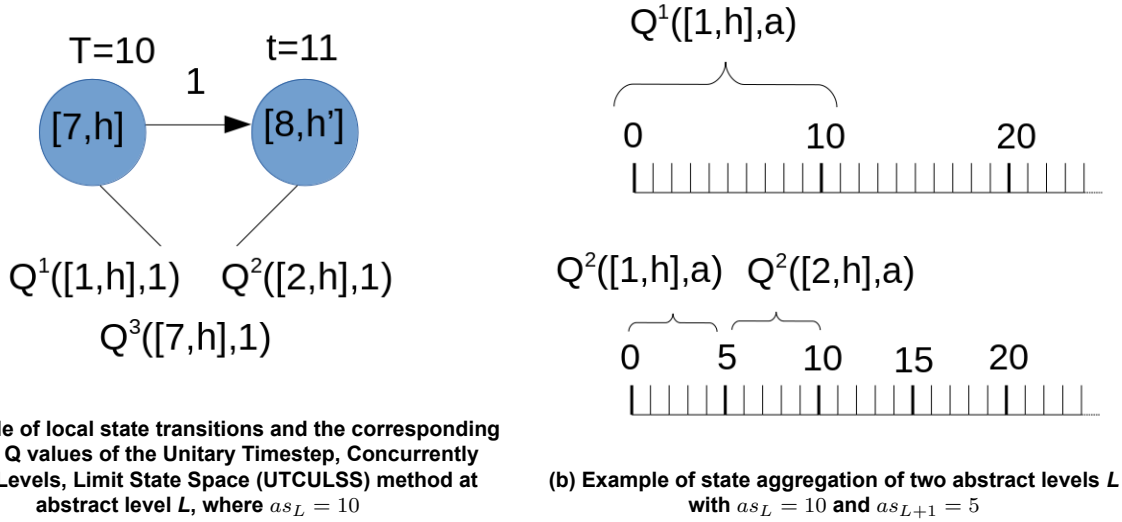
Again let us consider an example of 3 levels, where $as_1 = 10$,$as_2 = 5$,$as_3 = 1$ and $ts_L = 1$ at all levels L. As seen in figure 6.4a, let us suppose that at level L=1, at time $t = 5$ the agent is in state [3,h1]. The agent chooses action 1 and after the corresponding transition it observes state [4,h2], after ts_L time points, at time $t = 6$. At this timestep the agent chooses again action 1 resulting to state [5,h3], after ts_L time points, at time $t = 7$. If $h1 = h2 = h$, these two updates will both update the Q value of $Q^1([1, h], 1)$, because as seen in 6.4b both states correspond to the first ten minute interval. For all states that belong in the same interval regarding as_L, we update the same Q values, always w.r.t. the hotspots and action. This process continues until the end of the episode. When level 1 finishes, we map the Q values from level 1 to level 2. As depicted in 6.2b the Q values of the first ten minute interval are mapped to the Q values of the first and second five minute intervals of level 2 w.r.t. the corresponding hotspots and action. This process is repeated until we solve the problem at level 3.

6.2.3. Unitary Timestep, Concurrently Update Levels, Limit State Space (UTCULSS)

The timestep of this method is unitary at all levels, while the abstraction step descends while the level of abstraction increases. Also, the state space explored at any level is

restricted to a set of options based on the delay estimation provided by the previous level. The key difference here is that following option (b) in phase 5, when solving the MDP at space **State**^{L} we concurrently update the Q-values of all levels *L+k*, up to *h*. The phases are as follows:

1. **Consider the original state space, State.**
2. **Map State to an abstract space State^L, where |State^L| « |State|.** This method abstracts each state s_i in **State** with delay between consecutive time points t and $t + as_L$ to the same abstract state s_i^L.
3. **Solve MDP in State^L space.** Solve MDP in **State**^{L} space using unitary timestep.
4. **Map solution from abstract space State^L to the next abstract State^{L+1} space.** The presented method uses option (b) of generic framework phase 4 to initialize Q-values in the state space **State**$^{L+1}$.
5. **Solve MDP in the next State^{L+1} space.** Solve MDP in the next **State**$^{L+1}$ space using a smaller abstraction step. As described in subsection 6.2.2 we exploit the delay estimation decided at the abstract **State**^{L} space and effectively limit the state space **State**$^{L+1}$: After solving the MDP at the abstract space **State**^{L}, an estimation of the delay $delay_i$ decided by the agent, is available. We exploit this estimation to limit the agent's options D_i^{L+1} considered at the next level of abstraction: The set of options assumed by A_i in the next **State**$^{L+1}$ space is in $D_i^{L+1} = \{max\{0, delay_i - d^{L+1}\}, ..., delay_i\}$, where d^{L+1} is the amount of time instants we subtract from the delay estimation $delay_i$ to provide more option for the agent to consider while solving the problem in state space **State**$^{L+1}$.



(a) Example of local state transitions and the corresponding updated Q values of the Unitary Timestep, Concurrently Update Levels, Limit State Space (UTCULSS) method at abstract level *L*, where $as_L = 10$

(b) Example of state aggregation of two abstract levels *L* with $as_L = 10$ and $as_{L+1} = 5$

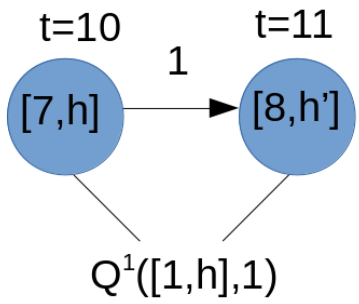
Figure 6.5: Example of the Unitary Timestep, Concurrently Update Levels, Limit State Space (UTCULSS). Case (a) shows the transitions of the local state of an agent regarding the action at time *t* and the corresponding updated Q value for an abstract level L with $as_L = 10$, whereas case (b) shows the state aggregation of two abstract levels *L* with $as_L = 10$ and $as_{L+1} = 5$

Again let us consider an example of 3 levels, where $as_1 = 10$,$as_2 = 5$,$as_3 = 1$ and $ts_L = 1$ at all levels *L*. As seen in figure 6.5a, let us suppose that at level *L=1*, at time $t = 10$ the agent is in state [7,h]. The agent chooses action 1 and after the corresponding transition it observes state [8,h'], after ts_L time points, at time $t = 11$. This transition results to updating $Q^1([1, h], 1)$ as 7 is in the first ten minute interval, $Q^2([2, h], 1)$ as 7 is in the second

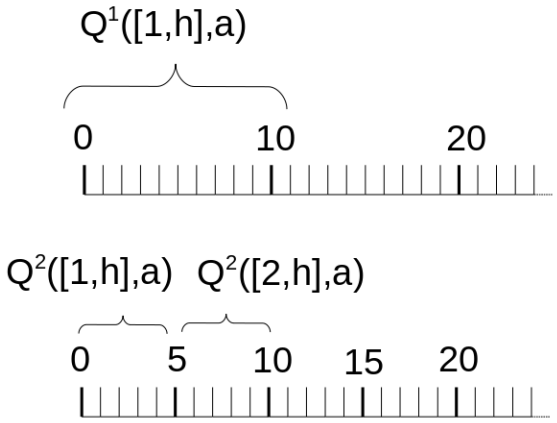
five minute interval and also $Q^3([7,h],1)$.

6.2.4. Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET)

This method uses a unitary timestep, descending abstraction steps as the abstraction level increases, and limits the state space to be explored at any level based on the solution decided at the previous level of abstraction. Additionally, we do not map the solution from abstract space **State**^{L} to the next **State**$^{L+1}$ space nor we update concurrently the Q-values of levels as described in subsection 6.2.3. Following option (c) of phase 5 different abstraction levels operate independently. Phases of the generic method are as follows:



(a) Example of local state transitions and the corresponding updated Q values of the Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) method at abstract level *L*, where $as_L = 10$



(b) Example of state aggregation of two abstract levels *L* with $as_L = 10$ and $as_{L+1} = 5$.

Figure 6.6: Example of the Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET). Case (a) shows the transitions of the local state of an agent regarding the action at time *t* and the corresponding updated Q value for an abstract level L with $as_L = 10$, whereas case (b) shows the state aggregation of two abstract levels *L* with $as_L = 10$ and $as_{L+1} = 5$

1. **Consider the ground (original) state space, State.**
2. **Map State to an abstract space State^L, where |State^L| « |State|.** his method abstracts each state s_i in **State** with delay between consecutive time points t and $t + as_L$ to the same abstract state s_i^L.
3. **Solve MDP in State^L space.** Solve MDP in **State**^{L} space using unitary timestep.
4. **Map solution from abstract space State^L to the next abstract State^{L+1} space.** The presented method uses option (c) of generic framework phase 4 to initialize Q-values in the state space **State**$^{L+1}$.
5. **Solve MDP in the next State^{L+1} space.**Solve MDP in the next **State**$^{L+1}$ space using a smaller abstraction step. As described in subsection 6.2.2 we exploit the delay estimation decided at the abstract **State**^{L} space and effectively limit the state space **State**$^{L+1}$: After solving the MDP at the abstract space **State**^{L}, an estimation of the delay delayi decided by the agent, is available. We exploit this estimation to limit the agent's options D_i^{L+1} considered at the next level of abstraction: The set of options assumed by A_i in the next **State**$^{L+1}$ space is in $D_i^{L+1} = \{max\{0, delay_i - d^{L+1}\}, ..., delay_i\}$, where d^{L+1} is the amount of time instants we subtract from the

delay estimation $delay_i$ to provide more option for the agent to consider while solving the problem in state space **State**$^{L+1}$.

Again let us consider an example of 3 levels, where $as_1 = 10$,$as_2 = 5$,$as_3 = 1$ and $ts_L = 1$ at all levels *L*. As seen in figure 6.6a, let us suppose that at level L=1, at time $t = 10$ the agent is in state [7,h]. The agent chooses action 1 and after the corresponding transition it observes state [8,h'], after ts_L time points, at time $t = 11$. This transition results to updating only $Q^1([1, h], 1)$ as 7 is in the first ten minute interval. Also we do not map the Q values of level 1 to level 2, but we instead initialize the Q table to 0.

7. RESULTS ON SCENARIOS AND COMPARISON BETWEEN VERSIONS AND WITH OTHER METHODS

To evaluate the proposed methods we have constructed evaluation cases, each corresponding to a specific day of 2016 above Spain. In this thesis we present evaluation results for the July 2 case, which seems to be one of the hardest among the available cases [1].

At first we compare our methods for 2 and 3 levels and for 10000 episodes at each level and observe their efficiency and effectiveness. We then continue to reducing the episodes of the most promising methods and for 3 levels to assess them in more demanding cases. Finally having the methods managed provide qualitative solutions using 8000 episodes in total, we further reduce the episodes to assess them in even more strenuous conditions. We explore cases where the first level does not achieve convergence nor results to 0 hotspots given the limited number of available episodes. Even in this case 2 of the methods manage to produce qualitative solutions regarding our metrics that result to 0 hotspots in 6000 episodes in total.

In subsections 7.3, 7.4 we present evaluation results for 2 and 3 levels and for 10000 episodes at each level following an ε-greedy exploration-exploitation strategy starting from ε =0.9, which every 80 rounds is diminished by 0.01. At episode 7200 the probability becomes 0.001 and is henceforth considered zero. Then, pure exploitation phase begins until the end of the episodes.

In subsection 7.3,7.4 we also present evaluation results for 3 levels for the most promising of our methods for the 5000 episodes at the first (abstract) level and 1500 episodes at the second (abstract) and third (original) levels. We also follow an ε-greedy exploration-exploitation strategy starting from ε =0.9 at every level, which is diminished by the value of 0.01 every 33 rounds at the first (abstract) level and every 13 rounds at the second (abstract) and third (original) levels. At the first (abstract) level the probability becomes 0.001 at episode 2970 and is henceforth considered zero. At the second (abstract) and third (original) levels the probability becomes 0.001 at episode 1170 and is henceforth considered zero. Then, pure exploitation phase begins until the end of the episodes for the operating level.

7.1. Evaluation Criteria

To report on the efficiency of the methods and their effectiveness w.r.t. the quality of solutions achieved, we provide per case the following information:

- **Number of flights with delay**: Also mentioned as “regulated flights”.
- **Average delay per flight**: We report on the average delay per flight (i.e. the ratio of total delay to the number of flights), while ignoring delays less than 4 minutes.

- **Learning Curves**: These curves show per methods' application episode the average delay per flight in the evaluation case, as agents learn their policies. As algorithms converge to solutions, the number of hotspots should be reduced and eventually reach to zero, signifying the computation of a solution, while the average delay should be reduced. Therefore, the speed of reaching that point (zero hotspots) and the round at which methods stabilize agents' joint policy (remaining to zero hotspots and to a specific value for flights' average delay - without oscillating between non-solutions and/or solutions, and/or different average delay values) signify the computational efficiency of the method to reach solutions. It must be noted that, in case a method cannot reach a solution for a specific case, it may converge to a joint policy that do not resolve all hotspots.
- **Distribution of delays to flights**: To show how delays are distributed to flights, we provide histograms showing the number of flights with (a) 5-9 minutes of delay, (b) 10-29 minutes of delay, (c) 30-59 minutes of delay, etc. Of course, if it happens that MaxDelay is less than 60 min, 30 min etc., the histogram does not provide data for the corresponding slots of delays.

Results reported are averages of results computed by 10 independent experiments per method.

7.2. Methods' configuration

7.2.1. Descending Timesteps (DT)

Table 7.1: Configuration table for DT method with 2 levels

level	as_L	ts_L
1	10	10
2	5	1

Table 7.1 depicts the as_L and ts_L values per level for the Descending Timesteps (DT) method and for the 2 levels case.

Table 7.2: Configuration table for DT method with 3 levels

level	as_L	ts_L
1	10	10
2	5	5
3	5	1

Table 7.2 depicts the as_L and ts_L values per level for the Descending Timesteps (DT) method and for the 3 levels case.

7.2.2. Other Methods

Table 7.3 depicts the as_L, ts_L and d^L values per level for the Unitary Timestep, Limit State Space (UTLSS), Unitary Timestep, Concurrently Update Levels, Limit State Space

Table 7.3: Configuration table for methods using unitary timestep ($ts_L = 1$) at all levels for the 2 levels case

level	as_L	ts_L	d^L
1	10	1	-
2	1	1	10

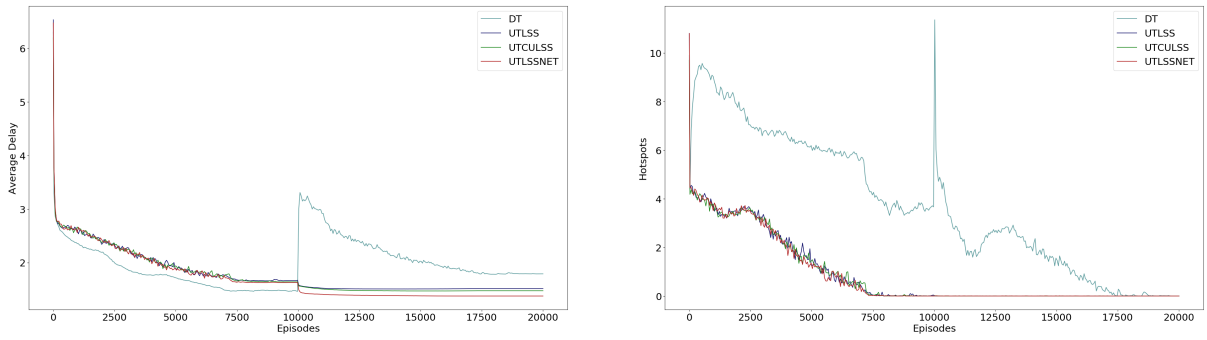
(UTCULSS), Unitary Timestep, Limit State Space, No Transfer Experience (UTLSSNET) methods and for the 2 levels case.

Table 7.4: Configuration table for methods using unitary timestep ($ts_L = 1$) at all levels for the 3 levels case

level	as_L	ts_L	d^L
1	10	1	-
2	5	1	10
3	1	1	5

Table 7.4 depicts the as_L, ts_L and d^L values per level for the Unitary Timestep, Limit State Space (UTLSS), Unitary Timestep, Concurrently Update Levels, Limit State Space (UTCULSS), Unitary Timestep, Limit State Space, No Transfer Experience (UTLSSNET) methods and for the 3 levels case.

7.3. Efficiency of the methods



(a) Average Delay **(b) Hotspots**

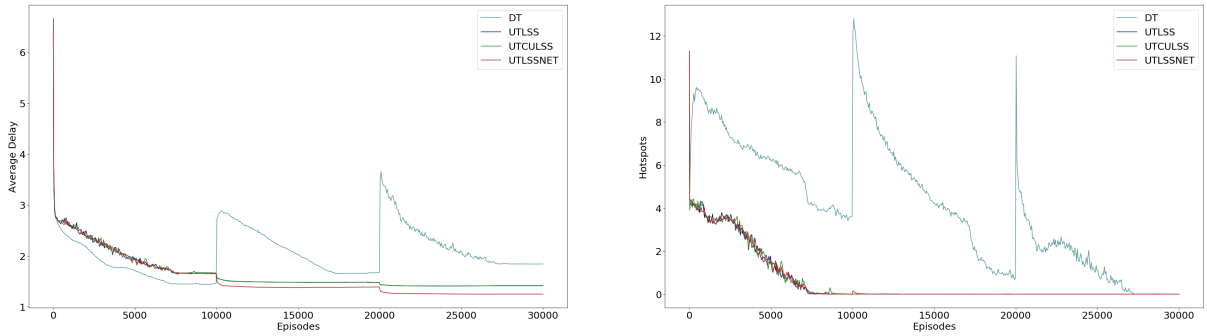
Figure 7.1: Result of 10 independent experiments. The learning curves given two levels and 10k episodes at each level, showing how agents manage to learn joint policies to resolve DCB problems, while reducing the average delays for all flights. The x axis corresponds to the episodes, while the y axis to the average delay per flight case (a) or the hotspots case (b).

Learning curves presented in Fig. 7.1 depict how given two levels and 10000 episodes at each level, agents manage to learn joint policies to resolve DCB problems, while reducing the average delays for all flights.

Learning curves presented in Fig. 7.1a and 7.2a depict how, given two and three levels correspondingly and 10000 episodes at each level, agents manage to learn to reduce the

average delay per flight: The methods converge effectively, but there is significant difference regarding the computational efficiency between methods that limit the state space explored and methods that do not.

More specifically, given two levels, methods that limit the state space explored converge effectively quite early, around episode 13000, compared to the Descending Timesteps (DT) method that converges around episode 18800, as Fig. 7.1a shows.



(a) Average Delay (b) Hotspots

Figure 7.2: Result of 10 independent experiments. The learning curves given three levels and 10k episodes at each level, showing how agents manage to learn joint policies to resolve DCB problems, while reducing the average delays for all flights. The x axis corresponds to the episodes, while the y axis to the average delay per flight case (a) or the hotspots case (b).

Learning curves presented in Fig. 7.2 depict how given three levels and 10000 episodes at each level, agents manage to learn joint policies to resolve DCB problems, while reducing the average delays for all flights.

As depicted in Fig. 7.2a, given three levels, methods that limit the state space explored converge effectively quite early, around episode 13000 at the second level and also around episode 22500 at the third level, compared to the Descending Timesteps (DT) method that converges around episode 28000.

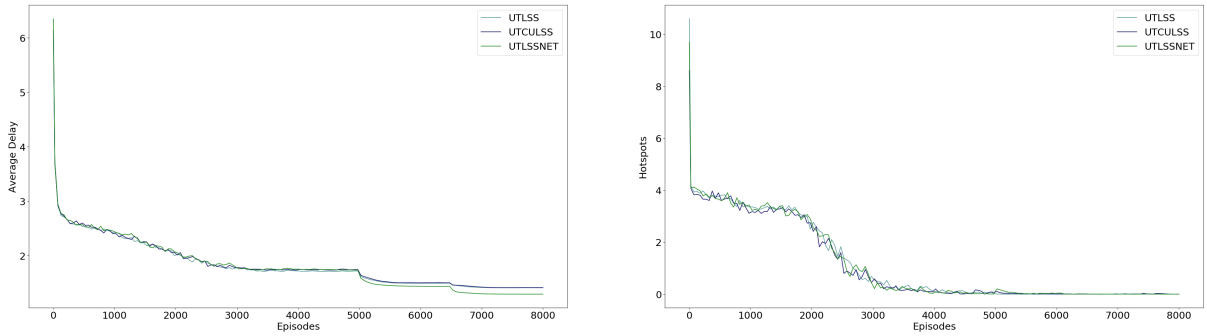
As results show in Fig. 7.1b and Fig. 7.2b methods that use unitary time step and also limit the state space almost eliminate hotspots before episode 10000, where as the Descending Timesteps (DT) method results to 0 hotspots approximately at the last 2500 episodes.

As a conclusion, methods that exploit state space limitation achieve faster convergence compared to the Descending Timesteps (DT) method. Next, we reduce the number of episodes for these methods in order to further assess their efficiency.

Learning curves presented in Fig. 7.3 depict how given three levels with 5000 episodes at level 1 and 1500 episodes at levels 2 and 3, methods that limit the state space manage to resolve DCB problems, while reducing the average delays for all flights.

As depicted in Fig. 7.3a, methods that exploit state space limitation converge effectively quite early, before episode 5000 at the first level, around episode 6000 at the third level and before episode 8000 at the third level.

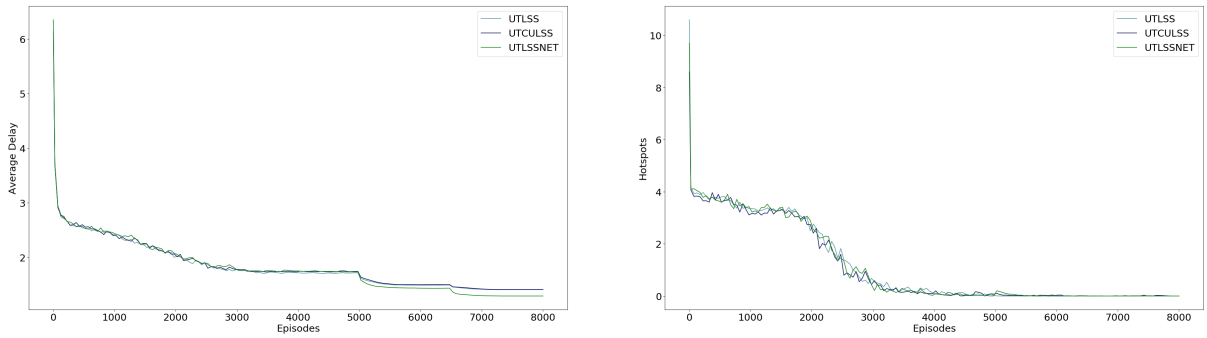
As depicted in Fig. 7.3b methods result to 0 hotspots even when reducing the number of episodes to 5000 at the first level and 1500 at the second and third levels. Next we re-



(a) Average Delay (b) Hotspots

Figure 7.3: Result of 10 independent experiments. The learning curves given three levels, 5000 episodes at level 1 and 1500 episodes at levels 2 and 3 for methods that limit the state space. Showing how agents manage to learn joint policies to resolve DCB problems, while reducing the average delays for all flights. The x axis corresponds to the episodes, while the y axis to the average delay per flight case (a) or the hotspots case (b).

duce further the number of episodes at the first level at 3000 episodes.



(a) Average Delay (b) Hotspots

Figure 7.4: Result of 10 independent experiments. The learning curves given three levels, 3000 episodes at level 1 and 1500 episodes at levels 2 and 3 for methods that limit the state space. Showing how agents manage to learn joint policies to resolve DCB problems, while reducing the average delays for all flights. The x axis corresponds to the episodes, while the y axis to the average delay per flight case (a) or the hotspots case (b).

Learning curves presented in Fig. 7.4 depict how given three levels with 3000 episodes at level 1 and 1500 episodes at levels 2 and 3, methods that limit the state space manage to resolve DCB problems, while reducing the average delays for all flights.

Results in Fig. 7.4a show that even though the first level is not able to converge, levels 2 and 3 achieve convergence.

As shown in Fig.7.4b 3000 episodes are not enough for methods to compute solutions towards zero hotspots at the first level. Even under these circumstances the Unitary Timestep,Concurrently Update Levels, Limit State Space (UTCULSS) method and the Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) method manage to find a solution that results to 0 hotspots really early at the second level. Unfortunately, this is not the case for the Unitary Timestep, Limit State Space (UTLSS) method that results to 0.2 hotspots on average over 10 experiments.

7.4. Effectiveness of the methods to resolve imbalances

Delving into the quality of results, in the following paragraphs we present the average delay, number of regulated flights, total delay and number of resulting hotspots reported by each method and also in comparison with the non hierarchical Independent Learners method presented in [1].

Table 7.5: Average over 10 independent experiments, hierarchical methods consist of 2 levels and each level comprises of 10000 episodes. Average Delay per Flight, Number of Regulated Flights Total Delay and Number of Resulting Hotspots as reported by each method and also in comparison to the non hierarchical Independent Learners method presented in [1].

Method	Average Delay per Flight	Number of Regulated Flghts	Total Delay	Number of Resulting Hotspots
Non Hierarchical	1.866	461.6	10396.5	0
DT	1.789	474.5	9976.2	0
UTLSS	1.513	348.4	8440.4	0
UTCULSS	1.474	340.8	8213.7	0
UTLSSNET	1.376	329.3	7669.6	0

Table 7.5 shows the Average Delay per Flight, Number of Regulated Flights Total Delay and Number of Resulting Hotspots as reported by each method and also in comparison to the non hierarchical Independent Learners method presented in [1]. Hierarchical methods consist of 2 levels and each level comprises of 10000 episodes.
As reported in Table 7.5 all methods result to zero hotspots and hierarchical methods produce more qualitative solutions when utilizing 2 levels in terms of average delay per flight, number of regulated flights and total delay. Also all methods that use unitary timestep and limit the state space explored, significantly outperform the Descending Timesteps (DT) method. The Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) method seems to be the most dominant, achieving the best results regarding all metrics.

Table 7.6: Average over 10 independent experiments, hierarchical methods consist of 3 levels and each level comprises of 10000 episodes. Average Delay per Flight, Number of Regulated Flights Total Delay and Number of Resulting Hotspots as reported by each method and also in comparison to the non hierarchical Independent Learners method presented in [1].

Method	Average Delay per Flight	Number of Regulated Flghts	Total Delay	Number of Resulting Hotspots
Non Hierarchical	1.866	461.6	10396.5	0
DT	1.841	496.4	10261.7	0
UTLSS	1.416	333.3	7892.8	0
UTCULSS	1.416	334.5	7886.0	0
UTLSSNET	1.251	288.1	6968.5	0

Table 7.6 shows the Average Delay per Flight, Number of Regulated Flights Total Delay and Number of Resulting Hotspots as reported by each method and also in comparison to the non hierarchical Independent Learners method presented in [1]. Hierarchical meth-

ods consist of 3 levels and each level comprises of 10000 episodes.

As reported in Table 7.6 all methods result to zero hotspots and hierarchical methods produce more qualitative solutions when utilizing 2 levels in terms of average delay per flight, number of regulated flights and total delay. The Descending Timesteps (DT) method seems to produce less qualitative solutions compared to the two levels case. Also all methods that use unitary timestep and limit the state space explored, significantly outperform the Descending Timesteps (DT) method. Again Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) method seems to be the most dominant, achieving the best results regarding all metrics.

As reported in tables 7.5 and 7.6 the Descending Timesteps (DT) method produces less qualitative solutions regarding all metrics with 3 levels compared to the solutions produced by 2 levels. All other hierarchical methods manage to significantly decrease the average delay per flight, the number of flights and also the total delay when utilizing 3 levels, compared to the results produced by 2 levels.

Next we present results regarding the average delay per flight, the number of regulated flights, the total delay and the number of resulting hotspots for the cases where we reduce significantly the number of episodes.

Table 7.7: Average over 10 independent experiments for the methods that use unitary timestep also in comparison to the non hierarchical Independent Learners method presented in [1]. Hierarchical methods consist of 3 levels and utilize 8000 episodes in total. The first level comprises of 5000 episodes while the second and third levels of 1500 episodes. Showing the Average Delay per Flight, Number of Regulated Flights Total Delay and Number of Resulting Hotspots as reported by each method.

Method	Average Delay per Flight	Number of Regulated FIghts	Total Delay	Number of Resulting Hotspots
Non Hierarchical	1.866	461.6	10396.5	0
UTLSS	1.411	319.1	7860.4	0
UTCULSS	1.411	316.7	7866.1	0
UTLSSNET	1.292	282.7	7194.1	0

Table 7.7 shows the Average Delay per Flight, Number of Regulated Flights Total Delay and Number of Resulting Hotspots as reported by the methods that use unitary timestep and also in comparison to the non hierarchical Independent Learners method presented in [1]. Hierarchical methods consist of three levels and utilize 8000 episodes in total. The first level comprises of 5000 episodes while the second and third levels of 1500 episodes.

As shown in Table 7.7, all methods result to zero hotspots. Also hierarchical methods outperform the non hierarchical regarding all metrics even with less episodes. We executed the non hierarchical method for 10000 episodes while the hierarchical methods were executed for 8000 episodes in total. Again the Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) method seems to be the most dominant, achieving the best results regarding all metrics.

Table 7.8 shows the Average Delay per Flight, Number of Regulated Flights Total Delay

Table 7.8: Average over 10 independent experiments for the methods that use unitary timestep also in comparison to the non hierarchical Independent Learners method presented in [1]. Hierarchical methods consist of 3 levels and utilize 6000 episodes in total. The first level comprises of 3000 episodes while the second and third levels of 1500 episodes. Showing the Average Delay per Flight, Number of Regulated Flights, Total Delay and Number of Resulting Hotspots as reported by each method.

Method	Average Delay per Flight	Number of Regulated FIghts	Total Delay	Number of Resulting Hotspots
Non Hierarchical	1.866	461.6	10396.5	0
UTLSS	1.399	313.8	7804.7	0.2
UTCULSS	1.404	320.8	7829.8	0
UTLSSNET	1.288	280.5	7176.0	0

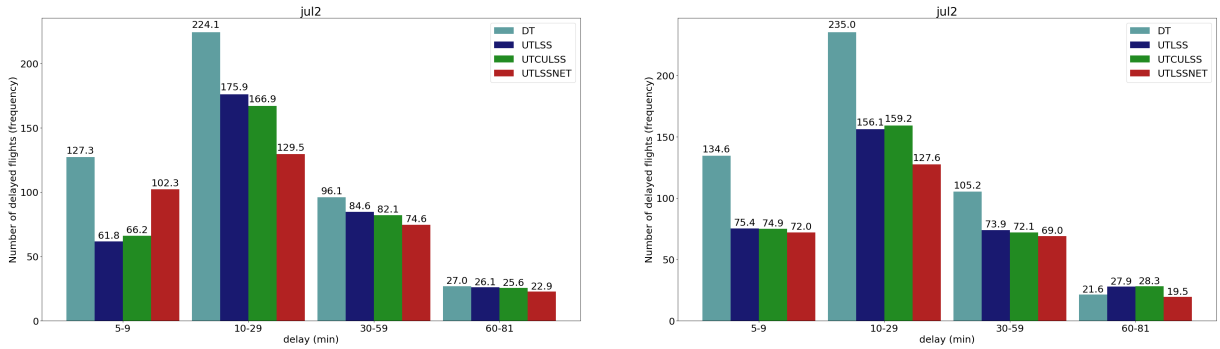
and Number of Resulting Hotspots as reported by the methods that use unitary timestep and also in comparison to the non hierarchical Independent Learners method presented in [1]. Hierarchical methods consist of three levels and utilize 6000 episodes in total. The first level comprises of 3000 episodes while the second and third levels of 1500 episodes.

As shown in Table 7.8, all methods except the Unitary Timestep, Limit State Space (UTLSS) method result to zero hotspots. Also hierarchical methods except the Unitary Timestep, Limit State Space (UTLSS) method that results to 0.2 hotspots, outperform the non hierarchical regarding all metrics even with less episodes. We executed the non hierarchical method for 10000 episodes while the hierarchical methods were executed for 6000 episodes in total. Again the Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) seems to be the most dominant, achieving the best results regarding all metrics.

As reported in tables 7.7 and 7.8 all hierarchical methods reduce the Average Delay per Flight and the Total Delay when utilizing 3000 episodes at the first level, compared to the case where the first level comprises of 5000 episodes. Regarding the Number of Regulated Flights, the Unitary Timestep, Limit State Space (UTLSS) and the Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) methods manage to reduce them whereas the Unitary Timestep Concurrently Update Levels, Limit State Space (UTCULSS) method increases them when the first level comprises of 3000 episodes, in comparison with the case where the first level comprises of 5000 episodes. Unfortunately as already stated the Unitary Timestep, Limit State Space (UTLSS) method results to 0.2 hotspots when it is executed for 3000 episodes at the first level. All other methods at all reported cases manage to eliminate hotspots.

Further to the quality of solutions, Fig.7.5-Fig.7.6 provide the distribution of delays to flights. All methods reduce drastically the number of regulated flights, while moving from small to large delay intervals.

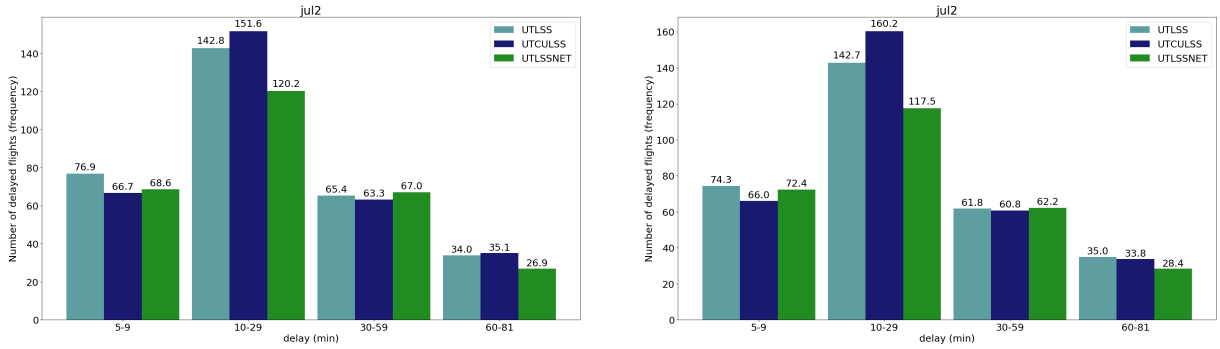
As depicted in Fig. 7.5 the Descending Timesteps (DT) and Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) methods reduce the number of flights with delay in the 60-81 interval when utilizing 3 levels, compared to the 2 levels case. Whereas the Unitary Timestep, Limit State Space (UTLSS) and Unitary Timestep, Concurrently Update Levels, Limit State Space (UTCULSS) methods increase the number of flights with



(a) 2 levels **(b) 3 levels**

Figure 7.5: The distribution of delays to flights for 2 (case (a)) and 3 (case (b)) levels and 10000 episodes at each level. The x axis shows the delay imposed while the y axis corresponds to the number of flights.

delay in the 60-81 interval when utilizing 3 levels, in comparison with the 2 levels case.



(a) 5000 episodes at first level **(b) 3000 episodes at first level**

Figure 7.6: The distribution of delays to flights for methods that utilize unitary timestep and limitation of the state space explored, with 3 levels and 2 levels (case (b)). The first level in case (a) comprises of 5000 episodes, whereas in case (b) it comprises of 3000 episodes, while the second and third levels of 1500 episodes. The x axis shows the delay imposed while the y axis corresponds to the number of flights.

As depicted in Fig. 7.6 the Unitary Timestep, Concurrently Update Levels, Limit State Space (UTCULSS) method reduces the number of flights with delay in the 60-81 interval when utilizing 3000 episodes at the first level, compared to the 5000 episodes at first level case. Whereas the Unitary Timestep, Limit State Space (UTLSS) and Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) methods increase the number of flights with delay in the 60-81 interval when utilizing 3000 episodes at the first level, in comparison with the 5000 episodes at first level case.

8. CONCLUSIONS AND FURTHER WORK

In this thesis we formulated the problem of resolving demand-capacity imbalances in the ATM as a hierarchical multiagent Markov Decision Process (MA-MDP) at multiple levels of abstraction.

We also explored the effectiveness of alternative hierarchical multiagent reinforcement learning methods and alternative schemes of transferring experience between levels, towards reducing delays and number of delayed flights. We demonstrated the effectiveness of our methods by evaluating them on one of the hardest real-world scenarios available to us. Experimental results show the potential of the proposed methods, in terms of efficiency (i.e. speed of convergence) and effectiveness (in terms of quality of solutions achieved) also compared to non hierarchical state of the art methods presented in [1].

By exploiting state-action abstractions we managed to achieve faster convergence and produce more qualitative solutions compared to non hierarchical state of the art methods presented in [1]. The DT method was outperformed by methods using unitary time step and limitation of the state space. Methods using unitary time step manage to gather experience to an abstract state regarding many ground states that abstract to it. By doing so abstract levels navigate to more promising parts of the state space compared to the Descending Timesteps (DT) method. By limiting the state space based on the solution provided by the previous level we were able to achieve faster convergence and more thorough exploration of promising parts of the state space. According to experimental results the Unitary Timestep, Limit State Space, No Experience Transfer (UTLSSNET) method achieved the most qualitative results. By working independently of the experience gained at previous levels this method was able to achieve better exploration in relation to other methods, as the exploration of the other methods was biased by the Q-values towards the solution of the previous level. Thus in order to transfer effectively and exploit experience gained at previous levels we should consider more sophisticated methods.

A direction of future work is to consider methods that assign weights while transferring the Q values between levels, depending on the distance between corresponding states s_i^L and s_i^{L+1}. A possible approach is kernel-based function approximation methods. Such methods utilize kernel functions to numerically express how relevant knowledge about any state is to any other state.

Another possible direction of future work is to run multiple instances of each level and aggregate the produced Q tables before transferring the experience to the next level. As instances will probably explore different parts of the state space, we hope to achieve more thorough exploration of the state space and thus produce more qualitative solutions compared to methods presented in this thesis and in [1].

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
RL	Reinforcement Learning
MA	Multiagent
MARL	Multiagent Reinforcement Learners
H-MARL	Hierarchical Multiagent Reinforcement Learners
ATM	Air Traffic Management
TBO	Trajectory Based Operations
DCB	Demand Capacity Balance
MDP	Markov Decision Process
MA-MDP	Multiagent-Markov Decision Process
IRL	Independent Reinforcement Learners
DT	Descending Timesteps
UTLSS	Unitary Timestep, Limit State Space
UTCULSS	Unitary Timestep, Concurrently Update Levels, Limit State Space
UTLSSNET	Unitary Timestep, Limit State Space, No Experience Transfer
TDC	Total Duration in Congestions
DC	Delay Cost

REFERENCES

[1] G. Vouros, T. Kravaris, A. Bastas, K. Blekas, C. Spatharis, J. Manuel Cordero Garcia, G. Andrienko, and N. Andrienko. Resolving congestions in the air traffic management domain via multiagent reinforcement learning methods.

[2] Gennady Andrienko, Natalia Andrienko, Peter Bak, Daniel Keim, and Stefan Wrobel. *Visual analytics of movement*. Springer Science & Business Media, 2013.

[3] Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 165–172. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[4] Robert W Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[5] Andrew J Cook and Graham Tanner. European airline delay cost reference values. 2015.

[6] Eurocontrol. Air traffic flow and capacity management (atfcm), 2011.

[7] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234. Citeseer, 2002.

[8] Jelle R Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(Sep):1789–1828, 2006.

[9] Collaborative reinforcement learning for resolving hotspots in the air traffic management domain. Master's thesis, National and Kapodistrian University of Athens, 2017.

[10] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

[11] Nicholas K Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, volume 8, pages 752–757, 2005.

[12] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[13] George Konidaris and Andrew G Barto. Efficient skill learning using abstraction selection. In *IJCAI*, volume 9, pages 1107–1112, 2009.

[14] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM, 2004.

[15] Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

[16] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[17] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[18] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[19] Natalia Hernandez-Gardiol and Sridhar Mahadevan. Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1047–1053, 2001.

[20] Csaba Szepesvari, Richard S Sutton, Joseph Modayil, Shalabh Bhatnagar, et al. Universal option models. In *Advances in Neural Information Processing Systems*, pages 990–998, 2014.

[21] Jonathan Sorg and Satinder Singh. Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 31–38. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[22] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[23] Daniel Rasmussen, Aaron Voelker, and Chris Eliasmith. A neural model of hierarchical reinforcement learning. *PloS one*, 12(7):e0180234, 2017.

[24] Benjamin Van Roy. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Ph. D. Dissertation, Massachusetts Institute of Technology, 1998.