Reinforcement Learning in Non-Stationary Environments

THESIS

submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

delivered by

Université de Toulouse
Institut Supérieur de l'Aéronautique et de l'Espace

in the field of Artificial Intelligence.

Presented and defended by

Erwan Lecarpentier

on July 6th, 2020.

Advisors
Emmanuel Rachelson and Guillaume Infantes

Acknowledgments

No words could describe how thankful I am to all the people who surrounded me during the realization of this PhD thesis.

First of all, I would like to thank Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson , my thesis advisors. I was very lucky to be supervised by such a wonderful trio. All the ingredients were there for me to have a great time doing my thesis. There were the technical aspects, we sometimes spent evenings writing equations on a board because we were so passionate about the subject. There were the hindsight aspects, the discussions together really made me grow and I now have a totally different perspective on the field and on related things such as what it is to be a researcher. There were the friendly aspects, as we are human first of all and I am glad I can say we successfully mixed work, fun and friendship during those four years. Overall I am very grateful to my advisors and really want to thank them for the great times we have had together.

I would like to thank Marc Toussaint, Vien Ngo, Peter Englert, Matthew Bernstein, Andrea Baisero, and Hung Ngo from the Machine Learning & Robotics Lab of the University of Stuttgart. I had the chance to discover the field of reinforcement learning with them during my Master's thesis and today I think that it is largely thanks to them that I developed the taste for research and that I decided to do a doctorate. Again, thank you all for welcoming me in Stuttgart for a year and I cant wait to see you again in conferences or others.

I would like to thank all the people of ISAE-SUPAERO and ONERA who contributed to make every day an over-great day, full of fun and productivity: Alexandre, Anass, Andrea, Andrés, Anh-toan, Anne, Antoine, Aymeric, Bastien, Carlos, Caroline, Clement, David, Dennis, Dmitry, Doriane, Eyal, Emmanuel, Florestan, Florian, Franco, François, Francesco, Frédéric, Guillaume, Heinrich, Hugo, Ilia, Ilyass, Jasdeep, Jean-Alexis, Juan jose, Lea, Luca, Mélody, Mina, Narjes, Paolo, Paul, Pierre-Antoine, Sana, Sandrine, Selma, Sophia, Thibault, Tristan, Valentin, Vatsal, and Zoe. I feel frustrated to write only your name in this list, as each one of you brought me so much and I would like to write so much about you. Hence I save those personal acknowledgments in my head and I promise to tell each one of you when I will meet you again, which doubles the appeal of this prospect.

I would like to thank Dave Abel, Kavosh Asadi, Yuu Jinnai, and Michael Littman from Brown University who welcomed me as a visiting PhD student during winter 2019. They largely contributed to the realization of the work reported in Chapter 5 and it was a real pleasure to share with them, from a human, cultural and scientific perspective.

I would like to thank Olivier Buffet, Tristan Cazenave, Aurélien Garivier, Émilie Kaufmann, Régis Sabbadin, and Bruno Zanuttini who accepted to be the reviewers of this PhD thesis and largely contributed to many improvements of this dissertation. Also, sharing with them in conferences and meetings has always been a great source of inspiration and of very pleasant moments.

I have been fortunate to find many many friends during my life and I believe friendship to be responsible for a good part of who we are at any given time. Thus, generally, I want to thank all my friends, for everything they have given me. My long time friends: Alex, Arthur, Aymeric, Charlo, Chloé, David, Éléonore, Gaëtan, Guillaume, Jérémy, Laura, Louis, Martha, and Pierre.My friends from preparatory classes: Antoine, Aziliz, Camille, Charles, Clémence, Coline, Coline, Edern, Étienne, Fabian, Fabian, Florent, Frédérik, Gabriel, Gabrielle, Guillhem, Jeanne, Julia, Kévin, Kévin, Lénaïc, Mathieu, Matthieu, Matthieu, Mohammed, Nadège, Nafie, Nicolas, Nicolas, Nolwenn, Romane, Ronan, Tangi, and Titouan.My friends from engineering school: Alicia, Amélie, David, Florestan, Jean-Charles, Jimi, Maxime, Murielle, Noémie, Quentin, Samuel, Thibault, Valentin, and Vivien.My friends from Stuttgart: Anas, Eliza, Grigor, Jan, Lysander, Moritz, Petrus, Sara, Tarek, Thu, and Yasmine.And my friends from other horizons: Arthur, Clémence, Éléonore, Guillaume, Heinrich, Huggo, Jojo, Mathou, Patrick, and Vincent.

I would like to thank my family, for their love, their support, and their endorsement. I have always had the chance of feeling loved and encouraged among you, and I think it makes a huge difference in the way I developed in all aspects of life. I was fortunate to have wonderful parents and equally wonderful siblings. They filled my life with plenty of kindness, affection, devotion and even pets. Talking about pets, I take this opportunity to thank, in chronological order, the amazing creatures we have shared our home with: Wisty, Réglisse, Walter, Câline, Frimousse, Sweety, Thalia, Titus, Safran and all its family, Malibu, Actimel, and Papuche. I forget Roberto and its huge family of hen, chicken and chicks, but, as birth control is difficult in a household of gallinaceous, I am unable to remember their dates of birth. I think taking care of pets also participates in a good personal development and I want to thank them all for accepting living with us. I also want to extend this acknowledgment to my grandmothers, my cousins, my aunts, my uncles and everyone in my family as I have always felt a benevolent presence, coming from all of you, that definitely made me more confident and optimistic in life. Last but not least, I want to thank Tata Isabelle, for the love she brought when she raised me. Being part of a big, united, family as mine has always been an asset to me and I want to thank you all for that.

I reserve my last acknowledgments to Emma, my new family here in Toulouse. She supported me during three years and brought me love, along with an exquisite point of view on life in general. I would be a whole different person if our paths hadn't crossed and I want to thank you for that. As I am talking about family, I also must thank Poun's, the little ball of fur and character that follows us and that fills us with happiness.

Generally, I want to thank everyone that I met during the thesis. I am sure I must be forgetting some people and I apologies in advance. The good news is that life goes on and I cant wait to have more great experiences with all of you and plenty of others so that I could re-write (at least in my heart) pages of acknowledgments.

Contents

List of Acronyms **ix**

1 Preliminaries **1**

2 Introduction **5**

2.1 Model and optimization criterion 5

2.2 Planning 17

2.3 Learning 26

3 The planning *vs.* re-planning trade-off in stationary Markov Decision Processes **33**

3.1 State of the art 34

3.2 Open-loop control 38

3.3 Open-loop search trees 41

3.4 Open-Loop Tree search Algorithm 44

3.5 Theoretical Analysis of the Open-Loop Tree search Algorithm 49

3.6 Empirical Analysis of the Open-Loop Tree search Algorithm 53

3.7 Conclusion 58

4 Planning in gradually evolving Markov Decision Processes **61**

4.1 State of the art 62

4.2 Non-Stationary Markov Decision Process 67

4.3 Worst case approach 72

4.4 Risk Averse Tree Search algorithm 76

4.5 Experiments 85

4.6 Conclusion 88

5 Learning in abruptly evolving Markov Decision Processes 91

5.1 State of the art . . . 93

5.2 Framework . . . 95

5.3 Lipschitz continuity of Q-functions . . . 100

5.4 Transfer using Lipschitz continuity . . . 105

5.5 The Lipschitz RMAX algorithm . . . 111

5.6 Experiments . . . 117

5.7 Conclusion . . . 120

6 Conclusion 123

A Proofs of the dissertation 127

A.1 Proofs of Chapter 2 . . . 127

A.2 Proofs of Chapter 3 . . . 128

A.3 Proofs of Chapter 4 . . . 130

A.4 Proofs of Chapter 5 . . . 136

B Complete version of the MCTS algorithm 151

C Additional experimental results for the OLTA algorithm 155

D Additional information on the experiments of Chapter 4, Section 4.5 159

E Additional information on the experiments of Chapter 5, Section 5.6 161

Bibliography 171

List of Figures

2.1 Cart-pole MDP illustration 10

2.2 An example of MCTS search tree 21

2.3 Illustration of one step of the MCTS procedure 23

2.4 Illustration of the reinforcement learning principle 27

2.5 Common reinforcement learning schemes 29

3.1 Reuse scheme of a closed-loop tree. 40

3.2 General representation of an open-loop tree. 43

3.3 Reuse scheme of an open-loop tree. 45

3.4 Upper bound on the probability of failure displayed for several depths 52

3.5 Illustration of the 1D track environment 54

3.6 Experimental comparison of OLTA to OLUCT on the discrete 1D track environment 55

3.7 Illustration of the continuous Physical Traveling Salesman Problem 56

3.8 Experimental comparison of OLTA to OLUCT on the continuous PTSP environment 57

4.1 Tree developped by the RATS algorithm 77

4.2 Illustration of the relaxed problem solved by the RATS algorithm 78

4.3 The non-stationary bridge environment 85

4.4 Average results for the non-stationary bridge experiment for several values of epsilon. 87

4.5 Distributional results for the non-stationary bridge experiment for three specific values of epsilon. 87

5.1 The T-shaped MDP transfer task. 97

5.2 Comparison of the RMAX upper bound, the MaxQInit upper bound and the LRMAX upper bound. 100

5.3 Illustration of the prior knowledge on the maximum pseudo-distance between models 115

5.4 The tight grid-world environment. 117

5.5 Experimental results of RMAX, LRMAX, MaxQInit and LRMaxQInit in the "tight" lifelong Reinforcement Learning experiment. 118

C.1 Illustration of the continuous 1D track environment. 155

C.2 Experimental comparison of OLTA to OLUCT on the continuous 1D track environment 156

C.3 Illustration of the discrete Physical Traveling Salesman Problem 157

C.4 Comparison between OLUCT and OLTA on the discrete PTSP for varying values of the misstep probability q. 158

List of Tables

2.1 Summary of the different policy classes. 13

4.1 Expected return $\mathbb{E}\left(\sum r\right)$ and CVaR at 5% for RATS, DP-snapshot and DP-NSMDP for $\epsilon \in \{0, 0.5, 1\}$. 88

D.1 Summary of the number of experiment repetition, number of sampled tasks, number of episodes, maximum length of episodes and upper bounds on the number of collected samples. 159

E.1 Summary of the number of experiment repetition, number of sampled tasks, number of episodes, maximum length of episodes and number of collected samples. 161

List of Acronyms

CMDP	Contextual Markov Decision Process
CVaR	Conditional Value at Risk
DP	Dynamic Programming
GVGP	General Video Game Playing
HMMDP	Hidden Mode Markov Decision Process
HS3MDP	Hidden-Semi-Markov-Mode Markov Decision Process
HOLOP	Hierarchical Open-Loop Optimistic Planning
LC	Lipschitz Continuous
LRMAX	Lipschitz R-Max
MC	Monte Carlo
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
ML	Machine Learning
MOMDP	Mixed Observability Markov Decision Process
NSMDP	Non-Stationary Markov Decision Process
OLETS	Open-Loop Expectimax Tree Search
OLOP	Open-Loop Optimistic Planning
OLTA	Open-Loop Tree search Algorithm
OLUCT	Open-Loop Upper Confidence bound applied to Trees
PAC-MDP	Probably Approximately Correct in Markov Decision Processes
POMDP	Partially Observable Markov Decision Process
POMCP	Partially Observable Monte Carlo Planning
PTSP	Physical Traveling Salesman Problem
RATS	Risk Averse Tree Search
RDV	Return Distribution Variance

RL	Reinforcement Learning
SDM	State Distribution Modality
SDSD	State Distance to State Distribution
SDV	State Distribution Variance
SMDP	Semi Markov Decision Process
THTS	Trial-based Heuristic Tree Search
TSP	Travelling Salesman Problem
UAV	Unmanned Aerial Vehicle
UCB	Upper Confidence Bound
UCT	Upper Confidence bound applied to Trees

Chapter 1

Preliminaries

Planning a navigation route from point A to point B with a boat on a calm sea is a stationary problem. The conditions do not change over time. If we consider unstable currents that may progressively deviate the trajectory of the boat, then the problem becomes non-stationary. Its properties evolve smoothly with time as the water flow changes. If we add a violent wind to this task with unpredictable gusts that may push the boat in any direction, then the problem is still non-stationary but its properties evolve abruptly with time. One can characterize the temporal evolution of a task in different ways. The three operation modes described in this boat example may lead to different ways of addressing the problem. This dissertation is a study of the possible resolutions of a task under different temporal evolution modes.

How to act in an environment that can change over time?

Before trying to answer this quite general question, we will define our terms and lay a broad framework of this study in the next section. Then, we will describe the progression of ideas and the contributions made during the realization of this work, in conjunction with the organization of this thesis.

Problem statement and summary of contributions

We are asking the question of acting in an environment that can change over time. What is acting in the first place? In this dissertation, acting is taking decisions or performing actions at discrete temporal slices, that we will call *decision epochs*. Since we just wrote that decisions were taken at discrete decision epochs, we restricted the topic of this dissertation to a field best known as sequential decision making. Who is acting? An agent is acting. The word agent, in our understanding, stands for any entity bound to take actions or decisions and therefore endowed with some kind of decision rule for that purpose. We will only deal with the case of a single agent. Where is the agent acting? Within an environment, which is defined by its state and a set of rules characterizing how its state change between each decision epoch. Intuitively, those rules take into account the actions taken by the agent. Thus, the agent may have the possibility to influence the way things evolve through its actions. The possibility for an environment to be non-stationary, as a central concern of this dissertation, will be modeled by this set of rules which can potentially evolve through time. Consequently, as an agent takes actions at different decision epochs, the reaction of the environment may not be the same for each one of them. The study we undertake takes into account different sets of hypotheses on the way temporal evolution is characterized. These hypotheses give rise to different contributions that we now present, along with the organization of the dissertation.

In the first chapter, we formally define the framework of our analysis. We start from the conceptual definition of what an agent and an environment are. We introduce the formal model of a Markov Decision Process (Puterman, 2014), used to represent those two entities and define an optimality criterion for an agent's decision rule. Finally, we recall the notions of learning (Sutton and Barto, 2018) and planning (Puterman, 2014; Ghallab, Nau, and Traverso, 2016), seen as mechanisms used by an agent to improve its decision rule with respect to this criterion.

In the second chapter, we focus on the case of unchanging environments, *i.e.*, characterized by little to no temporal evolution. We will see this assumption of stationary environment as a first step in our analysis. In such a case, one can leverage the fact that things do not evolve over time to extend the predictions an agent makes at some point to future decision epochs. In other words, what is valid now will always be valid. In this line of thought, we propose a framework for a planning agent to stop planning once a plan has been conceived. The plan is then followed without further refinement, reducing the computational complexity of the algorithm. This kind of behavior can be seen as lazy planning, in reference to the general programming principle of lazy evaluation. Subsequently, planning can be re-triggered if a deviation from the original plan is detected. We analyze our approach in terms of optimality and show that strong guarantees can be obtained. Overall, this first contribution consists in a lazy planning procedure, implementing a planning re-planning trade-off suited for slowly evolving environments. This chapter stems from the work carried out in Lecarpentier, Infantes, Lesire, and Rachelson (2018).

In the third chapter, we tackle non-stationary problems through the scope of continuous, gradual, evolution. The environment is now allowed to change smoothly over time. We formalize this assumption with a Lipschitz continuity hypothesis of the transition and reward functions of the Markov Decision Process (MDP) with respect to time. In other words, we put a mathematical constraint on the evolution rule of the environment, allowing it to evolve between subsequent decision epochs but forcing it to remain similar to what it was before. This makes sense from a physical point of view if we consider gradually changing conditions. Given that things evolve quietly, it becomes possible to predict the worst case of future outcomes from a planning perspective. To that end, our contribution here is the design of a risk averse planning agent that tries to avoid the pitfalls allowed by the gradual evolution. This agent is conservative and we demonstrate its decision rule to converge to the optimal one of the worst case scenario. This can be related to a minimax strategy against an adversary, given that the adversary is the environment itself. This chapter stems from the work carried out in Lecarpentier and Rachelson (2019).

In the fourth chapter, we complete our analysis by looking at abruptly changing problems. This comes as a complement to the previous chapter, where changes were constrained to be small. Here, things can become chaotic as we allow for any evolution of the environment. We show that this extreme non-stationary case can be addressed by reasoning on the similarity between "before" and "after" the appearance of a change. Precisely, we develop a learning agent that learns everything from scratch every time a change occurs. During learning, if the data appear to be similar to those gathered before the change, the algorithm re-uses the previously harvested knowledge. This mechanism, commonly referred to as transfer, allows to speed-up the convergence to the optimal behavior. This practical contributions comes from a theoretical analysis that we also carry out in this fourth chapter. We study some properties of the way two distinct environments can differ from one another. Precisely, we propose a metric to measure the distance between environments and show that small distances allow for a large amount of knowledge transfer. This chapter stems from the work carried out in Lecarpentier, Abel, Asadi, Jinnai, Rachelson, and Littman (2020).

Finally, we conclude and open on future questions raised during the realization of this work. Importantly, all the proofs of the original theoretical results of the dissertation, *i.e.* Lemmas and Theorems, are reported in the Appendix, Chapter A. We thought that this would allow the reader to keep a more focused reading flow and to better appreciate the presented results.

Chapter 2

Introduction

Sommaire

2.1 Model and optimization criterion 5
2.1.1 Model 6
2.1.2 Policy 12
2.1.3 Value function 13
2.1.4 Optimality criterion 15
2.2 Planning 17
2.2.1 Definition 17
2.2.2 Dynamic programming 18
2.2.3 Monte Carlo Tree Search 20
2.3 Learning 26
2.3.1 Definition 26
2.3.2 RMAX: an example of Reinforcement Learning algorithm 28

In this chapter, we formally define the framework of this thesis, which falls within the field of study of sequential decision making. First, we define the concepts of agent and environment along with the notions of action and state. By formalizing the ways these elements interact, this leads us to the definition of the Markov Decision Process (Puterman, 2014) model, that we use throughout this dissertation. Secondly, we introduce an optimality criterion for an agent's decision rule. Finally, we present the notions of learning (Sutton and Barto, 2018) and planning (Puterman, 2014; Ghallab, Nau, and Traverso, 2016), allowing an agent to improve its decision rule with respect to this criterion.

2.1 Model and optimization criterion

In this section, we lay the basic framework onto which we built our analysis. We start with the conceptual descriptions of an agent and an environment with its components. Then we define the formal framework of Markov Decision Processes that comprehends all those elements. Finally we define a decision rule as an object we want to optimize, along with the optimization criterion.

2.1.1 Model

2.1.1.1 Agent

An *agent* is any entity able to *sense*, to *decide* and to *act* (Ghallab, Nau, and Traverso, 2016). *Sensing* is done with respect to the environment. It is the acquisition of an observational input by the agent. This observation can be more or less precise, depending on the construction of the agent. Ability to sense is optional in the sense that an agent could have no senses at all. This extreme case may never be encountered but we keep it in our description of an agent for the sake of generality. *Deciding* is the application of any decision rule to select an action. This supposes a set of actions and maybe a criterion on which the agent can base its decision. *Acting* is simply the application of the selected action in the real world. Doing so may have an effect on the state of the world and maybe on the state of the agent itself.

Example 2.1 (Random agent)**.** The *random agent* is an agent whose decision rule is to select actions uniformly at random among the set of available actions. It may have the ability to sense but does not use the acquired observations.

Practically, we design agents for special tasks. Having a task means having a goal. Therefore, the decision rule of the agent should be designed so that the correct actions are performed at the correct moments to eventually reach this goal. The whole problem of sequential decision making is to construct a decision rule that meets our needs. The random agent can be seen as a baseline, representing a simple decision rule for which no design effort has been made. On some practical examples, this random decision rule may even be optimal, for instance, playing rock paper scissors against a second random policy. However, acting randomly is generally sub-optimal and, for this reason, this decision rule is often seen as a baseline representing a poor performance. We delay our answer to the question of the design of a decision rule to a later section and detail now our conceptual view of an environment.

2.1.1.2 Environment

An agent evolves within an *environment*. Substantially, it encompasses everything that is not the agent. Let us consider such a setting. If we add a second agent, an ambiguity appears in that there are now two decision makers. In this dissertation, we will never consider multiple agents and always center the analysis on a single entity. What could be viewed as an agent but whose actions are not determined by the decision process will be referred to as *external agents*. Although they fall in the definition of Section 2.1.1.1, we will see them as parts of the environment. We will always consider the simplified dichotomy of a single agent acting in an environment. Conversely, research on the topic of multi-agent systems aim at designing the decision rules of multiple agents to reach particular objectives. Often, the necessity to distinguish agents from their peers is due to the fact that they do not share the same set of information. This constraint makes it impossible to take global decisions as each agent will have to behave with respect to a subset of the total information. For instance, a fleet of autonomous cars sharing non-perfect information (*e.g.*, because of delayed communications), must be considered as several agents for the sake of realism. In our case, if we were to control several cars, all sharing their information perfectly as a single entity, then we would consider a single-agent system. The selected action at each decision epoch would be the joint actions performed by each car. Other cars, not controlled by us but by humans or autonomous robots for instance, are viewed as parts of the environment.

2.1.1.3 State, action and transition function

The *state* is a set of information characterizing the agent and / or the environment. The *actions* taken by the agent have the ability to modify the state. The way the state evolves given an action of the agent constitutes a rule that we call *transition function.* Given a current state and an action, the transition function outputs a resulting state from the application of this action at the current state. For instance, the rules of the board game of chess define the resulting state from the moving of a pawn given a state of the board.

It is important to make the distinction between the state and the whole set of information necessary to describe the interaction process of agent and an environment. Take the example of a maze where the state is the position of the agent. The information contained in the state is not enough to describe the effect of an action as it does not contain, *e.g.*, the information of the position of the walls. As a result, solely relying on the state, one cannot tell if moving forward would result in actually moving forward since one does not know if there is a wall in front of the agent. The remainder of the information is encoded “inside” the transition function which, in the example of a maze, “possesses” the information of the position of the walls. However, it is important to remark that, given the transition function, the information of the state is enough to describe any interaction between the agent and the environment. This fact is important as we will later assume that an agent can observe the state and base its decisions on this information. Thus, if the transition function is known, an agent can model any interaction *exactly*, based on the information of the state. Removing a part of the total information from the state allows to model problems where things are hidden and is thus more general.

2.1.1.4 Reward function

Here we introduce the important concept of *reward.* In neuroscience, the reward system is a set of mechanisms happening in our brain, responsible for the cognition, the sensing, of pleasurable stimuli like happiness or euphoria. After lunch, the feeling of satiety is pleasant and rewards us from having eaten. The same goes for drinking water, which creates a satisfying feeling. Sexual contact and parental investment are inherently pleasurable. Those three mechanisms are known as primary reward cognition (Schultz, 2015). They allow species to learn healthy behaviors for their survival, such as feeding, hydrating and copulating. Conversely, some natural mechanisms act as punishment. Pain is felt by any animal touching an injuring object, such as fire, allowing it to immediately stop doing so with a mere reflex. The feeling of hunger is unpleasant and urges us to eat. It is caused by a temporary decrease of the blood sugar level, a natural mechanism that triggers this feeling. Similarly to the way reward encourages reproduction of rewarding events, punishment discourages the converse.

Reward and punishment are useful for learning, which is their primary function. The agents considered in this dissertation will also follow this rule. We suppose the existence of a reward function. This function encompasses the notions of reward and punishment presented above in a single scalar signal. The higher this signal, the more rewarding for the agent. Conversely, the lower the signal, the more punitive it is. Quite intuitively, the reward signal depends on the state and the actions selected by the agent. This function is going to be the core of the learning process and materializes the goal of a task: collecting more rewards!

2.1.1.5 Markov Decision Process

In Sections 2.1.1.1, 2.1.1.2, 2.1.1.3 and 2.1.1.4, we introduced the concepts of agent, environment, state, action, transition function and reward function that constitute the basis of our framework. In this section, we define those elements formally, into a single object called a *Markov Decision Process* (MDP) (Puterman, 2014). An MDP is an "idealized" model of the world. Its mathematical simplicity made it the predilection tool for derivation of accurate mathematical results in the field of sequential decision making. It allows to answer questions like "What is an optimal behaviour?", "When can I tell that my agent learned an accurate representation of the world?", "How long will it take to reach a certain performance?", *etc.* The formal definition of an MDP follows in Definition 2.1 after a quick notation introduction.

Notation 2.1 (Set of probability distributions). Let X be a measurable space. We will write $\mathcal{P}(X)$ the set of probability distributions on the set X.

Notation 2.2 (Functional spaces and functions). For X and Y any two non-empty sets, we denote by $\mathcal{F}(X, Y)$ the set of functions defined on the domain X with Y as a codomain. For $f \in \mathcal{F}(X, Y)$, we employ the following notation with the understanding that "f is a function in $\mathcal{F}(X, Y)$ mapping each element $x \in X$ to $f(x) \in Y$":

$$\begin{array}{rccl} f: & X & \rightarrow & Y \\ & x & \mapsto & f(x) \,. \end{array}$$

Definition 2.1 (Markov Decision Process). A *Markov Decision Process*, is defined by a 4-tuple $\{\mathcal{S}, \mathcal{A}, T, r\}$, where:

- $\mathcal{S}$ is a set of states;
- $\mathcal{A}$ is a set of actions;
- T is a transition function mapping state-action pairs to the conditional probability distribution on the resulting state:

$$\begin{array}{rccl} T: & \mathcal{S} \times \mathcal{A} & \rightarrow & \mathcal{P}(\mathcal{S}) \\ & (s, a) & \mapsto & \mathbf{Pr}\left(\cdot \mid s, a\right) \,; \end{array}$$

- r is a reward function mapping state-action-state triples to the reward associated to the transition from state s to state s' by application of action a:

$$\begin{array}{rccl} r: & \mathcal{S} \times \mathcal{A} \times \mathcal{S} & \rightarrow & \mathbb{R} \\ & (s, a, s') & \mapsto & r(s, a, s')\,; \end{array}$$

For convenience, we will adopt the following notations in the remainder of the dissertation.

Notation 2.3. For all $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, we denote by $T^a_{s\cdot} \triangleq T(s, a)$ the conditional probability distribution on the resulting state from the application of action a at state s. Further, $T^a_{ss'} = \mathbf{Pr}\left(s' \mid s, a\right)$ denotes the probability to reach state s' when applying action a at state s. Additionally, we denote by $r^a_{ss'} \triangleq r(s, a, s')$ the scalar reward signal associated to the transition from state s to state s' by application of action a.

An important feature of an MDP is the *Markov property*. This property states that the transition probability to $s' \in \mathcal{S}$ while undertaking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ is only conditioned by the pair (s, a), and therefore independent from the previously encountered state-action pairs. This is a non-trivial assumption that implies that the current state-action pair contains all the information required for the environment to deduce the next state. The decisions are taken by the agent among the set of actions $\mathcal{A}$. Actions influence the evolution of the system from a particular state, as seen in the conditioning of the transition function T by the state-action pair. Collected rewards also depend on the current state-action pair, as well as the resulting state from the transition. MDPs are objects to model discrete time sequential decision making problems, where actions are taken at discrete decision epochs. We denote by $\mathcal{T} = \{0, \dots, H\}$ the *set of decision epochs* with H the *horizon*. It is either finite, if $H \in \mathbb{N}$ is a finite number, or infinite, if $H = \infty$. The set of decision epochs is voluntarily omitted in Definition 2.1. Its existence is implicit and — unless specified otherwise — we will take $\mathcal{T} = \mathbb{N}$. As a result, an MDP is a description of the dynamics, *i.e.*, the transitions and rewards, of the interaction process between an agent and an environment. Note that the transition and reward functions are stationary, *i.e.*, they do not depend explicitly on the decision epoch. We will sometimes refer to $s_t, a_t \in \mathcal{S} \times \mathcal{A}$ as the state-action pair of decision epoch t. Similarly, the collected reward corresponding to the transition (s_t, a_t, s_{t+1}) is denoted by r_t. We call *trajectory* a sequence of state-action-reward triples

$$\{(s_t, a_t, r_t)\}_{t \in \mathcal{T}}$$

collected while interacting with the MDP by selecting actions at each decision epoch. As a result, trajectories comply with the following rules:

$$\begin{aligned} s_{i+1} &\sim T^{a_i}_{s_i\cdot}, \\ r_i &= r^{a_i}_{s_i s_{i+1}}. \end{aligned}$$

We also define the notion of *terminal states* as specific states that have the property to end the realization of a trajectory before reaching the horizon H of the MDP. They could correspond to the reaching of a goal, or a failure situation where the system is defective and no more actions should be taken. For instance, a checkmate state in the game of chess is a terminal state. Overall, a trajectory ends either by reaching a terminal state or the horizon H of the process in the finite horizon case.

Notation 2.4. By convention, we use the notation s' to refer to the resulting state from the application of an action in state s. This notation along with the indexing by decision epoch $s_i \to s_{i+1}$ indicate the notion of predecessor in the transition function. We may use them interchangeably.

Example 2.2 (The cart-pole MDP example)**.** Consider a pole rotating on a cart sliding along a single axis. The goal of the task is to balance the pole while controlling the force applied to the cart. We denote by x the position of the cart, $\dot{x}$ its velocity, θ the angle between the pole and the vertical, $\dot{\theta}$ the angular velocity, M the cart-mass, m the pole-mass and l the pole length as illustrated in Figure 2.1. The state of the cart is defined by $s \triangleq \begin{pmatrix} x & \dot{x} & \theta & \dot{\theta} \end{pmatrix} \in \mathcal{S}$

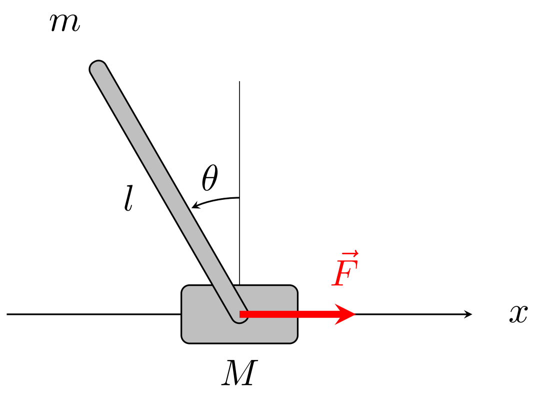


Figure 2.1: Cart-pole illustration with M the cart mass, m the pole mass, l the pole length, x the position of the cart on the horizontal axis, θ the angle between the pole and the vertical axis and $\vec{F}$ the force applied to the cart.

where $\mathcal{S} \equiv \mathbb{R}^4$. The action space is $\mathcal{A} \triangleq \{-F, 0, +F\}$ and corresponds to the magnitude and direction of the force vector $\vec{F}$ applied to the cart. The force is applied periodically with period τ, defining the set of decision epochs as $\mathcal{T} \triangleq [0, \tau, 2\tau, \dots, \infty)$. The dynamics of the system is defined by a differential equation obtained by applying the laws of mechanics. As a first order approximation, we use the Euler method to define the transition function of the MDP, resulting in the following set of equations:

$$s_{t+1} = s_t + \tau \begin{pmatrix} \dot{x}_{t+1} & \ddot{x}_{t+1} & \dot{\theta}_{t+1} & \ddot{\theta}_{t+1} \end{pmatrix}^\top,$$

with, in order of calculation, the following computations:

$$\begin{aligned}
\ddot{\theta}_{t+1} &= \frac{1}{l}\left(g\sin(\theta_t) - \cos(\theta_t)\frac{a_t + ml\dot{\theta}_t^2\sin(\theta_t)}{m+M}\right)\left(\frac{4}{3} - m\frac{\cos(\theta_t)^2}{m+M}\right)^{-1}, \\
\dot{\theta}_{t+1} &= \dot{\theta}_t + \tau\ddot{\theta}_{t+1}, \\
\ddot{x}_{t+1} &= \frac{a_t + ml\dot{\theta}_t^{\,2}\sin(\theta_t)}{m+M} - \frac{ml\ddot{\theta}_{t+1}}{m+M}, \\
\dot{x}_{t+1} &= \dot{x}_t + \tau\ddot{x}_{t+1},
\end{aligned}$$

where we denote by $s_t = \begin{pmatrix} x_t & \dot{x}_t & \theta_t & \dot{\theta}_t \end{pmatrix}$ the state indexed by the decision epoch $t \in \mathcal{T}$. Notice that, in this case, the transition function is deterministic. The next state is analytically derived from the current state-action pair and no stochasticity is introduced in this computation. The reward function is solely defined on the angle θ of the pole and is given by:

$$r^{a_t}_{s_t s_{t+1}} = \begin{cases} 1 \text{ if } \theta_{t+1} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \\ 0 \text{ else.} \end{cases}$$

In Example 2.2, the Markov property is verified since the resulting state s' depends only on the current state s and the action a. If the velocity of the cart and the angular velocity of the pole were non-observable to the agent, *i.e.*, $s = \begin{pmatrix} x & \theta \end{pmatrix}$, then the resulting state from a transition would not be deducible from a single state. Two subsequent states would allow to retrieve the velocity components which would in turn make this computation feasible. In such a case, the transition function would depend on the two previous states which is totally fine from a computational point of view but does not comply with the Markov property, hence, is not a valid MDP model.

In Definition 2.1, page 8, the reward function r defines the reward associated to a full transition $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$. Additionally, we define the *expected reward function*, representing the expected reward from applying an action in a state.

Definition 2.2 (Expected reward function)**.** Consider an MDP $\{\mathcal{S}, \mathcal{A}, T, r\}$, the *expected reward function* is defined as

$$\begin{array}{rccl} R: & \mathcal{S} \times \mathcal{A} & \to & \mathbb{R} \\ & (s, a) & \mapsto & R^a_s \triangleq \mathbb{E}_{s' \sim T^a_{s\cdot}} \left(r^a_{ss'}\right) . \end{array}$$

This definition being sometimes more convenient to deal with, we may use it interchangeably with the definition of the reward function in the remainder of the dissertation, allowing to define an MDP as $\{\mathcal{S}, \mathcal{A}, T, R\}$. Keep in mind that both co-exist, whatever the definition. Note that, if the expected reward function can be deduced from the reward function, the converse is not true.

Remark 2.1. No assumptions were made on the state-action space $\mathcal{S} \times \mathcal{A}$ of Definition 2.1, page 8. The set of admissible spaces is relatively broad and encompasses the following (Puterman, 2014):

- arbitrary finite sets;
- arbitrary countably infinite sets;
- compact subsets of finite dimensional Euclidean space;
- non-empty Borel subsets of complete, separable metric spaces.

The most "simple" class of state-action space may be finite sets such as $\mathcal{S} \times \mathcal{A} = \{s_1, \ldots, s_n\} \times \{a_1, \ldots, a_m\}$ where $n, m \in \mathbb{N}$. This example actually constitutes a very important problem class where theoretical results can be derived thanks to those properties. Most of the examples considered in this dissertation fall into that category for this particular reason.

Notation 2.5. In the case where $\mathcal{S}$ and $\mathcal{A}$ are finite sets, we respectively write $S \triangleq |\mathcal{S}|$ and $A \triangleq |\mathcal{A}|$ their cardinality.

Remark 2.2. The action space $\mathcal{A}$ introduced in Definition 2.1, page 8, contains all the actions the agent can take. In some problems, some actions may be illegal in some states, making the set of available actions dependent on the current state. If this is the case, we will denote by $\mathcal{A}(s) \subset \mathcal{A}$ the set of available actions at $s \in \mathcal{S}$.

2.1.2 Policy

A *policy* is a decision rule. In the most general case, we define a policy as in Definition 2.3.

Definition 2.3 (Non-stationary stochastic policy)**.** A *non-stationary stochastic policy* is a mapping,

$$\begin{array}{rccc} \pi : & \mathcal{S} \times \mathcal{T} & \to & \mathcal{P}(\mathcal{A}) \\ & (s, t) & \mapsto & \pi_t(\cdot \mid s) \; . \end{array}$$

A non-stationary stochastic policy provides a probability distribution over actions given a state $s \in \mathcal{S}$ and a decision epoch $t \in \mathcal{T}$. Taking a decision with respect to π amounts to sampling an action according to this probability distribution. Notice the conditioning of the distribution on the decision epoch t. This implies that the rule may change over time, *e.g.*, in a learning setting. If there is no decision epoch dependency, the policy is said to be *stationary*. Notice also the conditioning of the distribution on the state s. This accounts for the fact that an agent takes decisions according to the state it observes. To be even more general than Definition 2.3, one can also condition the decision rule on the whole history of sampled states from the first decision of the agent to the current one. This class of policy, called *history-based* policies, is discussed with more detail by Puterman (2014). Importantly, he shows that — for the optimality criterion we will later define — there exist a state-based policy that achieves the optimal performance. For this reason, we do not need to consider history-based policies and will restrict the scope of the considered policies in this dissertation to Definition 2.3. Additionally, the policy is said to be *deterministic* if the mapping is performed from $\mathcal{S}$ — and possibly $\mathcal{T}$ — to $\mathcal{A}$. A deterministic policy can be seen as stochastic, mapping each state or state-decision epoch pair to a Dirac distribution centered on a particular action. Without additional details, we will always refer to policies as stationary policies, only making the distinction between the deterministic case and the stochastic case. The four classes of policies introduced so far are summarized in Table 2.1.

Notation 2.6. We will denote by π_{random} the policy corresponding to the random agent of Example 2.1, page 6. Given a state $s \in \mathcal{S}$, π_{random} selects the action uniformly at random among the set of available actions $\mathcal{A}(s)$.

	Deterministic	Stochastic
Stationary	$\begin{array}{rccc} \pi: & \mathcal{S} & \to & \mathcal{A} \\ & s & \mapsto & \pi(s) \end{array}$	$\begin{array}{rccc} \pi: & \mathcal{S} & \to & \mathcal{P}(\mathcal{A}) \\ & s & \mapsto & \pi(\cdot \mid s) \end{array}$
Non-stationary	$\begin{array}{rccc} \pi: & \mathcal{S}\times\mathcal{T} & \to & \mathcal{A} \\ & (s,t) & \mapsto & \pi_t(s) \end{array}$	$\begin{array}{rccc} \pi: & \mathcal{S}\times\mathcal{T} & \to & \mathcal{P}(\mathcal{A}) \\ & (s,t) & \mapsto & \pi_t(\cdot \mid s) \end{array}$

Table 2.1: Summary of the different policy classes.

2.1.3 Value function

Previously, we roughly stated that our objective was to maximize the sum of all the rewards collected in trajectories. This brings us to the definition of the *total return* Z, as the summation of all the collected rewards along a single trajectory $\{(s_i, a_i, r_i)\}_{i\in\mathcal{T}}$ of finite horizon $H < \infty$,

$$Z^H \triangleq \sum_{t=0}^{H-1} r_t,\ H < \infty.$$

Notice that a trajectory may end by reaching a terminal state, prior to the horizon H of the process. We implicitly set the value of the rewards collected after such an event to 0, so that the definition complies with this case. In the infinite horizon case, where $H = \infty$, the return is ill-defined because of the possible divergence of the series. To allow a definition in that case, we introduce a *discount factor* $\gamma \in [0, 1)$ and define the *discounted return* as

$$Z_\gamma^\infty \triangleq \sum_{t=0}^{\infty} \gamma^t r_t.$$

Notation 2.7. We denote by $Z \triangleq Z_\gamma^\infty$ the infinite horizon discounted return. Without additional details, we always refer to the infinite horizon case and will specifically mention if it is the total or discounted case.

The discount factor γ acts as a weighting parameter in the discounted return. Weights decrease with the decision epoch of the collected reward in the summation. In other words, the further the reward is in the sequence of decisions, the less it contributes to the total sum. The discounted return is thus a criterion putting more importance on immediate rewards. Practically, this is a desirable feature, allowing to define the importance of long-term collected rewards via the value of γ. Without loss of generality, we will assume the reward function to be positive and bounded by a constant term $R_{\max} \in \mathbb{R}$, *i.e.*

$$0 \leq r_{ss'}^a \leq R_{\max},\ \forall s, a, s' \in \mathcal{S}\times\mathcal{A}\times\mathcal{S}.$$

From this hypothesis, one can easily show that the maximum achievable discounted return is given by

$$Z_{\max} \triangleq \frac{R_{\max}}{1-\gamma}.$$

This quantity is not always reachable in any MDP. $Z_{\max}$ corresponds to collecting $R_{\max}$ from the first decision epoch to infinity and constitutes an upper bound on the maximum achievable return. Our objective is to optimize a policy to maximize the value of the collected return as we will now see in our optimality criterion definition.

Return and discounted return are defined for a particular trajectory. Quite naturally, we define our objective as maximizing the *expected* return over multiple trajectories. In other words, we would like the highest return on average. We define the *value function of a policy* as the expected discounted return of this policy within an MDP.

Definition 2.4. Consider an MDP $M = \{\mathcal{S}, \mathcal{A}, T, r\}$, a set of decision epochs $\mathcal{T} = \{0, \dots, H\}$, $H \leq \infty$, a discount factor $\gamma \in [0, 1)$ and a non-stationary stochastic policy π. The value of π in M is defined for all $s \in \mathcal{S}$ by the *value function* V_M^π as

$$V_M^\pi(s) \triangleq \mathbb{E}\left(\sum_{t \in \mathcal{T}} \gamma^t r_{s_t s_{t+1}}^{a_t} \middle| s_0 = s,\, a_t \sim \pi_t\left(\cdot \mid s_t\right),\, s_{t+1} \sim T_{s_t \cdot}^{a_t}\ \forall t \geq 0 \right).$$

Notation 2.8. If there is no ambiguity on the considered MDP M of Definition 2.4, we omit M in the writing of the value function and write $V^\pi \triangleq V_M^\pi$.

The definition of the value function is written in the case of a non-stationary stochastic policy for the sake of generality. Naturally, it extends to any class of policy defined in Table 2.1, page 13. Intuitively, $V^\pi(s)$ is the expected discounted return given that we apply policy π at state $s \in \mathcal{S}$. In the finite horizon case, *i.e.*, when $H < \infty$, Definition 2.4 can be extended to the total sum of reward rather than discounted by choosing $\gamma = 1$. Additionally, we define the expected return of applying an action $a \in \mathcal{A}$ and then a policy π at a particular state $s \in \mathcal{S}$ as the Q-value function.

Definition 2.5. Consider an MDP $M = \{\mathcal{S}, \mathcal{A}, T, r\}$, a set of decision epochs $\mathcal{T} = \{0, \dots, H\}$, $H \leq \infty$, a discount factor $\gamma \in [0, 1)$ and a non-stationary stochastic policy π. The Q-value of π is defined for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ by the *Q-value function* Q_M^π as

$$Q_M^\pi(s, a) \triangleq \mathbb{E}\left(\sum_{t \in \mathcal{T}} \gamma^t r_{s_t s_{t+1}}^{a_t} \middle| s_0 = s,\, a_0 = a,\, a_t \sim \pi_t\left(\cdot \mid s_t\right)\ \forall t \geq 1,\, s_{t+1} \sim T_{s_t \cdot}^{a_t}\ \forall t \geq 0 \right).$$

Notation 2.9. Like the value function, if there is no ambiguity on the considered MDP M of Definition 2.5, we omit M in the writing of the Q-value function and write $Q^\pi \triangleq Q_M^\pi$.

Similarly to Definition 2.4, Definition 2.5 is written in the case of a non-stationary stochastic policy and extends to any class of policy defined in Table 2.1, page 13.

So far, we defined the value and Q-value functions formally as the expectation of the discounted return for a specific policy π. It turns out that V^π and Q^π are also solutions of two equations, namely, the *Bellman evaluation equations*.

Theorem 2.1 (Bellman evaluation equations, Puterman (2014))**.** *Consider an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$ and a stationary deterministic policy π, the value function V^π and Q-value func-*

tion Q^π satisfy the following equations for all $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$V^\pi(s) = R_s^{\pi(s)} + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^{\pi(s)}} \left(V^\pi(s') \right), \tag{2.1}$$

$$Q^\pi(s,a) = R_s^a + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^a} \left(Q^\pi(s', \pi(s')) \right). \tag{2.2}$$

By defining the Bellman operators for a policy π as follows:

$$\mathscr{T}_V^\pi(f)(s) = R_s^{\pi(s)} + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^{\pi(s)}} \left(f(s') \right), \qquad \text{for } f : \mathcal{S} \to \mathbb{R},$$

$$\mathscr{T}_Q^\pi(f)(s,a) = R_s^a + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^a} \left(f(s', \pi(s')) \right), \qquad \text{for } f : \mathcal{S} \times \mathcal{A} \to \mathbb{R},$$

Theorem 2.1 states that V^π and Q^π respect the two following fixed-point Equations:

$$V^\pi = \mathscr{T}_V^\pi(V^\pi), \tag{2.3}$$

$$Q^\pi = \mathscr{T}_Q^\pi(Q^\pi). \tag{2.4}$$

Further, the following theorem establishes that the solutions to Equations 2.3 and 2.4 exist and are unique. As a result, finding the solutions to these two equations amounts to finding V^π and Q^π.

Theorem 2.2 (Puterman (2014))**.** *Consider an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$ and a deterministic stationary policy π. If the discount factor γ verifies $0 \leq \gamma < 1$, then the operators $\mathscr{T}_V^\pi$ and $\mathscr{T}_Q^\pi$ are contractions in maximum norm. It follows, from application of Banach fixed-point Theorem, that Equations 2.3 and 2.4 have unique solutions.*

We introduced policies as objects we want to optimize along with the value of a policy. This allows us to define our optimality criterion in the next section.

2.1.4 Optimality criterion

For an MDP M, we call *optimal policy* and denote by π^* a policy maximizing the value function uniformly on $\mathcal{S}$, *i.e.*,

$$\pi^*(s) \overset{\triangle}{\in} \operatorname*{argmax}_\pi V^\pi(s), \ \forall s \in \mathcal{S}. \tag{2.5}$$

Solving an MDP means finding such an optimal policy. Importantly, one can show that in any MDP, there exist at least one *deterministic, stationary, state-dependent* policy satisfying Equation 2.5 (Puterman, 2014). Also, as mentioned in Section 2.1.2, page 12, the state dependency refers to the difference between history-based policies and state-based policies. Let us now consider the value and Q-value function of an optimal policy π^* by first introducing their notation.

Notation 2.10. Considering an MDP M, we will write the value and Q-value functions of an optimal policy defined by Equation 2.5, $V^* \triangleq V^{\pi^*}$ and $Q^* \triangleq Q^{\pi^*}$.

Although V^* and Q^* are also defined with Definitions 2.4 and 2.5, page 14, they share different properties than the value and Q-value functions of a non-optimal policy. First, resulting from the fact that an optimal policy maximizes the expected return, V^* and Q^* are linked with the following two equations:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s,a), \\ Q^*(s,a) &= R_s^a + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^a} \left(V^*(s') \right). \end{aligned}$$

Remark 2.3. The fact that we used the *maximum* operator on $\mathcal{A}$ in the former equation rather than the *supremum* operator comes from the fact that $\mathcal{A}$ is a compact set by hypothesis.

Secondly, V^* and Q^* are solutions of a different version of the Bellman evaluation Equations 2.1 and 2.2, page 15, namely, the Bellman optimality equations, reported in the following Theorem.

Theorem 2.3 (Bellman optimality equations)**.** *Consider an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$, the optimal value function V^* and optimal Q-value function Q^* satisfy the following equations for all $(s,a) \in \mathcal{S} \times \mathcal{A}$:*

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} \left\{ R_s^a + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^a} \left(V^*(s') \right) \right\}, \\ Q^*(s,a) &= R_s^a + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^a} \left(\max_{a' \in \mathcal{A}} Q^*(s',a') \right). \end{aligned}$$

Again, if we define the Bellman optimality operators as follows:

$$\begin{aligned} \mathscr{T}_V^*(f)(s) &= \max_{a \in \mathcal{A}} \left\{ R_s^a + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^a} \left(f(s') \right) \right\}, && \text{for } f : \mathcal{S} \to \mathbb{R}, \\ \mathscr{T}_Q^*(f)(s,a) &= R_s^a + \gamma \mathbb{E}_{s' \sim T_{s\cdot}^a} \left(\max_{a' \in \mathcal{A}} f(s',a') \right), && \text{for } f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \end{aligned}$$

we have that V^* and Q^* respect the two following fixed-point Equations:

$$V^* = \mathscr{T}_V^*(V^*), \tag{2.6}$$

$$Q^* = \mathscr{T}_Q^*(Q^*). \tag{2.7}$$

From those definitions, the same result as Theorem 2.2, page 15 applies for V^* and Q^*.

Theorem 2.4. *Consider an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$ and a deterministic stationary policy π. If the discount factor γ verifies $0 \leq \gamma < 1$, then the operators $\mathscr{T}_V^*$ and $\mathscr{T}_Q^*$ are contractions in maximum norm. It follows, from application of Banach fixed-point Theorem, that Equations 2.6, and 2.7 have unique solutions.*

The existence and uniqueness of the optimal value and Q-value functions is a fundamental result. In particular, it allows to search for Q^* rather than directly an optimal policy π^* to solve an MDP, which can be more convenient in some cases. Let us now explain why finding

Q^* is equivalent to finding π^*. The Q-value function of a policy π is a tool that defines the advantage of taking an action a and then follow π, rather than following π from the start. It allows to *improve* on a policy if, for a specific state s in an MDP, the following equation is verified:

$$Q^\pi(s, a) > V^\pi(s).$$

In such a case, the policy applying a at s and π otherwise improves on policy π since it yields higher expected discounted return. Choosing the action maximizing Q^π uniformly on $\mathcal{S}$ allows to derive a policy uniformly better than π. We call it the *greedy policy with respect to* Q^π which, for any $s \in \mathcal{S}$, selects the action

$$a = \operatorname*{argmax}_{\tilde{a} \in \mathcal{A}} Q^\pi(s, \tilde{a}).$$

From what we just wrote, if we know the optimal Q-value Q^*, we can deduce the optimal policy π^* by acting greedily with respect to Q^*. Indeed, since π^* yields maximum expected discounted return, the uniformly improving policy performs exactly the same since, by definition, it is not possible to do better. Hence, we have for any MDP,

$$\pi^*(s) \in \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s, a), \; \forall s \in \mathcal{S}. \tag{2.8}$$

As a result, π^* can be deduced from Q^*. To solve an MDP, one can interchangeably compute π^* or Q^*. In the literature, many algorithms — that we call value-based — aim at deriving the optimal Q-value function to solve an MDP. This dissertation focuses essentially on value-based algorithms, spanning from *planning* algorithms to *learning* algorithms. In the remainder of this introductory chapter, we explain the difference between those two different classes of algorithms.

2.2 Planning

2.2.1 Definition

Planning refers to the popular idea of “figuring out things in one’s head” (Dennett, 1975; Sutton, 1991). When acting within a maze, an agent can *plan* by “thinking” about the path it experienced and reason on this path to deduce whether to explore rather one part of the maze or another part. It could also deduce how far it is from the starting point. If the agent had a map of the maze, it could use it to deduce a path to the exit through a planning procedure. Planning is any reasoning related to a sequence of actions carried out in an agent’s mind, based on the knowledge it has on its environment. In this dissertation, we will focus on the case where an agent utilizes an MDP model of the environment for planning. For instance, this model can be used to deduce an optimal action given the current state of the agent. A model can be anything capturing the transition and reward functions of the MDP. It can be more or less accurate, its quality often being linked to the optimality of the planning procedure. Unlike learning methods, planning algorithms do not rely on data gathering. For the case where a model is available, we will suppose it to be already known. Of course, the processes of first estimating a model and then using it for planning can be combined in a single algorithm. This will be the case for the RMAX algorithm presented in Section 2.3.2, page 28. However, in this dissertation we also focus on vanilla planning algorithms, which justifies the distinction between planning and learning methods.

Among the models usable by a planning algorithm, we distinguish between generative and complete models whose definitions follow.

Definition 2.6 (Model and generative model)**.** Given a state-action space $\mathcal{S} \times \mathcal{A}$, a *model* is a pair of functions

$$\begin{array}{rcll} \hat{T} & : & \mathcal{S} \times \mathcal{A} & \rightarrow & \mathcal{P}(\mathcal{S}), \\ \hat{r} & : & \mathcal{S} \times \mathcal{A} \times \mathcal{S} & \rightarrow & \mathbb{R}, \end{array}$$

whose analytical expressions are known. If the functions $\hat{T}$ and $\hat{r}$ are black box functions, *i.e.*, we can only sample their outputs given an input but have no information on their analytical expressions, then the model is said to be a *generative model.*

Given an MDP $M = \{\mathcal{S}, \mathcal{A}, T, r\}$, a model $\hat{T}, \hat{r}$ is said to be the *true model* of M if $\hat{T} = T$ and $\hat{r} = r$. Correspondingly, a *true generative model* associated to M is such that the outputs of the functions $\hat{T}$ and $\hat{r}$ could have been generated with T and r.

Remark 2.4. Equivalently, when dealing with the expected reward function R rather than the reward function r, we consider models and generative models of the form $\hat{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$, $\hat{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The same notion of correspondence with the true model applies.

Obviously, the assumption of having access to a generative model is less restrictive than having a complete model. Therefore, methods using a generative model to compute an optimal policy are more appealing practically although they face a more complex challenge. In the next two sections, we successively present planning algorithms making use of a complete model and a generative model.

2.2.2 Dynamic programming

Dynamic Programming (DP) (Bellman, 1957) refers to a collection of methods for mathematical optimization and computer programming relying on the general principle of breaking the complexity of a problem by solving several sub-problems in a recursive way. Among the numerous applications of DP are methods to solve an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$ given that we know the true model (T, R). *Value iteration* and *policy iteration* both are DP algorithms that compute Q^* given T and R and deduce the optimal policy thanks to Equation 2.8, page 17. In this section, we present the value iteration algorithm, first applied to the derivation of V^* and by extension Q^*. Value iteration relies on Theorem 2.4, page 16 which, in addition to the existence and uniqueness of the optimal value and Q-value functions, provides a practical methods to compute them. The algorithm relies on three hypotheses:

1. the true model (T, R) is known;

2. $\mathcal{S}$ and $\mathcal{A}$ are finite sets;

3. the discount factor γ verifies $0 \leq \gamma < 1$.

Banach fixed-point theorem provides a method to compute the fixed-point of $\mathscr{T}_V^*$, which is V^* in this case. By initializing a sequence of functions with $V_0 : \mathcal{S} \to \mathbb{R}$ arbitrary and $V_{n+1} = \mathscr{T}_V^*(V_n), \forall n \in \mathbb{N}$, the sequence is guaranteed to converge to V^* in maximum norm. Practically, the value iteration method consists in repeatedly applying $\mathscr{T}_V^*$ on the computed function. Williams and Baird (1993) define the following stopping criterion:

$$\|V_n - V_{n-1}\|_\infty \leq \epsilon,$$

with $\epsilon > 0$ a precision parameter and the maximum norm is defined by $\|V_n - V_{n-1}\|_\infty \triangleq \max_{s \in \mathcal{S}} |V_n(s) - V_{n-1}(s)|$. Once this condition is fulfilled, they prove that the computed value function differs from V^* by no more than $\frac{2\gamma\epsilon}{1-\gamma}$ in maximum norm. This result also extends to the calculation of the Q-value function, which allows to control the optimality of the associated greedy policy by controlling the value of ϵ. In this dissertation, we will take a different stopping criterion, namely, directly reaching a precision ϵ in maximum norm, *i.e.*, when $n \in \mathbb{N}$ is such that:

$$\|V_n - V^*\|_\infty \leq \epsilon, \tag{2.9}$$

To that end, the following theorem provides the number of iterations needed for Equation 2.9 to hold.

Theorem 2.5 (Strehl, Li, and Littman (2009))**.** *Given an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$, a discount factor $\gamma \in [0, 1)$, a precision $\epsilon \in \left[0, \frac{R_{max}}{1-\gamma}\right)$ and the sequence of functions defined by $V_0 : \mathcal{S} \to \mathbb{R}$ arbitrary and $V_{n+1} = \mathscr{T}_V^*(V_n), \forall n \in \mathbb{N}$. If $n \geq \left\lceil \frac{1}{\ln(1/\gamma)} \ln\left(\frac{R_{max}}{\epsilon(1-\gamma)}\right) \right\rceil$,*

$$\|V_n - V^*\|_\infty \leq \epsilon.$$

The proof of Theorem 2.5 is reported in the Appendix, Chapter A, Section A.1. The number of operations $\left\lceil \frac{1}{\ln(1/\gamma)} \ln\left(\frac{R_{\max}}{\epsilon(1-\gamma)}\right) \right\rceil$ is smaller than the one reported by Strehl, Li, and Littman (2009). Indeed, their analysis required this number to be expressed as a polynomial, which slightly increases its magnitude. Since both results differ, we report the proof of Theorem 2.5 in the Appendix, Chapter A, Section A.1, page 127.

Using Proposition 2.5, the value iteration algorithm is described in Algorithm 1.

In the same way, the result of Theorem 2.5 can be extended to Q^*. The corresponding value iteration algorithm computing Q^* with accuracy ϵ in maximum norm is the same as Algorithm 1 with the following update rule:

$$Q(s, a) \leftarrow R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a \max_{a' \in \mathcal{A}} Q(s', a') \tag{2.10}$$

Algorithm 1 Value Iteration

```
**Input:** MDP model $T, R$; discount factor $\gamma \in [0, 1)$; precision $\epsilon \in \left[0, \frac{R_{\max}}{1-\gamma}\right)$
Initialize $V$ randomly
$m = \left\lceil \frac{1}{\ln(1/\gamma)} \ln\left(\frac{R_{\max}}{\epsilon(1-\gamma)}\right) \right\rceil$   # Compute the required number of iterations to reach precision $\epsilon$.
**for** $i \in \{1, \dots, m\}$ **do**
  **for** $s \in \mathcal{S}$ **do**
    $V(s) \leftarrow \max_{a \in \mathcal{A}} \{R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a V(s')\}$   # Update $V$ for every state.
  **end for**
**end for**
**return** $V$
```

Value iteration allows to solve an MDP given its transition and reward function. It is a planning algorithm in that it does not rely on an online interaction with the MDP, as opposed to learning algorithms that we will present later. Value iteration can be applied offline to compute Q^* which can be used to derive the optimal policy using Equation 2.8, page 17. The cost of applying value iteration can be evaluated with its *computational complexity* that we define as an upper bound on the number of operations needed by a computer to run an algorithm until completion. The computational complexity of value iteration is given in Theorem 2.6.

Theorem 2.6 (Computational complexity of value iteration)**.** *The value iteration algorithm (Algorithm 1) requires*

$$\mathcal{O}\left(S^2 A \left\lceil \frac{1}{\ln(1/\gamma)} \ln\left(\frac{R_{max}}{\epsilon(1-\gamma)}\right) \right\rceil\right)$$

computing operations to reach completion.

The proof of Theorem 2.6 is reported in the Appendix, Chapter A, Section A.1. It can be shown with a similar reasoning that computing the optimal Q-function of an MDP has the same complexity. Value iteration requires the knowledge of the complete model (T, R) of the MDP. This assumption can be unrealistic in some cases. For instance, any problem with some uncertainty on the available model is unsolvable with value iteration as it requires the an exact analytical formulation of the reward and transition functions. Further, from Theorem 2.6, the computational complexity of value iteration may be too high for problems with large state-action spaces. In the following section, we present planning algorithms designed to alleviate those two bottlenecks.

2.2.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a tree search algorithm using a tree structure to carry out a search in the trajectory space of an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$. The general idea is to build an estimate $\hat{Q}(s, a)$ of the optimal Q-value function $Q^*(s, a)$ for all actions $a \in \mathcal{A}$ and in the current state s. Then, it becomes possible to act greedily with respect to this estimate.

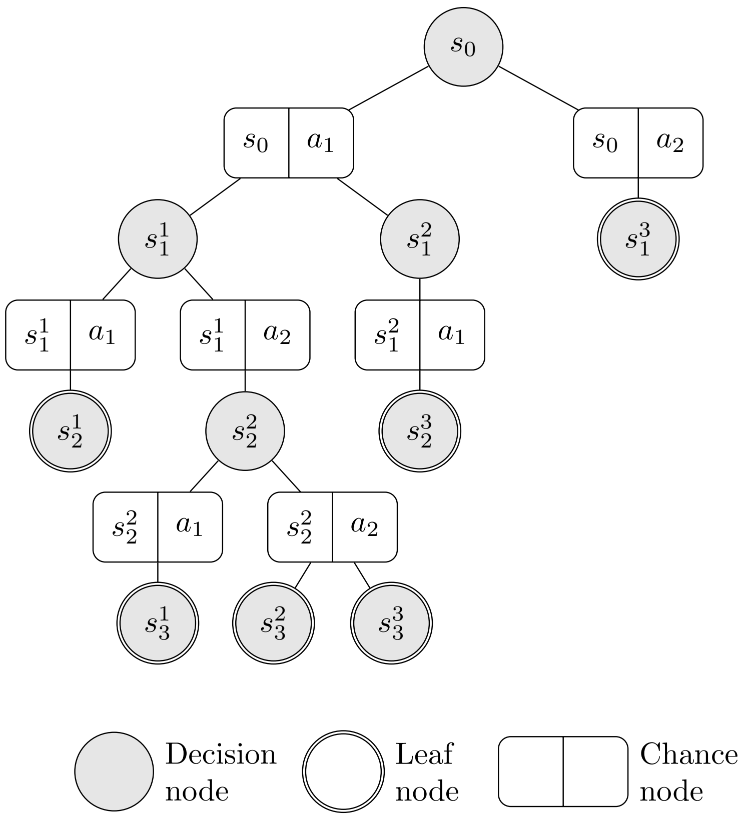


Figure 2.2: An example of MCTS search tree with $\mathcal{A} = \{a_1, a_2\}$. The current state is denoted by s_0. States are denoted by s_i^j where i is the decision epoch inside the trajectory initiated from s_0 and j an index to differentiate states corresponding to the same decision epoch.

If it turns out that $\text{argmax}_{a \in \mathcal{A}} \hat{Q}(s,a) = \text{argmax}_{a \in \mathcal{A}} Q^*(s,a)$, the derived policy is optimal. MCTS uses a generative model $\hat{M} = (\hat{T}, \hat{R})$ to incrementally build a tree of the possible trajectories from the current state of the agent. If $\hat{M} = (T, R)$, then the search happens in the true trajectories space of the MDP, in the sense that the simulated trajectories match those that would be generated in the real MDP. Once the tree built, an — hopefully optimal — action is selected by acting greedily with the Q-value function estimate $\hat{Q}$ and applied in the real MDP. The transition happens and a new state is reached. The tree building procedure is then repeated from this new state.

Let s_0 denote the current state. Executing an exhaustive search among all the possible trajectories starting from s_0 is a very costly process, if not unfeasible. Indeed, in the worst case, the number of possible trajectories is $(SA)^H$ with H the horizon, which is too large for most of the practical cases. If $H = \infty$, then the search is simply impossible to carry out. Given this constraint, the basic idea of MCTS is to prune the search, only exploring some trajectories by incrementally constructing a search tree. Those trajectories are called Monte Carlo (MC) simulations. Before explaining the building process, we explain the structure of a search tree. An MCTS search tree is represented in Figure 2.2. It is composed of trajectories initiated from s_0. Following the nomenclature of Trial-based Heuristic Tree Search (THTS)

(Keller and Helmert, 2013), the root node of the tree is what we will call a *decision node*, solely labeled by a state, which is naturally s_0. Its direct children are called *chance nodes*, labeled by state-action pairs. Chance nodes necessarily are labeled with the same state as their parent decision node as they only correspond to the selection of an action at that state. The children of a chance node are decision nodes corresponding to the states reached after application of the parent's action at the parent's state. We will respectively denote by ν_s and ν_s^a the decision and chance nodes.

During the MC-simulations, if the same state $s' \in \mathcal{S}$ is sampled twice from the state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ labeling a chance node ν_s^a, then one single child $\nu_{s'}$ is created. This means that, once a state is sampled, an equality operator is used to test whether it corresponds to the labeling state of each child node or not. From what we have defined so far, one can deduce the facts that each chance nodes has a unique decision node parent and conversely that each decision node has a unique chance node parent. This is always true except for the root node which has no parent. Quite naturally, a decision-chance-decision node sequence corresponds to a $(s, a, r, s') \in \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$ transition generated with $\hat{M}$ and the tree is solely built with such sequences. A *fully expanded node* is a decision node ν_s having all of its chance nodes children constructed, *i.e.*, one per available action at s. In the case of an MCTS tree, we will call *leaf nodes* the partially expanded nodes, lacking at least one chance node child.

Importantly, such a planning method is called *closed-loop planning* because the search happens in the trajectories space, of the form $\{s_i, a_i\}_{i=0}^{H}$ (Bubeck and Munos, 2010; Weinstein and Littman, 2012). The states along the trajectory are identified and distinguished. The search branches according to the selected actions *and* the different sampled states. This has the immediate advantage of allowing reasoning at the state level, *i.e.*, evaluating the optimality of a actions at specific states. Conversely, *open-loop planning* is a search in the *action sequences* or *plans* space, of the form $\{a_i\}_{i=0}^{H}$ (Bubeck and Munos, 2010; Weinstein and Littman, 2012). The states generated with $\hat{M}$ along the action sequence are not identified and the search only branches according to the selected actions in the plans. The consequence of this is that optimality is not considered with respect to a single state but to the distribution of states after application of a certain sequence of actions. Notice that for deterministic MDPs, closed-loop and open-loop planning are equivalent. We briefly introduced the difference between closed and open-loop planning for the sake of Chapter 3 where the latter is discussed more thoroughly.

The initial purpose of MCTS is identify the optimal action $\pi^*(s)$ at the current state $s \in \mathcal{S}$ by finding the action with the highest Q-value. Therefore, we define the values of decision and chance nodes, respectively mirroring V^* and Q^*. Intuitively, we use the notations $V(\nu_s)$ and $Q(\nu_s^a)$ to denote those values. Since $V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a)$ for all $s \in \mathcal{S}$, we naturally define the value of a decision node with the maximum value of its children chance nodes:

$$V(\nu_s) \triangleq \max_{\nu' \in \{\text{children of } \nu_s\}} Q(\nu') .$$

To define the value of a chance node, we use the statistics of the search tree. During the construction, complete trajectories are sampled, starting from s_0. For each chance node ν_s^a, a

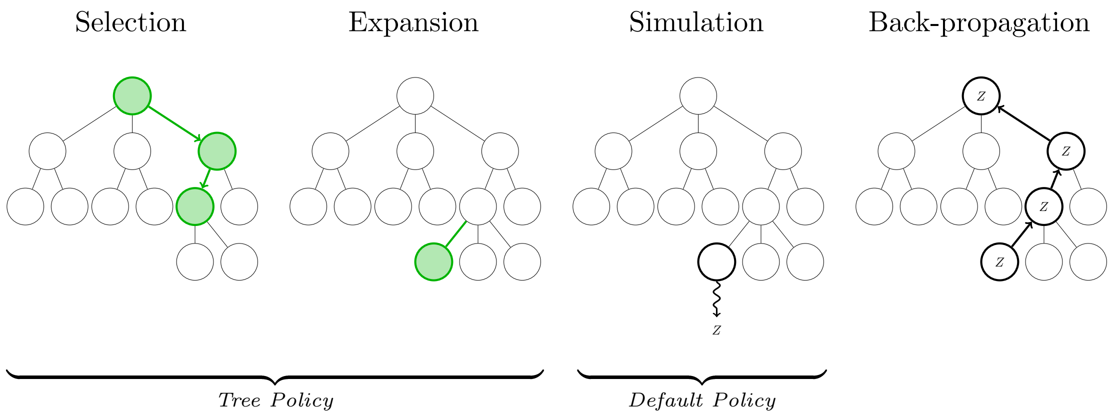


Figure 2.3: Illustration of one step of the MCTS procedure. For clarity, only decision nodes are represented and chance nodes are symbolized by the branches of the tree between subsequent decision nodes. Notice that — according to the tree building process — chance nodes never exist without at least one decision node child, which motivates this simplified illustration.

subset of those trajectories *went through* the parent decision node ν_s by sampling s; and then selected the action a, leading them to ν_s^a. Let us consider the end of those trajectories, starting from the moment where s is sampled and a selected. This allows us to define a collection of returns by the discounted sum of the collected rewards along those sub-trajectories. If we denote by $\{Z_i\}_{i=1}^k$ the set of sampled returns with k the number of trajectories going through ν_s^a, we define the value of the chance node by the empirical mean of the collected returns, *i.e.*,

$$Q(\nu_s^a) \triangleq \frac{1}{k}\sum_{i=1}^{k} Z_i. \tag{2.11}$$

Obviously, if the trajectories are generated with an optimal policy and the true model, then we have that

$$Q(\nu_s^a) \underset{k\to\infty}{\longrightarrow} Q^*(s,a). \tag{2.12}$$

We now detail the different steps of the tree building. This construction is the sequential realization of a four steps process illustrated in Figure 2.3. Each time the process is realized, a full trajectory starting from s_0 is generated. We call *budget* and denote by $B \in \mathbb{N}$ the total number of trajectories used to create the tree. We now detail each one of the four steps:

1) Selection. This first step aims at selecting one of the leaf nodes of the existing tree. If the tree is only composed of ν_{s_0}, then this node is selected. Else, we recursively apply a *tree policy* π_{tree} — which can select actions based on the values of the nodes — from s_0 and use the generative model $\hat{M}$ to reach a leaf node. From a decision node ν_s, the chance node corresponding to the action $\pi_{\text{tree}}(s)$ is selected. From a chance node ν_s^a, a state $s' \sim \hat{T}_{s\cdot}^a$, resulting from the application of action a in state s, is sampled. If s' corresponds to an existing decision node, the selection step continues from this decision node. Else, a new decision node, labeled with s', is created. In such a case, there is no need to apply the second step and the procedure jumps to the third step.

2) Expansion. Once a leaf node ν_s is reached, a new action a is randomly selected among the non-explored actions of ν_s and the corresponding chance node ν_s^a is created. A call to the generative model is performed, sampling a state $s' \sim \hat{T}_{s\cdot}^a$, resulting from the application of action a in state s. The decision node $\nu_{s'}$ is immediately created as a child of ν_s^a.

3) Simulation. The expansion step always ends with the creation of a decision node $\nu_{s'}$. From s', a full trajectory is drawn using $\hat{M}$ and a *default policy* π_{default} until termination, *i.e.*, reaching of a terminal state or the horizon H. This yields a realization of the discounted return Z that will be back-propagated to update the values of the nodes in the fourth step.

Remark 2.5. In the case where there are no terminal states and $H = \infty$, there could be no termination. One can introduce an artificial horizon $\tilde{H} < \infty$ and stop the trajectory when reaching $\tilde{H}$ to prevent this case.

4) Back-propagation. During the selection, expansion and simulation steps, a full trajectory of length H has been simulated with the generative model $\hat{M}$ and the subsequent applications of π_{tree} and π_{default}. The back-propagation step consists in updating the values of all the traversed chance nodes during the selection and expansion steps, according to their definition in Equation 2.11, page 23.

The MCTS algorithm is summarized in Algorithm 2. A more detailed version, including the sub-method's exact definitions is available in the Appendix, Chapter B.

Algorithm 2 MCTS

```
Set: MDP $\{\mathcal{S}, \mathcal{A}, T, r\}$; initial state distribution $\mathcal{P}_0$; horizon $H$.
Input: generative model $\hat{M} = (\hat{T}, \hat{R})$; budget $B$; tree policy $\pi_{\text{tree}}$; default policy $\pi_{\text{default}}$.
$s \sim \mathcal{P}_0$  # Set the initial state.
for $t \in \{1, \dots, H\}$ do
    $\nu_s \leftarrow$ decisionNode($s$)  # Initialize the search tree with the current state $s$.
    for $i \in \{1, \dots, B\}$ do
        $\nu_{\tilde{s}} \leftarrow$ selection($\nu_s, \hat{M}, \pi_{\text{tree}}$)  # Apply $\pi_{\text{tree}}$ to select a leaf node.
        $\nu_{\tilde{s}'} \leftarrow$ expansion($\nu_{\tilde{s}}, \hat{M}$)  # Expand the tree with a new decision node.
        $Z \leftarrow$ simulation($\tilde{s}', \pi_{\text{default}}, \hat{M}, H$)  # Apply $\pi_{\text{default}}$ to generate a trajectory and record
        the return.
        backPropagation($\nu_{\tilde{s}'}$)  # Update the values of the nodes with the collected return.
    end for
    $s' \sim T_{s\cdot}^a$  # Sample the next state.
    $s \leftarrow s'$
end for
```

To complete the definition of the MCTS algorithm, the specification of π_{tree} and π_{default} remains. As hinted by Equation 2.12, page 23, a policy as close to optimal as possible should be used for the values of the nodes to correspond to optimal values. In the literature, many different policies have been designed for that purpose (Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, and Colton, 2012). The famous variation of MCTS called Upper Confidence bound applied to Trees (UCT) proposed by Kocsis and

Szepesvári (2006) focuses the search in more promising areas of the sampled trajectories. To that purpose, they see the problem of deciding which action to select in a decision node as a multi-armed bandit problem (Katehakis and Veinott Jr., 1987). They consider using the Upper Confidence Bound (UCB) (Auer, Cesa-Bianchi, and Fischer, 2002) utility function defined for every chance node ν_s^a as

$$UCB(\nu_s^a) = Q(\nu_s^a) + 2C_p\sqrt{\frac{\ln(n)}{n(s,a)}}, \tag{2.13}$$

where $Q(\nu_s^a)$ is the value of the chance node defined in Equation 2.11, page 23, $C_p > 0$ is a constant exploration term, $n \in \{0,\dots,B\}$ is the total number of complete trajectories generated and $n(s,a)$ is the number of trajectories going through (s,a). The first term, $Q(\nu_s^a)$, is an exploitation term, favoring actions that yield high returns. The second term, $2C_p\sqrt{\frac{\ln(n)}{n(s,a)}}$, is an exploration term, favoring less visited nodes. The UCT tree policy π_{UCT} is the greedy policy with respect to the UCB utility function, *i.e.*, for a decision node ν_s, the selected chance node is

$$\pi_{\text{UCT}}(s) = \text{ action of } \underset{\nu\in\nu_s.\text{children}}{\text{argmax}}\ UCB(\nu). \tag{2.14}$$

Intuitively, Equation 2.13 realizes a trade-off between exploration and exploitation. Every chance node of the tree is visited infinitely often as its utility increases logarithmically with the total number n of trajectories. Given a finite budget $B \in \mathbb{N}$, UCT often features improved performances compared to flat MCTS. As B tends to infinity, UCT is proven to converge to the optimal action. Before stating this important result, let us introduce a few notations. They may seem cumbersome for the presentation of MCTS we make here but we will re-use them in Chapter 3.

Suppose we are running UCT from the current state $s_0 \in \mathcal{S}$. We denote by $\hat{Z}_{a,t}$ the estimated value of node $\nu_{s_0}^a$ after t simulations starting from (s_0, a) with $a \in \mathcal{A}$ and $t \in \{1,\dots,B\}$. We denote by $T_{a,t}$ the total number of visits to node $\nu_{s_0}^a$ after t simulations. Hence, $\hat{Z}_{a,T_{a,B}}$ represents the final estimate of the value of $\nu_{s_0}^a$ after a total of B MC-simulations. We denote by $a_B = \text{argmax}_{a\in\mathcal{A}}\, \hat{Z}_{a,T_{a,B}}$ the recommended action by UCT after B MC-simulations. Recall that our objective is to select the optimal action that we will write a^* at s_0. We denote by $Z_{a,t} = \mathbb{E}\left(\hat{Z}_{a,t}\right)$ the expected value of $\hat{Z}_{a,t}$, seen as a random variable subject to the randomness induced by the generative model $\hat{M}$ combined with the policies used by UCT. Asymptotically, this value converges to $Z_a = \lim_{t\to\infty} Z_{a,t}$. Finally, we introduce $\Delta_a = Z_{a^*} - Z_a$ for all $a \in \mathcal{A}$. It corresponds to the expected (discounted) return difference between a sub-optimal action and an optimal one.

Theorem 2.7 (Convergence of Failure Probability of UCT (Kocsis and Szepesvári, 2006))**.** *Consider Algorithm 2 executed with a budget $B \in \mathbb{N}$ and the UCT tree policy of Equation 2.14. Then, there exists some positive constant ρ such that*

$$\boldsymbol{Pr}(a_B \neq a^*) \leq \left(\frac{1}{B}\right)^{\frac{\rho}{2}\left(\frac{\min_{a \neq a^*} \Delta_a}{36}\right)^2} .$$

In particular, it holds that $\lim_{B \to \infty} \boldsymbol{Pr}(a_B \neq a^*) = 0.$

This results implies the convergence of UCT to an optimal policy of the solved MDP. This algorithm scales to larger classes of problem than the value iteration algorithm presented in Section 2.2.2, page 18. It relies on weaker hypotheses which makes it convenient for many practical uses. Particularly, MCTS and UCT both use a generative model which is often more convenient than the full model required by value iteration. However, there can be problems where even a generative model is not available. In this case, solving approaches generally consist in learning a model, or learning the optimal Q-value function, or directly learning an optimal policy. We focus on learning algorithms in the next section.

2.3 Learning

Commonly speaking, learning refers to all the processes of memorization used by animals or humans to conceive or modify their behavioral patterns under the influence of their environment and their experience. In the context of sequential decision making, the definition is not really different. Instead of the general concept of “learning”, we will narrow down our study to the specific framework of *Reinforcement Learning*, because it focuses on learning methods designed *for* the purpose of sequential decision making. In the next two sections, we successively detail our definition of the concept and introduce an important instance of Reinforcement Learning algorithm.

2.3.1 Definition

Reinforcement Learning (RL) (Sutton and Barto, 2018) is the systematization of the trial and error process in sequential decision making. An agent takes an action in an environment given the current state, it receives a reward signal and observes the state resulting from the transition. This process, illustrated in Figure 2.4, page 27, is repeated until completion. In this dissertation, the mathematical model of the Reinforcement Learning problem is an MDP $\{\mathcal{S}, \mathcal{A}, T, r\}$ (Definition 2.1, page 8) and we use T and r to generate the transitions. The goal of an agent in such a context is unchanged: we seek to maximize the expected (discounted) return. To that end, we also optimize a policy. To do so, interactions with the environment allow an agent to collect data from which it can *learn* what actions are good to take in what states. According to the RL interaction process, data points are of the form $(s, a, r, s') \in \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$ where s is a state, a the selected action at s, and (r, s') the

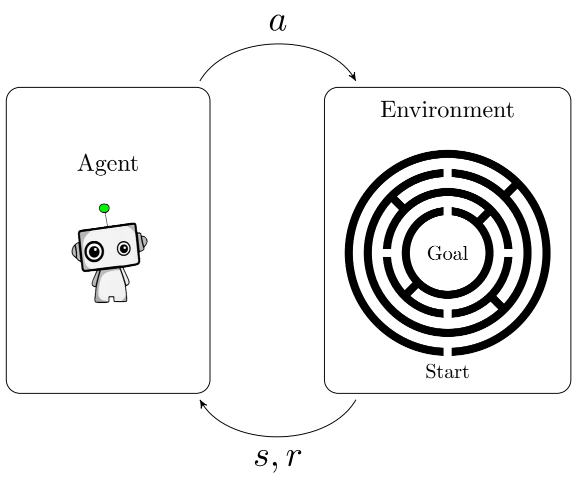


Figure 2.4: Illustration of the Reinforcement Learning principle: an agent interacts with an environment by taking actions. After each action, it receives a reward feedback and observes the state resulting from the application of the action. In the figure, a, s, r respectively represent the action, the state and the reward. Credit: inserted robot picture from Krypton / Doragon (licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license).

resulting reward-state pair from the transition. The intuition is that if a particular action in a particular state yields high reward, it could be a good thing to repeat this action at that state. However, the overall goal is more complex as we are interested in the sum of *all* the collected rewards. Hence, a myopic approach such as the one described before may be too greedy and perhaps the preferable action to maximize the long term sum of rewards is not the one yielding higher reward.

Learning necessarily implies repetition of the experienced task. We define *episodic tasks* as the sequential repetition of multiple trajectories within the same MDP run until completion, *i.e.*, when reaching the horizon H or a terminal state. To suggest the idea of sequential realization, we will call *episode* the realization of one single trajectory run until completion. The starting state of each episode is sampled according to a distribution that we write $\mathcal{P}_0 \in \mathcal{P}(\mathcal{S})$. Learning data do not need to be originated from the same episode as they correspond to the same MDP, hence, are still valid. At the end of an episode, the state is reset to an initial state sampled with $\mathcal{P}_0$. Each time a data point (s, a, r, s') is collected, it can be used by the agent within a learning procedure. Hence, additionally to the *capability to act* explained in the definition of an agent in Section 2.1.1.1, page 6, a Reinforcement Learning agent has a *capability to learn*. The general Reinforcement Learning procedure is detailed in Algorithm 3, page 28.

What does an agent learn? The ultimate goal of Reinforcement Learning is to learn an optimal policy of the MDP. The learning procedure consists in using the collected data to learn the shape of certain function(s) to that purpose. There are no rules defining *which function should be learned*, but many methods — with their own assets and disadvantages — have been derived to learn different objects, keeping in sight the eventual goal of computing

Algorithm 3 General Reinforcement Learning procedure

```
Input: agent with methods act and learn; MDP $\{\mathcal{S}, \mathcal{A}, T, r\}$; initial state distribution $\mathcal{P}_0$;
number of episodes $p \in \mathbb{N}$; horizon $H \in \{1, \dots, \infty\}$.
for each episode 1 to $p$ do
    $s \sim \mathcal{P}_0$ # Set the initial state.
    for $t \in \{1, \dots, H\}$ do
        $a \leftarrow$ agent.act($s$) # Apply the current policy.
        $s' \sim T^a_{s\cdot}$ # Sample the next state.
        $r \leftarrow r^a_{ss'}$ # Sample the reward signal based on the transition.
        agent.learn($s, a, r, s'$) # Process the collected transition in the learning method.
        $s \leftarrow s'$
        if $s$ is terminal then
            break the for loop # Jump to the next episode.
        end if
    end for
end for
```

an optimal policy. Commonly, RL methods comply with the three schemes illustrated in Figure 2.5, page 29. First, one could straightforwardly seek to learn an optimal policy π^*. We will call those methods *direct policy search* (Sigaud and Stulp, 2019). Some methods interleave the learning of the optimal value function V^* or the optimal Q-value function Q^* in the process. As seen before in Equation 2.8, page 17, the knowledge of Q^* is sufficient to deduce π^*. We call those methods *value-based* learning algorithms. Finally, some methods, first seek to learn the transition and reward functions of the MDP, use them to learn V^* or Q^* and in turn deduce the optimal policy from those functions. We call those methods *model-based* learning algorithms or *indirect Reinforcement Learning*. In this dissertation, we will mostly be dealing with model-based methods. In the following section, we give the example of an important model-based RL algorithm, called RMAX.

2.3.2 RMAX: an example of Reinforcement Learning algorithm

RMAX is a conceptually simple RL algorithm. First, a model of the environment is learned. Secondly, the optimal Q-value function associated to the learned model is computed. Lastly, the greedy policy with respect to the computed Q-value function is followed. The only hypothesis RMAX does is that $\mathcal{S}$ and $\mathcal{A}$ are finite spaces.

Consider an MDP $\{\mathcal{S}, \mathcal{A}, T, R\}$ and a discount factor $\gamma \in [0, 1)$. In RMAX, we keep track of a learned model $\left(\hat{T}, \hat{R}\right)$ that respects the principle of *optimism in the face of uncertainty*. This principle can be stated as follows: when the expected (discounted) return of an action is unknown, we assume it to yield the maximum value. This heuristic allows in turn to force the exploration of unknown state-action pairs since they are believed to yield maximum return. If it turns out that it is not true, then the real reward and transition function of those state-action pairs is learned and the belief is updated accordingly. Practically in RMAX, this is

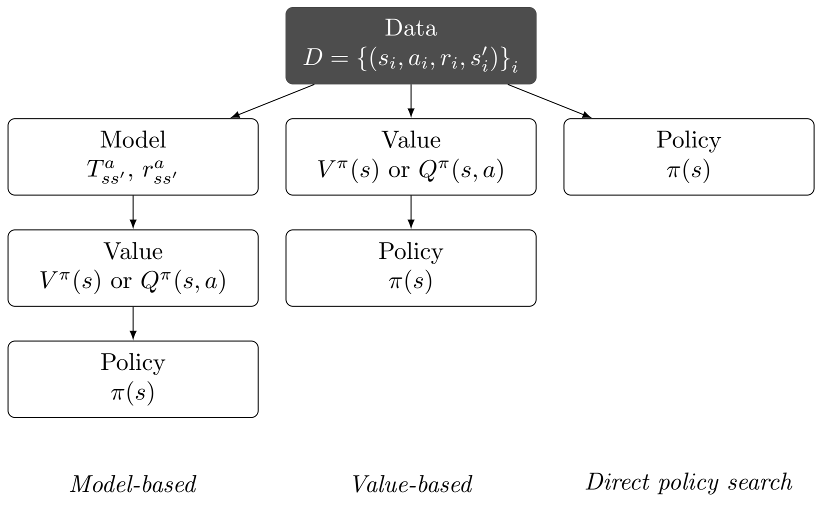


Figure 2.5: Common reinforcement learning schemes

expressed by the optimistic initialization of $\left(\hat{T}, \hat{R}\right)$. Prior to any learning, the transitions are assumed to be self transitions and rewards to yield $R_{\max}$, hence the name of the algorithm. Formally, we have the following initialization for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ and for all $s' \in \mathcal{S}, s' \neq s$:

$$\hat{T}^a_{ss} = 1,\ \hat{T}^a_{ss'} = 0,\ \hat{R}^a_s = R_{\max}. \tag{2.15}$$

Notation 2.11. If A is a set included in another set X containing all the elements under study, we denote by A^c the complement of A in X. A^c contains all the elements of X not in A, *i.e.*, $A^c \triangleq \{x \in X \mid x \notin A\}$.

As the MDP is explored, the real value of the transition and reward functions is assessed. Let us consider a particular state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$. We denote by $n(s,a)$ the number of times (s,a) was encountered during learning. We denote by $\{r_i\}_{i=1}^{n(s,a)}$ the reward samples collected at (s,a). We denote by $n(s,a,s')$ the number of times state $s' \in S$ was sampled after taking action a in s. In RMAX, the learned model at (s,a) is the empirical model defined by

$$\hat{T}^a_{ss'} = \frac{n(s,a,s')}{n(s,a)},\ \forall s' \in \mathcal{S}, \tag{2.16}$$

$$\hat{R}^a_s = \frac{1}{n(s,a)} \sum_{i=1}^{n(s,a)} r_i. \tag{2.17}$$

The empirical model of Equations 2.16 and 2.17 is used instead of the optimistic model of Equation 2.15 once we have high statistical confidence in the empirical estimates. To that end, we introduce $n_{\text{known}} \in \mathbb{N}$ the minimum number of samples required for each state-action pair to be considered known. This allows us to define the set of known state-action pairs by

$$K \triangleq \{(s,a) \in \mathcal{S} \times \mathcal{A} \mid n(s,a) \geq n_{\text{known}}\}.$$

As a result, if we write Q the optimal value function associated to $\left(\hat{T}, \hat{R}\right)$, its value can be defined as the solution to:

$$Q(s,a) = \begin{cases} \hat{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \hat{T}^a_{ss'} \max_{a' \in \mathcal{A}} Q(s',a') \text{ if } (s,a) \in K, \\ \frac{R_{\max}}{1-\gamma} \text{ else.} \end{cases} \tag{2.18}$$

We introduce the variables $\delta \in [0,1)$ and $\epsilon_M \in \mathbb{R}$ and consider a state-action pair to be known once its empirical model is ϵ_M-accurate in $\mathcal{L}_1$-norm with probability at least $1-\delta$. The value of n_{known} must be chosen in compliance with this fact and can be deduced from the following theorem.

Theorem 2.8 (Strehl, Li, and Littman (2009))**.** *Consider $(s,a) \in \mathcal{S} \times \mathcal{A}$. Suppose m successor states and rewards were independently drawn from $T^a_{s\cdot}$ and R^a_s respectively. Let $\delta \in [0,1)$ and $\epsilon_M \in \mathbb{R}$ be positive constants. Let $\left(\hat{T}, \hat{R}\right)$ be the empirical model estimate defined in Equations 2.16 and 2.17, page 29. If*

$$m \geq \left\lceil \frac{2\left(\ln(2^S - 2) - \ln(\delta/2)\right)}{\epsilon_M^2} \right\rceil,$$

then $\left\| T^a_{s\cdot} - \hat{T}^a_{s\cdot} \right\|_1 \leq \epsilon_M$ and $\left| R^a_s - \hat{R}^a_s \right| \leq \epsilon_M$ *with probability at least* $1-\delta$.

The proof of Theorem 2.8 is reported in the Appendix, Chapter A, Section A.1. To complete the definition of RMAX, one should specify an algorithm for computing the Q-value function Q defined in Equation 2.18. This equation is a fixed-point equation and can be solved by the value iteration algorithm presented in Section 2.2.2, page 18. From Theorem 2.5, page 19, the required number of iterations of the algorithm can be deduced to obtain an ϵ-accurate Q-value function in maximum norm with $\epsilon > 0$ a precision parameter. The RMAX procedure is detailed in Algorithm 4, page 31.

Several analyses of RMAX exist in the literature. We here briefly summarize the results reported by Strehl, Li, and Littman (2009) about the computational, space, and sample complexity of the algorithm. The computational complexity was already defined with the value iteration algorithm in Section 2.2.2, page 18. The *space complexity* refers to the amount of memory used by an algorithm. The *sample complexity* refers to the number of samples an algorithm requires to solve an MDP. It can be shown that RMAX is Probably Approximately Correct in Markov Decision Processes (PAC-MDP), with the understanding that all those complexities are bounded by a polynomial in the size of the problem. Formally, the definition follows.

Definition 2.7 (PAC-MDP algorithm)**.** An algorithm $\mathscr{A}$ yielding a policy whose value is ϵ-close in maximum norm to V^* is said to be an *efficient PAC-MDP* algorithm if, for any $\epsilon > 0$ and $0 < \delta < 1$, the per-timestep, computational complexity, space complexity, and the sample complexity of $\mathscr{A}$ are less than some polynomial in the relevant quantities $\left(S, A, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{R_{\max}}{1-\gamma}\right)$ with probability at least $1-\delta$. It is *simply PAC-MDP* if we relax the definition to have no computational complexity requirement.

Algorithm 4 RMAX

```
Set: MDP $\{\mathcal{S}, \mathcal{A}, T, r\}$; initial state distribution $\mathcal{P}_0$; horizon $H$.
Input: $n_{\text{known}}$; discount factor $\gamma$; $R_{\max}$.
for $(s, a) \in \mathcal{S} \times \mathcal{A}$ do
    Initialize $Q(s, a) \leftarrow \frac{R_{\max}}{1-\gamma}$
    Initialize $n(s, a) \leftarrow 0$
    Initialize $\hat{R}_s^a \leftarrow 0$
    for $s' \in \mathcal{S}$ do
        Initialize $\hat{T}_{ss'}^a \leftarrow 0$
    end for
end for
for each episode do
    $s \sim \mathcal{P}_0$ # Set the initial state.
    for $t \in \{1, \ldots, H\}$ do
        $a \leftarrow \text{argmax}_{\tilde{a} \in A} Q(s, \tilde{a})$ # Act greedily with respect to the current Q-function upper bound.
        $s' \sim T_{s\cdot}^a$ # Sample the next state.
        $r \leftarrow r_{ss'}^a$ # Sample the reward signal based on the transition.
        if $n(s, a) < n_{\text{known}}$ then
            $n(s, a) \leftarrow n(s, a) + 1$ # Increment the counter of current pair $s, a$.
            $\hat{R}_s^a \leftarrow \hat{R}_s^a + \frac{r - \hat{R}_s^a}{n(s,a)}$ # Update the reward model with the collected reward $r$.
            $\hat{T}_{ss'}^a \leftarrow \hat{T}_{ss'}^a + \frac{1 - \hat{T}_{ss'}^a}{n(s,a)}$ # Update the transition model with the outcome state $s'$.
            for $\tilde{s} \in \mathcal{S} \backslash \{s'\}$ do
                $\hat{T}_{s\tilde{s}}^a \leftarrow \hat{T}_{s\tilde{s}}^a \left(1 - \frac{1}{n(s,a)}\right)$ # Update the transition model for the other states.
            end for
            if $n(s, a) = n_{\text{known}}$ then
                # A new $(s, a)$ pair is known, update the upper bound $Q$ with value iteration.
                for $i \in \left\{1, \ldots, \left\lceil \frac{1}{\ln(1/\gamma)} \ln\left(\frac{R_{\max}}{\epsilon(1-\gamma)}\right) \right\rceil \right\}$ do
                    for $(s, a) \in \mathcal{S} \times \mathcal{A}$ do
                        if $n(s, a) \geq n_{\text{known}}$ then
                            $Q(s, a) \leftarrow \hat{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \hat{T}_{ss'}^a \max_{a' \in \mathcal{A}} \{Q(s', a')\}$ # Q-value iteration update.
                        end if
                    end for
                end for
            end if
        end if
        $s \leftarrow s'$
    end for
end for
```

First, the total computational complexity of RMAX using the value iteration algorithm as depicted earlier is

$$\tilde{\mathcal{O}}\left(\tau + \frac{S^2A^2\left(S + \ln(A)\right)}{1-\gamma} \ln\left(\frac{R_{\max}}{\epsilon(1-\gamma)}\right)\right)$$

when logarithmic factors are ignored, where τ is the total number of time steps. Secondly, the space complexity of RMAX is straightforwardly

$$\mathcal{O}\left(S^2A\right),$$

as it needs memory capacity for the empirical transition and reward model estimates as well as the Q-function estimate. Thirdly, the sample complexity result of RMAX follows.

Theorem 2.9 (RMAX sample complexity (Strehl, Li, and Littman, 2009))**.** *With probability $1-\delta$, the greedy policy with respect to Q computed by Algorithm 4, page 31, achieves an ϵ-optimal return in MDP M for all but (when logarithmic factors are ignored)*

$$\tilde{\mathcal{O}}\left(\frac{S\left|\{(s,a) \in \mathcal{S} \times \mathcal{A} \mid U(s,a) \geq V^*_M(s) - \epsilon\}\right|}{\epsilon^3(1-\gamma)^3}\right)$$

time steps, with U an upper bound on Q^*, the optimal Q-value function of M.

In this result is included the possibility to benefit from a tighter upper bound U on Q^* than $\frac{R_{\max}}{1-\gamma}$. Importantly, this accounts for the fact that the sample complexity of RMAX can be decreased thanks to the precision of U. We will see in Chapter 5 how to leverage this fact in a non-stationary environment.

Chapter 3

The planning *vs.* re-planning trade-off in stationary Markov Decision Processes

Sommaire

3.1 State of the art **34**
3.2 Open-loop control **38**
3.2.1 The case of closed-loop search trees 38
3.3 Open-loop search trees **41**
3.3.1 The Open-Loop Upper Confidence bound applied to Trees algorithm . . . 43
3.4 Open-Loop Tree search Algorithm **44**
3.4.1 Description 45
3.4.2 Distributional optimality criterion and decision criterion 46
3.5 Theoretical Analysis of the Open-Loop Tree search Algorithm **49**
3.6 Empirical Analysis of the Open-Loop Tree search Algorithm **53**
3.6.1 Heuristic decision criteria 53
3.6.2 1D Track Environment 54
3.6.3 Physical Traveling Salesman Problem 56
3.7 Conclusion **58**

Before tackling the possible ways for an environment to evolve over time, we focus on the stationary case, featuring no evolution. In this setting, we consider a planning agent building a plan, or sequence of actions, to be applied in an MDP. We ask ourselves the following question:

How long can an agent follow a plan without further re-planning in a stationary MDP?

Blindly following a plan, without analyzing if its realization corresponds to the prediction, consists in an open-loop control that presents a considerable advantage, namely, it spares potential future re-planning steps that often result in a costly process. A major use case is the one of tree search algorithms that often do not rely on persistent information. For instance, the MCTS algorithm presented in Chapter 2 repeats a complete tree construction procedure at each decision epoch, based on the current state. In this chapter, we propose to leverage the fact that the MDP is stationary to avoid re-planning at subsequent decision epochs. Instead, an open-loop plan is followed as long as it is *valid*, with a particular understanding of validity. Therefore, the resulting control is open-loop .

The approach we propose is a general framework, suitable to any planning algorithms performing tree search either in the trajectories space or the plans space of an MDP. Particularly, we will apply the method to an open-loop variant of the UCT algorithm introduced in Chapter 2. We assess the optimality guarantees of the resulting Open-Loop Tree search Algorithm (OLTA) algorithm.

This chapter is organized as follows. First, in Section 3.1, we review the state of the art of tree search algorithms and motivate the use of such a planning strategy. Then we report what has been done in terms of using persistent information in that field. Secondly, we define open-loop control with search trees in Section 3.2, page 38. We distinguish between closed and open-loop search trees, yielding two different ways of considering open-loop control with trees. Thirdly, we define our framework to avoid re-planning in Section 3.4, page 44. We introduce the OLTA algorithm, an open-loop control tree search algorithm allowing reduced computational complexity compared to its closed-loop counterpart. In Section 3.5, page 49, we assess theoretically the optimality guarantees of OLTA. In Section 3.6, page 53, we demonstrate empirically the performances of OLTA and show that it achieves a compromise between computational complexity and optimality. Finally, we conclude in Section 3.7, page 58.

3.1 State of the art

Solving a Markov Decision Process can be a computationally intensive process, depending on the size of its state-action space. The size of such a problem grows exponentially with the number of dimensions of $\mathcal{S} \times \mathcal{A}$, which makes the search for an optimal policy difficult. Richard E. Bellman coined the expression “curse of dimensionality” to refer to this phenomenon in his work on Dynamic Programming. Let us illustrate this with an example to have a better grasp on how hard could solving an MDP be. Consider an agent evolving within $M = \{\mathcal{S}, \mathcal{A}, T, R\}$. As we have seen it in Chapter 2, computing an optimal policy can be achieved by computing the optimal Q-value function. If a true model (Definition 2.6, page 18) of M is available, one could use the value iteration algorithm (Algorithm 1, page 20) to deduce an ϵ-accurate estimate of Q^* in maximum norm. However, with a discount factor $\gamma \in [0, 1)$, we saw in Theorem 2.6, page 20 that the computational complexity of value iteration is

$$\mathcal{O}\left(S^2 A \frac{1}{\ln(1/\gamma)} \ln\left(\frac{R_{\max}}{\epsilon(1-\gamma)}\right)\right),$$

which scales with $\mathcal{O}\left(S^2 A\right)$. Let us now take the example of the game of Go and figure out how much computation we should apply to solve it. This game is famous for its combinatorial nature, it features a board game with a grid pattern of size $19 \times 19 = 361$. At each turn, one of the two players can put a stone in one of the cells of the grid, resulting in a number of actions $A = 361$. The cells are either free or occupied by a single stone, hence all the actions are not always available. To be completely fair in our calculation, let us take $A = C \geq 1$, a positive constant, which is enough to serve our purpose of showing the tremendous effort needed to solve the game of Go by value iteration. The problem becomes combinatorial when it comes to enumerating all the possible states. The state of the game is described by 361 variables, each taking a value among three possibilities, hence $S = 3^{361} = 1.7 \times 10^{172}$. From Theorem 2.6, page 20, applying value iteration to the game of Go results in a number of operations upper bounded by $CS^2 = C \times 10^{344}$. Let us now assume that we have a super computer with a clock rate of 10 GHz, which, surpasses any commercial computer's clock rate in 2020. Solving the game of Go using such a super computer would take more than 10^{334} seconds, which is 10^{317} times the age of the universe ($\approx 10^{17}$ seconds). If we were able to make a perfect parallel computation in this case, it would require more than 10^{327} parallel processes to solve the game in one year. Even further, if — instead of a true model — we would use a true *generative* model, the analogous resolution method would first run into the problem of estimating the true model and then applying the Q-value iteration algorithm, similarly to the RMAX algorithm (Algorithm 4, page 31). No need to say that the problem becomes overly unfeasible in this case.

Despite all these inconveniences, the game of Go has been "solved" already (Coulom, 2006; Gelly and Silver, 2008; Enzenberger, Muller, Arneson, and Segal, 2010; Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, et al., 2016), in the sense of finding a close to optimal policy. The use of quotation marks in this last sentence is not trivial as the employed methods are not exact and approximate an optimal policy. Still, those approximations were able to outperform the best human players, seen as an unreachable baseline among the community a few years before. This achievement is mainly due to the progresses related to both the MCTS planning algorithm and the deep Reinforcement Learning methods.

MCTS has many appealing features that made it a first choice algorithm for planning within an MDP. First, it scales better to the size of the problem than exact methods such as value iteration. Secondly, it is any-time, meaning that the algorithm can be stopped at any time of the computation. The quality of the provided solution depending of course on the amount of time the algorithm is run. As seen in Chapter 2, an MCTS procedure is defined with a budget $B \in \mathbb{N}$, corresponding to the number of complete Monte Carlo simulations performed. It can be proved that the resulting policy converges to the optimal policy when B tends to infinity. Thirdly, MCTS only requires a generative model of the environment (Definition 2.6, page 18) which is less restrictive than a complete model. For all those reasons, the MCTS algorithm received a lot of interest from the scientific community over the past decades. Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, and Colton (2012) realized a quite complete survey of the enhancements brought to the algorithm from the first time it was proposed until 2011. Contributions encompass tree policy enhancements (Gelly and Silver, 2007; Keller and Eyerich, 2012; Keller and Helmert, 2013); default policy enhancements (Gelly and Silver, 2007; Silver and Tesauro, 2009; Rimmel and Teytaud, 2010; Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, et al., 2016); pruning for large, possibly continuous, state-action spaces (Mansley, Weinstein, and Littman, 2011; Couëtoux, Hoock, Sokolovska, Teytaud, and

Bonnard, 2011; Gelly and Silver, 2011; Keller and Eyerich, 2012; Auger, Couëtoux, and Teytaud, 2013; Hostetler, Fern, and Dietterich, 2014); back-propagation enhancements (Xie and Liu, 2009; Keller and Helmert, 2013; Feldman and Domshlak, 2014); and other enhancements linked to specific domains (Silver and Veness, 2010; Bouzy, Métivier, and Pellier, 2011; Perez, Rohlfshagen, and Lucas, 2012a; Baker, Ramchurn, Teacy, and Jennings, 2016). Overall, more than four hundred papers published in ten years illustrate this scientific enthusiasm. We will see MCTS as a canonical model-based planning algorithm and greatly inspire ourselves from its structure to derive our open-loop control framework. As we will see in Section 3.2.1, page 38, the application of this framework is straightforward in the case of closed-loop search such as preformed by MCTS. Therefore, we extend to the case of open-loop search. Notice here that the terminology may be ambiguous: open-loop *search* refers to a search carried in the plan space of an MDP whereas open-loop *control* refers to the sequential application of actions independently from the reached states in-between actions. Let us now review the state of the art in terms of open-loop tree search.

As introduced in Chapter 2, Section 2.2.3, page 20, open-loop planning is a search in the action sequences or plan space of an MDP. The tree structure associated to such a search method is solely composed with action sequences and the optimality of an action within a node is considered with respect to the distribution of states at this node. This distribution depends on the initial state at the root of the tree and the course of actions leading to this node. Of course, such a method is generally sub-optimal compared to closed-loop planning such as vanilla MCTS. Despite the sub-optimality, open-loop planning is useful for stochastic environments, especially with a high branching factor. We will develop the reasons why this approach is attractive in Section 3.2.1, page 38. On this line of thought, Bubeck and Munos (2010) proposed the Open-Loop Optimistic Planning (OLOP) algorithm, an open-loop planning algorithm applying the principle of optimism in the face of uncertainty, as seen with the UCT and RMAX algorithms in Chapter 2, Sections 2.2.3, page 20 and 2.3.2, page 28. The search is carried in the plans space and focuses on the most promising plan. They provide an analysis of the performance of OLOP, with respect to a measure of the proportion of near-optimal plans. Similarly, Weinstein and Littman (2012) proposed the Hierarchical Open-Loop Optimistic Planning (HOLOP) algorithm, also planning in the space of trajectories. They demonstrate strong empirical performances, close to optimal, despite the sub-optimal nature of open-loop planning. Perez Liebana, Dieskau, Hunermund, Mostaghim, and Lucas (2015) propose an experimental study of several open-loop planning algorithms in General Video Game Playing (GVGP) (Levine, Bates Congdon, Ebner, Kendall, Lucas, Miikkulainen, Schaul, and Thompson, 2013), including MCTS. Although their analysis does not focus on the benefit of open-loop planning, which is seen as a necessity to deal with stochastic games, the tested algorithms demonstrated state of the art results. The Open-Loop Expectimax Tree Search (OLETS) algorithm (Perez Liebana, Samothrakis, Togelius, Schaul, Lucas, Couëtoux, Lee, Lim, and Thompson, 2015) proposed to the 2014 General Video Game Playing competition is a variant of the HOLOP algorithm (Weinstein and Littman, 2012) in the context of GVGP. It conserves the open-loop planning strategy and is specifically designed for the competition with the use of a dedicated evaluation function and a finite set of actions. In this case, it was proven to outperform every competitors, especially closed-loop planners.

The framework we derive in this chapter is based on the following observation: tree search algorithms discard the computed tree at each decision epoch and never reuse it thereafter. Precisely, at each decision epoch, all the aforementioned planning algorithms compute a complex tree of the possible trajectories starting from the current state, apply the recommended action at the root node and start this cumbersome process once again after transitioning to the next state. From this observation and the fact that the environment does not change between decision epochs, we ask the following question:

Can a search tree be reused between subsequent decision epochs?

On this line of thought, Perez, Rohlfshagen, and Lucas (2012a) considered keeping the tree constructed with MCTS after applying the selected action $a \in \mathcal{A}$ at the root node. At the next decision epoch, the MCTS procedure is repeated with the same budget B but starting from a non-empty tree corresponding to the sub-tree whose root node is the decision node reached with a. This results in a new tree initialization for each decision epoch. However, they reported no empirical gain in their experiments and concluded to the unsuitability of this method in the environments they showcase. Heusner (2011) also applied a systematic reuse of the tree in the game of Ms. Pac-Man where the only source of uncertainty was the change of direction of the ghosts. He considered discarding the whole tree if one of these changes occurred, hence reducing the problem to a deterministic version. The AlphaGo Zero algorithm, developed for the game of Go, also retains the information of sub-trees between decision epochs (Silver, Schrittwieser, Simonyan, Antonoglou, Huang, Guez, Hubert, Baker, Lai, Bolton, et al., 2017). Like Heusner (2011), they keep the sub-tree reached by the application of the selected action only if its root node corresponds to the reached state in the real game. Powley, Cowling, and Whitehouse (2017) proposed a method that they called tree flattening for low computational resources ensemble MCTS (Fern and Lewis, 2011). The latter refers to the sequential application of MCTS procedures to produce an ensemble of decision makers and decide with respect to a vote between them. The approach proposed by Powley, Cowling, and Whitehouse (2017) consists in keeping statistics of the root node of subsequently constructed tree in ensemble MCTS. They keep track of the back-propagated rewards and the visit counts and show the positive effect of this method in experiments. Soemers, Sironi, Schuster, and Winands (2016) studied several enhancements to the MCTS algorithm in the context of GVGP. Among them, the possibility to reuse the sub-tree whose root node corresponds to the reached state after application of the recommended action in the initial tree. The developed idea, already suggested by Perez Liebana, Dieskau, Hunermund, Mostaghim, and Lucas (2015), was proven to yield successful experimental results in the context of GVGP. Although applied to a different domain, this corresponds to the same approach as the one used by Perez, Rohlfshagen, and Lucas (2012a), Heusner (2011), and Silver, Schrittwieser, Simonyan, Antonoglou, Huang, Guez, Hubert, Baker, Lai, Bolton, et al. (2017).

Overall, all the aforementioned methods in terms of tree reuse have been empirically tested. Vanilla MCTS-like algorithms are generally proven to converge to an optimal policy as B tends to infinity. Such a guarantee may disappear when keeping track of a previously constructed tree. For instance, the state reached by application of an action at the current state may not correspond to the root state of the corresponding sub-tree. In this chapter, we propose a theoretical analysis of the optimality guarantees in that setting, grounded on the hypothesis that the environment does not change over time. More than re-using a tree, we propose to avoid re-planning to decrease the computational complexity of an algorithm. We study the application of open-loop control in both cases of closed and open-loop search.

3.2 Open-loop control

Usually, an agent determines the action it takes based on the current state as illustrated in the RL principle of Figure 2.4, page 27. When the decision depends on the state, we say the control is *closed-loop*. This is the case for planning algorithms such as MCTS: a lookahead search is performed starting from the current state at each decision epoch. This is also the case for learning algorithms that select actions based on a policy — generally among those described in Table 2.1, page 13 — whose action selection is conditioned on the current state. Conversely, we say that the control is *open-loop* when the decision rule is not dependent on the current state. For instance, if we consider an initial state s_0 and a sequence of actions $\alpha \in \mathcal{A}^k, k \in \mathbb{N}$, applying subsequently the actions in α, starting at s_0, without taking into account the states reached along the trajectory, consists in an open-loop control. Such a decision rule is generally sub-optimal as it is blind to the state reached by the agent. Particularly, in stochastic MDPs, *i.e.*, with a non-deterministic transition function, the reached state after a transition may vary across executions, which could create unpredictable outcomes. Despite its sub-optimal nature, open-loop control generally presents the advantage of having a minimum computational complexity. For instance, applying a sequence of actions requires a constant number of computing operations $\mathcal{O}(1)$ at each decision epoch, which is appealing.

Our objective is to design a generic method for open-loop control with tree search algorithms. We propose to investigate methods achieving this by means of search tree reuse, as reviewed in Section 3.1, page 34. We already defined a certain type of search trees created by the MCTS algorithm in Section 2.2.3, page 20, namely closed-loop search trees. The scope of our study will extend to the case of open-loop search trees. Hence, in the next two sections, we will view open-loop control in both cases.

3.2.1 The case of closed-loop search trees

We defined *closed-loop planning* in Section 2.2.3, page 20 as a search happening in the trajectories space of an MDP where we sample trajectories of the form $\{(s_i, a_i)\}_{i=0}^{H}$. Particularly, this means that we keep track of the states reached along the trajectory and distinguish between them. If we were to represent this search with a tree, this would amount to the MCTS search tree represented in Figure 2.2, page 21. This type of search tree is composed of decision nodes, labeled by a state, alternating with chance nodes, labeled by a state-action pair. The fact that this search is closed-loop, *i.e.*, that we distinguish reached states along the path, allows to define the value of decision and chance nodes with respect to their unique labeling states or state action pairs. Indeed, for the MCTS algorithm, we defined those values in Section 2.2.3, page 20 as follows:

$$V(\nu_s) = \max_{\nu' \in \nu_s.\text{children}} V(\nu') ,$$

$$Q(\nu_s^a) = \frac{1}{k} \sum_{i=1}^{k} Z_i,$$

where ν_s and ν_s^a are the decision and chance nodes corresponding to $(s, a) \in \mathcal{S} \times \mathcal{A}$ and $\{Z_i\}_{i=1}^k, k \in \mathbb{N}$ is the collection of the discounted returns collected during Monte-Carlo simulations starting from (s, a). From there, let us formulate a simple way to perform open-loop control with a closed-loop search tree by means of tree reuse.

In Section 3.1, page 34, we reported contributions performing closed-loop search tree reuse to improve the quality of an algorithm (Heusner, 2011; Perez, Rohlfshagen, and Lucas, 2012a; Perez Liebana, Dieskau, Hunermund, Mostaghim, and Lucas, 2015; Soemers, Sironi, Schuster, and Winands, 2016; Silver, Schrittwieser, Simonyan, Antonoglou, Huang, Guez, Hubert, Baker, Lai, Bolton, et al., 2017). Unlike what we present, they do not take advantage of the tree reuse to perform open-loop control, but with the hope to increase the quality of the solution provided by the planning procedure. They have in common the method of re-using a search tree, plus the fact that this one is closed-loop. To perform open-loop control, we build on this reuse method, defined as follows: once a tree is built and the recommended action at the root applied in the real world, the sub-tree corresponding to the reached state at the following decision epoch is used as the new tree. If this sub-tree does not exist, mainly because the reached state was not sampled during the search, the whole tree is discarded. Let us explain this method in detail with an example. We put ourselves in the case where we want to build a tree at decision epoch $t_0 \in \mathcal{T}$ and reuse it at subsequent decision epoch $t_1 \in \mathcal{T}$. The root node of the tree is a decision node corresponding to the current state $s_0 \in \mathcal{S}$. Its subsequent child nodes are the chance nodes corresponding to the available actions at s_0. Those chance nodes themselves have one or several decision node children corresponding to the states sampled from the applications of the actions at s_0. Let us denote by $\{s_1^i\}_{i=1}^k$ this population of $k \in \mathbb{N}$ sampled states, corresponding to the decision epoch t_1. The search is performed in the memory of a computer, but once the tree is built, an action $a_0 \in \mathcal{A}$ is selected and a transition occurs in the *real* environment, yielding a new state s_1. The closed-loop search tree reuse method is then applied: if $s_1 \in \{s_1^i\}_{i=1}^k$, then the search sub-tree whose root node is ν_{s_1} serves as a new tree for decision epoch t_1, the remainder of the tree is discarded; else, s_1 is a new state, never sampled before, the whole tree is discarded and a new one is built with s_1 as the labeling state of the root node. This process is illustrated in Figure 3.1, page 40.

Given this method for closed-loop search tree reuse, we now conceive a simple algorithm for open-loop control in this setting. We develop on the simple idea that after reaching a sub-tree with the procedure illustrated in Figure 3.1, page 40, one can directly rely on this sub-tree without developing it further. The process can then be repeated as long as there

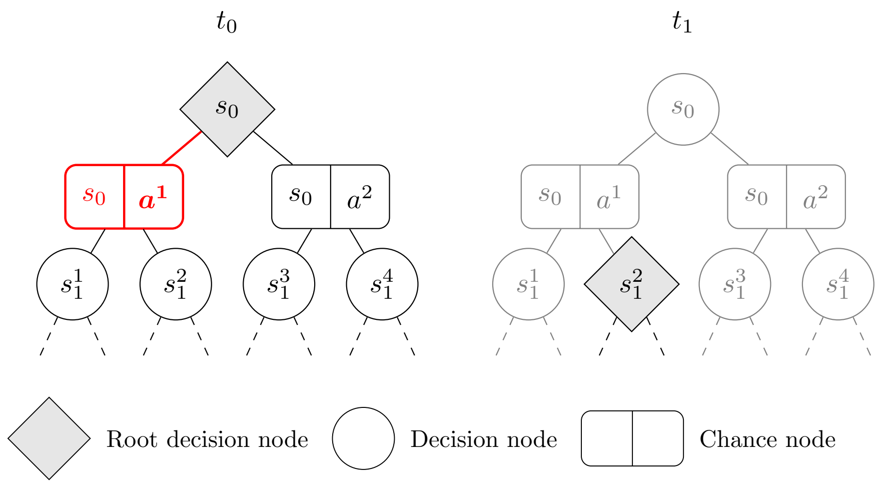


Figure 3.1: Reuse scheme of a closed-loop tree. On the left is represented the search tree at decision epoch t_0 with the root decision node labeled with state s_0. In red, the chance node corresponding to the maximum estimated Q-value is selected, action a^1 is taken. We suppose state s_1^2 was reached after applying a^1 at s_0. On the right, at decision epoch t_1, is represented the reused tree in black and the discarded part of the tree in gray. The new tree is the one whose root node is s_1^2.

exist a sub-tree corresponding to the reached states of the trajectory. Algorithm 5, page 41 describes this procedure in an online context.

Notice that this algorithm is not strictly open-loop — hence the name *pseudo* open-loop control — since the action selection is conditioned on the current state. However, the only operation is a state-wise comparison and no computational resources are dedicated to the estimation of the optimal action, which contrasts with closed-loop control algorithms such as MCTS. This can allow a considerable saving in computational complexity. The counterpart of this benefit is a loss of guarantees concerning the optimality of the selected action in a sub-tree. We saw in Theorem 2.7, page 26 that one can provide an upper bound on the probability of selecting a sub-optimal optimal action with the UCT algorithm. Applying Algorithm 5, page 41 with $\mathscr{A}$ = UCT allows to extend this upper bound to open-loop control and, by extending Theorem 2.7, page 26, we have the following upper bound on the probability of selecting a sub-optimal action in any reused sub-tree:

$$\mathbf{Pr}\left(a_b \neq a^*\right) \leq \left(\frac{1}{b}\right)^{\frac{\rho}{2}\left(\frac{\min_{a \neq a^*} \Delta_a}{36}\right)^2}, \tag{3.1}$$

with $b \in \{1, \dots, B\}$ the budget with which the sub-tree has been developed, *i.e.*, the number of times MC-simulations sampled the root node of the sub-tree following the UCT procedure.

Notice that the decision criterion to select and reuse a sub-tree only based on the fact that the sampled state corresponds to an existing sub-tree could be improved. Indeed, thanks to Equation 3.1, one could argue that with a small budget $b \in \{1, \dots, B\}$, the sub-tree may provide poor guarantees regarding the optimality of the proposed action at the root

Algorithm 5 Pseudo open-loop control with a closed-loop tree search algorithm

```
Set: MDP $\{\mathcal{S}, \mathcal{A}, T, r\}$; initial state distribution $\mathcal{P}_0$; horizon $H$.
Input: closed-loop tree search algorithm $\mathscr{A}$.
$s \sim \mathcal{P}_0$  # Set the initial state.
$\Gamma \leftarrow \mathscr{A}(s)$  # Initialize a tree with the initial state labeling the root node.
for $t \in \{1, \dots, H\}$ do
    Select action $a$ with respect to $\Gamma$
    $s' \sim T^a_{s\cdot}$  # Sample the next state.
    if $s'$ corresponds to a decision node of depth 1 in $\Gamma$ then
        $\Gamma \leftarrow$ sub-tree of $\Gamma$ corresponding to $s'$
    else
        $\Gamma \leftarrow \mathscr{A}(s')$  # Re-build a new tree starting from $s'$.
    end if
    $s \leftarrow s'$
end for
```

node. Straightforwardly, the condition that the budget is "large enough" to provide sufficient optimality guarantees with respect to Equation 3.1 can be added to Algorithm 5, page 41 to improve the decision of re-planning or not. Other decision criterion based on other statistics of the tree such as depth, width, *etc.* could also be used, depending on the application.

In this section, we presented a simple method for open-loop control re-using closed-loop search trees for which we provided optimality guarantees. Developing the equivalent method for open-loop trees is not straightforward as we will now see. The remainder of this chapter is dedicated to the formal description of this methodology and its analysis both theoretically and empirically. First, let us describe open-loop search trees in the next section.

3.3 Open-loop search trees

What motivates the use of open-loop search trees is the fact that closed-loop search becomes unfeasible in MDPs with a large or continuous state spaces. This fact is accentuated when many different states can result from a single transition. To better understand this, let us take the example of an MDP $M = \{\mathcal{S}, \mathcal{A}, T, R\}$ with $\mathcal{S}$ continuous and $T^a_{s\cdot}$ a continuous probability distribution for all $s, a \in \mathcal{S} \times \mathcal{A}$ (for instance Gaussian). Building an MCTS tree from a root state $s_0 \in \mathcal{S}$ according to Algorithm 2, page 24 results in the following phenomenon: every time an action a is performed from s_0, a *new*, never encountered before, state $s' \in \mathcal{S}$ is sampled. Indeed, since $T^a_{s_0\cdot}$ is continuous, the probability to sample twice the resulting state s' is zero. This is why the MCTS algorithm is generally modified to cope with this case in the context of closed-loop search trees, for instance with progressive strategies (Chaslot, Winands, Uiterwijk, Van Den Herik, and Bouzy, 2007; Coulom, 2007; Couëtoux, Hoock, Sokolovska, Teytaud, and Bonnard, 2011).

Open-loop search trees prevent such a phenomenon from occurring. Before detailing their structure, we enumerate the hypotheses that we make for the remainder of this chapter.

1. We assume a true generative model (T, R) to be available.
2. We assume the number of actions to be finite, *i.e.*, $A < \infty$. We make no particular hypothesis on $\mathcal{S}$.
3. For simplicity, we assume that the available actions are independent of the state the agent lies in, *i.e.*, $\mathcal{A}(s) = \mathcal{A}$ for all $s \in \mathcal{S}$.

Notation 3.1. We consider $A \in \mathbb{N}$ actions and write $\mathcal{A} = \{a^i\}_{i=1}^{A}$ the set of actions.

Open-loop search trees consist in a look-ahead search of the possible outcomes while following some action plans $\{a_i\}_{i=0}^{H}$ starting from the current state $s_0 \in S$. Thus, the root node of the tree is labeled by the unique state s_0. The edges correspond to the actions, hence A is the branching factor of the tree. The tree itself conforms to an ensemble of action sequences, or plans, originating from its root node.

We emphasize the fact that this tree structure implies that we search for a state-independent optimal sequence of actions (open-loop plan) which is in general sub-optimal compared to a state-dependent search, performed for instance by MCTS. The THTS family of algorithms in particular (Keller and Helmert, 2013) defines trees with chance and decision nodes while the open-loop structure does not apply an equality operator on the sampled states. Bubeck and Munos (2010) and Weinstein and Littman (2012) argue that closed-loop application of the first action in open-loop plans, although theoretically sub-optimal, can be competitive with these methods in practice, while being more sample-efficient.

Since the transition model is stochastic, the non-root nodes are not labeled by unique states. Instead, every such node is associated to a state distribution resulting from the application of the action plan leading to the considered node and starting from s_0. During the exploration, we consider saving all sampled states at each non-root node. In open-loop search trees, the leaf nodes correspond to nodes where the A actions have not all been tested. A comprehensive illustration of such a tree can be found in Figure 3.2, page 43. Within an open-loop search tree, the *depth* of a node is defined by the length of the action sequence required to reach this node.

For future needs, we here introduce the notion of recommended sub-trees. Let us consider an open-loop search tree built by an algorithm with a recommendation function. Generally, this function is to select the action leading to the node of maximum value. We will write Γ_d and call *recommended sub-tree at depth* $d \in \mathbb{N}$ the sub-tree resulting from the application of the d first recommended actions. Hence Γ_0 denotes the whole tree, Γ_1 the tree starting from the node reached by the application of the first recommended action and so on.

As seen in Section 3.1, page 34, several algorithms using open-loop search trees have been designed for different purposes. The framework we propose in this chapter for open-loop control builds on a new open-loop search tree algorithm that we introduce in the following section.

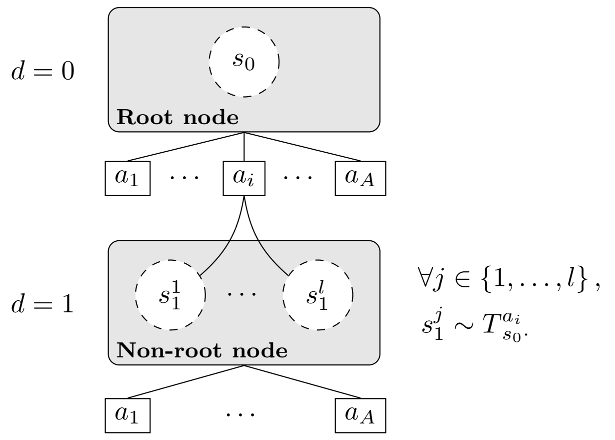

$d = 0$
s_0
Root node
a_1 $\cdots$ a_i $\cdots$ a_A
$d = 1$
s_1^1 $\cdots$ s_1^l
Non-root node
$\forall j \in \{1, \ldots, l\}$,
$s_1^j \sim T_{s_0}^{a_i}$.
a_1 $\cdots$ a_A

Figure 3.2: General representation of an open-loop tree, where $l \in \mathbb{N}$ is the number of times the sub-tree reached by action a^i has been developed. Two nodes are represented in this tree with their respective depths on the left.

3.3.1 The Open-Loop Upper Confidence bound applied to Trees algorithm

Open-Loop Upper Confidence bound applied to Trees (OLUCT) is an open-loop version of the UCT algorithm. The fundamental difference between both is that OLUCT is not provided with an equality operator over states. In other words, we do not distinguish between reached states during the search but only between sequences of actions, as illustrated in Figure 3.2, page 43. With the closed-loop search terminology, this means that decision and chance nodes do not correspond to unique states but to the state distribution reachable by the action plan leading to the node. Hence, decision and chance nodes are associated to the state distribution which makes OLUCT an open-loop planning algorithm. The fundamental consequence is that an action value within our tree is computed with respect to the parent node's state distribution rather than a single state. Apart from this, OLUCT uses the same exploration procedure as UCT. We now explain in detail the OLUCT procedure.

Let us consider planning with a finite budget $B \in \mathbb{N}$. Within a node, we denote by $\hat{Z}_{i,u}$ the estimated expected (discounted) return of action $i \in \{1, \ldots, A\}$ after $u \in \mathbb{N}$ samples of this action. We also denote by $T_i(t)$ the number of trials of action $i \in \{1, \ldots, A\}$ up to time $t \in \{1, \ldots, B\}$. This number of trials is the total number of MC-simulations taking action i after applying the plan leading to the considered node. The value of the nodes are different from the definition we had with closed-loop trees. First, the value of an action at time t within a node ν is defined by the estimated expected (discounted) return $\hat{Z}_{i,T_i(t)}$. In turn, the value of ν at time t is defined by its maximizing action:

$$V(\nu) = \max_{i \in \{1, \ldots, A\}} \left\{ \hat{Z}_{i,T_i(t)} \right\} \tag{3.2}$$

A node being labeled by a set of states instead of a unique state, its value reflects the expected (discounted) weighted by the state distribution. Like UCT, a UCB strategy (Auer, Cesa-Bianchi, and Fischer, 2002) is applied at each node where each action is seen as an arm of a bandit problem. At time $t \in \{1, \dots, B\}$, the tree policy selects the action I_t with the highest UCB, defined as follows:

$$I_t = \underset{i \in \{1,\dots,A\}}{\text{argmax}} \left\{ \hat{Z}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\},$$

where, similarly to Equation 2.13, page 25 defining the UCT tree policy,

$$c_{t,u} = 2C_p \sqrt{\frac{ln(t)}{u}} \tag{3.3}$$

is an exploration term ensuring that all actions will be sampled infinitely often. The C_p parameter is a positive constant that drives the exploration *vs.* exploitation trade-off. We do not specify the default policy used by the algorithm. Unless given a better one, the random policy can be used. The OLUCT procedure is detailed in Algorithm 6.

Algorithm 6 OLUCT

```
Set: MDP $\{\mathcal{S}, \mathcal{A}, T, R\}$; initial state distribution $\mathcal{P}_0$; horizon $H$.
Input: generative model $M = (T, R)$; budget $B$; default policy $\pi_{\text{default}}$.
$s \sim \mathcal{P}_0$  # Set the initial state.
for $t \in \{1, \dots, H\}$ do
  Create root node $\nu_{root}(s)$  # Initialize the search tree with the current state $s$.
  for $i \in \{1, \dots, B\}$ do
    $\nu_{leaf} \leftarrow$ select$(\nu_{root}, M)$;  # Select a leaf node following the OLUCT tree policy strategy and sample a new state for each encountered node.
    Expand$(\nu_{leaf}, M)$;  # Expand the node using the generative model.
    $Z \leftarrow$ simulation$(\nu_{leaf}, \pi_{\text{default}}, M, H)$;  # Simulate a roll-out using $\pi_{default}$, starting from the last sampled state in $\nu_{leaf}$.
    backPropagation$(\nu_{leaf}, Z)$;  # Update the values of the encountered nodes with the collected return (see Equation 3.2).
  end for
  $s' \sim T^a_{s\cdot}$  # Sample the next state.
  $s \leftarrow s'$
end for
```

3.4 Open-Loop Tree search Algorithm

In this section, we elaborate on the open-loop control methodology using open-loop search trees. This is the counterpart of Section 3.2.1, page 38 with open-loop trees. We first describe the OLTA algorithm, practically implementing the proposed framework. Then we discuss implementation details related to the algorithm.

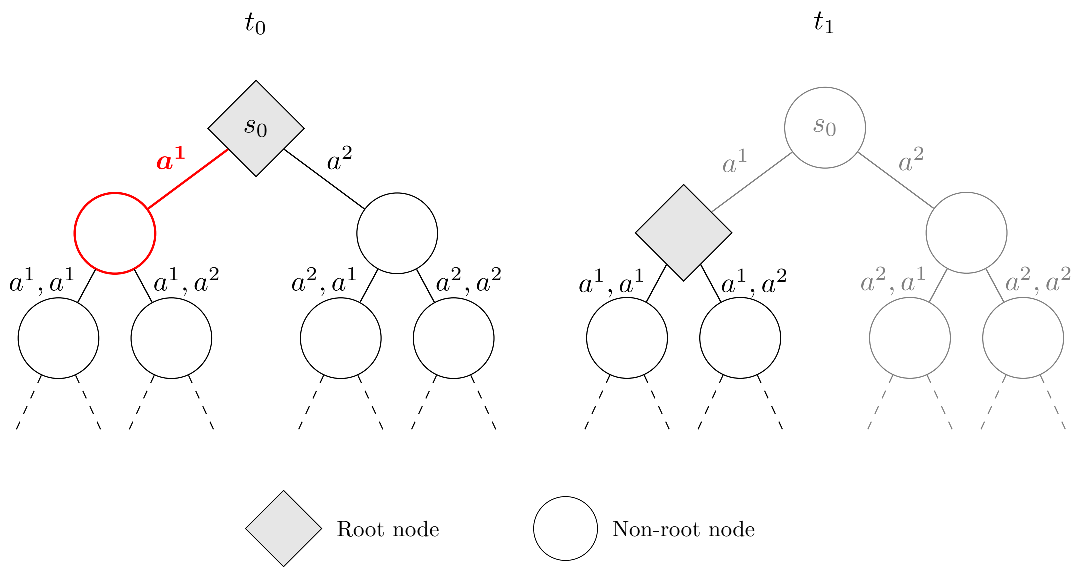


Figure 3.3: Reuse scheme of an open-loop tree. At decision epoch t_0, action a^1 is recommended by the algorithm and applied in the real world. The non-root node reached with a^1 is the new root node. The remainder of the tree is discarded.

3.4.1 Description

The OLTA algorithm achieves open-loop control with open-loop search trees of the form described in Figure 3.2, page 43. It relies on a generic open-loop planning algorithm to generate a tree, rooting from the current state. This algorithm could be the OLUCT procedure described in Algorithm 6 for instance. The OLTA procedure is described below.

1. A tree is created with the generic open-loop tree search algorithm.
2. The recommended action at the root of the whole tree Γ_0 is selected and applied.
3. Γ_1 is proposed to be reused as the new tree to a decision criterion that we will detail further. If the tree is selected, the procedure goes back to step 2, else, to step 1.

This open-loop search tree reuse scheme is illustrated in Figure 3.3. For subsequent execution time steps, OLTA decides either to use the sub-tree reached by the recommended action or to trigger a re-planning by building a new tree. If no re-planning is triggered, then the recommended action of the sub-tree is applied without using the additional information of the new state observed by the agent after the transition. This results in an open-loop control process and spares the cost of developing a new tree starting at this state. Overall, the intuition behind OLTA is that several consecutive recommended actions in an optimal branch of the tree can be reliable, despite the randomness of the environment. A major example is low-level control, where consecutive sampled states are close to each other. Even more, the planning rate in such a case is generally intense, which motivates the use of low computational complexity algorithms. OLTA is summarized in Algorithm 7, page 46.

One important feature of OLTA is the so-called *"decisionCriterion"*, based on which the agent decides to either use the first recommended sub-tree, or to re-build a new tree from the current state. The decision is based on a comparison with the characteristics of the resulting sub-tree and the current state. In the next section, we discuss different decision criteria, leading to the consideration of a family of different algorithms.

Notice that — similarly to Algorithm 5, page 41, its closed-loop counterpart — OLTA does not achieve strict open-loop control. Indeed, the action selection is conditioned on the current state since it is used by the decision criterion. However, the only operation is a state analysis and no computational resources are dedicated to the estimation of the optimal action, which contrasts with closed-loop control algorithms such as MCTS. We argue that, like Algorithm 5, page 41, this analysis has comparatively low computational complexity compared to re-planning strategies. This fact will be demonstrated empirically in a latter section.

Algorithm 7 OLTA

```
Set: MDP $\{\mathcal{S}, \mathcal{A}, T, R\}$; initial state distribution $\mathcal{P}_0$; horizon $H$.
Input: Open-loop planning algorithm $\mathscr{A}$; re-planning criterion decisionCriterion; action
selection procedure recommendedAction.
$s \sim \mathcal{P}_0$  # Set the initial state.
$\Gamma \leftarrow \mathscr{A}(s)$  # Initialize a complete open-loop search tree.
for $t \in \{1, \dots, H\}$ do
  if decisionCriterion$(s, \Gamma)$ recommends re-planning then
    $\Gamma \leftarrow \mathscr{A}(s)$  # Create a new open-loop search tree from the current state.
  end if
  $a \leftarrow$ recommendedAction$(\Gamma)$  # Select the recommended action at the root node.
  $\Gamma \leftarrow$ subTree$(\Gamma, a)$;  # Set the sub-tree reached from the application of $a$ as the main
  tree.
  $s' \sim T^a_{s\cdot}$  # Sample the next state.
  $s \leftarrow s'$
end for
```

The OLTA procedure starts by using a root node labeled by a unique state and then solely relies on sub-trees whose root nodes are labeled by distribution of states. This echoes to the Partially Observable Markov Decision Process (POMDP) (Kaelbling, Littman, and Cassandra, 1998) literature where uncertainty on the current state implies the creation of a belief state which is a probability distribution on $\mathcal{S}$ (Silver and Veness, 2010). Since OLTA can operate without certainty on the root state as seen in the tree reuse phase, it should be noted that the extension to the POMDP case is straightforward.

3.4.2 Distributional optimality criterion and decision criterion

The simplest implementation of the decision criterion is to keep the sub-tree only if its root node is fully expanded. This means that each action has been sampled at least once and the action values estimates updated with at least one roll-out with respect to π_{default}. This is the minimum requirement to recommend an action. We call the resulting algorithm *Plain OLTA*. It naively trusts the value estimates of the sub-tree, thus applies the whole plan of recommended actions until it reaches a partially expanded node. Therefore, Plain OLTA is expected to perform better in deterministic environments. In stochastic cases however, those estimates may be biased because of the different sources of uncertainty within the MDP. For this reason, we seek more robust criteria to base the decision on.

An ideal way to decide whether to keep the sub-tree or not is to track if the recommended action is optimal with respect to the new state s. Here we make an important distinction between a *state-wise optimal action* and a *distribution-wise optimal action*. The first one is the action selected by the optimal policy at a specific state. We write it $a^* = \text{argmax}_{a \in \mathcal{A}} Q^*(s, a)$, with $Q^* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ the optimal Q-value function. In order to define the second one, we introduce S_d, the state random variable at the root node of Γ_d. Its distribution results from the application of the d first recommended actions starting from s_0. Let us denote by $a_0, \dots, a_{d-1} \in \mathcal{A}^d$ the d first recommended actions. We denote by P_{S_d} the probability distribution associated to the random variable S_d. Formally, P_{S_d} is defined by

$$\begin{array}{rccl} P_{S_d} : & \mathcal{S} & \rightarrow & \mathbb{R} \\ & s & \mapsto & \mathbf{Pr}(S_d = s) = \int_{\mathcal{S}^{d-1}} T^{a_{d-1}}_{s_{d-1} s_d} \prod_{i=1}^{d-1} \left(T^{a_{i-1}}_{s_{i-1} s_i} ds_i \right) , \end{array}$$

where we used the integral instead of a discrete summation over $\mathcal{S}$ for the sake of generality. The distribution-wise optimal action maximizes the expected return given the state *distribution* of the node. To better explain the differences between state-wise and distribution-wise optimality, we introduce more notions. Following Bellemare, Dabney, and Munos (2017), given a policy π, a distributional Bellman equation can be expressed in terms of three sources of randomness that are:

- $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ the stochastic reward function;
- $P^\pi : \mathcal{F}(\mathcal{S}, \mathcal{A}) \rightarrow \mathcal{F}(\mathcal{S}, \mathcal{A})$ the transition operator associating to each element $f \in \mathcal{F}(\mathcal{S}, \mathcal{A})$

$$\begin{array}{rccl} P^\pi f : & \mathcal{S} \times \mathcal{A} & \rightarrow & \mathbb{R} \\ & (s, a) & \mapsto & f(S', A') \end{array}$$

$$\text{with} \begin{cases} S' & \sim T^a_{s \cdot} \\ A' & \sim \pi(\cdot | S') \end{cases}$$

- and Z the random return associated to a state-action pair and defined by

$$\begin{array}{rccl} Z : & \mathcal{S} \times \mathcal{A} & \rightarrow & \mathbb{R} \\ & (s, a) & \mapsto & R^a_s + \gamma P^\pi Z(s, a) \, . \end{array}$$

Additionally to those three sources of randomness, we introduced S_d, the random variable of the state resulting from the application of the d first recommended actions. Mathematically, we have the following distributional Bellman equations, classically defining the Q value function for any policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi,T,R}\left(Z(s,a)\right) .$$

We now define the Q-value function with respect to the state distribution P_{S_d} by

$$Q^{\pi}_{d}(a) \triangleq \mathbb{E}_{s\sim P_{S_d}}\left(Q^{\pi}(s,a)\right) .$$

where Q^{π} is the expected return given the three aforementioned sources of randomness and Q^{π}_{d} includes the fourth source of randomness caused by the introduction of S_d. In terms of optimality, we shall define the *optimal Q-value function with respect to* P_{S_d} *as*

$$Q^{*}_{d}(a) \triangleq \max_{\pi} Q^{\pi}_{d}(a) .$$

This allows to introduce $a^{*}_{d} \triangleq \operatorname{argmax}_{a\in A} Q^{*}_{d}(a)$ the distribution-wise optimal action at the root node of Γ_d. Overall, we made the distinction between Q^{*} and Q^{*}_{d}. The first function represents the optimal expected return given the interaction of R and T. The second function represents the optimal expected return given the interaction of R, T and the fact that the initial state is distributed along P_{S_d}. In turn, those two functions allowed us to distinguish between the state-wise optimal action a^{*} and the distribution-wise optimal action a^{*}_{d}.

Unfortunately, at the root node of Γ_d for $d > 0$, open-loop tree search algorithms do not estimate Q^{*} but Q^{*}_{d}. The bias introduced by the state distribution implies that, in the general case, we have no guarantee that $a^{*} = a^{*}_{d}$. The risk is that the set Ω_{S_d} of possible realizations of S_d may include states where a^{*} is sub-optimal, in which case the resulting return evaluations would weight in favor of a different action than a^{*}. In other words — by introducing the notion of domination domain for an action $a \in \mathcal{A}$ as $\mathcal{D}_a \triangleq \{s \in \mathcal{S} | \pi^{*}(s) = a\} \subset \mathcal{S}$ — if Ω_{S_d} is not included in $\mathcal{D}_{a^{*}}$, then the risk of the recommended action to be state-wise sub-optimal is increased. Conversely, if Ω_{S_d} is included in the domination domain of a^{*}, then the optimal action will be selected given that the budget is "large enough" with respect to the chosen tree search algorithm's performance. This fact comes from the asymptotic convergence of OLTA implemented with the OLUCT algorithm, studied later in Section 3.5. Consequently, one should base the decision criterion on the analysis of P_{S_d} and the action domination domains. For instance, Rachelson and Lagoudakis (2010) use the properties of Lipschitz-MDPs to compute these domains. Although the following discussion is inspired by this work, the consideration of Lipschitz-MDPs is out of the scope of this chapter. Instead, we describe below three intuitive axis of analysis onto which an empirical decision criterion can be implemented. Those criteria will be illustrated on real experiments in Section 3.6.

Current state and state distribution analysis. The current state s can be compared to the empirical state distribution $\hat{P}_{S_d}$ at the root node of the sub-tree, where $\hat{P}_{S_d}$ is the collection of sampled states at this root node. If the estimated probability $\hat{P}_{S_d}(s)$ of sampling s is large, then the return estimators are related to the locality of the state space the agent lies in. If not, then the distribution-wise optimal action may not be state-wise optimal. This consideration supposes to identify a state-metric for which two close states have a high chance to be in the same action domination domain. Alternatively, in the case of a POMDP, a belief distribution on the current state is available instead of the current state itself (Kaelbling, Littman, and Cassandra, 1998). In such a case, a direct comparison between this distribution and $\hat{P}_{S_d}$ can be performed (for instance with a Wasserstein metric).

State distribution analysis. The dispersion and multi-modality of $\hat{P}_{S_d}$ could motivate not to reuse a sub-tree. A high dispersion involves the possibility that $\hat{\Omega}_{S_d}$ does not belong to a single action domination domain and a re-planning should be triggered. The same consideration applies in terms of multi-modality. Conversely, a narrow, mono-modal, state distribution is a good hint for Ω_{S_d} to be comprised into a single action domination domain. Of course, those considerations are problem dependent and the concepts of spread *vs.* narrow or multi-modal *vs.* mono-modal should be refined accordingly.

Return distribution analysis. A widespread or a multi-modal return distribution for the recommended action in a node may indicate a strong dependency on the region of the state space we lie in. If Ω_{S_d} covers different action domination domains, each of these domains may contribute a different return distribution to the node's return estimates, thus inducing a high variance on this distribution or even a multi-modality. In this case, it is intuitively beneficial to trigger re-planning. Conversely, a narrow or mono-modal return distribution ensures the fact that, no matter the shape of $\hat{P}_{S_d}$, the return will always be the same. In other words, the optimal Q-value is not dependent on the state. Naturally, even after re-planning, widespread or multi-modal return distributions can naturally arise as a result of the MDP's reward and transition models.

We do not provide a unique generic method to base the decision criterion on. Indeed, we believe that it is a strongly problem-dependent issue and that efficient heuristics can be built accordingly. However, the analysis of the state and return distributions constitute promising indicators and we exemplify their use in the experiments of the last section. Studying theoretically the design of a decision criterion that could bring quantifiable optimality guarantees is an interesting perspective. Notice however that such a criterion might be too conservative and therefore trigger re-planning too often.

3.5 Theoretical Analysis of the Open-Loop Tree search Algorithm

In this section, we demonstrate that OLTA asymptotically provides distribution-wise optimal actions for any sub-tree Γ_d of depth d. For its performance and simplicity, we chose to implement OLUCT as the open-loop planning algorithm utilized by OLTA. However, any other algorithm generating trees as described in Section 3.1, page 34 could be used in the same way, for instance, the OLOP algorithm (Bubeck and Munos, 2010) or the HOLOP algorithm (Weinstein and Littman, 2012). We first derive an upper bound on the failure probability that converges towards zero when the initial budget B of the algorithm goes to infinity. Then, we characterize the loss of performance guarantees between subsequent depths and show a logarithmic decay of the upper bound. Intuitively, this means that deeper sub-trees provide less guarantees. The demonstration unfolds as follows: first we calculate a lower bound for the number of trials of the actions at the root of Γ_d in Lemma 3.1, page 51; then we derive an upper bound on the failure probability given a known budget at depth d in Lemma 3.2, page 51; finally we derive a recursive relation between the upper bounds of subsequent trees that leads to our result in Theorem 3.1, page 52.

We denote by $b(d) \in \mathbb{N}$ the total budget used to develop Γ_d, *i.e.*, the number of times the d first recommended actions have been selected by the tree policy during MC-simulations. This budget is known after the construction of the total tree Γ_0 by OLUCT. We denote by $T^d_{i,t}$ the number of times the i^{th} action at the root node of Γ_d has been selected by the OLUCT tree policy after $t \in \mathbb{N}$ expansions of Γ_d. Necessarily, after completion, $0 \leq T^d_{i,b(d)} \leq b(d)$. Similarly, $\hat{Z}^d_{i,T^d_{i,t}}$ denotes the estimate of the return of the i^{th} action at depth d after t expansions of the sub-tree Γ_d. This estimate is the average of the discounted return collected during MC-simulations starting from one of the states sampled at the root node of Γ_d followed by application of a^i. For notation convenience, we write

$$\hat{Z}^d_{i,t} \triangleq \hat{Z}^d_{i,T^d_{i,t}} .$$

We write I^d_t the index of the action chosen by the tree policy at depth d after t expansions of Γ_d. By definition of the tree policy, we have:

$$I^d_t = \underset{i \in \{1,\ldots,A\}}{\operatorname{argmax}} \left\{ \hat{Z}^d_{i,t-1} + c_{t-1,T^d_{i,t-1}} \right\} .$$

The index of the recommended action at depth d given a budget $b(d)$ is

$$\widehat{I}^d = \underset{i \in \{1,\ldots,A\}}{\operatorname{argmax}} \hat{Z}^d_{i,b(d)} .$$

Hence, $a^{\widehat{I}^0}, \ldots, a^{\widehat{I}^d} \in \mathcal{A}^{d+1}$ denotes the whole recommended plan. From this, deciding to perform open-loop control amounts to apply these actions subsequently, regardless of the reached states. We recall that the objective of OLTA is to allow this open plan application in a conservative way. Following Kocsis and Szepesvári (2006), we assume that the empirical estimates $\hat{Z}^d_{i,t}$ converge and write $Z^d_{i,t} = \mathbb{E}\left(\hat{Z}^d_{i,t}\right)$ and $Z^d_i = \lim_{t\to\infty} Z^d_{i,t}$. Then, we define the return difference between a sub-optimal action and the distribution-wise optimal one by

$$\Delta^d_i \triangleq Z^d_{i^*_d} - Z^d_i$$

for $i \in \{1,\ldots,A\} \setminus \{i^*_d\}$, where we write i^*_d the index of the distribution-wise optimal action at the root node of Γ_d. For simplicity, we make the assumption that only one action is optimal in a given node. The minimum return difference between a sub-optimal action and the optimal one at depth d is

$$\delta^d = \min_{i \in \{1,\ldots,A\} \setminus \{i^*_d\}} \Delta^d_i .$$

We now state our results. The first lemma provides a lower bound on the number of trials of the recommended action for any sub-tree Γ_d. This quantity being dependent on the budget used to develop the parent tree, we establish a recurrence relation on a lower bound between the budgets of subsequent sub-trees. The result guarantees that the budget of a child sub-tree will be bigger than a quantity proportional to the natural logarithm of the parent's budget.

Lemma 3.1 (Lower bound for the number of trials)**.** *For any sub-tree Γ_d developed with a budget $b(d) > A$, there exist a constant $\rho \geq 0$ such that $T^d_{i,b(d)} \geq \lceil \rho \ln(b(d)) \rceil$ for all $i \in \{1, \dots, A\}$. Furthermore, we have the following sequence of lower bounds for the budget with $\lceil \cdot \rceil$ the ceiling function:*

$$\begin{cases} b(d=0) = B \\ b(d) \geq \lceil \rho \ln\left(b(d-1)\right) \rceil \end{cases}.$$

The proof of Lemma 3.1 is reported in the Appendix, Chapter A, Section A.2. The second lemma is an upper bound on the failure probability, *i.e.*, the probability to select a distribution-wise sub-optimal action at the root node of a sub-tree. The results implies that this failure probability tends to zero as $b(d)$ tends to infinity. Lemma 3.2 shares many links with Theorem 2.7, page 26 stating the convergence of the failure probability for the UCT algorithm. Indeed, the lemma is its extension for any sub-tree of the recommended actions of Γ_0 developed with OLUCT.

Lemma 3.2 (Convergence of the failure probability at depth d conditioned on $b(d)$)**.** *For any sub-tree Γ_d developed with a budget $b(d) > A$, there exists a constant $\rho > 0$ such that we have the following upper bound on the failure probability, conditioned by the value of $b(d)$:*

$$\boldsymbol{Pr}\left(\widehat{I}^d \neq i^*_d \,\middle|\, b(d)\right) \leq 2b(d)^{-\frac{\rho}{2}(\delta^d)^2}.$$

Particularly, we have that $\boldsymbol{Pr}\left(\widehat{I}^d \neq i^*_d \,\middle|\, b(d)\right)$ tends to zero as $b(d)$ tends to infinity.

The proof of Lemma 3.2 is reported in the Appendix, Chapter A, Section A.2. This result provides that the probability of failure at the root node of Γ_d conditioned on the budget $b(d)$ tends to zero. Now we prove in Theorem 3.1, page 52 that this will also be the case conditioned on the *total budget B* used to develop the tree Γ_0. Indeed, conditioning on $b(d)$ is not enough as we need the guarantee that $b(d)$ will actually increase given that we increase B. This last result proves this, which concludes our analysis of OLTA by the fact that the probability to select the distribution-wise optimal action at any sub-tree Γ_d tends to 1 as B tends to infinity.

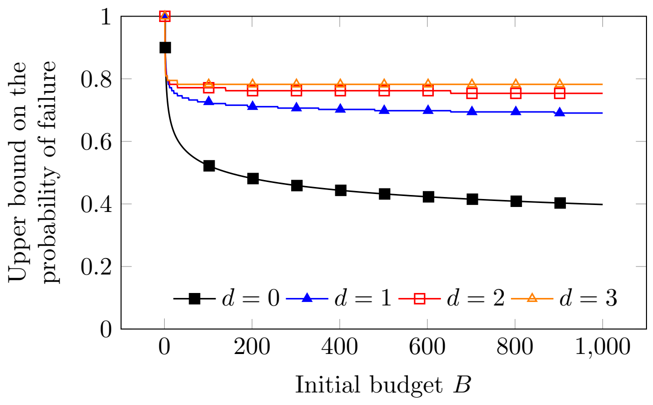


Figure 3.4: Upper bound on the probability of failure at depths $d \in \{0, 1, 2, 3\}$ for $C_p = 0.7$ and $\delta^d = 0.27$ for any depth d.

Theorem 3.1 (Convergence of the failure probability at depth d conditioned on the total budget B)**.** *For a total budget of B and for any sub-tree Γ_d developed with a budget $b(d) > A$, there exists a constant $\rho > 0$ such that we have the following recursive relation for the upper bound on the failure probability, conditioned by B:*

$$\boldsymbol{Pr}\left(\widehat{I}^d \neq i_d^* \,\middle|\, B\right) \leq 2\lceil \rho \ln \left(b(d-1)\right)\rceil^{-\frac{\rho}{2}(\delta^d)^2}.$$

Additionally, for any depth $d \geq 1$ given the total budget B:

$$\boldsymbol{Pr}\left(\widehat{I}^d \neq i_d^* \,\middle|\, B\right) \leq f^d(B)^{-\frac{\rho}{2}(\delta^d)^2},$$

with the function f defined by

$$f: \begin{array}{ccc} \mathbb{R} & \rightarrow & \mathbb{R} \\ x & \mapsto & 2\lceil \rho \ln(x) \rceil \end{array},$$

and the function composition defined by $f^1 = f$ and $\forall d > 1$, $f^d = f \circ f^{d-1}$.

The proof of Theorem 3.1 is reported in the Appendix, Chapter A, Section A.2. As mentioned earlier, Theorem 3.1 provides that the failure probability converges towards zero as the total budget B increases. The speed at which this failure probability decreases is driven by the provided upper bound. The result shows a logarithmic growth between the upper bounds on the failure probability of two subsequent trees. An illustration of the upper bound as a function of the total budget can be found in Figure 3.4 for several different depths. This highlights the fact that the deeper the sub-tree is, the less reliable is the recommended action at the root node, which motivates the introduction of the decision criteria in Section 3.4.2, page 46. We should note that these upper bounds are derived without making further hypotheses on the MDP and express a worst case value. Practically, depending on the problem, subsequent sub-trees could be highly relevant with respect to the current state. We show in the next section that equal performances to OLUCT can be reached with a smaller computational budget and number of calls to the generative model.

3.6 Empirical Analysis of the Open-Loop Tree search Algorithm

We compared OLTA with OLUCT on a discrete 1D track environment[1] and a continuous version of the Physical Traveling Salesman Problem[2] (PTSP) (Perez, Rohlfshagen, and Lucas, 2012b). We implemented five decision criteria, leading to five variations of OLTA. We first define those variants and then we describe the environment and discuss the obtained results.

3.6.1 Heuristic decision criteria

A relevant decision criterion with respect to the treated problem allows OLUCT to discard a sub-tree when its first recommended action may not be state-wise optimal given the current state. Such a practice leads to re-planning. The key idea developed in OLTA is to trigger this re-planning as rarely as possible while maintaining a conservative behavior. We implemented five different tests to base the decision on, and evaluated them independently, which led to the following variations of OLTA.

Plain OLUCT. The simplest decision criterion that discards a sub-tree only if its root-node is not fully expanded.

State Distribution Modality (SDM-OLTA). We test whether the empirical state distribution is multi-modal or not. We implemented this test for countable state spaces only, which allows to maintain a count vector in $\mathbb{N}^S$ containing the number of times each state has been sampled in each node. If there are several modes, *i.e.*, several values of the count vector are non-zero, the tree is discarded only if the current state does not belong to a majority mode. We define a majority mode by a mode comprising more than τ_{SDM} % of the sampled states, τ_{SDM} being a chosen parameter. The extension of this decision criterion to uncountable state spaces, for instance $\mathbb{R}^n$ with $n \in \mathbb{N}$, requires a notion of metric between states and a notion of mode dispersion which is problem-dependent.

State Distribution Variance (SDV-OLTA). We test whether the empirical state distribution variance is above a certain threshold τ_{SDV}. The tree is discarded if it is the case. For multi-dimensional state spaces such as in the Physical Traveling Salesman Problem (PTSP) problem, the variance-to-mean ratio is considered for the different orders of magnitude to be comparable.

State Distance to State Distribution (SDSD-OLUCT). We compute the Mahalanobis distance (De Maesschalck, Jouan-Rimbaud, and Massart, 2000) of the current state from the empirical state distribution. Intuitively, it reflects how far is the current state from the states of the empirical distribution with respect to a certain metric. The tree is discarded if the distance is above a selected threshold τ_{SDSD}.

Code available at:
[1]https://github.com/SuReLI/1dtrack.git
[2]https://github.com/SuReLI/flatland.git

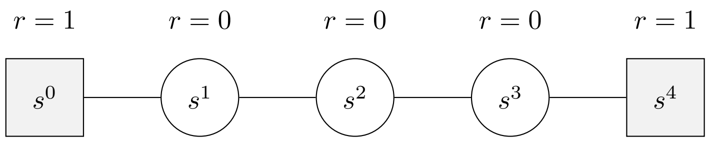


Figure 3.5: Illustration of the 1D track environment. On top of each cell, representing a state, is the immediate reward of the transition to this state.

Return Distribution Variance (RDV-OLUCT). Similarly as the state distribution variance, we test whether the empirical return distribution variance is above a certain threshold τ_{RDV}. The tree is discarded if it is the case.

A more selective decision criterion can easily be derived by combining the previously described decision criteria and discarding the tree if one of them recommends to do so. This would result in a more conservative re-planning strategy.

3.6.2 1D Track Environment

The 1D track environment, illustrated in Figure 3.5, is a 1D discrete environment where an agent can either go to the right cell or the left cell. Hence, the action space is $\mathcal{A} = \{\text{Right}, \text{Left}\}$. The initial state is the cell in the middle of the grid, *i.e.*, following the notation of Figure 3.5, $s_0 = s^2$. The reward is 0 everywhere except for the transition to one of the two terminal states s^0 and s^4 for which it is equal to 1. We introduce a transition misstep probability $q \in [0, 1]$ which is the probability to end up in the opposite state after taking an action. Formally, the transition function is defined as

$$\begin{cases} T^{\text{right}}_{s^i s^{i-1}} & = q \\ T^{\text{right}}_{s^i s^{i+1}} & = 1-q \end{cases}, \forall i \in \{1, 2, 3\}\,.$$

The same applies for the left action. For the states s^0 and s^4, the transition function is equivalent to a transition to an absorbing extra state — meaning that any transition from this extra state would yield the same state — with zero reward. If $q < 0.5$, the optimal policy π_{optimal} is to go left at s^1, to act randomly at s^2 and to go right at s^3.

The simulation settings are the following: the tree development budget is $B = 20$; the default policy is $\pi_{\text{default}} = \pi_{\text{optimal}}$; the simulation horizon for π_{default} during the MC-simulations is $H = 10$; the exploration term defined in Equation 3.3, page 44 is set to $C_p = 0.7$; we test eleven misstep probabilities $q \in \{0.0, 0.05, \ldots, 0.5\}$; finally the discount factor is $\gamma = 0.9$. The different decision criteria parameters were selected empirically and set as follows: $\tau_{\mathrm{SDM}} = 80$, $\tau_{\mathrm{SDV}} = 0.4$, $\tau_{\mathrm{SDSD}} = 1$, and $\tau_{\mathrm{RDV}} = 0.9$. We generated 1000 episodes for each value of q and recorded 3 performance measures:

1. the *performance loss*, which is equal to the number of time steps to termination;
2. the *computational cost*, which is equal to the time needed for the computer used in

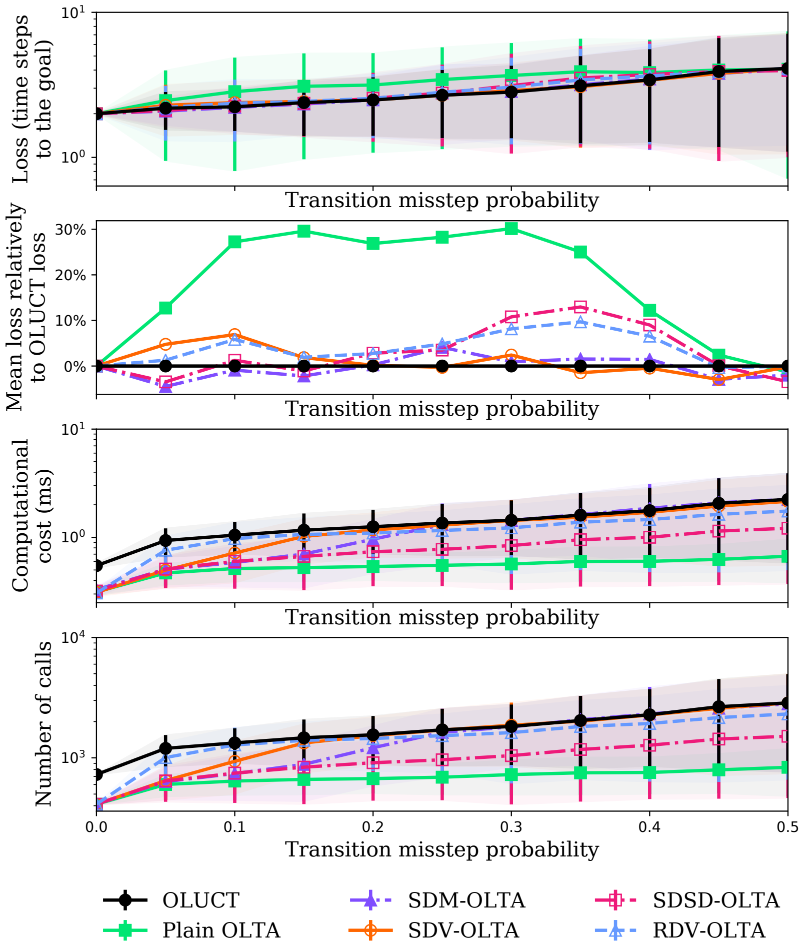


Figure 3.6: Comparison between OLTA and OLUCT on the discrete 1D track environment for varying values of the misstep probability q.

experiments to run each algorithm;

3. and the *number of calls to the generative model.*

The results are presented in Figure 3.6. We display two different graphs of the loss. The first one is the raw mean values displayed with 90% standard deviation. The second one highlights the relative performance between OLTA and OLUCT where only the mean is displayed.

The motivation behind the use of the 1D track environment is to test open-loop control in a highly stochastic environment where feedback of the current state is highly informative about the optimal action. First, notice that the parameters are tuned so that the OLTA algorithm can easily find the optimal action and that the derived plan at the root node of Γ_0 is optimal. In case of misstep for the first action, OLTA has to guess that a re-planning should be triggered while OLUCT re-plans systematically. As seen on Figure 3.6, the non-plain OLTA

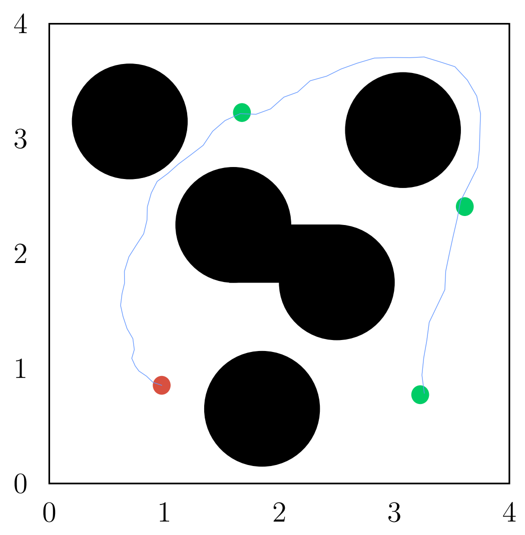


Figure 3.7: Illustration of the continuous Physical Traveling Salesman Problem. A trajectory derived by an OLUCT algorithm is displayed in green. The starting point is displayed in red, the waypoints in green and the walls in black.

and OLUCT achieved a very comparable loss. Plain-OLUCT had a weaker performance due to its systematic reuse of the sub-trees. Notice that some variations of OLTA such as SDV-OLTA achieved a better mean loss than OLUCT for some values of q. Due to the high variance, this observation cannot lead to the conclusion that OLTA can outperform OLUCT. However, this emphasizes the fact that the performance are very similar. In terms of both computational cost and number of calls to the generative model, OLTA widely outperforms OLUCT. As q increases, this computational gain vanishes and catches up with OLUCT for SDM-OLTA and SDV-OLTA. This accounts for the discriminating power of their decision criteria that discard more trees. RDV-OLTA and SDSD-OLTA kept a lower computational cost while reasonably matching the performance of OLUCT. Obviously, the computational cost of Plain-OLTA stays low. The apparent similarity between the number of calls to the generative model and the computational cost proves that computing our decision criteria is less expensive than re-planning. This result demonstrates the benefit of the approach since calls to the generative model induce most of the computational cost of tree search algorithms.

3.6.3 Physical Traveling Salesman Problem

The PTSP, illustrated in Figure 3.7, is a continuous navigation problem in which an agent must reach all the waypoints within a maze. The state is $s = (x, y, \theta, v) \in \mathbb{R}^4$ where (x, y) is the 2D position of the agent, θ its orientation and v its velocity. The action space is $\mathcal{A} = \{+d\theta, 0, -d\theta\}$, the actions respectively consist in increasing, conserving or decreasing the orientation. The reward is $+1$ when a waypoint is reached for the first time; -1 when a wall is attained, which corresponds to a crash; and 0 otherwise. The setting differs from

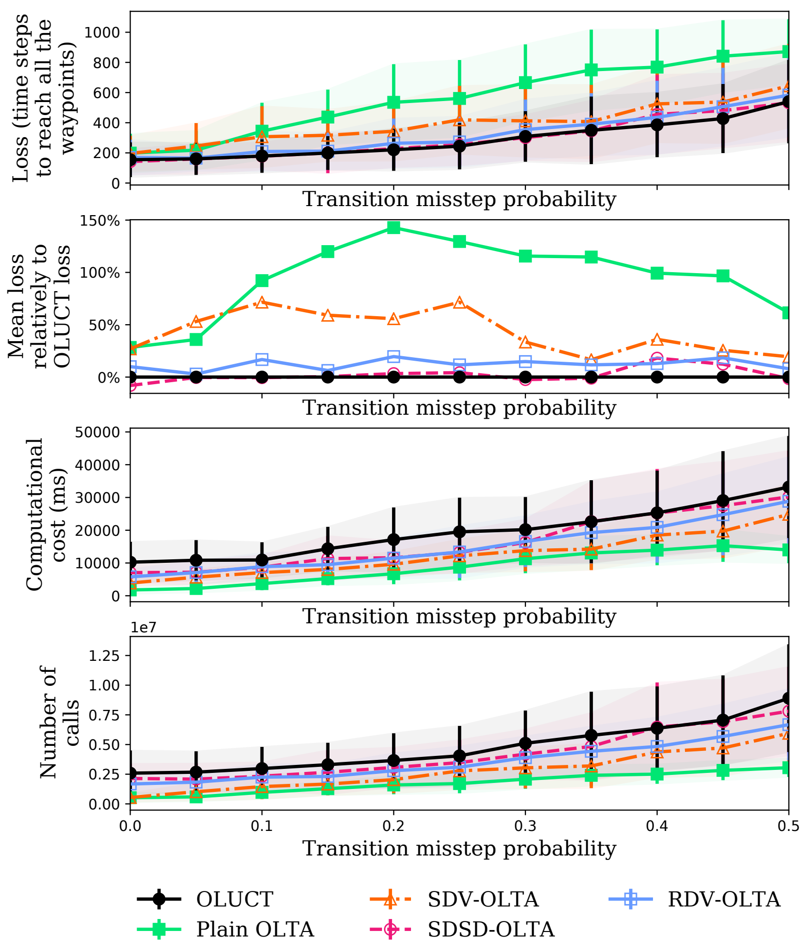


Figure 3.8: Comparison between OLTA and OLUCT on the continuous PTSP for varying values of the misstep probability q.

the hypothesis made in Section 2.1.3, page 13 because the reward function of the PTSP can be negative. However, a reward signal can still be scaled into $[0, R_{\max}]$ and negative values have no consequences on the theoretical results. The episodes of the PTSP terminate when the agent reaches all the waypoints or the horizon H. The walls cannot be crossed and the orientation is flipped when a crash occurs. Similarly, as in the 1D track environment, we introduce a misstep probability $q \in [0, 1]$ which is the probability for another action to be undertaken instead of the current one. A Gaussian noise of standard deviation σ_{noise} is added to each component of the resulting state from a transition.

The simulation settings are the following: the initial state is $s_0 = (1.1, 1.1, 0, 0.1)$; the standard deviation of the transition Gaussian noise is $\sigma_{\text{noise}} = 0.02$; the tree development budget is $B = 300$; the default policy is $\pi_{\text{default}} = \pi_{\text{go-straight}}$ which maintains the orientation constant; the simulation horizon for π_{default} during the MC-simulations is $H = 50$; the exploration term defined in Equation 3.3, page 44 is set to $C_p = 0.7$; we test the same eleven misstep probabilities $q \in \{0.0, 0.05, \ldots, 0.5\}$ as in the 1D track environment; finally, the discount factor is $\gamma = 0.99$. The map of the PTSP used in the experiments is the one depicted in Figure 3.7, page 56 with three waypoints. The different decision criteria parameters were tuned to: $\tau_{\text{SDV}} = 0.02$; $\tau_{\text{SDSD}} = 1$; $\tau_{\text{RDV}} = 0.1$. We reserve the development of SDM-OLTA in the continuous case for future work. We generated 100 episodes for each transition misstep probability and recorded the same performance measures as in the 1D track case.

The results are presented in Figure 3.8, page 57. OLUCT, SDSD-OLTA and RDV-OLTA achieved a comparable loss for all the values of q, which shows that our method is applicable to larger problems than the 1D track environment. It shows that open-loop control with a careful analysis of the reached states can yield performance equal to closed-loop control. SDV-OLTA reached a lower level of performance. Plain OLTA still achieved the highest loss since it is highly sensitive to the stochasticity of the environment. It shows that open-loop control without a careful analysis of the reached states can yield poor performance. In terms of both computational cost and number of calls to the generative model, the same trade-off between performance and computational cost is observed. Plain OLTA and SDV-OLTA considerably lowered the number of calls at the cost of the performance while SDSD-OLTA and RDV-OLTA reached a better compromise. The number of calls to the generative model and the computational cost are quite similar, meaning that — even with the higher dimensionality of the PTSP compared to the 1D track — the cost incurred by the computation of the decision criteria is negligible in comparison to the one incurred by the re-planning procedure. Again, reducing the cost of calls to the generative model being one of the main concerns when dealing with tree search algorithms, this demonstrates the benefits of the OLTA approach. Notice that SDV-OLTA achieved a good cost-performance trade-off in the 1D track environment while not in the PTSP relatively to the other algorithms. This is explained by the sensitivity of the decision criteria to parameter tuning and by the problem-dependent relevance of such a criterion. Indeed, the use of such a heuristic rather than a systematic criterion implies to empirically set the parameters for each problem. For the sake of completeness, we also generated experiments on the continuous 1D track and the discrete PTSP. The results are available in the Appendix, Chapter C. We chose to only illustrate the discrete 1D track and the continuous PTSP for the theoretical interest of the first one and the complexity of the second one.

3.7 Conclusion

We introduced OLTA, a new class of tree search algorithms performing open-loop control by re-using subsequent sub-trees of a main tree built with the OLUCT algorithm. A decision criterion based on the analysis of the current state and the current sub-tree allows the agent to efficiently determine if the latter can be exploited. Practically, OLTA can achieve the same level of performance as OLUCT, given that the decision criterion is well designed. Furthermore, the computational cost is strongly lowered by decreasing the number of calls to the generative model. This saving is the main interest of the approach and can be exploited in two ways: it decreases the energy consumption which is relevant for critical systems with low resources such as Robots, Unmanned Vehicles or Satellites; it allows a system to re-allocate the computational effort to other tasks rather than controlling the robot. We emphasize the fact that this method is generic and can be combined with any other tree search algorithm than OLUCT. Open questions include building non problem-dependent decision criteria, for instance by making more restrictive hypothesis on the considered class of MDPs, but also applying the method to other benchmarks and other open-loop planners.

The content of this chapter suggests a few perspectives. As mentioned in Section 3.4.1, page 45, the OLTA algorithm could straightforwardly be extended to the POMDP setting. Indeed, in such a case, states are non-observable which implies reasoning with belief states which are probabilistic distributions on $\mathcal{S}$. The same reasoning is applied within the non-root nodes of OLTA, giving rise to the notion of distribution-wise optimality. Therefore, studying the theoretical guarantees brought by OLTA within a POMDP is a natural extension as well as studying its empirical performance. The decision criterion introduced in Section 3.4.2, page 46 formalizes a rule to trigger re-planning or not, given the statistics of the tree. The proposed criteria were based on a qualitative interpretation of the decision problem's nature which yielded convincing empirical performances. An interesting perspective would be to study theoretically the formalization of a decision criterion. For instance, one could define an optimality measure of the decision from which a criterion can be derived. We can expect the resulting algorithm to be less effective than the proposed approaches in this chapter — as theoretical considerations may be too conservative — but it would produce a more robust algorithm.

Overall, the approach developed in this chapter is a practical method to address the planning *vs.* re-planning trade-off problem in tree search algorithms. Noticeably, a formal analysis allows to quantify the optimality of not re-planning which contrasts with the existing literature. The proposed algorithm relies on the standard assumption that the environment does not change over time. This allows to have high confidence on past predictions and to follow an open-loop plan without further refinements. In the following chapters, we change the set of hypotheses, first to "slowly" evolving MDPs in Chapter 4, then to rapid evolution allowing abrupt changes in Chapter 5.

Chapter 4

Planning in gradually evolving Markov Decision Processes

Sommaire

4.1 State of the art 62

4.1.1 Temporal evolution as a robust MDP problem 62

4.1.2 Temporal evolution as a multi-MDP model 64

4.2 Non-Stationary Markov Decision Process 67

4.2.1 Definition 67

4.2.2 Optimality criterion 68

4.2.3 Lipschitz Continuous NSMDP 69

4.3 Worst case approach 72

4.3.1 Snapshot of an NSMDP 72

4.3.2 Planning with a snapshot model 74

4.4 Risk Averse Tree Search algorithm 76

4.4.1 Presentation of the algorithm 76

4.4.2 Theoretical analysis of the RATS algorithm 81

4.4.3 Heuristic function 83

4.5 Experiments 85

4.6 Conclusion 88

This chapter is dedicated to the study of environments whose properties evolve gradually over time. This is, for instance, a problem that an autonomous car meets in practice. The traffic conditions change slowly, sometimes a road becomes increasingly busy or, conversely, less and less crowded. To model this class of tasks, we introduce in this chapter the notion of temporal evolution in our definition of an MDP, yielding the definition of Non-Stationary Markov Decision Process (NSMDP). We will use this model as a generic way of characterizing evolving MDPs. We make an additional hypothesis to describe a specific class of NSMDPs, namely gradually changing NSMDPs. The resulting model captures tasks whose properties evolve gradually over time such as the autonomous car depicted earlier. The first idea that comes to mind in such an environment is probably that non-predicted changes can lead to poor performance. This brings us to the following question:

How to conservatively behave in a gradually evolving environment?

In this chapter, we answer this question through the scope of planning agents. We assume that evolution in this setting is not chaotic. It has regularity properties and the environment cannot change completely from one decision epoch to the following one. The idea we try to develop is to exploit this regularity property to reason on the set of possible futures. In that context, we propose a planning agent, producing a conservative choice of action given the non-stationarity hypothesis, and the constraint of gradual temporal evolution.

This chapter is organized as follows. First, in Section 4.1, we review the state of the art in terms of temporal evolution in MDPs. Secondly, in Section 4.2, page 67, we describe the NSMDP setting and the regularity assumption. Thirdly, in Section 4.3, page 72, we describe the worst case approach proposed in this chapter. In Section 4.4, page 76, we design the Risk Averse Tree Search (RATS) algorithm, practically implementing the worst case approach; In Section 4.5, page 85, we illustrate experimentally the behavior of RATS in NSMDPs. Finally, we conclude in Section 4.6, page 88.

4.1 State of the art

Solving Non-Stationary Markov Decision Processes can be considered from at least two different point of views. First, it can be treated as a specific case of MDP model with uncertainty, which is the topic of the robust MDP literature. Secondly, it can be viewed as a multi-MDP model where an agent interacts sequentially with multiple environments. We review related contributions from those two points of view.

4.1.1 Temporal evolution as a robust MDP problem

Iyengar (2005) introduce the framework of robust MDPs, where the transition function is allowed to evolve within a set of possible functions due to uncertainty. Formally, the transition function associated to a state - action - decision epoch triple $(s, a, t) \in \mathcal{S} \times \mathcal{A} \times \mathcal{T}$, belongs a set of possibilities that also depends on (s, a, t). The reward function is the same, regardless of the decision epoch. The set of possibilities is constrained by a *rectangularity assumption*, which is equivalent to an independence relation between the models of different decision epochs. In that setting, Dynamic Programming algorithms are developed to find the optimal policy for the worst case evolution of the transition function, similar to a minimax approach. This differs from the approach we present in this chapter in two fundamental aspects: first we consider uncertainty in the reward model as well; secondly we use a stronger Lipschitz formulation on the set of possible transition and reward functions, this last point being motivated by its relevance to the non-stationary setting with gradual temporal evolution. The latter is an alternative to the rectangularity assumption in Robust MDPs. Further, we propose an online tree search planning algorithm, differing from DP in terms of applicability. Notice that the robust MDP setting introduced by Iyengar (2005) consider uncertainty on the underlying transition model in general. In our study, we focus on the uncertainty brought by the possibility for the model to evolve gradually over time, which is more specific. The contributions of the robust MDP literature discussed in Section 4.1.1 mostly consider uncertainty in general, without specifying a temporal evolution. Kalmár, Szepesvári, and Lőrincz (1998) introduce the framework of ϵ-stationary MDPs that can be seen as robust MDPs with a different constraint on the set of possible transition functions than the rectangularity hypothesis. More precisely, they make the assumption that, considering a reference transition function T^*, the distance between the transition functions corresponding to different decision epochs and T^* is bounded by a scalar $\epsilon > 0$. In other words, the model is non-stationary and cannot evolve "too far" from T^*. The notion of distance between transition functions in their paper is not formally discussed as the contribution of the paper is a qualitative analysis of a method to bridge the gap between MDP model and real-life. On that matter, they develop the notion of ϵ-stationary MDP to account for the uncertainty on the model and show experimentally that traditional RL approaches achieve good performances in such a setting. Szita, Takács, and Lőrincz (2002) extend the ϵ-stationary MDP setting presented by Kalmár, Szepesvári, and Lőrincz (1998) using the generalized MDP framework of Szepesvári and Littman (1996). The resulting model is called ϵ-MDP. They show in such a setting that an RL algorithm able to find the optimal policy of an MDP can find a near optimal policy in the associated ϵ-MDP. In this case, near optimal is defined by an upper bound on the distance between the estimated value function and the optimal one. Interestingly, they prove this upper bound to be proportional to ϵ. Lim, Xu, and Mannor (2013) consider learning in robust MDPs where the transition function evolves in an adversarial manner for a subset of $\mathcal{S} \times \mathcal{A}$. In that setting, they propose to learn to distinguish between state-action pairs featuring adversarial evolution and those whose temporal discrepancy of the transition function is only due to the stochasticity. The derived approach is shown to achieve similar regret to a minimax approach that would know which pairs have adversarial evolution.

Unlike the robust MDP framework, some contributions focus on the case of MDPs with non-stationary reward functions with fixed transition models. Even-Dar, Kakade, and Mansour (2009) studied such a case, allowing for any temporal evolution of the reward function, possibly adversarial. They propose an algorithm achieving sub-linear regret with respect to the best stationary policy. Dick, Gyorgy, and Szepesvari (2014) viewed a similar setting from the perspective of online linear optimization. They distinguish two cases: first, when perfect information is available to the agent, meaning that the full reward function is available at each decision epoch; secondly, when bandit information is available, meaning that only the sampled reward is available. They provide two methods to solve these problems, yielding near-optimal regret bounds in the case of perfect information. In the bandit information case, they show their method to improve on existing results on a specific class of MDP.

Finally, some contributions focus on the general case — that we consider in this chapter — of both non-stationary transition and reward functions. Csáji and Monostori (2008) generalized the model of ϵ-MDP mentioned earlier to this more general setting by introducing the (ϵ, δ)-MDP. Considering a *base MDP* $M = \{\mathcal{S}, \mathcal{A}, T, r\}$, an (ϵ, δ)-MDP is a MDP whose transition and reward functions depend on the decision epoch and are respectively ϵ-close and δ-close to T and r. In other words, an (ϵ, δ)-MDP is a time varying MDP that cannot evolve too far from the base MDP. In this setting, they study theoretically the convergence of general stochastic iterative RL algorithms. More specifically, they extend the study to the classical cases of asynchronous DP, Q-learning and temporal difference learning. Abbasi, Bartlett, Kanade, Seldin, and Szepesvári (2013) consider the same setting extended to the adversarial case, *i.e.*, where the evolution of the transition and reward functions is potentially determined by an adversary. Instead of the (ϵ, δ) constraint on the evolution of the model, they make a *mixing assumption* that roughly translates to the fact that the transition function must be sufficiently stochastic. Precisely, that the transition probability must not be concentrated on a single state as in deterministic MDPs. In this setting, they develop an algorithm achieving sublinear regret with respect to a set of comparison policies.

Despite not falling in the scope of robust RL, there exist some model-free RL methods that have been studied in the case of gradually evolving environments. We here review some of them for the sake of completeness. In the context of online learning of NSMDPs, Lecarpentier, Rapp, Melo, and Rachelson (2017) come with the proof of concept that an algorithm designed to learn stationary tasks such as the Q-Learning algorithm can be extended to this setting by disabling the convergence properties with adequate hyperparameters. They empirically show that a good performance can be achieved on the task of updraft exploitation for a gliding Unmanned Aerial Vehicle (UAV). Abdallah and Kaisers (2016) introduced Repeated Update Q-learning, a variant of the Q-Learning algorithm where updates are repeated for actions with a low probability of being sampled. Their approach is intended to mitigate the policy-dependency of the update rule and proven to converge. Additionally, Repeated Update Q-learning is showed to yield better performance than standard Q-Learning on non-stationary MDPs.

4.1.2 Temporal evolution as a multi-MDP model

We reviewed non-stationary environments seen as MDPs with uncertain transition and reward functions. Temporal evolution has also been extensively studied by assuming the existence of a collection of latent MDP models and the possibility to switch from one latent model to the other during the interaction process. Those approaches generally have the common feature that the switching occurrence between models is relatively infrequent. In other words, the evolution of the environment is assumed to be scarce. This contrasts from the point of view we take in this chapter, where we consider potential evolution between each decision epoch.

In that matter, the framework of Hidden Mode Markov Decision Process (HMMDP) introduced by Choi, Yeung, and Zhang (1999) is one of the first developed settings. An HMMDP model encompasses a collection of MDPs called *modes* sharing the same state-action space, along with a *mode transition function*. Between decision epochs, the current mode's transition function is applied to execute a state transition and select the next state. Additionally, the mode transition function is applied to select the next mode. Consequently, the current mode (or MDP, the agent is interacting with) may change between decision epochs, making the overall process non-stationary. The variable corresponding to the index of the current mode is a hidden value, making the process partially observable. Apart from this hidden value, the state of the current MDP is fully observable. In this setting, Choi, Yeung, and Zhang (1999) and Choi, Yeung, and Zhang (2000) proposed to extend the Baum-

Welch algorithm to the learning of an HMMDP modeled as a POMDP (Kaelbling, Littman, and Cassandra, 1998). They derive both a direct method assuming the knowledge that the environment *is* an HMMDP and an indirect method derived from general POMDP solving algorithms. The direct method was further refined by Choi, Zhang, and Yeung (2001) by introducing a decomposed value function taking into account the specificity of each latent MDP. Based on this, they re-implement a value iteration algorithm for HMMDPs. It should be noticed that the HMMDP model is a particular case of Mixed Observability Markov Decision Process (MOMDP) (Ong, Png, Hsu, and Lee, 2009; Araya-López, Thomas, Buffet, and Charpillet, 2010; Chanel, 2013), which are POMDPs featuring a partly hidden state space. Precisely, a subset of the state variables is fully observable while the remaining variables are hidden, which is the case of the value of the current mode in HMMDPs.

Wiering (2001) studied a specific case of non-stationary environment where some objects may move over time, altering the transition function in a specific way. In this setting, they consider a planning agent and propose to track the movement of those objects. If changes occur, a re-planning is triggered. No assumption is made on the evolution rate of the environment, only on the knowledge of the position of the moving objects.

Doya, Samejima, Katagiri, and Kawato (2002) tackle a non-stationary problem similar to the HMMDP framework. They propose to decompose a task in multiple domains in space and time. For each domain, they postulate that a simple (learned) controller can achieve a reasonable performance. A *responsibility signal* is introduced to determine with which domain is the agent interacting and therefore which controller should be learned based on those data. Along with the modeling of a scarce model switch occurrence, their approach shares the fact with the HMMDP literature that the current task domain is hidden.

Jaulmes, Pineau, and Precup (2005) tackle the case of non-stationary POMDPs with an algorithm that represents the model uncertainty with a parameterized Dirichlet distribution. During execution, a number of models are sampled according to this distribution, their optimal policies are computed and one of them is followed with a probability corresponding to the model's weight in the Dirichlet distribution. The Dirichlet parameters are learned with the data from the current trajectory of the agent. To cope with non-stationarity, past experience is weighted less than recent experience in the learning process.

Da Silva, Basso, Bazzan, and Engel (2006) propose a context detection RL algorithm. They consider a similar setting to HMMDP where a collection of latent MDP models exists and the information of the current interaction MDP is hidden. Like the RMAX algorithm (Chapter 2, Section 2.3.2, page 28), they learn an empirical model of each latent MDP. To deduce which latent MDP is the current one, they compare the ability to predict the environment of each learned model with a *quality signal.* This signal reflects both the quality of transition and reward predictions, in the same way as the responsibility signal introduced by Doya, Samejima, Katagiri, and Kawato (2002).

Hadoux, Beynier, and Weng (2014b) extend the HMMDP framework to the Hidden-Semi-Markov-Mode Markov Decision Process (HS3MDP) model. The extension consists in the introduction of a new hidden random variable corresponding to the duration of the interaction process with each mode. The estimation of this duration can in turn be used to increase the quality of the actions performed by a planning agent. In that setting, they adapt the Partially Observable Monte Carlo Planning (POMCP) algorithm (Silver and Veness, 2010) designed to solve large POMDP instances to the HS3MDP framework. The adaptation is performed by exploiting the structure of HS3MDPs during tree search and by replacing the particle filters used in the POMCP algorithm by an exact representation of beliefs to solve larger problem instances. An application of this framework to argumentative debates can be found in Hadoux (2015).

Hadoux, Beynier, and Weng (2014a) build on the approach developed by Da Silva, Basso, Bazzan, and Engel (2006) and propose another approach to detect environment changes. They use tools developed in statistics to detect those changes, resulting in a more principled method that, importantly, does not assume the knowledge of the number of latent MDPs.

Banerjee, Liu, and How (2017) also propose a change detection method in this context. Unlike Hadoux, Beynier, and Weng (2014a) who use the optimal policy of the currently identified MDP to direct the search, they propose to first actively explore with a sub-optimal exploration policy before exploiting the identified MDP. Despite the loss of reward caused by the application of the exploration policy, they prove their method to yield faster change detection. Noticeably, they assume knowledge of the models of the latent MDPs which reduces the applicability of the approach. This information is used to define the exploration policy that seeks to maximize the information gain with respect to the latent MDP models.

Sindhu, J., and Shalabh (2019) consider another change point detection method borrowed form Prabuchandran, Singh, Dayama, and Pandit (2019) and use it in a variant of the Q-Learning algorithm (Watkins and Dayan, 1992). The change point detection method is based on a transformation of the data by fitting a parameterized Dirichlet family of distributions. Change points are detected by testing the maximum likelihood estimates of Dirichlet parameters on a sliding window of the sampled time series. The resulting algorithm is proven to be competitive with the context detection RL algorithm developed by Da Silva, Basso, Bazzan, and Engel (2006). However, the approach is limited in applicability since the evolution pattern, *i.e.*, “which MDP comes after this one”, is assumed to be known.

Lifelong Reinforcement Learning (Silver, Yang, and Li, 2013; Brunskill and Li, 2014; Abel, Jinnai, Guo, Konidaris, and Littman, 2018) is an important class of problems featuring non-stationarity. It consists in experiencing a series of tasks drawn sequentially. At stake, an agent should be able to transfer some form of knowledge between tasks in order to accelerate learning. Lifelong Reinforcement Learning features a very similar temporal evolution mechanism to the HMMDP framework. The main difference between both settings is the fact that the information of the occurrence of a change is hidden in HMMDPs and related literature whereas it is revealed to the agent in lifelong RL. Therefore, the former focuses on the change detection mechanism and the latter focuses on the knowledge transfer mechanism. We postpone the review of the lifelong RL framework to the next chapter which is about transfer methods in lifelong RL.

4.2 Non-Stationary Markov Decision Process

In this section, we first define formally a Non-Stationary Markov Decision Process (NSMDP), extending the definition of an MDP introduced in Chapter 2, Section 2.1.1.5, page 8. Secondly, we define the corresponding optimality criterion, mirroring those introduced for an MDP in Chapter 2, Section 2.1.4, page 15. Thirdly, we introduce the Lipschitz Continuous NSMDP framework, modeling gradually evolving environments.

4.2.1 Definition

To define a Non-Stationary Markov Decision Process, we revert to the initial MDP model introduced by Puterman (2014) (Chapter 2), where the transition and reward functions depend on the decision epoch. An NSMDP is formally defined as follows.

Definition 4.1 (Non-Stationary Markov Decision Process)**.** A Non-Stationary Markov Decision Process (NSMDP) is an MDP whose transition and reward functions depend on the decision epoch. It is defined by a 5-tuple $\{\mathcal{S}, \mathcal{T}, \mathcal{A}, T, r\}$, where:

- $\mathcal{S}$ is a set of states;
- $\mathcal{T} = \{0, \dots, H\}$ is a set of decision epochs with $H \leq +\infty$;
- $\mathcal{A}$ is a set of actions;
- T is a transition function mapping state - decision epoch - action triples to the conditional probability distribution on the resulting state:

$$\begin{array}{rccl} T: & \mathcal{S} \times \mathcal{T} \times \mathcal{A} & \rightarrow & \mathcal{P}(\mathcal{S}) \\ & (s, t, a) & \mapsto & \mathbf{Pr}(\cdot \mid s, t, a)\ . \end{array}$$

- r is a reward function mapping state - decision epoch - action - state quadruples to the reward associated to the transition from state s to state s' by application of action a at decision epoch t:

$$\begin{array}{rccl} r: & \mathcal{S} \times \mathcal{T} \times \mathcal{A} \times \mathcal{S} & \rightarrow & \mathbb{R} \\ & (s, t, a, s') & \mapsto & r(s, t, a, s')\,. \end{array}$$

For convenience, we will adopt the following notations, specific to NSMDPs, in the remainder of the dissertation.

Notation 4.1. For all $(s, t, a, s') \in \mathcal{S} \times \mathcal{T} \times \mathcal{A} \times \mathcal{S}$, we write $T_t(\cdot \mid s, a) \triangleq T(s, t, a)$ the conditional probability distribution on the resulting state from the application of action a at decision epoch t and state s. Further, $T_t(s' \mid s, a) = \mathbf{Pr}(s' \mid s, t, a)$ denotes the probability to reach state s' when applying action a at decision epoch t and state s. Additionally, we denote by $r_t(s, a, s') \triangleq r(s, t, a, s')$ the scalar reward signal associated to the transition from state s to state s' by application of action a at decision epoch t.

This definition of an NSMDP can be viewed as that of a stationary MDP (Definition 2.1, page 8) whose state space has been enhanced with the decision epoch. While this addition is trivial in episodic tasks, *i.e.*, where an agent is given the opportunity to interact several times with the same MDP, it is different when the experience is unique. In non-episodic tasks, no exploration is allowed along the temporal axis since one cannot “go back in time”. This fact justifies the distinction made between state and decision epoch in Definition 4.1.
Viewing NSMDPs as MDPs including the decision epoch in the state echoes back to the definition of Semi Markov Decision Process (SMDP) (Howard, 1963; Sutton, Precup, and Singh, 1999; Puterman, 2014), where the duration of a state transition is a continuous random variable. However, in Definition 4.1, there is no idea of decision epoch “resulting from a transition”. In this dissertation, we consider discrete time decision processes with constant transition durations, which imply deterministic decision times in our definition of an NSMDP. In other words, the same amount of time elapses between any two subsequent decision epochs. Conversely, in an SMDP, the duration of an action — *i.e.*, the duration of a transition — is a random variable in the general case. Considering fixed action durations, as it is done in this Chapter, is a mild assumption since many discrete time sequential decision problems can be described with such a model.
The value of H distinguishes the finite horizon case $H \in \mathbb{N}$ from the infinite case $H = +\infty$. Similarly to the expected reward function of an MDP (Definition 2.2, page 11), we define the *non-stationary expected reward function*, representing the expected reward from applying an action in a state at a specific decision epoch.

Definition 4.2 (Non-stationary expected reward function)**.** The *non-stationary expected reward function* of an NSMDP $\{\mathcal{S}, \mathcal{T}, \mathcal{A}, T, r\}$ is defined as

$$\begin{array}{rlcl} R: & \mathcal{S} \times \mathcal{T} \times \mathcal{A} & \rightarrow & \mathbb{R} \\ & (s, t, a) & \mapsto & R_t(s, a) \triangleq \mathbb{E}_{s' \sim T_t(\cdot \mid s, a)} \left(r_t(s, a, s') \right) . \end{array}$$

Without loss of generality, we assume in this chapter that the reward function is bounded between $-R_{\max}$ and $R_{\max}$.

4.2.2 Optimality criterion

In Chapter 2, Section 2.1.4, page 15, we defined an optimal policy associated to an MDP as the one maximizing the value function uniformly on $\mathcal{S}$. We recall that a non-stationary policy is a mapping from $\mathcal{S} \times \mathcal{T}$ to $\mathcal{A}$ in the case of a deterministic policy and to $\mathcal{P}(\mathcal{A})$) in the case of a stochastic policy (See Chapter 2, Table 2.1, page 13 for a summary of the different policy classes considered in this dissertation). Similarly to the value function of a policy in an MDP introduced in Chapter 2, Definition 2.4, page 14, the value function of a policy can be defined in the case of an NSMDP.

Definition 4.3 (Non-stationary value function)**.** Consider an NSMDP $M = \{\mathcal{S}, \mathcal{T}, \mathcal{A}, T, r\}$, a discount factor $\gamma \in [0, 1)$ and a non-stationary stochastic policy π. The value of π in M is defined for all $(s, t) \in \mathcal{S} \times \mathcal{T}$ by the *non-stationary value function* $V^{\pi}_{t,M}$ as

$$V^{\pi}_{t,M}(s) \triangleq \mathbb{E}\left(\sum_{i=t}^{\infty} \gamma^{i-t} r_i\left(s_i, a_i, s_{i+1}\right) \,\middle|\, s_t = s,\, a_i \sim \pi_i\left(\cdot \mid s_i\right),\, s_{i+1} \sim T_i\left(\cdot \mid s_i, a_i\right),\ \forall i \geq t\right).$$

Notation 4.2. If there is no ambiguity on the considered NSMDP M of Definition 4.3, page 68, we omit M in the writing of the non-stationary value function and write $V^{\pi}_t \triangleq V^{\pi}_{t,M}$.

Mathematically, the non-stationary value function is not different from its stationary counterpart given that the decision epoch can be included in the state. From this point of view, the only difference is practical as the temporal axis cannot be explored at will in non-episodic tasks. In this chapter, although the generalization to the finite horizon is straightforward, we will focus on the discounted criterion and consider infinite horizon NSMDPs. The definition of the non-stationary Q-value function Q^{π}_t for any policy π within an NSMDP is straightforward and we have the following relation between the value and Q-value functions:

$$Q^{\pi}_t(s, a) = R_t\left(s, a\right) + \gamma \mathbb{E}_{s' \sim T_t(\cdot \mid s, a)}\left(V^{\pi}_{t+1}(s')\right). \tag{4.1}$$

We define an optimal policy associated to an NSMDP as a non-stationary deterministic policy maximizing the value function uniformly on $\mathcal{S} \times \mathcal{T}$.

$$\pi^*_t(s) \overset{\triangle}{\in} \operatorname*{argmax}_{\pi} V^{\pi}_t(s),\ \forall\left(s, t\right) \in \mathcal{S} \times \mathcal{T}. \tag{4.2}$$

Puterman (2014) showed that, for a stationary MDP (Definition 2.1, page 8), in the infinite horizon case with a discount factor $\gamma < 1$, there exists a Markovian deterministic *stationary* policy that is optimal. This is not the case within an NSMDPs. By viewing the decision epoch as a component of the state, one can show that an optimal policy of an NSMDP is non-stationary in the most general case. Intuitively, as the environment changes over time, the decision rule should adapt to these changes. In both cases, there exist a deterministic optimal policy.

4.2.3 Lipschitz Continuous NSMDP

Many real-world problems can be modeled as NSMDPs. For instance, the problem of path planning for a glider immersed in a non-stationary atmosphere (Chung, Lawrance, and Sukkarieh, 2015; Lecarpentier, Rapp, Melo, and Rachelson, 2017), or that of vehicle routing in dynamic traffic congestion (Van Woensel, Kerbache, Peremans, and Vandaele, 2008; Kok, Hans, and Schutten, 2012). Realistically, we consider that the expected reward and transition functions do not evolve arbitrarily fast over time. Conversely, if such an assumption was not made, any arbitrary evolution of the NSMDP would be allowed which is both unrealistic and hard to solve. Hence, we assume that changes occur gradually over time. Mathematically, we formalize this hypothesis by bounding the evolution rate of the transition and expected reward functions, using the notion of Lipschitz Continuous (LC) functions. Generally, taking advantage of Lipschitz continuity to infer bounds on the value of a function within a certain neighborhood is a widely used tool in the RL, bandit and optimization communities (Kleinberg, Slivkins, and Upfal, 2008; Rachelson and Lagoudakis, 2010; Pirotta, Restelli, and Bascetta, 2015; Pazis and Parr, 2013; Munos, 2014). Formally, Lipschitz continuity is defined as follows.

Definition 4.4 (Lipschitz Continuity)**.** Let (X, d_X) and (Y, d_Y) be two metric spaces and consider the function $f : X \rightarrow Y$. f is said to be *L-Lipschitz Continuous* (L-LC) with $L \in \mathbb{R}^+$ a positive constant if and only if

$$d_Y(f(x), f(\hat{x})) \leq L\, d_X(x, \hat{x}),\, \forall (x, \hat{x}) \in X^2.$$

L is called a *Lipschitz constant* of the function f.

Intuitively, an LC function is limited in how fast it can change on its definition set. We apply this hypothesis to the transition and reward functions of an NSMDP so that those functions are LC with respect to time. For the transition function, this leads to the consideration of a metric between probability density functions. For that purpose, we use the 1-Wasserstein distance (Villani, 2008) whose definition follows.

Definition 4.5 (1-Wasserstein distance)**.** Let (X, d_X) be a Polish metric space[1], μ, ν any probability measures on X and $\Pi(\mu, \nu)$ the set of joint distributions on $X \times X$ with marginals μ and ν. The 1-Wasserstein distance between μ and ν is defined by

$$W_1(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \int_{X \times X} d_X(x, y) d\pi(x, y).$$

The choice of the Wasserstein distance is motivated by the fact that it quantifies the distance between two distributions in a physical manner, respectful of the topology of the measured space (Dabney, Rowland, Bellemare, and Munos, 2018; Asadi, Misra, and Littman, 2018). Intuitively, we would like close transition probability functions to yield close states when sampled. To better understand this, let us consider an agent riding a rocket to the moon and three different transition probability distributions of the resulting position of this agent:

1. the first one has its support in the northern hemisphere of the moon;
2. the second one has its support in the southern hemisphere of the moon;
3. the third one has its support on the planet Jupiter.

Physically, we would like the first and second probability distributions to have a small distance compared to their distance to the third probability distributions since Jupiter is far away from the moon. This can be done by taking into account the distance between the elements of the supports of the different probability distributions, which is what the 1-Wasserstein distance

[1]A Polish space is a topological space that is separable and completely metrizable.

achieves. Formally, the 1-Wasserstein distance has two appealing properties. First, it admits distributions of disjoint supports. Comparatively, the Kullback-Leibler divergence is infinite for distributions with different supports, regardless of the magnitude of the discrepancies. Secondly, it is sensitive to the metric of the measured space. If we consider two regions of the support where two distributions differ, the Wasserstein distance is sensitive to the distance between the elements of those regions. Comparatively, the total-variation metric is the same regardless of this distance. We now introduce the notion of LC-NSMDP seen as gradually evolving NSMDPs.

Definition 4.6 ((L_T, L_r)-LC-NSMDP)**.** An (L_T, L_r)-*LC-NSMDP* is an NSMDP whose transition and reward functions are respectively L_T-LC and L_r-LC with respect to time, *i.e.*, $\forall (s, s', t, \hat{t}, a) \in \mathcal{S}^2 \times \mathcal{T}^2 \times \mathcal{A}$,

$$\begin{aligned} W_1 \left(T_t \left(\cdot \mid s, a \right), T_{\hat{t}} \left(\cdot \mid s, a \right) \right) &\leq L_T \left| t - \hat{t} \right|, \\ \left| r_t \left(s, a, s' \right) - r_{\hat{t}} \left(s, a, s' \right) \right| &\leq L_r \left| t - \hat{t} \right|. \end{aligned}$$

One should remark that the LC property should be defined with respect to actual decision times and not decision epoch indexes for the sake of realism. In the present case, both have the same value, and we choose to keep this convention for clarity. Our results however extend easily to the case where indexes and times do not coincide. In the remainder of this chapter, we consider (L_T, L_r)-LC-NSMDPs, making Lipschitz Continuity our regularity property. Intuitively, an LC-NSMDP is an NSMDP whose transition and reward functions cannot *vary too much* (with respect to the introduced metrics) between decision epochs. Notice that, along with the Lipschitz constraint we just introduced, the reward function cannot vary indefinitely as we assume it to be upper bounded by $R_{\max}$. Notice also that the expected reward function R is defined as a convex combination of r by the transition probability measure T. As a result, the notion of Lipschitz Continuity of R is strongly related to that of r and T as showed in Theorem 4.1, page 72. Before stating the result, we introduce another formulation of the 1-Wasserstein distance known as the dual formulation for the need of the mathematical proof.

Definition 4.7 (Dual formulation of the 1-Wasserstein distance)**.** Let (X, d_X) be a Polish metric space and μ, ν any two probability measures on X. The dual formulation of the 1-Wasserstein distance between μ and ν is defined by

$$W_1 \left(\mu, \nu \right) = \sup_{f \in \mathrm{Lip}_1(X)} \int_X f(x) d(\mu - \nu)(x)$$

where $\mathrm{Lip}_1 \left(X \right)$ denotes the set of the continuous mappings $X \to \mathbb{R}$ with a minimal Lipschitz constant bounded by 1, *i.e.*,

$$\mathrm{Lip}_1 \left(X \right) \triangleq \left\{ f : X \to \mathbb{R} \;\middle|\; \exists L \in \mathbb{R},\, 0 \leq L \leq 1,\, |f(x) - f(\hat{x})| \leq L d_X(x, \hat{x}),\, \forall \left(x, \hat{x} \right) \in X^2 \right\}.$$

Theorem 4.1. *Given an (L_T, L_r)-LC-NSMDP, the expected reward function R introduced in Definition 4.2, page 68 is L_R-LC with*

$$L_R = L_r + L_T.$$

The proof of Theorem 4.1 is reported in the Appendix, Chapter A, Section A.3. This result shows that the evolution rate of the expected reward function R is conditioned by the evolution rates of r and T. Practically, it allows to work with both functions interchangeably, benefiting from the same LC property under the hypothesis that r is Lipschitz Continuous. Generally, the information contained in a reward function defined on $\mathcal{S} \times \mathcal{A}$ along with the information contained in the transition function is enough to act optimally under most of the optimality criteria (Puterman, 2014). For this reason, some results of this chapter will be provided in the setting where the reward function is defined on $\mathcal{S} \times \mathcal{A}$. In such a case, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ will be considered independent from the transition function. All the results extend straightforwardly to the classical case of reward functions defined on $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ and its expected reward function.

4.3 Worst case approach

We consider the problem of planning within an NSMDP. As said earlier, the introduction of such a model compared to MDPs is motivated by the fact that one cannot explore at will along the temporal axis in non-episodic tasks. Therefore, knowing a model of the NSMDP itself would break this difficulty as it could be called as well in the future as in the past. Thus, in this section, we first introduce the notion of snapshot of an NSMDP seen as an instantaneous capture of the current model at planning time. Then, we formalize a robust planning method to the — unknown — evolution of the environment taking advantage of the knowledge of both the snapshot model and the Lipschitz continuity assumption.

4.3.1 Snapshot of an NSMDP

We consider finding an optimal policy within an LC-NSMDP under the non-episodic task hypothesis. The non-stationarity assumption discourages learning from previous experience since the data (interaction samples) may become unrelated to the current transition and reward functions over time, because of the evolution of those two functions. An alternative is to use model-based approaches, such as planning methods described in Chapter 2, Section 2.2, page 17. This class of algorithms requires access to either a true model or a generative model of the environment. However, we consider that using the true NSMDP model for this purpose is an unrealistic hypothesis. This would amount to know the exact future evolution of the environment and would break the interest of the distinction between state and time that motivated the introduction of the NSMDP model. Therefore, we assume the agent does not have access to the true NSMDP model. Instead, we introduce the notion of *snapshot* *model.* Intuitively, the snapshot associated to the decision epoch t_0 is a temporal slice of the NSMDP at t_0. This means that we consider its transition and reward functions as if they were "frozen" and could only be evaluated at t_0. One can see this as taking a photograph and performing planning within the resulting stationary MDP. Assuming a snapshot to be known is a much more realistic assumption than assuming the complete model to be known. Indeed, in many practical cases, one can easily gather information about the current transition and reward functions of the environment but hardly infer on their future evolution. For instance, in the thermal soaring problem of a glider, the current position of updrafts can be deduced with respect to the position of the clouds but their future position is hard to guess as it depends on complex mechanisms. Similarly, in the problem of vehicle routing in dynamic traffic congestion, the knowledge of the current traffic density is easy to acquire, but the way it will evolve through time is a complex process. A snapshot MDP is formally defined as follows.

Definition 4.8 (Snapshot of an NSMDP)**.** The snapshot of an NSMDP $\{\mathcal{S}, \mathcal{T}, \mathcal{A}, T, r\}$ at decision epoch t_0, denoted by MDP_{t_0}, is the stationary MDP defined by the 4-tuple $\{\mathcal{S}, \mathcal{A}, T_{t_0}, r_{t_0}\}$ where $\mathcal{S}$, $\mathcal{A}$ are shared with the NSMDP and

$$\begin{array}{rlcl} T_{t_0}: & \mathcal{S} \times \mathcal{A} & \to & \mathcal{P}(\mathcal{S}) \\ & (s, a) & \mapsto & T(s, t_0, a) \ , \\ r_{t_0}: & \mathcal{S} \times \mathcal{A} \times \mathcal{S} & \to & \mathbb{R} \\ & (s, a, s') & \mapsto & r(s, t_0, a, s') \end{array}$$

are the transition and reward functions of the NSMDP at t_0.

Notation 4.3. Following Notation 4.1, page 67, for a snapshot MDP_{t_0} of an NSMDP $\{\mathcal{S}, \mathcal{T}, \mathcal{A}, T, r\}$ with $t_0 \in \mathcal{T}$, we write, for all $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, $T_{t_0}(\cdot \mid s, a) \triangleq T_{t_0}(s, a)$ the conditional probability distribution on the resulting state from the application of action a at state s in MDP_{t_0}; and $T_{t_0}(s' \mid s, a) = \mathbf{Pr}(s' \mid s, t_0, a)$ the probability to reach state s' when applying action a at state s in MDP_{t_0}.

Similarly to the MDP and NSMDP cases, this definition induces the existence of the snapshot expected reward function R_{t_0} defined by

$$\begin{array}{rlcl} R_{t_0}: & \mathcal{S} \times \mathcal{A} & \to & \mathbb{R} \\ & (s, a) & \mapsto & \mathbb{E}_{s' \sim T_{t_0}(\cdot \mid s, a)} \left(r_{t_0}(s, a, s') \right) . \end{array}$$

Remarkably, the snapshot MDP_{t_0} is a stationary MDP and coincides with the NSMDP *only* at t_0 in the most general case. Particularly, one can generate a trajectory $\{(s_t, a_t, r_t)\}_{t \in \mathcal{T}}$ corresponding to an NSMDP using the sequence of snapshots $\{\text{MDP}_t\}_{t \in \mathcal{T}}$ as a model. Overall, the assumption of knowledge of snapshot models amounts to considering a planning agent only able to get the current stationary model of the environment.

4.3.2 Planning with a snapshot model

We consider a planning agent interacting with an NSMDP at state - decision epoch $(s_0, t_0) \in \mathcal{S} \times \mathcal{T}$, using the snapshot model MDP_{t_0}. By planning, we mean conducting a look-ahead search within the possible trajectories starting from (s_0, t_0). The search allows in turn to identify an optimal action with respect to the model by estimating the optimal Q-values of the actions at (s_0, t_0). This action is then undertaken and the agent transitions to the next state where the operation is repeated. This general procedure is the one of the MCTS algorithm. However, the consequence of planning with MDP_{t_0} is that the estimated optimal Q-values of the actions at (s_0, t_0) correspond to the optimal Q-values of the *snapshot*, not the true NSMDP model. In the general case, the true optimal Q-values at (s_0, t_0) within the NSMDP do not match these estimates because of the non-stationarity. Indeed, trajectories starting at (s_0, t_0) would require the knowledge of future snapshots $\text{MDP}_{t>t_0}$ to match real trajectories of the NSMDP, which is out of our assumptions.

In this section, we propose a risk averse alternative to this issue. The intuition we develop is that, given the *gradual* evolution rate of the environment, for a state s, seen at a future decision epoch $t > t_0$ during the search, we can predict the set into which the transition and reward functions of the future snapshot MDP_t lie. Given that set, we consider its element yielding the poorest performance to compute an action robust to the worst case evolution. Let us first define the set of admissible future snapshots and then detail the risk averse method. Formally, the Lipschitz continuity property of the transition and reward functions allows us to define such a domain in Theorem 4.2.

Theorem 4.2 (Set of admissible snapshot models)**.** *Consider an (L_T, L_r)-LC-NSMDP, $(s, t, a) \in \mathcal{S} \times \mathcal{T} \times \mathcal{A}$. The transition and expected reward functions (T_t, R_t) of the snapshot MDP_t at (s, a) belong to the set $\Delta_t(s, a)$ defined as follows:*

$$\Delta_t(s,a) \triangleq \mathcal{B}_{W_1}\left(T_{t-1}\left(\cdot \mid s,a\right), L_T\right) \times \mathcal{B}_{|\cdot|}\left(R_{t-1}(s,a), L_R\right)$$

where $L_R = L_T + L_r$ and $\mathcal{B}_d\left(c, r\right)$ denotes the ball of center c, defined with metric d and radius r.

The proof of Theorem 4.2 is reported in the Appendix, Chapter A, Section A.3. Borrowing the same notations, the same applies for the reward function. For all $(s, t, a, s') \in \mathcal{S} \times \mathcal{T} \times \mathcal{A} \times \mathcal{S}$,

$$r_t\left(s, a, s'\right) \in \mathcal{B}_{|\cdot|}\left(r_{t-1}\left(s, a, s'\right), L_r\right).$$

Recall that we choose to work with the expected reward as independent from the transition function rather than the reward function, keeping in mind that the LC property of the expected reward function can be deduced from the LC property of the reward function, as shown in Theorem 4.1, page 72. Intuitively, those results establish that the transition and reward models of subsequent decision epochs cannot differ “too much”, the evolution rate being controlled by the magnitude of the Lipschitz constants. This conforms to the gradually evolving NSMDP framework developed in this chapter.

For a future prediction at $(s, t) \in \mathcal{S} \times \mathcal{T}, t > t_0$, what model should be used for planning? Using T_{t_0}, R_{t_0} would result in planning in MDP_{t_0} which yields no theoretical guarantees. The underlying evolution of the NSMDP being unknown, a desirable feature would be to use a model leading to a policy that is *robust* to every possible evolution. To that end, we propose to use the snapshots corresponding to the worst possible evolution scenario under the constraints of Theorem 4.2. Such a practice — commonly used in the robust MDP literature mentioned in Section 4.1, page 62 — is a way to minimize the maximum regret of applying a policy in the NSMDP given any possible evolution, including adversarial. In other words, this is a way to minimize the maximum loss that could result from an unfavorable evolution of the environment. Notably, in NSMDPs presenting catastrophic terminal states corresponding to, for instance, crashes or a failures, the risk averse behavior minimizes the risk of reaching those states. Compared to using MDP_{t_0}, this boils down to using a different value estimate for s at t than the target $V^*_{\text{MDP}_{t_0}}(s)$ which provides no robustness guarantees.

We now define formally the worst possible evolution. Consider an NSMDP $\{\mathcal{S}, \mathcal{T}, \mathcal{A}, T, R\}$ and a policy π. We put ourselves in the case of a general reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, independent from the transition function T. A *worst case NSMDP complying with the snapshot MDP_{t_0}*, $t_0 \in \mathcal{T}$, corresponds to a sequence of transition and reward models minimizing the expected value of applying π in any pair $(s, t) \in \mathcal{S} \times \mathcal{T}$, $t \geq t_0$, while remaining within the bounds of Theorem 4.2, page 74. Given the snapshot at t_0, we write $\bar{V}^{\pi}_{t_0,t}(s)$ the value of a policy π in a worst case NSMDP at $(s, t) \in \mathcal{S} \times \mathcal{T}$, $t \geq t_0$ defined as

$$\bar{V}^{\pi}_{t_0,t}(s) \triangleq \min_{\left\{\left(\tilde{T}_i, \tilde{R}_i\right)\right\}_{i \in \{t_0, t_0+1, \dots\}}} \mathbb{E}\left(\sum_{i=t}^{\infty} \gamma^{i-t} \tilde{R}_i\left(s_i, a_i\right) \,\middle|\, \begin{array}{l} s_t = s,\; a_i \sim \pi_i(\cdot \mid s_i) \\ s_{i+1} \sim \tilde{T}_i\left(\cdot \mid s_i, a_i\right) \end{array}\right) \tag{4.3}$$
$$\text{such that} \begin{cases} \left(\tilde{T}_{t_0}, \tilde{R}_{t_0}\right) = \left(T_0, R_0\right) \\ \left(\tilde{T}_i\left(\cdot \mid s, a\right), \tilde{R}_i\left(s, a\right)\right) \in \Delta_i(s, a),\; \forall\left(s, i, a\right) \in \mathcal{S} \times \{t_0, t_0+1, \dots\} \times \mathcal{A} \end{cases}$$

Remark 4.1. It should be noticed that there may exist no valid Bellman equation (Bellman, 1957) for Definition 4.3. In fact, Iyengar (2005) proved in the general case that there exists such an equation if we assume the snapshots of different decision epochs to be independent from each other. This assumption is called the rectangularity hypothesis. However, in our case, the snapshots *are* interdependent, as detailed in Theorem 4.2, page 74. The motivation behind the introduction of Definition 4.3 is essentially the physical sense of the worst achievable value of a policy, rather than the theoretical existence of a Bellman equation.

Intuitively, the worst case NSMDP is a model of a non-stationary environment leading to the poorest possible performance for π, while being an admissible evolution of MDP_{t_0}. It reflects a worst case scenario and magnifies the possible pitfalls of the environment, for instance π leading to catastrophic terminal states. Additionally, we define $\bar{Q}^{\pi}_{t_0,t}(s,a)$ as the worst case Q-value for the pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ at decision epoch $t \in \mathcal{T}$, $t \geq t_0$:

$$\bar{Q}^{\pi}_{t_0,t}(s,a) \triangleq \begin{cases} R_{t_0}(s,a) + \gamma \mathbb{E}_{s' \sim T_{t_0}(\cdot \mid s,a)} \left(\bar{V}^{\pi}_{t_0,t+1}(s') \right) \text{ if } t = t_0, \\ \min\limits_{(\tilde{T},\tilde{R})} \tilde{R}(s,a) + \gamma \mathbb{E}_{s' \sim \tilde{T}(\cdot \mid s,a)} \left(\bar{V}^{\pi}_{t_0,t+1}(s') \right) \text{ else,} \\ \text{such that } \left(\tilde{T}, \tilde{R} \right) \in \Delta_t(s,a), \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \end{cases} \tag{4.4}$$

Similarly as the way an NSMDP can be described with a sequence of snapshot models, the worst case NSMDP can be described with a sequence of *worst case snapshot* models. A pair $(\overline{p}_t, \overline{R}_t)$ that participates in the minimization of Equation 4.4 defines a worst case snapshot at decision epoch t. Such a snapshot is recursively defined as, for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ as

$$\left(\bar{T}_t, \bar{R}_t \right) \overset{\triangle}{\in} \underset{(\tilde{T},\tilde{R}) \in \Delta_t(s,a), \forall (s,a) \in \mathcal{S} \times \mathcal{A}}{\text{argmin}} \mathbb{E}_{s' \sim \tilde{T}(\cdot \mid s,a)} \left(\tilde{R}(s,a) + \gamma \bar{V}^{\pi}_{t_0,t+1}(s') \right)$$

Identifying the worst case snapshots would allow the planning agent to derive a cautious, minimax behavior that provides a worst case performance guarantee given only MDP_{t_0}.

4.4 Risk Averse Tree Search algorithm

In this section, we present a planning algorithm, practically implementing the risk averse approach presented in Section 4.3.2, page 74. We first describe the algorithm, seen as a tree search planning method. Then, we perform a theoretical analysis of the guarantees brought by the algorithm along with its computational complexity. Finally, we develop possible improvements to the algorithm through the use of heuristic functions.

4.4.1 Presentation of the algorithm

Tree search algorithms have been described in Chapter 2, Section 2.2.3, page 20. Following (Keller and Helmert, 2013), we consider closed-loop search trees, composed of decision nodes alternating with chance nodes. We adapt their formulation to take time into account, resulting in the following definitions. A decision node at depth t, denoted by $\nu^{s,t}$, is labeled by a state - decision epoch pair (s,t). The edges leading to its children chance nodes correspond to the available actions at (s,t). A chance node, denoted by $\nu^{s,t,a}$, is labeled by a state - decision epoch - action triple (s,t,a). The edges leading to its children decision nodes correspond to the reachable state - decision epoch pairs (s',t') after performing action a at (s,t) as illustrated in Figure 4.1, page 77. Notice that in this type of tree, from the perspective of a node, the notion of *depth* is equivalent to the notion of decision epoch.

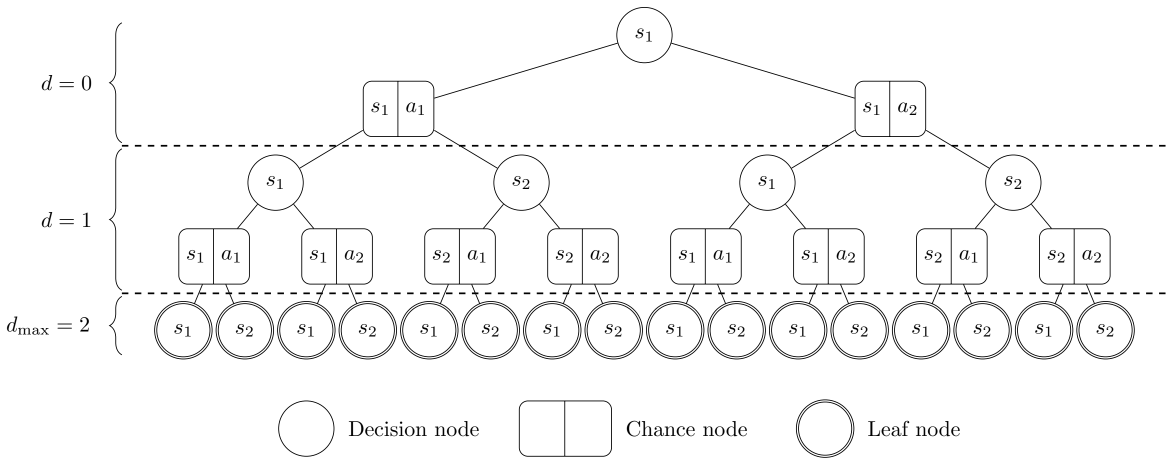


Figure 4.1: Tree structure illustration for a maximum depth of $d_{\max} = 2$ with state space $\mathcal{S} = \{s_1, s_2\}$ and action space $\mathcal{A} = \{a_1, a_2\}$. The current state s_1 of the agent when the tree is built labels the root node. Each depth, shown on the left side, corresponds to the decision epoch associated with the nodes.

We consider the problem of estimating the optimal action a_0^* at (s_0, t_0) within a worst case NSMDP complying with the known snapshot MDP$_{t_0}$. This problem is twofold. It requires 1) to estimate the worst case NSMDP model given MDP$_{t_0}$ and 2) to explore the latter in order to identify a_0^*. We propose to tackle both problems with an algorithm inspired by the minimax algorithm (Fudenberg and Tirole, 1991) where the *max* operator corresponds to the agent's policy, seeking to maximize the return; and the *min* operator corresponds to the worst case model, seeking to minimize the return. By analogy with the minimax literature, this approach amounts to see the environment as an adversarial agent. Estimating the worst case NSMDP requires to estimate the sequence of subsequent worst case snapshots minimizing Equation 4.4. The interdependence of those snapshots between subsequent decision epochs (Equation 4.3, page 75) makes the problem hard to solve (Iyengar, 2005), particularly because of the combinatorial nature of the adversary's action space. Especially, as the latter consists in modifying the properties of the model, it is continuous and thus infinite. For instance, for $(s, t, a) \in \mathcal{S} \times \mathcal{T} \times \mathcal{A}$, there is an infinite number of scalars $R_t(s, a)$ belonging to $\mathcal{B}_{|\cdot|}\left(R_{t-1}(s, a), L_R\right)$, and choosing any of those scalars can be viewed as an action of the environment, seen as an adversary. Instead, we propose to solve a relaxation of this problem, by considering snapshots only constrained by MDP$_{t_0}$. Making this approximation leaves a possibility to invalidate Theorem 4.2, page 74 but allows for an efficient search within the developed tree and — as will be shown experimentally — leads to robust policies. Figure 4.2, page 78 illustrates the effect of considering the relaxed problem compared to the original constraints depicted in Equation 4.3, page 75. For that purpose, we define the set of admissible snapshot models at $(s, t, a) \in \mathcal{S} \times \mathcal{T} \times \mathcal{A}$ with respect to MDP$_{t_0}$ by

$$\Delta_{t_0}^{t}(s, a) \triangleq \mathcal{B}_{W_1}\left(T_{t_0}\left(\cdot \mid s, a\right), L_T\left|t - t_0\right|\right) \times \mathcal{B}_{|\cdot|}\left(R_{t_0}(s, a), L_R\left|t - t_0\right|\right) .$$

The relaxed analogues of Equations 4.3, page 75 and 4.4, page 76 for $(s, t, a) \in \mathcal{S} \times \mathcal{T} \times \mathcal{A}$

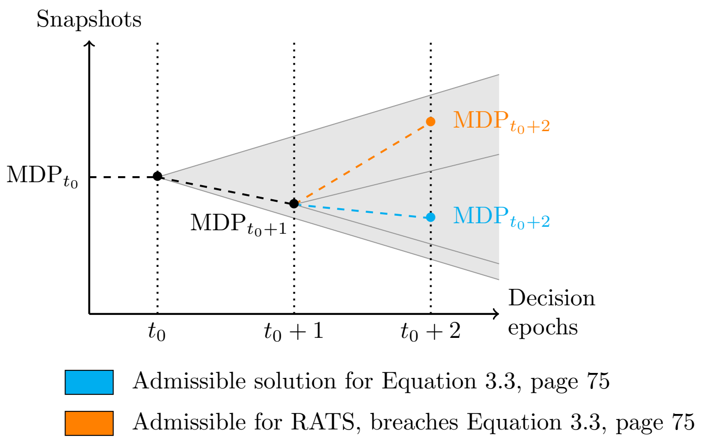


Figure 4.2: Illustration of the relaxed problem solved by the RATS algorithm. The horizontal axis represents time with the decision epochs and the vertical axis represents the space of snapshot MDPs, allowing to illustrate the Lipschitz constraint between models as if they were scalar values. Equation 4.3, page 75 constrains subsequent snapshots to respect the Lipschitz constraint while the relaxed version only constrains them with respect to the "origin" snapshot MDP_{t_0}.

are defined as follows:

$$\hat{V}^{\pi}_{t_0,t}(s) \triangleq \min_{\{(\tilde{T}_i,\tilde{R}_i)\}_{i\in\{t_0,t_0+1,\dots\}}} \mathbb{E}\left(\sum_{i=t}^{\infty}\gamma^{i-t}\tilde{R}_i\left(s_i,a_i\right) \,\middle|\, \begin{array}{l} s_t = s,\ a_i \sim \pi_i(\cdot \mid s_i) \\ s_{i+1}\sim \tilde{T}_i\left(\cdot \mid s_i,a_i\right)\end{array}\right) \tag{4.5}$$
$$\text{such that}\begin{cases}\left(\tilde{T}_{t_0},\tilde{R}_{t_0}\right) = (T_0,R_0) \\ \left(\tilde{T}_i\left(\cdot \mid s,a\right),\tilde{R}_i\left(s,a\right)\right)\in\Delta^i_{t_0}(s,a), \\ \forall\left(s,i,a\right)\in\mathcal{S}\times\{t_0,t_0+1,\dots\}\times\mathcal{A}\end{cases}$$

$$\hat{Q}^{\pi}_{t_0,t}(s,a) \triangleq \min_{(T,R)\in\Delta^t_{t_0}(s,a),\,\forall(s,a)\in\mathcal{S}\times\mathcal{A}} R(s,a)+\gamma\mathbb{E}_{s'\sim p}\left(\hat{V}^{\pi}_{t_0,t+1}(s')\right). \tag{4.6}$$

For an optimal policy with respect to the criterion so defined, we get the following value and Q-value functions:

$$\hat{V}^{*}_{t_0,t}(s) \triangleq \max_{a\in\mathcal{A}}\hat{Q}^{*}_{t_0,t}(s,a), \tag{4.7}$$

$$\hat{Q}^{*}_{t_0,t}(s,a) \triangleq \min_{(\tilde{T}_t,\tilde{R}_t)\in\Delta^t_{t_0}(s,a),\,\forall(s,a)\in\mathcal{S}\times\mathcal{A}} \tilde{R}_t\left(s,a\right)+\gamma\mathbb{E}_{s'\sim\tilde{T}_t(\cdot \mid s,a)}\left(\hat{V}^{*}_{t_0,t+1}(s')\right). \tag{4.8}$$

Remark 4.2. In Remark 4.1, we stated that there may exist no valid Bellman equations (Bellman, 1957) for the worst case value and Q-value functions. In contrast, it can be shown for the relaxed problem (Definitions 4.5, 4.6, 4.7 and 4.8) that such a Bellman equation exists. This straightforwardly stems from the results presented by Iyengar (2005), proving the case when the snapshot models are supposed to be independent from each other (rectangularity assumption). Indeed, in the definition of the relaxed problem, the rectangularity assumption is verified while it is not true in the full problem of Equations 4.3, page 75 and 4.4, page 76.

Let us now provide a method to calculate the value of state - decision epoch pairs and state - decision epoch - action triples within the nodes of the tree search algorithm.

Max nodes. A decision node $\nu^{s,t}$ corresponds to a max node due to the greediness of the agent with respect to the subsequent values of the children. We aim at maximizing the return while retaining a risk averse behavior. As a result, the value of $\nu^{s,t}$ follows Equation 4.7, page 78 and is defined as:

$$V(\nu^{s,t}) = \max_{a \in \mathcal{A}} V\left(\nu^{s,t,a}\right) . \tag{4.9}$$

Min nodes. A chance node $\nu^{s,t,a}$ corresponds to a min node due to the use of a worst case NSMDP as a model which minimizes the value of $\nu^{s,t,a}$ with respect to the reward and the subsequent values of its children. Writing the value of $\nu^{s,t,a}$ as the value of s, t, a, within the worst case snapshot minimizing Equation 4.8, page 78, and using the values of the children as values for the next reachable states, leads to Equation 4.10.

$$Q\left(\nu^{s,t,a}\right) = \min_{(T,R) \in \Delta_{t_0}^{t}} R(s,a) + \gamma \mathbb{E}_{s' \sim T}\left(V\left(\nu^{s',t+1}\right)\right) \tag{4.10}$$

Given that the values of the subsequent children estimate the optimal value $\bar{V}^*_{t_0,t+1}(s')$, this equation reflects the optimal Q-value of Equation 4.8, page 78.

Our approach considers the environment as an adversarial agent, as in an *asymmetric* two-player game, in order to search for a robust plan. Note that this game is *asymmetric*, since the agents have different action sets: the actions the environment can take are modifications to the model itself. The resulting algorithm, called Risk Averse Tree Search (RATS), is described in Algorithm 8, page 80. Given an initial state - decision epoch pair, a minimax tree is built using the snapshot MDP_{t_0} and the operators corresponding to Equations 4.9 and 4.10 to estimate the worst case snapshots at each depth. The tree is built, the action leading to the best possible value from the root node is selected and a real transition is performed. The next state is then reached, the new snapshot model MDP_{t_0+1} is acquired and the process re-starts from the beginning. Notice the use of $T(\nu' \mid \nu)$ and $R(\nu)$ in the pseudo-code: they are light notations respectively standing for the probability $T_t(s' \mid s, a)$ to transition to a decision node $\nu' \equiv \nu^{s',t+1}$ given a chance node $\nu \equiv \nu^{s,t,a}$ and $R_t(s,a)$ corresponding to a chance node $\nu \equiv \nu^{s,t,a}$. The tree built by RATS is entirely developed until the maximum depth $d_{\max}$. A heuristic function is used to evaluate the value of the states labeling the leaf nodes of the tree, this aspect of the algorithm being discussed later in Section 4.4.3, page 83.

For Algorithm 8, page 80 to be practically executable, the computation of the min operator should be detailed. We provide a closed-form expression of this minimization problem in Theorem 4.3, page 81. The resulting expression should be used to set the value of any chance node of the tree. To compute it, a formula giving the worst case snapshot model is evaluated and the latter is used to compute the value. Recall that this worst case snapshot model consists in the "optimal action" selected by the environment seen as an adversary in our risk averse approach. Notice that the information of the value of the Lipschitz constant, or at least a pessimistic estimate, should be made available to the agent for the computation

Algorithm 8 RATS

```
Set: NSMDP $\{\mathcal{S}, \mathcal{T}, \mathcal{A}, T, r\}$; initial state distribution $\mathcal{P}_0$; horizon $H$.
Input: maximum search depth $d_{\max}$; heuristic function $\mathcal{H}$.
$s \sim \mathcal{P}_0$  # Set the initial state.
for $t \in \{1, \dots, H\}$ do
  $M \leftarrow \text{MDP}_t$  # Acquire the current snapshot model.
  $\nu_0 \leftarrow$create a root node labeled with $(s, t)$
  $\text{Minimax}(\nu_0, M, d_{\max}, \mathcal{H})$  # Apply the minimax procedure described below.
  $\nu^* \leftarrow \text{argmax}_{\nu' \in \{\text{children of } \nu_0\}} \{\text{value of } \nu'\}$
  $a \leftarrow$ labeling action of $\nu^*$  # Select the action maximizing the value.
  $s' \sim T_t(\cdot \mid s, a)$  # Sample the next state.
  $s \leftarrow s'$
end for

Function Minimax:
Input: node $\nu$; snapshot $M$; maximum search depth $d_{\max}$; heuristic function $\mathcal{H}$.
if $\nu$ is a decision node then
  $s \leftarrow$ labeling state of $\nu$
  if $s$ is terminal or depth of $\nu$ is $d_{\max}$ then
    Set the value of $\nu$ to $\mathcal{H}(s, d_{\max})$  # Use the heuristic function as an estimator for the
    values of the leaf nodes.
  else
    Set the value of $\nu$ to $\max_{\nu' \in \{\text{children of } \nu\}} \text{Minimax}(\nu', M, d_{\max}, \mathcal{H})$  # Max operator.
  end if
else
  Set the value of $\nu$ to
  $\min_{(T,R) \in \Delta_{t_0}^t} R(\nu) + \gamma \sum_{\nu' \in \{\text{children of } \nu\}} T(\nu' \mid \nu) \text{Minimax}(\nu', M, d_{\max}, \mathcal{H})$  # Min operator.
end if
return Value of $\nu$
```

to be feasible. We are interested in characterizing Algorithm 8, page 80 without function approximation and therefore will consider finite $\mathcal{S} \times \mathcal{A}$ sets for the computation of the min operator and the theoretical analysis of the algorithm in Section 4.4.2, page 81. Before stating the result, we should demonstrate the convexity of the 1-Wasserstein distance in Lemma 4.1, page 81.

Lemma 4.1 (Convexity of the 1-Wasserstein distance)**.** *The 1-Wasserstein distance is convex,* i.e.*, for $\lambda \in [0, 1]$, (X, d_X) a Polish space and any three probability measures w_0, w_1, w_2 on X, the following holds:*

$$W_1(w_0, \lambda w_1 + (1 - \lambda)w_2) \leq \lambda W_1(w_0, w_1) + (1 - \lambda)W_1(w_0, w_2) \, .$$

Theorem 4.3 (Closed-form expression of the worst case snapshot of a chance node)**.** *Consider a finite state-action space $\mathcal{S} \times \mathcal{A}$. Following Algorithm 8, page 80, a solution to Equation 4.10, page 79 using the snapshot model MDP$_{t_0}$ is given by:*

$$\begin{aligned} \tilde{T}(\cdot \mid s, a) &= (1 - \lambda)T_{t_0}(\cdot \mid s, a) + \lambda T_{sat}(\cdot \mid s, a) \, , \\ \tilde{R}(s, a) &= R_{t_0}(s, a) - L_R \, |t - t_0| \, , \end{aligned}$$

with $T_{sat}(\cdot \mid s, a) = (0, \cdots, 0, 1, 0, \cdots, 0)$ with 1 at position $\operatorname{argmin}_{s'} V(\nu^{s',t+1})$ and

$$\lambda = \begin{cases} 1 \text{ if } W_1(T_{t_0}, T_{sat}) \leq L_T \, |t - t_0| \\ L_T \, |t - t_0| \, / W_1(T_{t_0}, T_{sat}) \text{ otherwise.} \end{cases}$$

The proofs of Lemma 4.1 and Theorem 4.3 are reported in the Appendix, Chapter A, Section A.3.

4.4.2 Theoretical analysis of the RATS algorithm

In this section, we perform a theoretical analysis of the RATS algorithm. We are interested in characterizing two things: first, how well is RATS approximating the value and Q-value functions of Equations 4.7, page 78 and 4.8, page 78; secondly, what computational complexity does the algorithm have.

As in vanilla minimax algorithms, Algorithm 8, page 80 bootstraps the values of the leaf nodes with a heuristic function if these leaves do not correspond to terminal states. Given such a leaf node $\nu^{s,t}$, a heuristic aims at estimating the value of the optimal policy at (s, t) within the worst case NSMDP, *i.e.*, $\hat{V}^*_{t_0,t}(s)$. Let $\mathcal{H}(s, t)$ be such a heuristic function. We call *heuristic error* in (s, t) the difference between $\mathcal{H}(s, t)$ and $\hat{V}^*_{t_0,t}(s)$. Assuming that the heuristic error is uniformly bounded, the following result provides an upper bound on the propagated error due to the choice of $\mathcal{H}$.

Theorem 4.4 (Upper bound on the propagated heuristic error within RATS)**.** *Consider an agent executing Algorithm 8, page 80 at $(s_0, t_0) \in \mathcal{S} \times \mathcal{T}$ with a heuristic function $\mathcal{H}$. We write $\mathcal{L}$ the set of all leaf nodes and we assume the state-action space $\mathcal{S} \times \mathcal{A}$ to be finite. Suppose that the heuristic error is uniformly bounded,* i.e.,

$$\exists \delta_{\mathcal{H}} > 0,\ \forall \nu^{s,t} \in \mathcal{L},\ \left|\mathcal{H}(s,t) - \hat{V}^*_{t0,t}(s)\right| \leq \delta_{\mathcal{H}} \,. \tag{4.11}$$

Then we have for every decision and chance nodes $\nu^{s,t}$ and $\nu^{s,t,a}$, at any depth $d \in \{0, \dots, d_{max}\}$:

$$\left|V(\nu^{s,t}) - \hat{V}^*_{t0,t}(s)\right| \quad\ \ \leq \gamma^{(d_{max}-d)}\delta_{\mathcal{H}} \,, \tag{4.12}$$

$$\left|Q(\nu^{s,t,a}) - \hat{Q}^*_{t0,t}(s,a)\right| \leq \gamma^{(d_{max}-d)}\delta_{\mathcal{H}} \,, \tag{4.13}$$

where d_{max} is the maximum depth of the tree.

The proof of Theorem 4.4 is reported in the Appendix, Chapter A, Section A.3. Given the parameter of the maximum depth of the tree $d_{\max}$ and the heuristic error $\delta_{\mathcal{H}}$, Theorem 4.4 quantifies the approximation error in terms of those parameters. The approximation error converges to zero given that $d_{\max}$ increases, *regardless* of the heuristic function $\mathcal{H}$. Importantly, for every chance node child $\nu^{s,t,a}$ of the root node of the tree,

$$\left|Q(\nu^{s,t,a}) - \hat{Q}^*_{t0,t}(s,a)\right| \leq \gamma^{d_{\max}}\delta_{\mathcal{H}} \,,$$

which quantifies the approximation error of the applied action in the real NSMDP after execution of the complete procedure. It should be noticed that Theorem 4.4 reports the error made by RATS for the approximation of the optimal worst case value and Q-value functions of the relaxed problem (Equations 4.7, page 78 and 4.7, page 78). In this dissertation, we do not quantify the error made for the approximation of the value and Q-value of the full problem (Equations 4.3, page 75 and 4.4, page 76). We believe such study to yield vacuous approximation bounds in the general case. Bridging the gap between both the relaxed and the full problem is however an especially interesting question, out of the scope of this chapter. We now state the computational complexity result of the RATS algorithm.

Theorem 4.5 (Computational complexity)**.** *Consider a heuristic function $\mathcal{H}$ with evaluation complexity $\mathcal{O}(1)$. The total computational complexity of Algorithm 8 executed with $\mathcal{H}$ is*

$$\mathcal{O}\left(\tau S^{1.5} A \, (SA)^{d_{max}}\right) \,,$$

with τ the total number of time steps and d_{max} the maximum depth of the tree.

The proof of Theorem 4.5 is reported in the Appendix, Chapter A, Section A.3. Notice that the result is expressed as a function of the total number of time steps — or decision epochs — denoted by τ. This is to be understood as any number of decision epochs for which the RATS tree search process would be used to select an action. For instance, in the case of Algorithm 8, page 80, the horizon H would be used in place of τ if H would be finite.

4.4.3 Heuristic function

In this section, we discuss possible heuristic functions that could be used within the RATS algorithm. Such a function is used to evaluate the leaf nodes of the tree built by the RATS algorithm, illustrated in Figure 4.1, page 77, when the labeling state of those nodes is not terminal. If the state is terminal, the true value of the nodes can be used by taking the worst case reward associated with the relaxed problem described by Equations 4.7, page 78 and 4.8, page 78. Namely, for a leaf node labeled with $(s, t) \in \mathcal{S} \times \mathcal{T}$, given the snapshot MDP_{t_0} with $t_0 \in \mathcal{T}$ corresponding to the decision epoch of the root node of the tree,

$$V(\nu^{s,t}) = \max \left\{ -R_{\max}, \max_{a \in \mathcal{A}} R_{t_0}(s, a) - L_R \left| t - t_0 \right| \right\} ,$$

where the minimum reward $-R_{\max}$ is assigned in case of overshooting. If the state labeling a leaf decision node is not terminal, then a heuristic function of the form

$$\begin{array}{rccc} \mathcal{H}: & \mathcal{S} \times \mathcal{T} & \rightarrow & \mathbb{R} \\ & (s, t) & \mapsto & \mathcal{H}(s, t) \end{array}$$

is used. Ideally, setting this heuristic to $\mathcal{H} : (s, t) \mapsto \hat{V}^*_{t_0,t}(s)$ would result in a perfect estimation of the value of the node and reduce the heuristic error to zero. However, this is an intractable problem since $\hat{V}^*_{t_0,t}$ is precisely the function RATS aims at estimating. Let us now discuss feasible alternatives.

Theorem 4.4, page 82 implies that with *any* heuristic function $\mathcal{H}$ inducing a uniform heuristic error, the propagated error at the root of the tree is guaranteed to be upper bounded by $\gamma^{d_{\max}} \delta_{\mathcal{H}}$. In particular, since the reward function is bounded by hypothesis, we have that

$$\frac{-R_{\max}}{1-\gamma} \leq \hat{V}^*_{t_0,t}(s) \leq \frac{R_{\max}}{1-\gamma}$$

Thus, selecting the zero function defined by

$$\mathcal{H} : (s, t) \mapsto 0 \,, \tag{4.14}$$

ensures the following error at the root node of the tree, labeled by $(s_0, t_0) \in \mathcal{S} \times \mathcal{T}$:

$$\left| V(\nu^{s_0,t_0}) - \hat{V}^*_{t_0,t}(s) \right| \leq \frac{\gamma^{d_{\max}} R_{\max}}{1-\gamma} \,, \tag{4.15}$$

and for any action $a \in \mathcal{A}$, the corresponding child node verifies

$$\left| Q(\nu^{s,t_0,a}) - \hat{Q}^*_{t_0,t}(s, a) \right| \leq \frac{\gamma^{d_{\max}} R_{\max}}{1-\gamma} \,. \tag{4.16}$$

Such a choice of a heuristic function has the advantage of being simple. The control of the error made at the root node provided by Theorem 4.4, page 82 allows to quantify the impact of choosing the zero function. Of course, this choice is problem-dependent and without further details, we can only provide the guarantees of Equations 4.15 and 4.16.

In order to improve the precision of the algorithm, we propose to guide the heuristic by using a function reflecting better the value of $(s,t) \in \mathcal{S} \times \mathcal{T}$ at leaf node $\nu^{s,t}$. As said earlier, the ideal function $\mathcal{H} : (s,t) \mapsto \hat{V}^*_{t_0,t}(s)$ reduces the heuristic error to zero, but is impossible to compute. Instead, we suggest to use the value of s within the snapshot MDP_t, using an evaluation policy π, *i.e.*, $\mathcal{H}(s,t) = V^\pi_{\text{MDP}_t}(s)$. This snapshot is also not available as we do not know the evolution of the underlying NSMDP. However, one can provide a range wherein this value lies, given the value of $V^\pi_{\text{MDP}_{t_0}}(s)$. Following the worst case approach presented in Section 4.3, page 72, we propose to use the minimum achievable value within this range as a heuristic function. Before stating the result, we should present a preliminary lemma on the distance between the finite horizon values of a policy at a particular state within the snapshots MDP_t and MDP_{t_0}.

Lemma 4.2. *Consider an (L_T, L_R)-LC-NSMDP with a finite state-action space $\mathcal{S} \times \mathcal{A}$. For $(s,t,t_0) \in \mathcal{S} \times \mathcal{T}^2$ and $n \in \mathbb{N}$, the finite horizon values of any policy π at s within the snapshots MDP_t and MDP_{t_0} verify:*

$$\left| V^{\pi,n}_{MDP_{t_0}}(s) - V^{\pi,n}_{MDP_t}(s) \right| \leq L_{V_n} \left| t - t_0 \right| ,$$

with $L_{V_n} = \sum_{i=0}^n \gamma^i L_R$, $n \in \mathbb{N}$ and the finite horizon value is defined as follows:

$$V^{\pi,n}_{MDP_{t_0}}(s) = \mathbb{E}\left(\sum_{i=0}^{n} \gamma^i r_{t_0}(s_i, a_i, s_{i+i}) \;\middle|\; \begin{array}{l} s_0 = s, \\ s_{i+1} \sim T_{t_0}\left(\cdot \mid s_i, a_i\right), \, i \geq 0 \\ a_i \sim \pi(\cdot), \, i \geq 0 \end{array} \right) .$$

Theorem 4.6. *Consider $s \in \mathcal{S}$, π a stationary policy, MDP_{t_0} and MDP_t two snapshot MDPs with $t, t_0 \in \mathcal{T}^2$. The following bound holds:*

$$\left| V^{\pi}_{MDP_{t_0}}(s) - V^{\pi}_{MDP_t}(s) \right| \leq \left| t - t_0 \right| \frac{L_R}{1-\gamma} ,$$

with $L_R = L_T + L_r$.

The proofs of Lemma 4.2 and Theorem 4.6 are reported in the Appendix, Chapter A, Section A.3. Since MDP_{t_0} is available, $V^\pi_{\text{MDP}_{t_0}}(s)$ can be estimated, for instance through Monte Carlo simulations such as performed by the MCTS algorithm. Let $\hat{V}^\pi_{\text{MDP}_{t_0}}(s)$ denote such an estimate. Following Theorem 4.6, we have the following:

$$V^\pi_{\text{MDP}_{t_0}}(s) - \frac{L_R}{1-\gamma} \left| t - t_0 \right| \leq V^\pi_{\text{MDP}_t}(s) \leq V^\pi_{\text{MDP}_{t_0}}(s) + \frac{L_R}{1-\gamma} \left| t - t_0 \right| .$$

Hence, a computable worst case heuristic function on $V^\pi_{\text{MDP}_t}(s)$ is given by

$$\mathcal{H} : (s,t) \mapsto \widehat{V}^\pi_{\text{MDP}_{t_0}}(s) - \frac{L_R}{1-\gamma} \left| t - t_0 \right| . \tag{4.17}$$

One should remark that the uncertainty brought by this heuristic increases with $|t - t_0|$, *i.e.*,

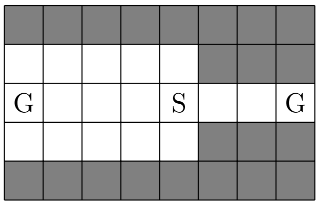


Figure 4.3: The non-stationary bridge environment

with $d_{\max}$, since t denotes the decision epoch of a leaf node. This is contrasted by Theorem 4.4, page 82 that shows that the overall accuracy of the algorithm increases with $d_{\max}$. Indeed, the bounds provided by Theorem 4.4, page 82 decrease quickly with $d_{\max}$, as the uncertainty of the heuristic function increases.

One should recall that neither the heuristic defined by Equation 4.14, page 83 nor the one defined by Equation 4.17, page 84 is better than the other. Indeed, no theoretical guarantee can be provided without additional hypotheses on the underlying NSMDP. The only result in the general case is the bound provided by Theorem 4.4, page 82 that holds for both presented heuristics.

4.5 Experiments

We compare the RATS algorithm with two other policies[2]. The first one, named DP-snapshot, uses Dynamic Programming to compute the optimal actions with respect to the snapshot models at each decision epoch. It corresponds to following a recommended action with respect to the current state of the NSMDP, regardless of its potential evolution. The second one, named DP-NSMDP, uses the real NSMDP as a model to provide its optimal action. It corresponds to an omniscient agent and should be seen as an upper bound on the performance.

We choose a particular grid-world domain coined "non-stationary bridge" illustrated in Figure 4.3. An agent starts at the "S" labeled state in the center and the goal is to reach one of the two terminal "G" labeled states where a reward of +1 is received. The gray cells represent holes that are terminal states where a reward of -1 is received. Reaching the goal on the right leads to the highest payoff since it is closest to the initial state and a discount factor $\gamma = 0.9$ is applied. The actions are $\mathcal{A} = \{\text{Right, Up, Left, Down}\}$. The transition function is stochastic and non-stationary. At decision epoch $t = 0$, any action yields the expected outcome in a deterministic way. With time, when applying the actions Left or Right, the probability to reach the positions usually stemming from either Up or Down increases symmetrically until reaching a maximum value of 0.45. We call this probability the *misstep probability.* For instance, after a *large enough* amount of decision epochs, the probability to reach the left cell while applying action Left is 0.55, the probability to reach the upper cell is 0.225 and the probability to reach the lower cell is 0.225. We set the Lipschitz constant $L_T = 1$. Aside, we

[2]The code of the experiments is available at https://github.com/SuReLI/rats-experiments – For information about the Machine Learning reproducibility checklist, see Appendix, Section D, page 159.

introduce a parameter $\epsilon \in [0, 1]$ controlling the behavior of the environment. If $\epsilon = 0$, only the left-hand side bridge becomes slippery with time. It reflects a close to worst case evolution for a policy aiming to the left-hand side goal. If $\epsilon = 1$, only the right-hand side bridge becomes slippery with time. It reflects a close to worst case evolution for a policy aiming to the right-hand side goal. In between, the misstep probability is proportionally balanced between left and right, *i.e.*, both sides become slippery. One should note that changing ϵ from 0 to 1 does not cover all the possible evolutions from MDP_{t_0} but provides a concrete, graphical illustration of RATS's behavior for various possible evolutions of the NSMDP.

We tested RATS with $d_{\max} = 6$ so that leaf nodes in the search tree all are terminal states. Hence, the optimal risk averse policy is applied and no heuristic approximation is made. Our goal is to demonstrate that planning in this worst case NSMDP allows to minimize the loss given any possible evolution of the environment. To illustrate this, we report results reflecting different evolutions of the same NSMDP using the ϵ factor. It should be noted that, at $t = 0$, RATS always moves to the left, even if it takes longer to reach the goal, since going to the right may be risky if the probabilities to go Up and Down increase. This corresponds to the careful, risk averse, behavior. Conversely, DP-snapshot always moves to the right since the first acquired snapshot MDP_0 does not capture this risk. As a result, the $\epsilon = 0$ case reflects a favorable evolution for DP-snapshot and a bad one for RATS. The opposite occurs with $\epsilon = 1$ where the cautious behavior dominates over the risky one, and the in-between cases mitigate this effect.

In Figure 4.4, page 87, we display the achieved expected return for each algorithm as a function of ϵ, *i.e.*, as a function of the possible evolutions of the NSMDP. As expected, the performance of DP-snapshot strongly depends on this evolution. It achieves high return for $\epsilon = 0$ and low return for $\epsilon = 1$. Conversely, the performance of RATS varies less across the different values of ϵ. The effect illustrated here is that RATS maximizes the minimal possible return given *any* evolution of the NSMDP. It provides the guarantee to achieve the best return in the worst case. This allows the agent to avoid catastrophic outcomes such as the performance reached by DP-snapshot in the case $\epsilon = 1$. This behavior is highly desirable when one requires robust performance guarantees as, for instance, in critical certification processes.

Figure 4.5, page 87 displays the return distributions of the three algorithms for $\epsilon \in \{0, 0.5, 1\}$. The effect seen here is the tendency for RATS to diminish the left tail of the distribution corresponding to low returns for each evolution. It corresponds to the optimized criteria, *i.e.*, robustly maximizing the worst case value. A common risk measure is the Conditional Value at Risk (CVaR) defined as the expected return in the worst $q\%$ cases. We illustrate the CVaR at 5% achieved by each algorithm in Table 4.1, page 88. Notice that RATS always maximizes the CVaR compared to both DP-snapshot and DP-NSMDP. Indeed, even if the latter uses the true model, the optimized criteria in DP is the expected return.

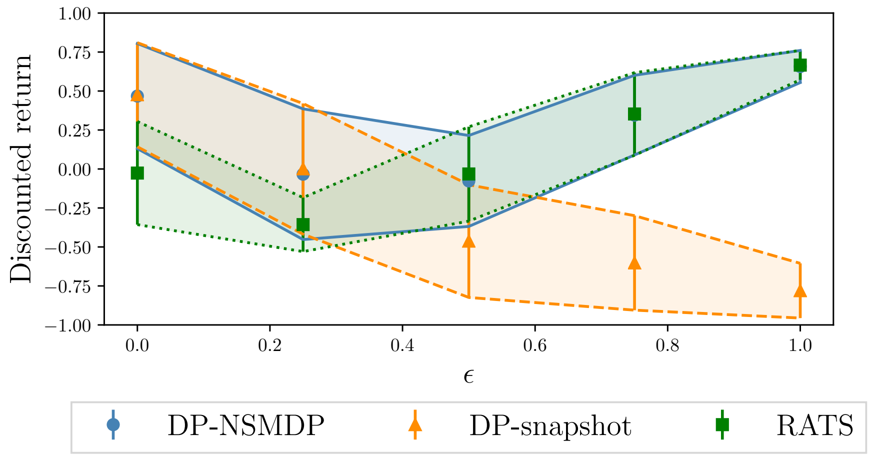


Figure 4.4: The discounted return is represented with 50% of the standard deviation for each algorithm.

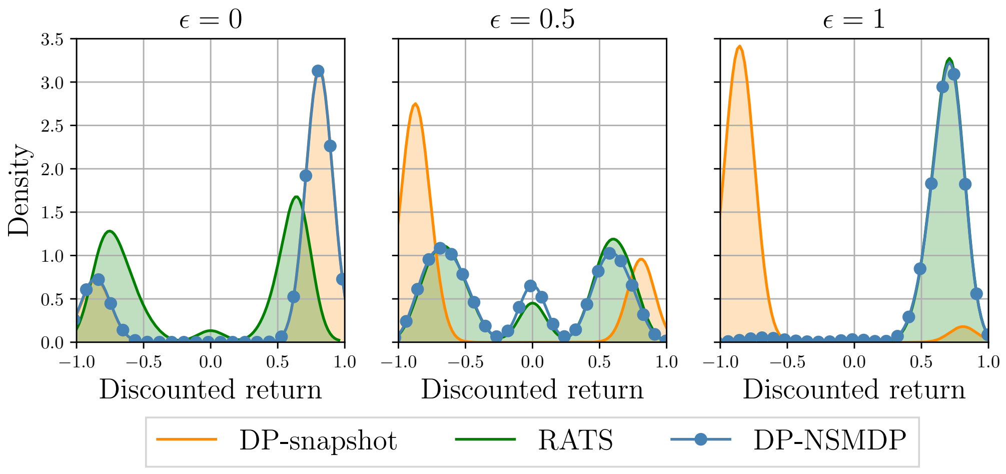


Figure 4.5: The distribution of the discounted return achieved by the three algorithms is represented for $\epsilon \in \{0, 0.5, 1\}$.

ϵ		RATS	DP-snapshot	DP-NSMDP
0	$\mathbb{E}\left(\sum r\right)$	-0.026	**0.48**	0.47
	CVaR	**-0.81**	-0.90	-0.9
0.5	$\mathbb{E}\left(\sum r\right)$	**-0.032**	-0.46	-0.077
	CVaR	**-0.81**	-0.90	**-0.81**
1	$\mathbb{E}\left(\sum r\right)$	**0.67**	-0.78	0.66
	CVaR	**0.095**	-0.90	-0.033

Table 4.1: Expected return $\mathbb{E}\left(\sum r\right)$ and CVaR at 5% for RATS, DP-snapshot and DP-NSMDP for $\epsilon \in \{0, 0.5, 1\}$.

4.6 Conclusion

We proposed an approach for robust planning in non-stationary stochastic environments. We introduced the framework of Lipschitz Continuous NSMDPs and derived the RATS algorithm, to predict the worst case evolution and to plan accordingly. We analyzed RATS theoretically and showed that it approximates a worst case NSMDP with a control parameter that is the depth of the search tree. The analysis also includes the study of the computational complexity of the algorithm. We showed empirically the benefit of the approach that searches for the highest lower bound on the worst achievable score. RATS is robust to every possible evolution of the environment, *i.e.*, maximizes the expected worst case outcome on the whole set of possible NSMDPs. Our method was applied to the uncertainty on the evolution of a model. Generally, it could be extended to any uncertainty on the model used for planning, given bounds on the set of the feasible models.

The content of this chapter suggests a few perspectives. The purpose of the proposed approach is to lay a basis of worst case analysis for robust solutions to NSMDPs. As is, RATS is computationally intensive and scaling the algorithm to larger problems is an exciting future challenge. On that matter, an interesting direction would be to adapt the planning *vs.* re-planning trade-off method presented in Chapter 3 to the RATS framework. Such a practice would result in a decrease of the computational requirements of RATS due to tree re-use. Another perspective arises from the fact that the RATS policy could be overly conservative. Indeed, we saw in experiments (not reported in this dissertation) that — as the algorithm aims at computing the optimal policy of the worst case evolution — the resulting behavior could perform poorly in environments that do not evolve in an adversarial manner. Constraining the model to stay close to a reference model as in the ϵ-stationary MDP literature (Kalmár, Szepesvári, and Lőrincz, 1998) would alleviate this effect by restricting the set of possible future models. On this line of thought, Lim, Xu, and Mannor (2013) proposed an approach to learn the portion of $\mathcal{S} \times \mathcal{A}$ featuring adversarial evolution of the model. Including such a practice in RATS would also make the algorithm less conservative. An interesting aspect of this chapter is the introduction of the relaxed min-max problem in Equations 4.7, page 78 and 4.8, page 78 to replace the true problem of Equations 4.3, page 75 and 4.4, page 76. Theoretically, it could be beneficial to study the impact of the relaxation on the computed Q-function. The question would be to quantify the difference between this Q-function and the one that is solution to the true problem. Obviously, an interesting perspective is to design an algorithm computing the solution of the true problem instead of the relaxation. As reported by Iyengar (2005), such a practice breaks an assumption they make when studying theoretically the robust MDP formulation, namely, the rectangularity assumption. The latter consists roughly in considering models corresponding to subsequent decision epochs as independent. Particularly, proving the contraction properties of the Bellman operator can be problematic without the rectangularity assumption. Therefore, studying the impact of breaking this assumption in the LC-NSMDP setting rises both an interesting theoretical question and an algorithmic challenge.

Overall, the approach developed in this chapter is a risk averse planning algorithm for gradually evolving environments. This assumption is used to constrain the set of future possible outcomes, making a worst case approach feasible. In the following chapter, we study the case of abruptly changing environments, potentially breaking any regularity assumption in the evolution.

Chapter 5

Learning in abruptly evolving Markov Decision Processes

Sommaire

5.1 State of the art **93**

5.2 Framework **95**

5.2.1 Lifelong Reinforcement Learning 95

5.2.2 Negative transfer 97

5.2.3 Proposed approach 98

5.3 Lipschitz continuity of Q-functions **100**

5.3.1 Main result 100

5.3.2 Similar Lipschitz continuity results 102

5.4 Transfer using Lipschitz continuity **105**

5.4.1 A transferable upper bound on the optimal Q-function 105

5.4.2 A computable upper bound on the optimal Q-function 107

5.5 The Lipschitz RMAX algorithm **111**

5.5.1 Definition of LRMAX 111

5.5.2 Refining the LRMAX bound with the maximum model distance 114

5.6 Experiments **117**

5.7 Conclusion **120**

In Chapter 4, we studied a method to derive a risk averse behavior in MDPs evolving gradually over time. The gradual evolution was modeled with a Lipschitz continuity assumption of the MDP model with respect to time. In this chapter, we remove this assumption and consider the general case of MDPs evolving through time in an unconstrained way. Therefore, we ask the following question:

How to learn a policy in an environment that may change abruptly over time?

Abrupt changes over time mean that, at any point, the current MDP can change arbitrarily. Consequently, we will consider the case where an agent faces a series of tasks, possibly very different from one another. We will assume that the agent is informed when a change has occurred. In the literature, this setting is often referred to as lifelong RL as it includes the idea of learning different tasks over a long period of time, similarly to what human-beings do along their lives.

In lifelong RL, a reasonable idea is that once a task has been learned, it may be possible to use the acquired knowledge to accelerate the learning of new tasks. This key question is known as *transfer* and we will refer to learned tasks as *source* tasks and to new tasks as *target* tasks. Hence, in lifelong RL, we aim at transferring knowledge from source to target tasks in order to learn them more quickly. The transferred knowledge between tasks can be of various nature. Common practices in the literature often suggest to transfer learned policies, collected samples, learned models, or even learned value functions. In this chapter, we will deal with this latter case of *value transfer*.

We elaborate on the intuitive idea that *similar* tasks should allow a large amount of transfer. An agent able to compute online a similarity measure between source tasks and the current target task should be able to perform transfer accordingly. By measuring the amount of inter-task similarity, we design a method for value transfer, practically applicable in the online lifelong RL setting.

Specifically, we introduce a metric between MDPs and prove that the optimal Q-function is Lipschitz continuous with respect to the MDP space. This property allows to compute a provable upper bound on the optimal Q-function of an unknown target task, given the learned optimal Q-function of a source task. Knowing this upper bound allows to accelerate the convergence of an RMAX-like algorithm, relying on an optimistic estimate of the optimal Q-function as described in Chapter 2, Section 2.3.2, page 28. Importantly, this method performs non-negative transfer (it cannot cause performance degradation) as the computed upper bound provably does not underestimate the optimal Q-function.

This chapter is organized as follows. First, in Section 5.1, page 93, we review the state of the art in terms of transfer methods in the lifelong RL setting. We focus on the approaches related to the quantification of the distance between MDPs, for instance with respect to a metric. Secondly, in Section 5.2, page 95, we define formally the lifelong RL framework, then we introduce the notion of negative transfer, and we describe the approach proposed in this chapter. Thirdly, in Section 5.3, page 100, we introduce a pseudometric between MDPs and show that the optimal Q-function — seen as a function of the MDPs — is Lipschitz Continuous with respect to this pseudometric. Intuitively, this result establishes that MDPs that are close from each other necessarily have close Q-functions. In Section 5.4, page 105, we use this continuity property to propose a value transfer method based on the calculation of this local pseudo distance between MDPs. Naturally, if the computed distance is small, the Q-function of the target task can practically be deduced from the one of the source task. Full knowledge of both MDPs is not required by the method and the transfer is non-negative, which makes the method both practical and safe in an online setting. In Section 5.5, page 111, we build a PAC-MDP algorithm called *Lipschitz R-Max (LRMAX)*, applying the transfer method in the online lifelong RL setting. We theoretically study the algorithm by providing sample and computational complexity bounds. In Section 5.6, page 117, we showcase the LRMAX algorithm in lifelong RL experiments and demonstrate its advantages for accelerating the learning of new target tasks. Finally, we conclude in Section 5.7, page 120.

5.1 State of the art

Transfer in the context of lifelong RL has been extensively studied in the past decades (Taylor and Stone, 2009; Lazaric, 2012). The general idea is to transfer some knowledge under any form from source tasks to target tasks to accelerate the learning of the latter. Transfer can be applied to the RL field but also in supervised or unsupervised tasks. In this literature review, we focus on transfer methods relying on any similarity measure between source and a target task. Using such a similarity measure between MDPs or related features has the appealing characteristic of quantifying the degree of analogy between tasks, which, intuitively, should be linked to the amount of transfer achievable. Therefore, the availability of a similarity measure can be useful to determine from which source task should transfer be performed or even how much transfer should be applied.

Carroll and Seppi (2005) define four task similarity measures, respectively quantifying: the "advantage", seen for instance as the reward gain or convergence speed, of applying any transfer method between two tasks; the number of states with identical optimal policy; the mean squared error between the optimal Q-functions of the source and target task; the mean squared error between the expected reward function of the source and target task. Regardless of the employed task similarity measure, *direct transfer* (Carroll, 2005) is used as the transfer method in the experiments. More precisely, direct transfer corresponds to the direct initialization of the learned Q-function of a target task with the learned Q-function of a source task. Notice that such a method may allow for negative transfer, a notion that we will precisely define in Section 5.2.2, page 97 but that roughly corresponds to a degradation of the learning speed compared to an algorithm learning from scratch. With those four similarity measures, they show that different measures can be helpful to capture different similarity aspects of two MDPs. However, no metric dominates the others uniformly in terms of transfer performance. Despite the fact that the measures require both source and target tasks to be completely learned, the authors propose to start applying the method prematurely after reaching a certain number of decision epochs for the framework to be applicable in the online lifelong RL setting. Additionally, to limit the number of inter-task comparisons that grows linearly with the number of source tasks, a clustering technique to group the data issued from similar source tasks is proposed.

Fernández and Veloso (2006) introduce a policy re-use method based on a similarity measure between policies. The degree of similarity is assessed in terms of the value of a policy in the current task. Given a library of policies, they propose an algorithm that selectively re-uses the most valuable policy with respect to this criterion. The method can be applied to the lifelong RL framework as a policy transfer approach.

Lazaric, Restelli, and Bonarini (2008) provide a practical method for sample transfer, computing a similarity metric reflecting the probability that the models of the source and target tasks are identical. This similarity metric is used to identify the most suited source task from which samples can be transferred. The selection being based on a similarity measure, the approach allows for more robustness to negative transfer. Additionally, a relevance score is computed for each sample to select those providing more advantage to speed-up the learning of the target task. The proposed approach is applicable in a batch RL setting as opposed to the online setting considered in this chapter.

Sorg and Singh (2009) study the problem of transfer learning in the case where there exist a *soft homomorphism* between the state spaces of the tasks. Formally, this can be interpreted as the existence of a mapping from each state of the target MDP to a probability distribution on the state space of the source task. They define a soft MDP homomorphism such that the reward and transition functions of the source MDP weighted by the soft homomorphism match the reward and transition functions of the target MDP. The existence of such a function allows to bound the optimal Q-value function of a target MDP. Similarly to the approach developed in this chapter, the bound can in turn be used to initialize the Q-function of the source MDP, resulting in a value transfer method. Noticeably, the soft homomorphism mapping described above can be learned during interaction with the target MDP.

Konidaris, Scheidwasser, and Barto (2012) introduce a notion of shared feature between tasks. They propose a value transfer method based on the learning of a *portable* — *i.e.*, defined for any task — value function mapping *features* to values. This function allows to initialize the learned value function of a target MDP by capturing the similar features with a source task.

Mahmud, Hawasly, Rosman, and Ramamoorthy (2013) propose a clustering technique to prune the set of source tasks and rather represent them by a subset of tasks. They introduce a similarity measure between MDPs by quantifying the regret of running the optimal policy of one MDP in the other MDP. This similarity measure is in turn used to cluster source tasks and form an ϵ-net over the MDP space, with the following understanding of an ϵ-net: once a new target task is sampled, there exist a source task in the resulting subset from the clustering process whose computed optimal policy has a value function that is ϵ-close to the optimal value function of the source task. Therefore, the transfer method is a policy transfer method where the learned optimal policy of a selected target task is selected.

Brunskill and Li (2013) also introduce a clustering method using the $\mathcal{L}_1$ norm between learned source MDP models. The data used to learn close models during interactions with those MDPs are grouped, forming pools of data issued from similar source tasks. In turn, during learning of a new target task, if the collected data provide sufficient evidence that the target task belongs to a single cluster, the data from this cluster are merged with the collected data to accelerate learning. The approach is proven to yield non-negative transfer and can be applied in the multi-task RL setting.

Ammar, Eaton, Taylor, Mocanu, Driessens, Weiss, and Tuyls (2014) propose to learn the model of a source MDP and to view the prediction error on a target MDP as a dissimilarity measure in the task space. The method makes use of samples collected from both source and target tasks. It allows to identify which target task would be best suited for transfer which is performed in different ways, namely, by respectively initializing the Q-function values, the Q-function parameters or the policy of an explored target task with the optimal Q-function values, the optimal Q-function parameters and the optimal policy of a learned source task. The method is studied in an offline setting where both target and source tasks have been explored. The extension to the online lifelong RL setting is seen as a future contribution.

Song, Gao, Wang, and An (2016) define a metric based on the bi-simulation metric introduced by Ferns, Panangaden, and Precup (2004) and the Wasserstein metric (Villani, 2008). The developed idea is to perform Value transfer between states with low bi-simulation distances. This metric requires knowing both MDPs completely and is thus unusable in the lifelong RL setting where we expect to perform transfer before having learned the target MDP.

Lastly, Abel, Jinnai, Guo, Konidaris, and Littman (2018) present the MaxQInit algorithm featuring many similarities to the approach proposed in this chapter. Although the method is not a transfer method based on a similarity measure — contrarily to the other contributions presented in this section — the way transfer is performed and the provided guarantees are exactly the same. MaxQInit is a generic method to compute with high probability the maximum over the Q-values of a finite set of MDPs in the lifelong RL setting. Using this maximum value for the initialization of the Q-value of a new source task potentially accelerates learning in a provably non-negative transfer method. Importantly, the presented algorithm is built on the RMAX algorithm (Chapter 2, Section 2.3.2, page 28) and preserves the PAC-MDP guarantees of this algorithm. Practically, the method consists in sequentially learning tasks among the finite set of MDPs to assess the maximum Q-value for each state-action pair over the set of sampled tasks. Once enough samples have been gathered, this maximum value is used as the initialization Q-value for all the visited state-action pairs, resulting in a tighter upper bound on the Q-function than the one used by RMAX, namely,

$$\frac{R_{\max}}{1-\gamma} \,. \tag{5.1}$$

Critically, the sample complexity of the latter algorithm is decreased by providing a tighter upper bound than $\frac{R_{\max}}{1-\gamma}$, making MaxQInit faster at learning new tasks.

5.2 Framework

In this section, we first describe formally the lifelong RL framework. Then, we outline the problem of negative transfer that can be met in lifelong RL. Finally, we qualitatively describe the approach developed in this chapter.

5.2.1 Lifelong Reinforcement Learning

Lifelong Reinforcement Learning (Silver, Yang, and Li, 2013; Brunskill and Li, 2014) is the problem of experiencing online a series of MDPs drawn from an unknown distribution. Each time an MDP is sampled, a classical RL problem takes place — as depicted in Chapter 2, Algorithm 3, page 28 — where the agent is able to interact with the environment to maximize its expected return. Additionally, each experience with a new MDP can be specified by sampling an horizon H and an initial state distribution $\mathcal{P}_0$, respectively defining the length of interaction between the agent and the current MDP and the distribution of the initial state of any episode.

As commonly done (Wilson, Fern, Ray, and Tadepalli, 2007; Lazaric, Restelli, and Bonarini, 2008; Ammar, Eaton, Taylor, Mocanu, Driessens, Weiss, and Tuyls, 2014; Brunskill and Li, 2014; Abel, Jinnai, Guo, Konidaris, and Littman, 2018) we restrict the scope of the study to the case where sampled MDPs share the same state-action space $\mathcal{S} \times \mathcal{A}$. These restricted sets of MDPs give a relevant insight on the question we try to answer and are of great importance in the RL literature. Similarly to the expected reward function (Chapter 2, Definition 2.2, page 11), we consider reward functions depending on state-action pairs, *i.e.*,

$$\begin{array}{rccl} R: & \mathcal{S} \times \mathcal{A} & \rightarrow & [0, R_{\max}] \\ & (s, a) & \mapsto & R_s^a \, . \end{array}$$

Without loss of generality, we will assume $R_{\max} = 1$ in this chapter, keeping in mind that scaling a reward signal between 0 and 1 has no effect on learning. Overall, in this chapter, we refer indifferently to MDPs, models or tasks, and write them $M \equiv (T, R)$ with the understanding that they share the same state-action space $\mathcal{S} \times \mathcal{A}$. Consequently, instead of sampling complete MDPs in the lifelong RL procedure described above, only models are sampled. We will write $\mathcal{M}$ the space of MDPs. The general lifelong RL procedure is described in Algorithm 9.

Algorithm 9 General lifelong RL procedure

```
Input: state-action space $\mathcal{S} \times \mathcal{A}$; distribution $\mathcal{D}$ over $(T, R, H, \mathcal{P}_0)$; learning agent $\blacktriangle$.
Repeat:
    $(T, R, H, \mathcal{P}_0) \sim \mathcal{D}$ # Sample a model, an horizon and an initial state distribution.
    Apply Algorithm 3, page 28, with arguments $\{\blacktriangle, \mathcal{S}, \mathcal{A}, T, R, \mathcal{P}_0, H\}$ # Classical RL procedure.
```

Noticeably, Hallak, Di Castro, and Mannor (2015) introduce the setting of Contextual Markov Decision Process (CMDP) presenting similarities with the lifelong RL framework. A CMDP consists of a fixed set of MDPs from which models are sampled and experienced during episodes of fixed durations. The sampling strategy could be random or adversarial. In this setting, they provide an algorithm achieving bounded regret with respect to the optimal policy. The proposed method consists in first clustering the experienced trajectories to differentiate between the MDPs; secondly classifying the current trajectory to identify if it belongs to one of the computed clusters (exploration); finally set the policy to the one learned from the trajectories of the identified cluster (exploitation).

Similarities can also be found between lifelong RL and the setting of HMMDPs described in Chapter 4, Section 4.1, page 62. Both frameworks have in common the sequential interaction with several different MDPs. The major difference is that the variable indicating that a new MDP has been sampled is hidden in the context of an HMMDP and revealed to the agent in lifelong RL.

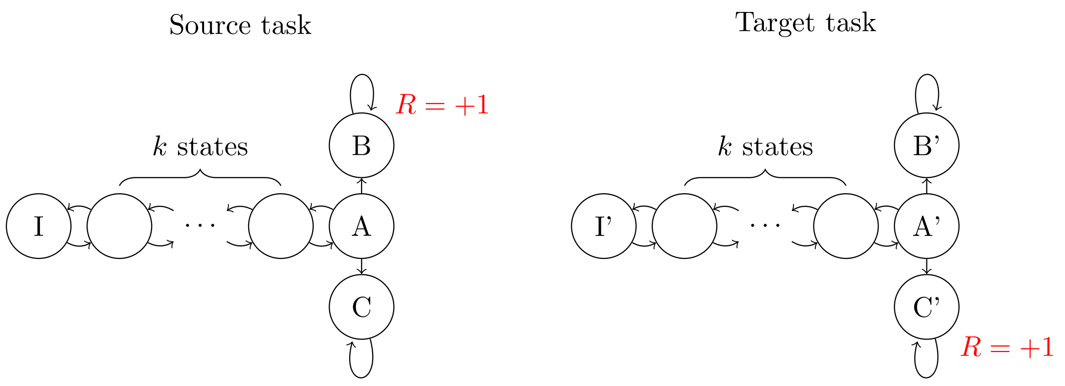


Figure 5.1: The T-shaped MDP transfer task.

5.2.2 Negative transfer

In the lifelong RL setting, it is reasonable to think that knowledge gained on previous MDPs could be re-used to improve the performance in new MDPs. Such a practice, known as knowledge transfer, sometimes does cause the opposite effect, *i.e.*, decreases the performance. In such a case, we talk about *negative transfer*. Several attempt to formally define negative transfer have been done, but researchers hardly agree on a single definition, as *performance* can be defined in various ways. For instance, it can be characterized by the speed of convergence, the area under the learning curve, the final score of the learned policy or classifier, and many other things. Defining negative transfer is out of the scope of this chapter, but let us give an example of why this phenomenon can be problematic.

In their paper, Song, Gao, Wang, and An (2016) propose a transfer methods based on the metric between MDPs they introduce, stemming from the bi-simulation metric introduced by Ferns, Panangaden, and Precup (2004). In their method, a bi-simulation metric is computed between each pair of states belonging respectively to the source and target MDPs. Roughly, this metric tells how *different* are the transition and reward models corresponding to the states pairs, for the action maximizing th distance. More precisely, if we write (T, R) and $(\bar{T}, \bar{R})$ the models of two MDPs, and $(s, s') \in \mathcal{S}$ a state pair, the distance d between s and s' is defined by

$$d(s, s') = \max_{a \in \mathcal{A}} \left\{ \left| R^a_s - \bar{R}^a_{s'} \right| + c\, W_1 \left(T^a_{s \cdot}, \bar{T}^a_{s' \cdot} \right) \right\} , \tag{5.2}$$

where $c \in \mathbb{R}$ is a positive constant and W_1 is the 1-Wasserstein metric defined in Chapter 4, Definition 4.5, page 70. For each state of the target model, the closest counterpart state (with the smallest bi-simulation distance) of the source MDP is identified and its learned Q-values are used to initialize the Q-function of the target MDP. In their experiments, Song, Gao, Wang, and An (2016) run a standard Q-Learning algorithm (Watkins and Dayan, 1992) with an ϵ-greedy exploration strategy thereafter.

Let us now consider applying this method to a similar task to the T-shaped MDP transfer task proposed by Taylor and Stone (2009). The source and target tasks are respectively described on the left and right sides of Figure 5.1. In each task, the states are represented by the circles and the arrows between them correspond to the available actions that allow to move from one state to the other. Notice that the letters representing the states of both tasks are differentiated by the presence of a "prime" (for instance *I'*) in the target task and the absence of a "prime" (for instance *I*) in the source task. This should be understood as the *same* states seen either in the source or in the target task. As a result, the two tasks effectively share the same state-action space. The initial state of both tasks is the left state I for the source task and I$'$ for the target task. Between the states I and A in the source task (respectively I$'$ to A$'$ in the target task) are k states, k being a parameter increasing the distance to travel from I to A (respectively I$'$ to A$'$). The tasks are deterministic and the reward is zero everywhere except for the state B in the source task and C$'$ in the target task where a reward of $+1$ is received. Consequently, the optimal policy in the source task is to go to the state A and then to the state B. In the target task, the same applies except that a transition to state C should be applied in place of state B$'$ when the agent is in state A$'$.

Regardless of the parameters used in the bi-simulation metric of Equation 5.2, page 97, the direct state transfer method from Song, Gao, Wang, and An (2016) maps the following states together as they share the exact same model:

$$\begin{array}{ccc} \text{I} & \longleftrightarrow & \text{I}' \\ \text{k states} & \longleftrightarrow & \text{k states} \\ \text{A} & \longleftrightarrow & \text{A}'\,. \end{array}$$

Hence, during learning, the Q-function of the target task is initialized with the values of the Q-function of the source task. Therefore, the behavior derived with the Q-Learning algorithm is the optimal policy of the source task, but in the target task. Depending on the value of the learning rate of the algorithm, the time to favor action DOWN in state A$'$ instead of action UP can be arbitrarily long. Also, depending on the value of ϵ, the exploration of state C$'$ due to the ϵ-greedy strategy can be arbitrarily unlikely. Finally, the time needed for one of those two events to occur increases proportionally to the value of k, which can be arbitrarily large.

This case illustrates the difficulty facing any transfer method in the general context of lifelong RL. The method proposed by Song, Gao, Wang, and An (2016) can be highly efficient in some cases as they show in experiments, but the lack of theoretical guarantees makes negative transfer possible. Generally, using a similarity measure such that the bi-simulation metric helps to discourage using some source tasks when the computed similarity is too low. However, as we saw in the T-shaped MDP example, this rule is not absolute and the choice of the metric is important. The approach we develop in this chapter aims at avoiding negative transfer by providing a conservative transferred knowledge that is simply of no use when the similarity between source and target tasks is too low. This is intuitive as we do not expect *any* task to provide transferable knowledge to *any* other task.

5.2.3 Proposed approach

The transfer method we propose is based on the following observation: the RMAX Algorithm, presented in Chapter 2, Section 2.3.2, page 28, is a PAC-MDP algorithm whose sample complexity scales with the number of state-action pairs for which the upper bound on the Q-value of a sub-optimal action overestimates the upper bound on the Q-value of an optimal action (see Theorem 2.9, page 32). Recall that RMAX applies the principle of optimism in the face of uncertainty and therefore drives its exploration *vs.* exploitation trade-off by using an upper bound on the optimal Q-function. The standard approach consists in taking the maximum achievable value $\frac{1}{1-\gamma}$ as an initialization for the upper bound. Hence, for any state, all the sub-optimal actions have overestimated Q-values, and the sample complexity of RMAX is maximized. In the context of lifelong RL, we propose to refine this upper bound for the exploration of new target tasks by measuring their similarity to source tasks.

In this chapter, we will use a local pseudometric between MDPs as a similarity measure. We will show in Section 5.3, page 100 that the optimal Q-function is Lipschitz Continuous (LC) with respect to the space of MDPs. Recall that LC functions are defined in Chapter 4, Definition 4.4, page 70. Written intuitively, we aim at having the following inequality, for two MDPs $M, \bar{M} \in \mathcal{M}$, and a pair $(s, a) \in \mathcal{S} \times \mathcal{A}$:

$$\left| Q^*_M(s,a) - Q^*_{\bar{M}}(s,a) \right| \leq d\left(M, \bar{M}\right) , \tag{5.3}$$

with d the aforementioned local pseudometric. In turn, if $\bar{M}$ is a learned source MDP and M a new target MDP, Equation 5.3 yields

$$Q^*_M(s,a) \leq Q^*_{\bar{M}}(s,a) + d\left(M, \bar{M}\right) . \tag{5.4}$$

Our main goal is to establish a result of the form of Equation 5.3 and to use the upper bound of Equation 5.4 to decrease the sample complexity of an RMAX-like algorithm in new target tasks. Overall, the proposed transfer method consists in using the upper bound described in Equation 5.4.

The method developed by Abel, Jinnai, Guo, Konidaris, and Littman (2018) is similar to ours in two fundamental points: first, a theoretical upper bound on optimal Q-functions across the MDP space is built; secondly, this provable upper bound is used to transfer knowledge between MDPs by replacing the maximum $\frac{1}{1-\gamma}$ bound in an RMAX-like algorithm, providing PAC guarantees. The difference between the two approaches is illustrated in Figure 5.2, page 100 where the MaxQInit bound is the one developed by Abel, Jinnai, Guo, Konidaris, and Littman (2018) and the LRMAX bound is the one we present in this chapter. On this figure, the essence of the LRMAX bound is noticeable. It stems from the fact that the optimal Q-value function is locally LC in the MDP space with respect to a specific metric. Confirming the intuition, close MDPs with respect to this metric have close optimal Q-values. It should be noticed that no bound is uniformly better than the other as intuited by Figure 5.2, page 100. Hence, combining all the bounds results in a tighter upper bound as we will later illustrate in experiments (Section 5.6, page 117). We first carry out the theoretical characterization of the Lipschitz continuity properties in the following section. Then, we build on this result to propose a practical transfer method for the online lifelong RL setting.

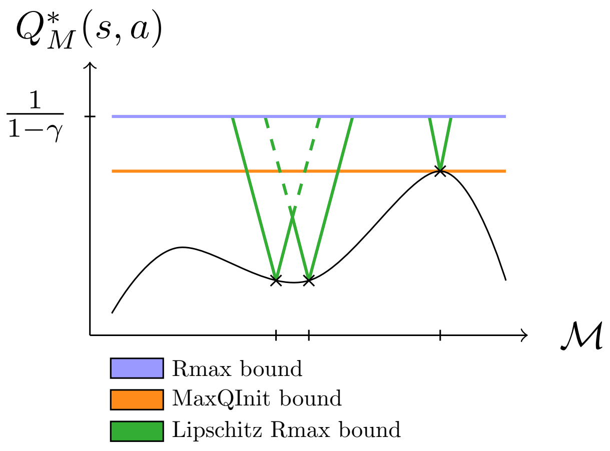


Figure 5.2: Comparison of the RMAX upper bound, the MaxQInit upper bound and the LRMAX upper bound. The optimal Q-function is represented for a particular s, a pair as a function of the MDP space $\mathcal{M}$. The RMAX, MaxQInit and LRMAX bounds are represented for three sampled target MDPs.

5.3 Lipschitz continuity of Q-functions

In this section, we first present the main result of Lipschitz continuity of the optimal Q-function with respect to the task space. Then, we will outline similar results that are interesting from a theoretical point of view but that we will not use in the remainder of the dissertation.

5.3.1 Main result

The intuition we build on is that *similar* MDPs should have *similar* optimal Q-functions. Formally, this insight can be translated into a continuity property of the optimal Q-functions over the MDP space $\mathcal{M}$. To that end, we introduce a local pseudometric characterizing the distance between the models of two MDPs at a particular state-action pair.

Definition 5.1. Given two tasks $M = (R, T)$ and $\bar{M} = (\bar{R}, \bar{T})$, and any function $f : \mathcal{S} \to \mathbb{R}^+$, we define the *pseudometric between models* at $(s, a) \in \mathcal{S} \times \mathcal{A}$ with respect to f as:

$$D^f_{sa}(M, \bar{M}) \triangleq \left| R^a_s - \bar{R}^a_s \right| + \sum_{s' \in \mathcal{S}} f(s') \left| T^a_{ss'} - \bar{T}^a_{ss'} \right| . \tag{5.5}$$

Recall that a *metric* on a set X is a function $m : X \times X \rightarrow \mathbb{R}$ which has the following properties for any $x, y, z \in X$:

P1. $m(x, y) \geq 0$ (positivity),

P2. $m(x, y) = 0 \Leftrightarrow x = y$ (positive definiteness),

P3. $m(x, y) = m(y, x)$ (symmetry),

P4. $m(x, z) \leq m(x, y) + m(y, z)$ (triangle inequality).

If property P2 is not verified by m, but instead we have that $m(x, x) = 0$ for any $x \in X$, then m is called a *pseudometric*. If m only verifies P1, P2 and P4 then m is called a *quasimetric*. If m only verifies P1 and P2 and if X is a set of probability measures, then m is called a *divergence*.

From this, the pseudometric between models that we introduced in Definition 5.1, page 100 is indeed a pseudometric as it is relative to a positive function f that could be equal to zero and break property P2 while the other properties are verified. We implicitly cast this definition in the context of discrete state spaces. The extension to continuous spaces is straightforward but beyond the scope of this chapter. For the sake of clarity in the remainder of this study, we introduce the following notation for any two MDPs $M, \bar{M} \in \mathcal{M}$ and $(s, a) \in \mathcal{S} \times \mathcal{A}$:

$$D_{sa}(M \| \bar{M}) \triangleq D_{sa}^{\gamma V_{\bar{M}}^{*}}(M, \bar{M}) \,,$$

corresponding to the pseudometric between models with the particular choice of $f = \gamma V_{\bar{M}}^{*}$. We now state the main result of the local pseudo-Lipschitz continuity.

Theorem 5.1 (Local pseudo-Lipschitz continuity). *For two MDPs $M, \bar{M} \in \mathcal{M}$, for all state action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$,*

$$\left| Q_{M}^{*}(s, a) - Q_{\bar{M}}^{*}(s, a) \right| \leq \Delta_{sa}(M, \bar{M}) \,, \tag{5.6}$$

with the local MDP pseudometric *defined by* $\Delta_{sa}(M, \bar{M}) \triangleq \min \left\{ d_{sa}(M \| \bar{M}), d_{sa}(\bar{M} \| M) \right\}$*, and the* local MDP dissimilarity $d_{sa}(M \| \bar{M})$ *defined as the unique solution to the following fixed-point equation for $d_{sa} \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$:*

$$d_{sa} = D_{sa}(M \| \bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^{a} \max_{a' \in \mathcal{A}} d_{s'a'} \,, \; \forall \, (s, a) \in \mathcal{S} \times \mathcal{A} \,. \tag{5.7}$$

The local MDP dissimilarity between MDPs $d_{sa}(M \| \bar{M})$ of Theorem 5.1 does not respect the metric properties P2 and P3, hence the name *dissimilarity*. Notice that, since this dissimilarity is asymmetric, it yields two valid upper bounds. Taking the minimum of both still produces a valid upper bound, hence the min operator in the definition of the local MDP pseudometric. The latter, $\Delta_{sa}(M, \bar{M}) = \min \left\{ d_{sa}(M \| \bar{M}), d_{sa}(\bar{M} \| M) \right\}$, however, regains property P3 and is hence a pseudometric. A noticeable consequence is that Theorem 5.1 is "in the spirit" of a Lipschitz continuity result but cannot be called as such, hence the name *pseudo-Lipschitz continuity.* To prove Theorem 5.1, we shall introduce a Lemma on the existence and uniqueness of the solution to the fixed-point equation. The proofs of Lemma 5.1 and Theorem 5.1 are reported in the Appendix, Chapter A, Section A.4.

Lemma 5.1. *Given two MDPs $M, \bar{M} \in \mathcal{M}$, the following equation on $d \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$ is a fixed-point equation admitting a unique solution for any $(s, a) \in \mathcal{S} \times \mathcal{A}$:*

$$d_{sa} = D_{sa}(M \| \bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \max_{a'} d_{s'a'} \,.$$

We refer to this unique solution as $d_{sa}(M \| \bar{M})$.

Lemma 5.1 guarantees the existence of $d_{sa}(M \| \bar{M})$. Theorem 5.1, page 101 establishes that the distance between the optimal Q-functions of two MDPs at $(s, a) \in \mathcal{S} \times \mathcal{A}$ is controlled by a local dissimilarity between the MDPs. The latter follows a fixed-point equation (Equation 5.7, page 101), which can be solved by DP, for instance, in a value iteration procedure (See Chapter 2, Section 2.2.2, page 18). Notice that, for the result to hold, the function

$$\begin{array}{rccc} f: & \mathcal{S} & \to & \mathbb{R}^+ \\ & s & \mapsto & V^*_{\bar{M}}(s) \end{array}$$

was selected in the definition of the model's pseudometric (Equation 5.5, page 100). Notice also that the policies in Equation 5.6, page 101 are the optimal ones for the two MDPs respectively and thus are generally different policies.

The approach we presented in Section 5.2.3, page 98 relies on an upper bound on the optimal Q-function of a target task in the lifelong RL setting. Let us derive such an upper bound based on the local pseudo-Lipschitz continuity results presented so far. Consider the case where $\bar{M} \in \mathcal{M}$ is a learned source task and $M \in \mathcal{M}$ is a new target task. Borrowing the notations of Theorem 5.1, page 101, given that $Q^*_{\bar{M}}$ is known, the function

$$\begin{array}{rcl} \mathcal{S} \times \mathcal{A} & \to & \mathbb{R} \\ (s, a) & \mapsto & Q^*_{\bar{M}}(s, a) + \Delta_{sa}(M, \bar{M}) \end{array} \tag{5.8}$$

can be used as an upper bound on Q^*_M. Still the question of how to compute this function remains. Indeed, in the context of lifelong RL where $\bar{M}$ is a source task and M a target task, $\bar{M}$ may be partially known and little to no experience of M has been acquired. Hence, the function of Equation 5.8 cannot be computed as it contains unknown quantities. Despite this difficulty, this function is the basis on which we will construct a computable and transferable upper bound in Section 5.4, page 105.

5.3.2 Similar Lipschitz continuity results

Similar results to Theorem 5.1, page 101 can be derived, namely for the optimal value function, and the value and Q-value functions of any policy. We state those results in this section for the sake of completeness, and for their theoretical interest. However, one should know that they will not be used further in this dissertation. Therefore, the reader only interested in the transfer method derived from the local pseudo-Lipschitz continuity result of Theorem 5.1, page 101 may skip this section and directly proceed to Section 5.4, page 105.

The first result is the equivalent local pseudo-Lipschitz continuity result of the optimal Q-function for the optimal value function, stated below.

Theorem 5.2 (Local pseudo-Lipschitz continuity of the optimal value function)**.** *For any two MDPs $M, \bar{M} \in \mathcal{M}$, for all $s \in \mathcal{S}$,*

$$\left| V^*_M(s) - V^*_{\bar{M}}(s) \right| \leq \max_{a \in \mathcal{A}} \Delta_{sa}(M, \bar{M})$$

where the local MDP pseudometric $\Delta_{sa}(M, \bar{M})$ has the same definition as in Theorem 5.1, page 101.

The proof of Theorem 5.2 is reported in the Appendix, Chapter A, Section A.4. Another result can be derived for any policy π that one wishes to evaluate in both MDPs. For the sake of generality, we state the results for any stochastic stationary policy mapping states to distributions over actions. Recall that a deterministic policy is a stochastic policy choosing a particular action with probability 1 and the others with probability 0. First, we state the result for the value function in Theorem 5.3 and then for the Q-function in Theorem 5.4, page 104. Each theorem relies on the existence and uniqueness of the solution to a fixed-point equation. Therefore, they both are preceded by a lemma, stating the result. The proofs of Lemma 5.2, Theorem 5.3, Lemma 5.3, page 104, and Theorem 5.4, page 104 are reported in the Appendix, Chapter A, Section A.4.

Lemma 5.2. *Given two MDPs $M, \bar{M} \in \mathcal{M}$, any stochastic stationary policy π, the following equation on $d \in \mathcal{F}(\mathcal{S}, \mathbb{R})$ is a fixed-point equation admitting a unique solution for any $s \in \mathcal{S}$:*

$$d_s = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(D_{sa}^{\gamma V^\pi_{\bar{M}}}(M, \bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} d_{s'} \right) .$$

We refer to this unique solution as $d^\pi_s(M \| \bar{M})$.

Theorem 5.3 (Local pseudo-Lipschitz continuity of the value function of any policy)**.** *For any two MDPs $M, \bar{M} \in \mathcal{M}$, for any stochastic stationary policy π, for all $s \in \mathcal{S}$,*

$$\left| V^\pi_M(s) - V^\pi_{\bar{M}}(s) \right| \leq \Delta^\pi_s(M, \bar{M})$$

where $\Delta^\pi_s(M, \bar{M}) \triangleq \min\left\{ d^\pi_s(M \| \bar{M}), d^\pi_s(\bar{M} \| M) \right\}$ and $d^\pi_s(M \| \bar{M})$ is defined as the fixed-point of the following fixed-point equation on $d \in \mathcal{F}(\mathcal{S}, \mathbb{R})$:

$$d_s = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(D_{sa}^{\gamma V^\pi_{\bar{M}}}(M, \bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} d_{s'} \right) .$$

Lemma 5.3. *Given two MDPs $M, \bar{M} \in \mathcal{M}$, any stochastic stationary policy π, the following equation on $d \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$ is a fixed-point equation admitting a unique solution for any $(s,a) \in \mathcal{S} \times \mathcal{A}$:*

$$d_{sa} = D_{sa}^{\gamma V_{\bar{M}}^{\pi}}(M, \bar{M}) + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} T_{ss'}^{a} \pi(a' \mid s') d_{s'a'} \,.$$

We refer to this unique solution as $d_{sa}^{\pi}(M \| \bar{M})$.

Theorem 5.4 (Local pseudo-Lipschitz continuity of the Q-function of any policy)**.** *For any two MDPs $M, \bar{M} \in \mathcal{M}$, for any stochastic stationary policy π, for all $(s,a) \in \mathcal{S} \times \mathcal{A}$,*

$$\left| Q_M^{\pi}(s,a) - Q_{\bar{M}}^{\pi}(s,a) \right| \leq \Delta_{sa}^{\pi}(M, \bar{M})$$

where $\Delta_{sa}^{\pi}(M, \bar{M}) \triangleq \min \left\{ d_{sa}^{\pi}(M \| \bar{M}), d_{sa}^{\pi}(\bar{M} \| M) \right\}$ and $d_{sa}^{\pi}(M \| \bar{M})$ is defined as the fixed-point of the following fixed-point equation on $d \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$:

$$d_{sa} = D_{sa}^{\gamma V_{\bar{M}}^{\pi}}(M, \bar{M}) + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} T_{ss'}^{a} \pi(a' \mid s') d_{s'a'} \,.$$

A consequence of Theorem 5.1, page 101 is a global pseudo-Lipschitz continuity result.

Theorem 5.5 (Global pseudo-Lipschitz continuity)**.** *For two MDPs $M, \bar{M} \in \mathcal{M}$, for all $(s,a) \in \mathcal{S} \times \mathcal{A}$,*

$$\left| Q_M^{*}(s,a) - Q_{\bar{M}}^{*}(s,a) \right| \leq \Delta(M, \bar{M}) \,, \tag{5.9}$$

with $\Delta(M, \bar{M}) \triangleq \min \left\{ d(M \| \bar{M}), d(\bar{M} \| M) \right\}$ and

$$d(M \| \bar{M}) \triangleq \frac{1}{1-\gamma} \max_{(s,a) \in \mathcal{S} \times \mathcal{A}} \left\{ D_{sa}(M \| \bar{M}) \right\} \,.$$

The proof of Theorem 5.5 is reported in the Appendix, Chapter A, Section A.4. Similarly to the local MDP dissimilarity between MDPs of Theorem 5.1, page 101, the quantity $d(M \| \bar{M})$ does not respect the metric properties P2 and P3, which makes it a *dissimilarity* measure between MDPs. However, the quantity $\Delta(M, \bar{M})$ allows to regain property P3 and results in a pseudometric between MDPs. Like Theorem 5.1, page 101, this fact justifies the name global pseudo-Lipschitz continuity of Theorem 5.5.

From a pure transfer perspective, Equation 5.9 is interesting since the right hand side does not depend on a state action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$. Hence, the counterpart of the upper bound of Equation 5.8, page 102, namely,

$$\begin{array}{lcl} \mathcal{S} \times \mathcal{A} & \to & \mathbb{R} \\ (s,a) & \mapsto & Q_{\bar{M}}^{*}(s,a) + \Delta(M, \bar{M}) \end{array}$$

is easier to compute. Indeed, $\Delta(M, \bar{M})$ can be computed once and for all, contrarily to

$\Delta_{sa}(M, \bar{M})$ that needs to be evaluated for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. However, we do not use this result for transfer because it is impractical to compute online. Indeed, estimating the maximum in the definition of $d(M \| \bar{M})$ might be as hard as solving both MDPs, which, when it happens, is too late for transfer to be useful. On the other hand, the result of the local Lipschitz continuity can yield a practical, computable upper bound in the online lifelong RL setting. In the next section, we detail the calculation and use of such a bound.

5.4 Transfer using Lipschitz continuity

We derived a local pseudo-Lipschitz continuity result in Section 5.3.1, page 100, Theorem 5.1, page 101. Given an MDP $M \in \mathcal{M}$, this result allowed us to propose an upper bound on Q^*_M via Equation 5.8, page 102. As such an upper bound can accelerate learning, it can be a useful information to transfer from one MDP to the other. In Section 5.4.1, we formally define this transferable upper bound. Then, we will see the limits of this approach as some quantities are unknown in the definition of this transferable upper bound, making it impractical to compute. Therefore, in Section 5.4.2, page 107, we propose a way to compute this quantity. Precisely, we introduce a surrogate upper bound that can be calculated online in the lifelong RL setting, without having completely explored the source and target tasks. Finally, we implement the method in an algorithm described in Section 5.5, page 111.

5.4.1 A transferable upper bound on the optimal Q-function

A purpose of value transfer, when interacting online with a new MDP, is to initialize the value function and drive the exploration to accelerate learning. We aim to exploit value transfer in a method guaranteeing three conditions:

- C1. the resulting algorithm is PAC-MDP (Strehl, Li, and Littman, 2009);
- C2. the transfer accelerates learning;
- C3. the transfer is non-negative.

From Theorem 5.1, page 101, one can naturally define a local upper bound on the optimal Q-function of an MDP given the optimal Q-function of another MDP.

Definition 5.2. Given two tasks $M, \bar{M} \in \mathcal{M}$, for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, the *Lipschitz upper bound on Q^*_M induced by $Q^*_{\bar{M}}$* is defined as

$$U_{\bar{M}}(s, a) \triangleq Q^*_{\bar{M}}(s, a) + \Delta_{sa}(M, \bar{M}) \,, \tag{5.10}$$

and verifies $U_{\bar{M}}(s, a) \geq Q^*_M(s, a)$.

The principle of *optimism in the face of uncertainty*, introduced in Chapter 2, Section 2.3.2, page 28, leads to consider that the long-term expected return from any state-action pair is the maximum return $\frac{1}{1-\gamma}$, unless proven otherwise. The RMAX algorithm in particular explores an MDP so as to shrink this upper bound that drives its exploration *vs.* exploitation trade-off. RMAX, introduced in Chapter 2 Section 2.3.2, page 28, is a model-based, online RL algorithm with PAC-MDP guarantees which means that convergence to a near-optimal policy is guaranteed in a polynomial number of steps with high probability. It relies on an optimistic model initialization that yields an optimistic upper bound U on the optimal Q-function, then acts greedily with respect to U. By default, it takes the maximum value

$$U(s,a) = \frac{1}{1-\gamma},$$

but any tighter upper bound is admissible. Thus, shrinking U with Equation 5.10, page 105 is expected to improve the learning speed or sample complexity for new tasks in lifelong RL. In RMAX, during the resolution of a task $M \in \mathcal{M}$, the state-action space $\mathcal{S} \times \mathcal{A}$ is split into a subset of known state-action pairs K and its complement K^c of unknown pairs. A state-action pair is *known* if the number of collected reward and transition samples allows estimating an ϵ-accurate model in $\mathcal{L}_1$-norm with probability higher than $1-\delta$. We refer to ϵ and δ as the *RMAX precision parameters*. This translates into a threshold n_{known} on the number of visits $n(s,a)$ to a pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ that are necessary to reach this precision. Consider executing RMAX in the context of lifelong RL, given the experience of a set of m MDPs $\bar{\mathcal{M}} = \{\bar{M}_1, \dots, \bar{M}_m\}$, we define the total bound as the minimum over all the Lipschitz bounds induced by each previous MDP.

Theorem 5.6 (Lipschitz upper bound on the optimal Q-function)**.** *Given a partially known task $M = (T, R) \in \mathcal{M}$ with the set of state-action pairs K for which the model $\hat{M} = (\hat{T}, \hat{R})$ of M is an ϵ-accurate estimate of (T, R) in $\mathcal{L}_1$-norm with probability at least $1-\delta$, $\delta \in (0,1]$. Given the set of Lipschitz bounds on Q^*_M induced by previous tasks $\left\{U_{\bar{M}_1}, \dots, U_{\bar{M}_m}\right\}$, the function Q defined below is an upper bound on Q^*_M for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ with probability at least $1-\delta$.*

$$Q(s,a) \triangleq \begin{cases} \hat{R}^a_s + \gamma \sum\limits_{s' \in \mathcal{S}} \hat{T}^a_{ss'} \max\limits_{a' \in \mathcal{A}} Q(s',a') + \frac{\epsilon}{(1-\gamma)^2} & \text{if } (s,a) \in K, \\ U(s,a) & \text{otherwise,} \end{cases} \tag{5.11}$$

with $U(s,a) = \min\left\{\frac{1}{1-\gamma}, U_{\bar{M}_1}(s,a), \dots, U_{\bar{M}_m}(s,a)\right\}$.

The proof of Theorem 5.6 is reported in the Appendix, Chapter A, Section A.4. Traditionally in RMAX, Equation 5.11 is solved to a precision ϵ_Q using an algorithm such as value iteration (see Algorithm 1, page 20). This yields a function Q that is a valid heuristic, *i.e.*, a provable upper bound on Q^*_M, for the exploration of the MDP M. We defined the Lipschitz upper bound on the optimal Q-function in Theorem 5.6 to accelerate learning. However, it is not directly computable using value iteration as the Lipschitz bounds induced by source MDPs require some knowledge about models of the source and target task that may be unavailable. We provide a solution to this issue in the next section.

5.4.2 A computable upper bound on the optimal Q-function

The key issue addressed in this section is how to practically compute the value of the upper bound $U(s,a)$ presented in Theorem 5.6. More precisely: how to compute the values of the induced Lipschitz bounds? Consider two tasks $M = (T, R)$ and $\bar{M} = (\bar{T}, \bar{R})$, on which vanilla RMAX has been applied, yielding the respective sets of known state-action pairs K and $\bar{K}$, along with the learned models $\hat{M} = (\hat{T}, \hat{R})$ and $\hat{\bar{M}} = (\hat{\bar{T}}, \hat{\bar{R}})$, and the upper bounds Q and $\bar{Q}$ respectively on Q^*_M and $Q^*_{\bar{M}}$.

Remark 5.1. Notice that, if $K = \emptyset$, *i.e.*, no state-action pair is known, then $Q(s,a) = \frac{1}{1-\gamma}$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. Conversely, if $K^c = \emptyset$, *i.e.*, all the state-action pairs are known, then Q is an ϵ-accurate estimate of Q^*_M in $\mathcal{L}_1$-norm with high probability.

Equation 5.11 of Theorem 5.6 allows the transfer of knowledge from $\bar{M}$ to M if the induced Lipschitz bound $U_{\bar{M}}(s,a)$ can be computed for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. Unfortunately, the true optimal value functions, transition and reward models, necessary to compute $U_{\bar{M}}$, are unknown (see Equation 5.10, page 105). Thus, we propose to compute a looser upper bound based on the learned models and value functions. The proposed idea is to make use of the available knowledge on the optimal value function and the model, and to use upper bounding quantities whenever no knowledge is available. This allows not to underestimate the resulting upper bound, which is a necessary requirement to avoid negative transfer.

First, we provide an upper bound $\hat{D}_{sa}(M \| \bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ on the pseudometric between models M and $\bar{M}$ in Theorem 5.7. Secondly, we provide an upper bound $\hat{d}_{sa}(M \| \bar{M})$ on the local MDP dissimilarity $d_{sa}(M \| \bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ in Theorem 5.8. In turn, using the two results yields a closed-form expression of an upper bound on the induced Lipschitz bound of Equation 5.10, page 105.

Theorem 5.7. *Given two tasks $M, \bar{M} \in \mathcal{M}$, K and $\bar{K}$ the respective sets of state-action pairs where their models are known with accuracy ϵ in $\mathcal{L}_1$-norm with probability at least $1-\delta$, where $\delta \in (0,1]$. We define the* upper bound on the pseudometric between models *$\hat{D}_{sa}(M \| \bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ by*

$$\hat{D}_{sa}(M \| \bar{M}) \triangleq \begin{cases} D_{sa}^{\gamma \bar{V}}(\hat{M}, \hat{\bar{M}}) + 2B & \text{if } (s,a) \in K \cap \bar{K} \\ \max\limits_{\bar{m} \in \mathcal{M}} D_{sa}^{\gamma \bar{V}}(\hat{M}, \bar{m}) + B & \text{if } (s,a) \in K \cap \bar{K}^c \\ \max\limits_{m \in \mathcal{M}} D_{sa}^{\gamma \bar{V}}(m, \hat{\bar{M}}) + B & \text{if } (s,a) \in K^c \cap \bar{K} \\ \max\limits_{(m, \bar{m}) \in \mathcal{M}^2} D_{sa}^{\gamma \bar{V}}(m, \bar{m}) & \text{if } (s,a) \in K^c \cap \bar{K}^c \end{cases} \tag{5.12}$$

where $B = \epsilon \left(1 + \gamma \max_{s' \in \mathcal{S}} \bar{V}(s')\right)$. For all $(s,a) \in \mathcal{S} \times \mathcal{A}$, $\hat{D}_{sa}(M \| \bar{M})$ is an upper bound on $D_{sa}^{\gamma V^*_{\bar{M}}}(M, \bar{M})$ with high probability, i.e.,

$$\boldsymbol{Pr}\left(\hat{D}_{sa}(M \| \bar{M}) \geq D_{sa}^{\gamma V^*_{\bar{M}}}(M, \bar{M})\right) \geq 1 - \delta\,.$$

The proof of Theorem 5.7 is reported in the Appendix, Chapter A, Section A.4.

Remark 5.2. The assumption of Theorem 5.7 that the models are known with probability at least $1-\delta$ with accuracy ϵ in $\mathcal{L}_1$-norm can be translated mathematically as

$$\mathbf{Pr}\left(\begin{array}{lll} \left|R^a_s - \hat{R}^a_s\right| & \leq \epsilon, & \forall(s,a) \in K \quad \text{and} \\ \left\|T^a_{ss'} - \hat{T}^a_{ss'}\right\|_1 & \leq \epsilon, & \forall(s,a) \in K \quad \text{and} \\ \left|\bar{R}^a_s - \hat{\bar{R}}^a_s\right| & \leq \epsilon, & \forall(s,a) \in \bar{K} \quad \text{and} \\ \left\|\bar{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'}\right\|_1 & \leq \epsilon, & \forall(s,a) \in \bar{K} \end{array}\right) \leq 1-\delta. \tag{5.13}$$

Importantly, the probabilistic event of Inequality 5.13 is the intersection of at most $4SA$ individual events of estimating either R^a_s, $T^a_{ss'}$, $\bar{R}^a_s$ or $\bar{T}^a_{ss'}$ with precision ϵ. Each one of those individual events is itself true with probability at least $1-\delta'$, where $\delta' \in (0,1]$ is a parameter, as described in Theorem 2.8, page 30. For *all* the individual events to be true at the same time, *i.e.* for Inequality 5.13 to be verified, one must apply Boole's inequality and set $\delta' = \delta/(4SA)$ to ensure a total probability — *i.e.*, probability of the intersection of *all* the individual events — of at least $1-\delta$.

The magnitude of the B term is controlled by ϵ. In the case where no information is available on the maximum value of $\bar{V}$, we have that $B = \frac{\epsilon}{1-\gamma}$. ϵ measures the accuracy with which the tasks are known: the smaller ϵ, the smaller B. Note that $\bar{V}$ is used as an upper bound on the true optimal value function $V^*_{\bar{M}}$. In many cases, $\max_{s' \in \mathcal{S}} V^*_{\bar{M}}(s') \leq \frac{1}{1-\gamma}$; for instance for stochastic shortest path problems, which feature rewards only upon reaching terminal states, we have that $\max_{s' \in \mathcal{S}} V^*_{\bar{M}}(s') = 1$ and thus one can take $B = \epsilon(1+\gamma)$ that provides a tighter bound for transfer.

Importantly, the upper bound on the pseudometric between models introduced in Theorem 5.7 can be computed analytically. We here propose a closed-form expression to compute this upper bound. Consider two tasks $M = (T, R)$ and $\bar{M} = (\bar{T}, \bar{R})$, with K and $\bar{K}$ the respective sets of state-action pairs where their learned models $\hat{M} = (\hat{T}, \hat{R})$ and $\hat{\bar{M}} = (\hat{\bar{T}}, \hat{\bar{R}})$ are known with accuracy ϵ in $\mathcal{L}_1$-norm with probability at least $1-\delta$. We denote by $V_{\max}$, a known upper bound on the maximum achievable value. In the worst case where one does not have any information on the value of $V_{\max}$, setting $V_{\max} = \frac{1}{1-\gamma}$ is a valid upper bound. We detail the computation of $\hat{D}_{sa}(M \| \bar{M})$ for each cases: 1) $(s,a) \in K \cap \bar{K}$, 2) $(s,a) \in K \cap \bar{K}^c$, and 3) $(s,a) \in K^c \cap \bar{K}^c$. The case $(s,a) \in K^c \cap \bar{K}$ being the symmetric of case 2), the same calculations apply.

1) If $(s,a) \in K \cap \bar{K}$, we have

$$\begin{aligned} \hat{D}_{sa}(M \| \bar{M}) &= D^{\gamma \bar{V}}_{sa}(\hat{M}, \hat{\bar{M}}) + 2B \\ &= \left|\hat{R}^a_s - \hat{\bar{R}}^a_s\right| + \gamma \sum_{s' \in \mathcal{S}} \bar{V}(s') \left|\hat{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'}\right| + 2\epsilon \left(1 + \gamma \max_{s' \in \mathcal{S}} \bar{V}(s')\right). \end{aligned}$$

Since (s,a) is a known state-action pair, everything is known and computable in this last equation. Note that $\max_{s' \in \mathcal{S}} \bar{V}(s')$ can be tracked along the updates of $\bar{V}$ and thus its computation does not induce any additional computational complexity.

2) If $(s,a) \in K \cap \bar{K}^c$, we have

$$\begin{aligned}
\hat{D}_{sa}(M\|\bar{M}) &= \max_{\bar{m}\in\mathcal{M}} D_{sa}^{\gamma\bar{V}}(\hat{M},\bar{m}) + B \\
&= \max_{\bar{R}^a_s, \bar{T}^a_{ss'}} \left(\left|\hat{R}^a_s - \bar{R}^a_s\right| + \gamma \sum_{s'\in\mathcal{S}} \bar{V}(s') \left|\hat{T}^a_{ss'} - \bar{T}^a_{ss'}\right| \right) + \epsilon \left(1 + \gamma \max_{s'\in\mathcal{S}} \bar{V}(s')\right), \\
&= \max_{r\in[0,1]} \left|\hat{R}^a_s - r\right| + \gamma \max_{\substack{t\in[0,1]^S \\ \text{s.t. } \sum_{s'\in\mathcal{S}} t_{s'}=1}} \left(\sum_{s'\in\mathcal{S}} \bar{V}(s') \left|\hat{T}^a_{ss'} - t_{s'}\right| \right) + \epsilon \left(1 + \gamma \max_{s'\in\mathcal{S}} \bar{V}(s')\right).
\end{aligned}$$

First, we have

$$\max_{r\in[0,1]} \left|\hat{R}^a_s - r\right| = \max\left\{\hat{R}^a_s, 1 - \hat{R}^a_s\right\}.$$

Maximizing over the variable $t \in [0,1]^S$ such that $\sum_{s'\in\mathcal{S}} t_{s'} = 1$ is equivalent to maximizing a convex combination of the positive vector $\bar{V}$ whose terms are not independent as they must sum to one. This is easily solvable as a linear programming problem. A straightforward (simplex-like) resolution procedure consists in progressively adding mass on the terms that will maximize the convex combination as follows:

- $t_{s'} = 0, \forall s' \in \mathcal{S}$
- l = Sort states by decreasing values of $\bar{V}$
- While $\sum_{s\in\mathcal{S}} t_s < 1$
  - s' = pop first state in l
  - Assign $t_{s'} \leftarrow \arg\max_{t\in[0,1]} \left|\hat{T}^a_{ss'} - t\right|$ to s' (note that $t_{s'} \in \{0,1\}$)
  - If $\sum_{s\in\mathcal{S}} t_s > 1$, then $t_{s'} \leftarrow 1 - \sum_{s\in\mathcal{S}\setminus s'} t(s)$

This allows calculating the maximum over transition models.

Notice that there is a simpler computation that almost always yields the same result (when it does not, it provides an upper bound) and does not require the burden of the previous procedure. Consider the subset of states for which $\bar{V}(s') = \max_{s\in\mathcal{S}} \bar{V}(s)$ (often these are states in $\bar{K}^c$). Among those states, let us suppose there exists s^+, unreachable from (s,a), according to $\hat{T}$, *i.e.*, $\hat{T}^a_{ss^+} = 0$. If $\bar{M}$ has not been fully explored, as is often the case in RMAX, there may be many such states. Then the distribution t with all its mass on s^+ maximizes the $\max_{t\in[0,1]^S}$ term. Conversely, if such a state does not exist (that is, if for all such states $\hat{T}^a_{ss^+} > 0$), then $\max_{s\in\mathcal{S}} \bar{V}(s)$ is an upper bound on the $\max_{t\in[0,1]^S}$ term. Therefore:

$$\max_{t\in[0,1]^S} \left(\sum_{s'\in\mathcal{S}} \bar{V}(s') \left|\hat{T}^a_{ss'} - t_{s'}\right| \right) \leq \max_{s\in\mathcal{S}} \bar{V}(s),$$

with equality in many cases.

3) If $(s,a) \in K^c \cap \bar{K}^c$, the resolution is trivial and we have

$$\begin{aligned}
\hat{D}_{sa}(M \| \bar{M}) &= \max_{m,\bar{m} \in \mathcal{M}^2} D_{sa}^{\gamma \bar{V}}(m, \bar{m}) \\
&= \max_{R_s^a, T_{ss'}^a, \bar{R}_s^a, \bar{T}_{ss'}^a} \left(\left| R_s^a - \bar{R}_s^a \right| + \gamma \sum_{s' \in \mathcal{S}} \bar{V}(s') \left| T_{ss'}^a - \bar{T}_{ss'}^a \right| \right) \\
&= \max_{r,\bar{r} \in [0,1]} |r - \bar{r}| + \gamma \max_{\substack{t,\bar{t} \in [0,1]^S \\ \text{s.t. } \sum_{s \in \mathcal{S}} t_s = 1 \\ \text{and } \sum_{s \in \mathcal{S}} \bar{t}_s = 1}} \sum_{s' \in \mathcal{S}} \bar{V}(s') \left| t_{s'} - \bar{t}_{s'} \right| \\
&= 1 + 2\gamma \max_{s \in \mathcal{S}} \bar{V}(s) \,.
\end{aligned}$$

Overall, computing the value of the provided upper bound in the three cases allows to compute $\hat{D}_{sa}(M \| \bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.

We derived a way to compute an upper bound on the pseudometric between models. Using $\hat{D}_{sa}(M \| \bar{M})$ in Equation 5.7, page 101, we now provide an upper bound $\hat{d}_{sa}(M \| \bar{M})$ on the local MDPs pseudometric $d_{sa}(M \| \bar{M})$ for any two MDPs $M, \bar{M} \in \mathcal{M}$ and any state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ in the following theorem.

Theorem 5.8. *Given two tasks $M, \bar{M} \in \mathcal{M}$, K the set of state-action pairs for which (R, T) is known with accuracy ϵ in $\mathcal{L}_1$-norm with probability at least $1-\delta$. If the condition $\gamma(1+\epsilon) < 1$ is met, the solution $\hat{d}_{sa}(M \| \bar{M})$ of the following fixed-point equation on $\hat{d} \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$ is an upper bound on $d_{sa}(M \| \bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ with probability at least $1-\delta$:*

$$\hat{d}_{sa} = \begin{cases} \hat{D}_{sa}(M \| \bar{M}) + \gamma \left(\sum_{s' \in \mathcal{S}} \hat{T}_{ss'}^a \max_{a' \in \mathcal{A}} \hat{d}_{s'a'} + \epsilon \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'} \right) & \text{if } (s,a) \in K, \\ \hat{D}_{sa}(M \| \bar{M}) + \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'} & \text{otherwise.} \end{cases} \tag{5.14}$$

Similarly as Theorem 5.7, page 107, Theorem 5.8 requires the knowledge of an ϵ-accurate estimate in $\mathcal{L}_1$-norm for both MDP models with high probability on a subset of $\mathcal{S} \times \mathcal{A}$. In other words, Inequality 5.13, page 108 must be verified, which implies learning an ϵ-accurate estimate of each reward or transition function with probability at least $1 - \delta/(4SA)$ to ensure a total probability of at least $1-\delta$. See Remark 5.2, page 108, for more details. Theorem 5.8 requires a fixed-point equation to admit a unique solution. This result is stated in the following lemma.

Lemma 5.4. *Given two tasks $M, \bar{M} \in \mathcal{M}$, K the set of state-action pairs for which (R, T) is known with accuracy ϵ in $\mathcal{L}_1$-norm with probability at least $1-\delta$. If $\gamma(1+\epsilon) < 1$, this equation on $\hat{d} \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$ is a fixed-point equation admitting a unique solution:*

$$\hat{d}_{sa} = \begin{cases} \hat{D}_{sa}(M \| \bar{M}) + \gamma \left(\sum_{s' \in \mathcal{S}} \hat{T}_{ss'}^a \max_{a' \in \mathcal{A}} \hat{d}_{s'a'} + \epsilon \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'} \right) & \text{if } (s,a) \in K, \\ \hat{D}_{sa}(M \| \bar{M}) + \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'} & \text{otherwise.} \end{cases}$$

We refer to this unique solution as $\hat{d}_{sa}(M \| \bar{M})$*.*

The proofs of Lemma 5.4 and Theorem 5.8 are reported in the Appendix, Chapter A, Section A.4. Similarly as in Theorem 5.7, page 107, the condition $\gamma(1+\epsilon) < 1$ illustrates the fact that for a large return horizon (*i.e.*, a large value of γ) a high accuracy (*i.e.*, a small value of ϵ) is needed for the bound to be computable.

Finally, a computable Lipschitz upper bound on Q^*_M induced by $Q^*_{\bar{M}}$ with high probability is given by

$$\hat{U}_{\bar{M}}(s,a) = \bar{Q}(s,a) + \min\left\{\hat{d}_{sa}(M \| \bar{M}), \hat{d}_{sa}(\bar{M} \| M)\right\} . \tag{5.15}$$

And the associated upper bound on $U(s,a)$ (Equation 5.11, page 106) given the source tasks $\bar{\mathcal{M}} = \{\bar{M}_i\}_{i=1}^m$ is given for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ by

$$\hat{U}(s,a) = \min\left\{\tfrac{1}{1-\gamma}, \hat{U}_{\bar{M}_1}(s,a), \ldots, \hat{U}_{\bar{M}_m}(s,a)\right\} . \tag{5.16}$$

Remark 5.3. In Remark 5.2, page 108, we noticed the requirement to estimate each individual reward or transition function with precision ϵ in $\mathcal{L}_1$-norm with probability at least $1-\delta'$, with $\delta' = \delta/(4SA)$. This ensures a total probability of estimating all the individual models correctly *at the same time* of at least $1-\delta$. This fact is true for two tasks, *i.e.*, in the case $m=1$, where only one source task has been sampled and learned. For the general case of m source tasks, one should set $\delta' = \frac{\delta}{2(m+1)SA}$ for Equation 5.16 to be an upper bound on $U(s,a)$ with probability at least $1-\delta$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.

We now have a way to practically compute a surrogate upper bound on the induced Lipschitz bounds by the optimal Q-function of source MDPs. By combining this surrogate with Theorem 5.6, page 106, we can deduce the upper bound on the optimal Q-function of a partially known target task induced by *all* the source tasks. In turn, is the resulting bound is smaller than $\frac{1}{1-\gamma}$, the performance of an RMAX-like algorithm should be increases by benefiting from a tighter heuristic function. In the next section, we propose an algorithm employing this method in the online lifelong RL setting.

5.5 The Lipschitz RMAX algorithm

In this section, we define an algorithm for lifelong RL applying the transfer method described in Section 5.4, page 105. First we describe the algorithm, then we detail potential improvements.

5.5.1 Definition of LRMAX

In lifelong RL, MDPs are encountered sequentially. Applying RMAX to task M yields the set of known state-action pairs K, the learned models $(\hat{T}, \hat{R})$, and the upper bound Q on Q^*_M. Saving this information when the task changes allows to compute the upper bound of

Equation 5.16 for a new target task, and to use it to shrink the optimistic heuristic of RMAX. This effectively transfers value functions between tasks based on task similarity. As the new target task is explored, the task similarity is assessed with better confidence, refining the values of $\hat{D}_{sa}(M\|\bar{M})$, $\hat{d}_{sa}(M\|\hat{M})$ and finally $\hat{U}(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$, allowing for more efficient transfer where the task similarity is appraised. The resulting algorithm, Lipschitz R-Max (LRMAX), is presented in Algorithm 10, page 113. To avoid ambiguities with $\bar{\mathcal{M}}$, we use the notation $\hat{\mathcal{M}}$, denoting the learned features ($\hat{T}$, $\hat{R}$, K and Q) about previously seen MDPs.

LRMAX requires a few parameters to control the precision on various estimated quantities. First, the parameter δ corresponds to the probability for the PAC-MDP results to hold. More precisely, Theorem 5.6, page 106, 5.7, page 107, and 5.8, page 110, onto which LRMAX is based, are true with probability $1 - \delta$. Consequently, this is an additional input parameter that quantifies the degree of uncertainty one can have on the policy of LRMAX. Secondly, similarly to RMAX, ϵ_M indicates the precision with which the empirical model estimates the true model of the current MDP. In Chapter 2, Section 2.3.2, page 28, we saw in Theorem 2.8, page 30 that a precision ϵ_M is reached if the minimum number of samples n_{known} to estimate a model is larger than

$$\left\lceil \frac{2\left(\ln(2^S - 2) - \ln(\delta)]\right)}{\epsilon_M^2} \right\rceil .$$

Consequently, n_{known} should be deduced from the input parameter ϵ_M. Lastly, the parameter ϵ_Q quantifies the error made by the value iteration algorithm — presented in Chapter 2, Section 2.2.2, page 18 — on the estimated Q-function. Indeed, we choose to use the value iteration algorithm as the DP method to solve Equations 5.14, page 110 and 5.11, page 106 in the LRMAX procedure. This choice is motivated by the simplicity of value iteration and the control over the approximated optimal Q-function it provides. Any other algorithm could be used in place. The ϵ_Q parameter combined with the ϵ_M parameter should be taken into account to estimate the total quality of the learned policy — *i.e.*, how close is its value function to the true optimal value function — after convergence of Algorithm 10, page 113.

Overall, the behavior of LRMAX on a given task $M \in \mathcal{M}$ is precisely that of RMAX, but with a tighter admissible heuristic $\hat{U}$ that becomes better as the new task is explored (while this heuristic remains constant, equal to $\frac{1}{1-\gamma}$, in vanilla RMAX). LRMAX meets Condition C1, *i.e.*, is PAC-MDP, as stated in Theorems 5.9 and 5.10, page 114. The sample complexity of vanilla RMAX is

$$\tilde{\mathcal{O}}\left(\frac{S^2 A}{\epsilon^3 (1-\gamma)^3}\right) ,$$

as stated in Chapter 2, Theorem 2.9, page 32. This sample complexity is improved by LRMAX in Theorem 5.9, page 114 which meets Condition C2. The upper bound $\hat{U}$ used by LRMAX is provably overestimating Q_M^* with high probability as stated in Theorem 5.6, page 106, which avoids negative transfer and meets Condition C3. As shown in Theorem 5.9, page 114, the sample complexity of LRMAX is no worse than that of RMAX, meaning that in the worst case, LRMAX achieves the same level of performance as RMAX. As discussed in Section 5.2.2, page 97, this last fact is equivalent to a provably non-negative transfer method.

Algorithm 10 Lipschitz RMAX algorithm

```
**Context:** state-action space $\mathcal{S} \times \mathcal{A}$; distribution $\mathcal{D}$ over $(T, R, H, \mathcal{P}_0)$.
**Input:** $\delta$ parameter for the high probability results; precision $\epsilon_M$ with which the models are learned allowing to deduce $n_{\text{known}}$; precision $\epsilon_Q$ of value iteration on the estimated Q-function; discount factor $\gamma$.
Initialize $\hat{\mathcal{M}} \leftarrow \emptyset$.
**for** each sampled MDP $(T, R, H, \mathcal{P}_0) \sim \mathcal{D}$ **do**
  Initialize $Q(s,a) \leftarrow \frac{1}{1-\gamma}, \forall (s,a) \in \mathcal{S} \times \mathcal{A}$, and $K \leftarrow \emptyset$
  Initialize $\hat{T}$ and $\hat{R}$ # RMAX optimistic model (see Chapter 2, Section 2.3.2, page 28).
  $Q \leftarrow$ UpdateQ$(\hat{\mathcal{M}}, \hat{T}, \hat{R})$
  $s \sim \mathcal{P}_0$ # Set the initial state.
  **for** $t \in \{1, \dots, H\}$ **do**
    $a \leftarrow \text{argmax}_{a' \in \mathcal{A}} Q(s, a')$ # Act greedily with respect to the current Q-function upper bound.
    $s' \sim T_{s\cdot}^a$ # Sample the next state.
    $r \leftarrow R_s^a$ # Sample the reward.
    $n(s,a) \leftarrow n(s,a) + 1$ # Increment the visitations counter for $(s,a)$.
    **if** $n(s,a) < n_{\text{known}}$ **then**
      Store $(s, a, r, s')$ # Memorize the transition sample for a future model update.
    **end if**
    **if** $n(s,a) = n_{\text{known}}$ **then**
      Update $K$, $\hat{T}_{ss'}^a$ and $\hat{R}_s^a$ # Override the optimistic model with the learned model.
      $Q \leftarrow$ UpdateQ$(\hat{\mathcal{M}}, \hat{T}, \hat{R})$ # Update the upper bound with the new learned model.
    **end if**
  **end for**
  Save $\hat{M} \leftarrow (\hat{T}, \hat{R}, K, Q)$ in $\hat{\mathcal{M}}$ # Save the target task as a source task.
**end for**

**Function** UpdateQ:
**Input:** learned model of the current target MDP $(\hat{T}, \hat{R})$; set of source MDPs $\hat{\mathcal{M}}$.
**for** $\bar{M} \in \hat{\mathcal{M}}$ **do**
  # Compute the induced Lipschitz upper bound for each source MDP.
  Compute $\hat{D}_{sa}(M \| \bar{M})$ and $\hat{D}_{sa}(\bar{M} \| M)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ (Equation 5.12, page 107)
  Compute $\hat{d}_{sa}(M \| \bar{M})$ and $\hat{d}_{sa}(\bar{M} \| M)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ (DP on Equation 5.14, page 110)
  Compute $\hat{U}_{\bar{M}}$ (Equation 5.15, page 111)
**end for**
# Compute the total upper bound.
Compute $\hat{U}$ (Equation 5.16, page 111)
Compute and return $Q$ (DP on Equation 5.11, page 106 using $\hat{U}$)
```

Theorem 5.9 (Sample complexity of Lipschitz RMAX (Strehl, Li, and Littman, 2009))**.** *With probability $1-\delta$, $\delta \in (0,1]$, the greedy policy with respect to Q computed by LRMAX achieves an ϵ-optimal return in MDP M for all but (when logarithmic factors are ignored)*

$$\tilde{\mathcal{O}}\left(\frac{S\left|\left\{(s,a) \in \mathcal{S} \times \mathcal{A} \mid \hat{U}(s,a) \geq V^*_M(s) - \epsilon\right\}\right|}{\epsilon^3(1-\gamma)^3}\right)$$

time steps, with $\hat{U}$ defined in Equation 5.16, page 111 a non-static, decreasing quantity, upper bounded by $\frac{1}{1-\gamma}$.

Theorem 5.10 (Computational complexity of Lipschitz RMAX)**.** *The total computational complexity of Lipschitz RMAX is*

$$\tilde{\mathcal{O}}\left(\tau + \frac{S^3A^2N}{(1-\gamma)} \ln\left(\frac{1}{\epsilon_Q(1-\gamma)}\right)\right)$$

with τ the number of time steps or decision epochs, ϵ_Q the precision of value iteration and N the number of source tasks.

The proof of Theorem 5.10 is reported in the Appendix, Chapter A, Section A.4.

5.5.2 Refining the LRMAX bound with the maximum model distance

LRMAX relies on upper bounds on the local distances between tasks defined in Equation 5.14, page 110. The quality of the Lipschitz bound on the optimal Q-function Q^*_M of a target task $M \in \mathcal{M}$ greatly depends on the quality of those estimates and can be improved accordingly. We discuss two methods to provide finer estimates.

Refining with prior knowledge. First, from Definition 5.1, page 100, the pseudometric between models can be shown to be a bounded quantity as stated in the following result.

Theorem 5.11 (Upper bound on the pseudometric between models)**.** *For any two MDPs $\left(M, \bar{M}\right) \in \mathcal{M}^2$, at any state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, we have the following upper bound on the pseudometric between models:*

$$D_{sa}(M \| \bar{M}) \leq \frac{1+\gamma}{1-\gamma} .$$

The proof of Theorem 5.11 is reported in the Appendix, Chapter A, Section A.4. The quantity $\frac{1+\gamma}{1-\gamma}$ consists in a upper bound on the pseudo-distance between *any* two MDPs. However, in practice, the tasks experienced in lifelong RL might not cover the full span of possible MDPs and therefore may be systematically closer to each other than $\frac{1+\gamma}{1-\gamma}$. For instance, the pseudo-distances between variations of the Breakout video game are much smaller.

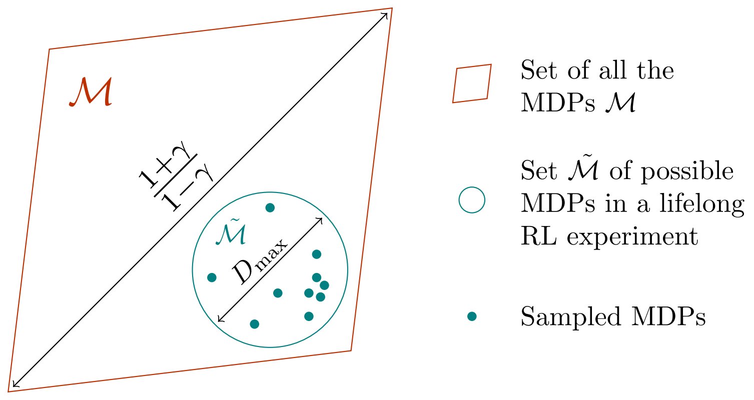

$\mathcal{M}$
$\frac{1+\gamma}{1-\gamma}$
$\tilde{\mathcal{M}}$
$D_{\max}$
Set of all the MDPs $\mathcal{M}$
Set $\tilde{\mathcal{M}}$ of possible MDPs in a lifelong RL experiment
Sampled MDPs

Figure 5.3: Illustration of the prior knowledge on the maximum pseudo-distance between models. The sets of MDPs are represented in 2D for the sake of illustration. The sampled MDPs denote tasks sampled in the lifelong RL experiment. The maximum pseudo-distance $\frac{1-\gamma}{1+\gamma}$ between two MDPs of the complete set of MDPs $\mathcal{M}$ is potentially larger than the maximum pseudo-distance $D_{\max}$ between two MDPs of the lifelong RL experiment $\tilde{\mathcal{M}}$.

In the same way, the pseudo-distance between two games in the Arcade Learning Environment (ALE) (Bellemare, Naddaf, Veness, and Bowling, 2013) — which is a common reference benchmark in the RL community — is also smaller than the maximum distance $\frac{1+\gamma}{1-\gamma}$ between any two MDPs defined on the common state-action space of the ALE. Let us denote by $\tilde{\mathcal{M}} \subset \mathcal{M}$ the set of possible MDPs for a particular lifelong RL experiment. We define the *maximum model distance* at $(s, a) \in \mathcal{S} \times \mathcal{A}$ by

$$D_{\max}(s, a) \triangleq \max_{M, \bar{M} \in \tilde{\mathcal{M}}^2} D_{sa}(M \| \bar{M}) \,.$$

This quantity is the maximum possible pseudo-distance between models for the same state-action pair in the complete set $\tilde{\mathcal{M}}$ of the possible tasks. Obviously, $D_{\max}(s, a)$ is also upper bounded by $\frac{1+\gamma}{1-\gamma}$. However, *prior knowledge* might indicate a smaller upper bound. We will write such an upper bound $D_{\max}$, considered valid for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ pairs, *i.e.*, such that

$$D_{\max} \geq \max_{s,a,M,\bar{M} \in \mathcal{S} \times \mathcal{A} \times \tilde{\mathcal{M}}^2} \Big(D_{sa}(M \| \bar{M}) \Big) \,.$$

In a lifelong RL experiment, $D_{\max}$ should be seen as a rough estimate of the maximum model discrepancy an agent may encounter, regardless of the state-action pair. Figure 5.3 illustrates the relative importance of $D_{\max}$ *vs.* $\frac{1+\gamma}{1-\gamma}$.

We saw that the transfer method used by LRMAX requires computing the upper bound described in Equation 5.14, page 110 on the local MDP dissimilarity. Solving this equation boils down to accumulating $\hat{D}_{sa}(M \| \bar{M})$ values in $\hat{d}_{sa}(M \| \bar{M})$. Reducing an estimate of $\hat{D}_{sa}(M \| \bar{M})$ in a single state-action pair actually tightens the upper bound $\hat{d}_{sa}(M \| \bar{M})$ in *all* the state-action pairs. Thus, replacing $\hat{D}_{sa}(M \| \bar{M})$ in Equation 5.14, page 110 by $\min\{D_{\max}, \hat{D}_{sa}(M \| \bar{M})\}$, provides a smaller upper bound $\hat{D}_{sa}(M \| \bar{M})$ on $D_{sa}(M \| \bar{M})$, and thus a smaller induced Lipschitz bound $\hat{U}$. Importantly, this last bound allows an efficient transfer only if it is lesser than $\frac{1}{1-\gamma}$. Consequently, such an upper bound $D_{\max}$ can make a difference between successful and unsuccessful transfer, even if its value is of little importance.

Conversely, setting a value for $D_{\max}$ quantifies the distance between the models of the MDPs of the task space. As this value can be linked to the efficiency of transfer, knowing it in advance given a task space $\mathcal{M}$ is a useful way to appraise if the transfer method will be efficient or not.

Refining by learning the maximum distances. Furthermore, one can estimate online the value of $D_{\max}(s,a)$ for each pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, lifting the previous hypothesis of available prior knowledge. Particularly, one can build an empirical estimate of the maximum model distance $D_{\max}(s,a)$ at $(s,a) \in \mathcal{S} \times \mathcal{A}$ with the following quantity:

$$\hat{D}_{\max}(s,a) \triangleq \max_{M,\bar{M} \in \hat{\mathcal{M}}^2} \hat{D}_{sa}(M \| \bar{M}) \,,$$

where $\hat{\mathcal{M}} \subset \tilde{\mathcal{M}}$ is the finite set of sampled tasks, *i.e.*, already explored. In Figure 5.3, page 115, $\hat{\mathcal{M}}$ is the set of the MDPs represented by blue dots. The pitfall being that, with few explored tasks, $\hat{D}_{\max}(s,a)$ could underestimate $D_{\max}(s,a)$ in the case where the maximizing tasks have not been picked. In Theorem 5.12 we provide a lower bound on the probability that $\hat{D}_{\max}(s,a)$ does not underestimate $D_{\max}(s,a)$, depending on the number of sampled tasks. In turn this indicates when $\hat{D}_{\max}(s,a)$ is an upper bound on $D_{\max}(s,a)$ with high probability, which can be combined with Algorithm 10, page 113 to improve the performance by tightening the bound.

Theorem 5.12. *Consider an algorithm producing ϵ-accurate model estimates $\hat{D}_{sa}(M\|\bar{M})$ for a subset K of $\mathcal{S} \times \mathcal{A}$ after interacting with any two MDPs $M, \bar{M} \in \mathcal{M}$. Assume $\hat{D}_{sa}(M\|\bar{M})$ to be an upper bound of $D_{sa}(M\|\bar{M})$ for any $(s,a) \notin K$. Consider $\delta \in (0,1]$. For all $(s,a) \in \mathcal{S} \times \mathcal{A}$, after sampling m tasks in the lifelong RL setting, if m is large enough to verify $2(1-p_{\min})^m - (1-2p_{\min})^m \leq \delta$, then,*

$$\boldsymbol{Pr}\left(\hat{D}_{\max}(s,a) + \epsilon \geq D_{max}(s,a)\right) \geq 1 - \delta \,.$$

The proof of Theorem 5.12 is reported in the Appendix, Chapter A, Section A.4. Notice that, similarly to the setting of the MaxQInit algorithm (Abel, Jinnai, Guo, Konidaris, and Littman, 2018), it is assumed in Theorem 5.12 that $\mathcal{M}$ is a finite set and that each task has a minimum sampling probability $p_{\min}$. This can also be interpreted as a non-adversarial task sampling strategy. Notice also that the result provided in Theorem 5.12 holds for an algorithm producing ϵ-accurate model estimates with probability 1. One should pay attention to the fact that the Lipschitz RMAX algorithm presented in Algorithm 10, page 113, *only* produces ϵ-accurate model estimates with probability at least $1-\delta$, $\delta \in (0,1]$ being a parameter of the algorithm. Hence, to apply the result of Theorem 5.12 to Algorithm 10, one should consider the joint probability of both having ϵ-accurate model estimates on a subset K of $\mathcal{S} \times \mathcal{A}$ *and* over-estimating $D_{\max}(s,a)$ with $\hat{D}_{\max}(s,a) + \epsilon$. This result can be achieved by applying a union bound to both probabilities.

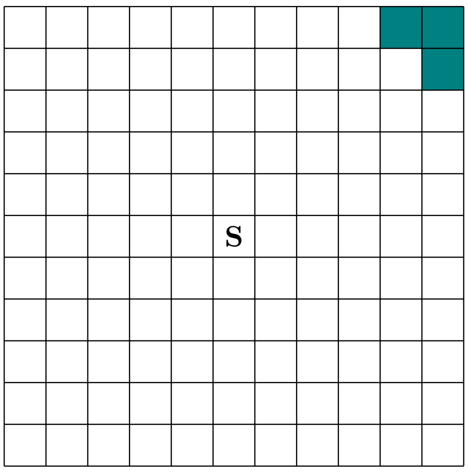


Figure 5.4: The tight grid-world environment.

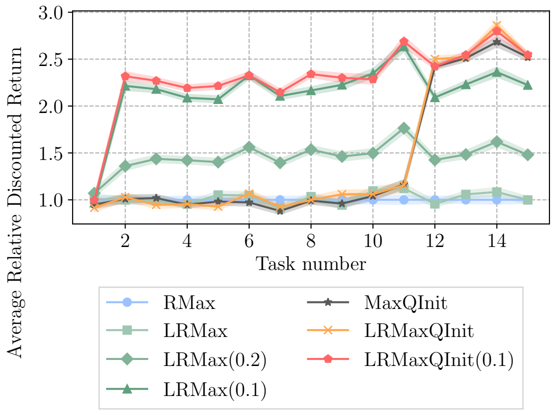
5.6 Experiments

The experiments reported here[1] illustrate how 1) LRMAX allows for early performance increase in lifelong RL by efficiently transferring knowledge between tasks; 2) the Lipschitz bound of Equation 5.15, page 111 improves the sample complexity compared to RMAX by providing a tighter upper bound on Q^*. Graphs are displayed with 95% confidence intervals.

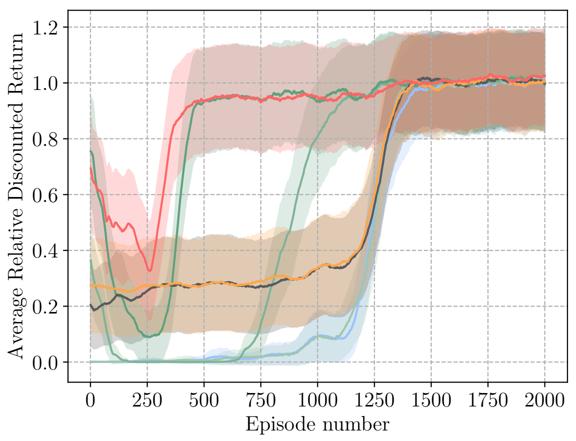
We evaluate different variants of LRMAX in a lifelong RL experiment. The RMAX algorithm will be used as a baseline performing no transfer. It achieves the performance of a model-based PAC-MDP algorithm learning any new task from scratch. LRMAX(x) denotes Algorithm 10, page 113 with prior $D_{\max} = x$. MaxQInit denotes the MaxQInit algorithm from Abel, Jinnai, Guo, Konidaris, and Littman (2018), consisting in a state-of-the art PAC-MDP algorithm achieving transfer in a non adversarial setting with PAC guarantees. Both LRMAX and MaxQInit algorithms achieve value transfer by providing a tighter upper bound on Q^* than $\frac{1}{1-\gamma}$. Taking the minimum over both upper bounds results in combining the two approaches. We include such a combination in our study with the LRMaxQInit algorithm. Similarly, LRMaxQInit(x) consists in the latter algorithm, benefiting from prior knowledge $D_{\max} = x$.

The environment we used in all experiments is a variant of the "tight" environment used by Abel, Jinnai, Guo, Konidaris, and Littman (2018). The tight environment is a 11×11 grid-world illustrated in Figure 5.4. The initial state of the agent is the central cell displayed with an "S". The actions are to move to the adjacent cell in one of the four cardinal directions, *i.e.*, $\mathcal{A} = \{\text{Right, Up, Left, Down}\}$. The reward is 0 everywhere, except for executing an action in one of the three teal cells in the upper-right corner. Each time a task is sampled, a slipping probability of executing another action as the one selected is drawn in $[0, 1]$ and the reward received in each one of the teal cells is picked in $[0.8, 1.0]$. Hence, tasks have different reward

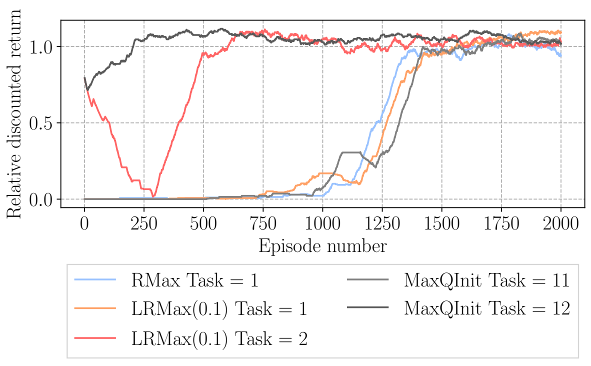
[1]Code available at https://github.com/SuReLI/llrl – For information about the Machine Learning reproducibility checklist, see Appendix, Section E, page 161.



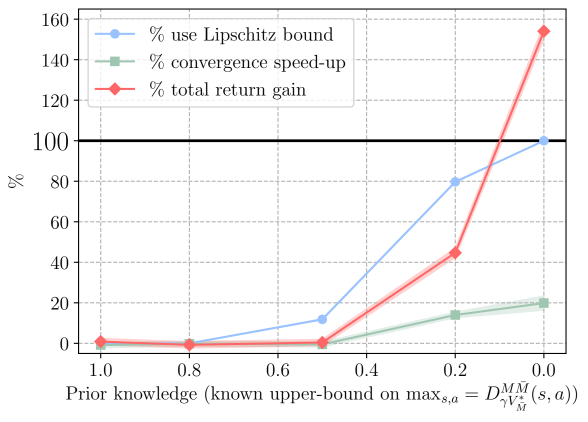
(a) Average discounted return *vs.* tasks



(b) Average discounted return *vs.* episodes



(c) Discounted return for specific tasks



(d) Algorithmic properties *vs.* $D_{\max}$

Figure 5.5: Experimental results of RMAX, LRMAX, MaxQInit and LRMaxQInit in the "tight" lifelong Reinforcement Learning experiment.

and transition functions.

To comply with the requirements of MaxQInit and Theorem 5.12, page 116, a finite set of tasks is pre-sampled and tasks are drawn with the same probability during the lifelong RL experiment. Precisely, we sample 15 tasks in sequence among a pool of 5 possible different sampled tasks. Note, however, that LRMAX does not require the set of MDPs to be finite, which is a noticeable advantage in applicability. Each is run for 2000 episodes of length 10. The complete operation is repeated 10 times to provide narrow confidence intervals. We used $n_{\text{known}} = 10$, $\delta = 0.05$, $\epsilon_M = 0.01$ and $\epsilon_Q = 0.01$. Theoretically, n_{known} should be a lot larger ($\approx 10^5$) in order to reach an accuracy of $\epsilon_M = 0.01$ according to Theorem 2.8, page 30 (Chapter 2). However, it is a common practice to assume such small values of n_{known} are sufficient to reach an acceptable model accuracy ϵ_M (Abel, Jinnai, Guo, Konidaris, and Littman, 2018). Interestingly, empirical validation did not confirm this assumption for any RMAX-based algorithm. We keep these values nonetheless for the sake of comparability between algorithms and consistency with the literature. Despite such absence of accuracy guarantees, RMAX-based algorithms still perform surprisingly well and are robust to model estimation uncertainties. This raises the interesting question of the robustness of RMAX to wrong models.

The results are reported in Figure 5.5, page 118. Figure 5.5a, page 118 displays the discounted return for each task, averaged across episodes. Similarly, Figure 5.5b, page 118 displays the discounted return for each episode, averaged across tasks and follows the same color code as Figure 5.5a, page 118. Figure 5.5c, page 118 displays the discounted return for five specific instances, detailed below. To avoid inter-task disparities, all the aforementioned discounted returns are displayed relatively to an estimator of the optimal expected return for each task. For readability purposes, Figures 5.5b, page 118 and 5.5c, page 118 display a moving average over 100 episodes. Figure 5.5d, page 118 reports the benefits of various values of $D_{\max}$ on the properties of Algorithm 10, page 113.

In Figure 5.5a, page 118, we first observe that LRMAX benefits from the transfer method, as the average discounted return increases as more tasks are experienced. Moreover, this advantage appears as early as the second task. Conversely, the MaxQInit algorithm needs to wait for task 12 before benefiting from transfer. As suggested in Section 5.5.2, page 114, various amounts of prior allow the LRMAX transfer method to be more or less efficient: a smaller known upper bound $D_{\max}$ on $\hat{D}_{sa}(M \| \bar{M})$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ implies a larger discounted return gain. As expected, combining both approaches in the LRMaxQInit algorithm outperforms all other methods. Indeed, this is explained by the fact that taking the minimum of the two upper bound provides the tightest bound.

Episode-wise, we observe in Figure 5.5b, page 118 that the LRMAX transfer method allows for faster convergence, hence decreases the sample complexity. Interestingly, LRMAX features three stages in the learning process.

1. The first episodes are characterized by a direct exploitation of the transferred knowledge, causing these episodes to yield high payoff. This is due to the combined facts that the Lipschitz bound of Equation 5.15, page 111 is larger on promising regions of $\mathcal{S} \times \mathcal{A}$ seen on previous tasks and the fact that LRMAX acts greedily with respect to that bound. Qualitatively, this can be translated by the fact that, given the information of the prior $D_{\max}$ characterizing the similarity between the tasks of the pool, LRMAX effectively exploits this information by first exploring the states that yielded high payoff in previous tasks.

2. This high performance regime is followed by the exploration of unknown regions of $\mathcal{S} \times \mathcal{A}$, in our case yielding low returns. Indeed, as promising regions are explored first, the bound becomes tighter for the corresponding state-action pairs, enough for the Lipschitz bound of unknown pairs to become larger, thus driving the exploration towards low payoff regions in our case. Such regions are quickly identified and never revisited thereafter. This behavior is sound in that it accounts for the fact that LRMAX follows the optimism in the face of uncertainty principle.

3. Eventually, LRMAX stops exploring and converges to the optimal policy.

Importantly, in all experiments, LRMAX never features negative transfer as supported by the provability of the Lipschitz upper bound with high probability. This is indeed demonstrated by the fact that it is at least as efficient in learning as the no-transfer RMAX baseline.

Figure 5.5c, page 118 displays the collected returns of RMAX, LRMAX(0.1), and MaxQInit for specific tasks. We observe that LRMAX benefits from the transfer as early as task 2, where the aforementioned 3-stages behavior is visible. Again, we observe that MaxQInit needs to wait for task 12 to leverage the transfer method. However, the bound it provides are tight enough to allow for almost zero exploration of the task. This supports the fact that LRMaxQInit reaches the best performance by taking the "best of two worlds", *i.e.*, the best bound between the LRMAX transfer method and the MaxQInit transfer method. This echoes back to the discussion of Section 5.2, page 95 where we see in Figure 5.2, page 100 that the two bounds are of different nature and thus not comparable.

In Figure 5.5d, page 118, we display the following quantities for various values of the prior $D_{\max}$:

- ρ_{Lip}, is the ratio of the number of decision epochs where the Lipschitz bound was tighter than the RMAX bound $\frac{1}{1-\gamma}$;
- $\rho_{Speed-up}$, is the relative gain of decision epochs before convergence when comparing LRMAX to RMAX. This quantity is estimated based on the decision epoch of the last updates of the empirical model $\bar{M}$;
- ρ_{Return}, is the relative total return gain on 2000 episodes of LRMAX compared to RMAX.

First, we observe an increase of ρ_{Lip} as $D_{\max}$ becomes tighter. This means that the Lipschitz bound of Equation 5.15, page 111 becomes effectively smaller than $\frac{1}{1-\gamma}$ as the prior increases. This phenomenon leads to faster convergence, indicated by $\rho_{Speed-up}$. Finally, this increased convergence rate allows for a net total return gain, illustrated by the increase of ρ_{Return}.

Overall, in this analysis, we have showed that LRMAX benefits from an enhanced sample complexity thanks to the used value transfer method. The knowledge of a prior $D_{\max}$ further increases this benefit. The method is comparable to the MaxQInit method and has some advantages such as the early fitness for use and the applicability to infinite sets of tasks. No computed upper bound by LRMAX or MaxQInit is uniformly better than the other, making the use of the combination of the two an even better heuristic for an RMAX algorithm. Importantly, the transfer is non-negative while preserving the PAC-MDP guarantees of LRMAX.

5.7 Conclusion

We have studied theoretically the Lipschitz continuity property of the optimal Q-function in the MDP space. This led to a local Lipschitz continuity result, establishing that the distance between the optimal Q-functions of two MDPs at the same state-action pair is upper bounded by a local (state-action dependent) distance between MDPs. This local distance can be computed by dynamic programming. A consequence of this result is a global Lipschitz continuity property of the optimal Q-function in the MDP space, with respect to a pseudometric between MDPs. We then proposed a value transfer method using the local continuity property with the Lipschitz RMAX algorithm, practically implementing this approach in the lifelong RL setting. The algorithm preserves PAC-MDP guarantees, accelerates the learning in subsequent tasks and performs non-negative transfer. Potential improvements of the algorithm were discussed in the form of prior knowledge introduction on the maximum distance between models and online estimation with high probability of this distance. We showcased the algorithm in lifelong RL experiments and demonstrated empirically its ability to accelerate learning. The results also confirm that no negative transfer occurs, regardless of the parameter settings. It should be noted that our approach can directly extend other PAC-MDP algorithms (Szita and Szepesvári, 2010; Rao and Whiteson, 2012; Pazis, Parr, and How, 2016; Dann, Lattimore, and Brunskill, 2017) to the lifelong RL setting.

The content of this chapter suggests a few perspectives. The upper bound computed by the LRMAX algorithm has the advantage to be a *true* upper bound on the optimal Q-function of *any* MDP with high probability. In turn, this allows us to claim that the transfer method is provably non-negative with high probability. However, we did not restrict the study to any sub-class of MDPs, making the set of hypotheses relatively wide. In turn, this makes the computed upper bound very conservative and, possibly, there could exist tighter upper bounds with respect to the particular set of MDPs that can be sampled in particular lifelong RL experiments. Therefore, an interesting perspective would be to *learn* problem-dependent metrics that could be used for transfer, in the same way as the general MDPs local pseudometric we introduced in Theorem 5.1, page 101. Conversely, widening the set of hypotheses is also possible. Precisely, we restricted the study to lifelong RL tasks sharing the same state-action space $\mathcal{S} \times \mathcal{A}$. To what extent is transfer possible or efficient if tasks have different state-action spaces? An answer is partly provided in the study carried out in this chapter, since the LRMAX algorithm can be applied to partially explored tasks. Therefore, the case of a state never encountered before arises in our study. In such a case, a conservative upper bound on the Q-function is provided by LRMAX. However, the case where the state-action spaces do not share the same structure — for instance $\mathcal{S} \equiv \mathbb{R}^2$ for a task and $\mathcal{S} \equiv \mathbb{R}^3$ for another task — remains an open question. The derivation of a Lipschitz upper bound on Q-functions in such a case would be an interesting perspective. A limitation of the LRMAX algorithm is its linear growth in terms of computational complexity with the number of source tasks. To prune such a phenomenon, Mahmud, Hawasly, Rosman, and Ramamoorthy (2013) and Brunskill and Li (2013) proposed clustering techniques to compress the information gathered in the pool of source tasks. Naturally, such a practice could benefit LRMAX as it could help retaining selected information. Indeed, as seen in the experiments of Section 5.6, page 117, LRMAX is able to perform efficient transfer as soon as a single source task has been learned. This suggests that additional knowledge about the same kind of tasks could be of little use to the algorithm and motivates pruning the gathered information.

Overall, the approach developed in this chapter is a similarity-based transfer method for abruptly evolving environments. It should be considered as a different setting from the previous chapter where temporal evolution was assumed to be gradual. Consequently, the approaches resulting from those two different settings feature different characteristics.

Chapter 6

Conclusion

At the beginning of this dissertation, we asked the following question:

How to act in an environment that can change over time?

We proposed an answer through the scope of Reinforcement Learning and considered planning and learning agents acting in Markov Decision Processes (MDP) under different temporal evolution hypotheses. In this conclusion chapter, we first summarize the three axes of contributions of the study, corresponding to Chapters 3, 4 and 5. Then, we conclude on the answer we provided to the initial question and the general perspectives it offers.

Chapter 3 is dedicated to the study of the planning *vs.* re-planning trade-off within stationary MDPs. A method designed to reuse the search tree constructed by a planning agent to prevent re-planning is proposed. The method is suited for both closed-loop and open-loop tree search algorithms. A theoretical study shows the provided guarantees in terms of optimality in both cases. Importantly, related work in the literature lack such a performance guarantees study, which makes the interest of the contribution. As a key feature of the derived algorithms, a so called decision criterion is introduced to balance the trade-off between optimality loss and computational complexity gain. The benefits of the method are demonstrated experimentally. As a perspective, the extension to the Partially Observable Markov Decision Process framework was considered as — similarly to the proposed approach — it comprehends reasoning on state distributions rather than particular states. Also, a formal analysis of the decision criterion was proposed as a way to extend the applicability of the proposed algorithms.

Chapter 4 is dedicated to the study of a planning strategy in the case of gradually evolving MDPs. First, a novel Lipschitz continuity hypothesis of the transition and reward functions with respect to time is introduced to model the assumption of gradual evolution. Secondly, the hypothesis that an instant snapshot of the complete model is available at each decision epoch is introduced. Intuitively, this assumption should be understood as the knowledge of the current transition and reward functions but not their evolutions, which has practical applications. On this basis, the proposed approach is a worst case planning method to derive the optimal policy of the worst case evolution. This results in a risk averse strategy corresponding to the Minimax behavior. The resulting algorithm, called RATS, was formally shown to estimate the optimal policy of the worst case scenario. Finally, the performances of RATS were demonstrated experimentally. As a perspective, we suggested a method to scale the algorithm to larger problems. We also proposed the idea of making the resulting policy less conservative by restricting the set of possible temporal evolutions or learning the set of state-action pairs evolving in an adversarial manner. Finally, we suggested an exact resolution of the risk averse problem as RATS practically computes the solution of a relaxed problem. Quantifying the impact of such a relaxation is also of interest.

Chapter 5 is dedicated to the study of a value transfer method in the lifelong Reinforcement Learning setting. This framework can be seen as a case of abruptly evolving MDPs, as opposed to the setting studied in Chapter 4. The proposed idea is to quantify the distance of the current transition and reward models to already experienced environments. In turn, the knowledge of such a distance allows to derive an upper bound on the optimal Q-function of the current MDP which accelerates learning in an RMAX-like algorithm. Such a transfer method, using similarity measures between tasks, have the appealing feature to quantify the amount of achievable transfer. Further, the transfer method is non-negative as the used upper bound is provably valid with high probability. Noticeably, the presented algorithm, called LRMAX, features PAC-MDP guarantees, which ensures the convergence to the optimal policy in a polynomial number of decision epochs with high probability. As a non-negative, similarity-based, PAC-MDP transfer method, the LRMAX algorithm is the first method of the literature combining those three appealing features. Finally, we studied LRMAX theoretically and experimentally and showed in both cases that it benefits from an improved sample complexity thanks to the transfer method. As a perspective, we proposed to integrate domain-specific knowledge in the form of a learned metric. Such practice could improve the performance of the LRMAX algorithm by providing a tighter upper bound. Then, we asked the question of transfer between tasks featuring different state-action spaces and the derivation of a Lipchitz upper bound in such a case. Lastly, the perspective of pruning the set of source tasks from which to perform transfer was proposed by using clustering methods.

Overall, our study comprehends three parts. First, a study of the planning *vs.* re-planning trade-off in stationary MDPs. Secondly, a study of the risk averse strategy in gradually evolving MDPs. Thirdly, a study of a transfer method in abruptly evolving MDPs. These three sets of hypotheses cover important cases of MDP models featuring different temporal evolutions. Each one of the three aspects was treated by designing a control algorithm adapted to the setting. Systematically, a theoretical analysis was carried out, demonstrating the theoretical guarantees. Additionally, the results were supported with empirical evidences.

Our answer to the question of acting in non-stationary environments starts by categorizing the types of non-stationarities. As shown, different evolution modes imply different solutions to the problem, relying on different assumptions and providing different guarantees. Considering the most general framework that comprehends everything is particularly interesting theoretically but often lacks practical applications. Nonetheless, in this dissertation, we tried to make a compromise by considering less general settings — by studying particular cases of non-stationarity — that still encompass many applications. Hence, our answers in each case do not allow to close the question, but they constitute contributions to the field that we hope to be important. One could argue that the initial question is very general, which motivated the variety of the presented results. Indeed, it is true that each one of the three settings presented in Chapters 3, 4 and 5 corresponds to a sub-field of the Reinforcement Learning problem, to which decades of scientific research are associated. This dissertation allows to put them in perspective from each other and to appreciate better their particularities. Precisely, this allows to highlight what differences make one framework particularly harder than the other and how to address these particular difficulties. In the remainder of this conclusion, we develop on general perspectives.

A recurrent assumption of the dissertation is the consideration of discrete, finite, state-action spaces $\mathcal{S} \times \mathcal{A}$. In most of the applications, particularly in Chapters 4 and 5, this resulted in a bottleneck in the applicability of the algorithms, having a computational complexity at least $\mathcal{O}\left(S^2 A^2\right)$. The same problem arises regarding their sample complexity. Therefore, a natural direction for the research carried out in this dissertation is to make use of some sort of approximations to prune the complexity of our algorithms. For instance, using function approximation for the derived optimal Q-function — as widely done in the literature — could achieve this. Another example would be to make use of state or action abstractions, which amounts to realize a mapping from a "large" MDP to a smaller one in terms of dimensions. On the other hand, working on the exact state-action space with tabular functions allows to conserve theoretical guarantees about the performance of the algorithms. Such guarantees are systematically lost when dealing with function approximations. Achieving to mix approximations *and* theoretical guarantees would probably constitute a major advance in the field. To that end, the answer could be the introduction of a new framework, into which guarantees are seen in a different way than sample or computational complexity. To cope with approximations, one should probably consider relaxed guarantees, maybe probabilistic as done in the PAC-MDP framework. An answer could even be to adopt a different formulation than the original MDP framework.

A more precise perspective that would benefit all the approaches presented in the dissertation is to learn emerging patterns in the observed state-action pairs and their associated models. In Chapter 3, learning what state distribution could motivate a decision criterion to trigger re-planning would allow domain-specific knowledge to improve the planning *vs.* re-planning trade-off. In Chapter 4, learning that some state-action pairs evolve in an adversarial manner or not would be useful to a risk averse planning agent. In Chapter 5, learning that some state-action pairs have invariant models across lifelong RL tasks or conversely that it constantly changes would drastically prune the search thanks to domain-specific knowledge. More generally, those ideas are on the same line of thought as the perspective presented in the last paragraph as it would consist in learning state-action abstractions or representations to allow for more efficient algorithms. Particularly, incorporating learned, domain-specific, knowledge, is a way to improve the range of applicability of an algorithm.

On the same idea of pruning the complexity of an algorithm, an interesting perspective coming from the Neuroscience and the Natural Language Processing fields is the use of features representing real-life concepts to build an abstract state space. More precisely, encoding states in a semantic space yields a simplified representation of an MDP composed of elements relative to known or learned features of a problem. For instance, an agent interacting with a maze could learn the concept of walls and in turn identify them online. This supposes the maintenance of an internal representation of the world that could be used for planning.

Necessarily, this representation would need corrections as it is learned over time and adapted to a specific domain, which raises the question of planning *vs.* acting (Ghallab, Nau, and Traverso, 2014). For instance, predictive coding is a theory from the Neuroscience field, stating that the biological brain generates an abstract model of its sensory input. When this model does not reflect accurately the reality, a prediction error is back-propagated through the network and the model is revised. Such a mechanism has recently proven to represent meaningful features able to capture the structural properties of a problem (Lotter, Kreiman, and Cox, 2016; Spratling, 2017; Boutin, Franciosini, Ruffier, and Perrinet, 2019).

Overall, allowing to scale the approaches presented in this dissertation to a wider range of real life problems is an exciting prospect. The proposed perspectives mainly highlight the need for abstract representations of a problem in order to reach good performances. Future studies should however retain theoretical insights to gain a formal quantification of the capabilities of a learning algorithm.

Appendix A

Proofs of the dissertation

A.1 Proofs of Chapter 2

Proof of Theorem 2.5, page 19. Consider the sequence of distances defined by $\Delta_n = \|V_n - V^*\|_\infty$. We have the following inequality between subsequent elements of this sequence:

$$\begin{aligned}
\Delta_{n+1} &= \max_{s\in\mathcal{S}} |V_{n+1}(s) - V^*(s)| \\
&= \max_{s\in\mathcal{S}} \left| \max_{a\in\mathcal{A}} \left\{ R_s^a + \gamma \sum_{s'\in\mathcal{S}} T_{ss'}^a V_n(s') \right\} - \max_{a\in\mathcal{A}} \left\{ R_s^a + \gamma \sum_{s'\in\mathcal{S}} T_{ss'}^a V^*(s') \right\} \right| \\
&\le \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \left| R_s^a + \gamma \sum_{s'\in\mathcal{S}} T_{ss'}^a V_n(s') - R_s^a + \gamma \sum_{s'\in\mathcal{S}} T_{ss'}^a V^*(s') \right| \\
&\le \gamma \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \left\{ \sum_{s'\in\mathcal{S}} T_{ss'}^a \left| V_n(s') - V^*(s') \right| \right\} \\
&\le \gamma \Delta_n.
\end{aligned}$$

Since $\Delta_0 \le \frac{R_{\max}}{1-\gamma}$, we have that $\Delta_n \le \frac{\gamma^n R_{\max}}{1-\gamma}$. Thus, if n is such that $\frac{\gamma^n R_{\max}}{1-\gamma} \le \epsilon$, then $\Delta_n \le \epsilon$. Equivalently, if $n \ge \ln\left(\frac{\epsilon(1-\gamma)}{R_{\max}}\right)(\ln(\gamma))^{-1}$, then $\Delta_n \le \epsilon$, which concludes the proof. $\square$

Proof of Theorem 2.6, page 20. As seen in Theorem 2.5, page 19, value iteration requires $\left\lceil \frac{1}{\ln(\gamma)} \ln\left(\frac{\epsilon(1-\gamma)}{R_{\max}}\right) \right\rceil$ full updates of the value function, which is randomly initialized. Each full update requires updating the current value of V for each states, resulting in S operations. Each one of those operations requires computing A values among which the maximum is picked for the update. Each one of those A values requires computing the value of $\mathcal{O}(S)$ possible successor states. As a result, a full update of V (for every states) requires $\mathcal{O}(S^2 A)$ operations. $\square$

Proof of Theorem 2.8, page 30. This result stems directly from Lemma 13 and 14 from Strehl, Li, and Littman (2009) that provide the minimal value of m for the two different events of:

A estimating $T_{s\cdot}^a$ with precision ϵ_M in 1-norm with probability at least $1-\delta'$, B estimating R^a_s with precision ϵ_M in absolute value with probability at least $1-\delta'$,

where $\delta' \in [0,1)$. For both events to be verified at the same time, we apply the union bound for independent events and have that

$$\begin{aligned}\mathbf{Pr}\,(\mathrm{A}\cap\mathrm{B}) &= 1-\mathbf{Pr}(\overline{\mathrm{A}\cap\mathrm{B}})\\ &= 1-\mathbf{Pr}(\bar{\mathrm{A}}\cup\bar{\mathrm{B}})\\ &\geq 1-\left(\mathbf{Pr}(\bar{\mathrm{A}})+\mathbf{Pr}(\bar{\mathrm{B}})\right)\,,\end{aligned}$$

where the negation of the probability event X is denoted by $\bar{X}$. Setting $\delta' = \delta/2$ for each event A and B yields $\mathbf{Pr}(\bar{\mathrm{A}}) \leq \delta/2$ and $\mathbf{Pr}(\bar{\mathrm{B}}) \leq \delta/2$, which concludes the proof by implying

$$\mathbf{Pr}\,(\mathrm{A}\cap\mathrm{B}) \geq 1-\delta\,.$$

□

A.2 Proofs of Chapter 3

Proof of Lemma 3.1, page 51. The first result is borrowed from Kocsis and Szepesvári (2006) where they show it for a generic bandit problem. The extension to our case with a given budget is straightforward. The sequence of lower bounds can be derived by observing that $b(d) = T^{d-1}_{\widehat{I}^{d-1},b(d-1)}$ and applying the previous lower bound. □

Proof of Lemma 3.2, page 51. Let us first bound the failure probability with the probability of overestimating a sub-optimal action and underestimating the optimal one up to $\Delta^d_{\widehat{I}^d}/2$.

$$\begin{aligned}&\mathbf{Pr}\left(\widehat{I}^d \neq i^*_d \,\middle|\, b(d)\right)\\ &= \mathbf{Pr}\left(\hat{Z}^d_{\widehat{I}^d,b(d)} \geq \hat{Z}^d_{i^*_d,b(d)} \,\middle|\, b(d)\right)\\ &\leq \mathbf{Pr}\left(\hat{Z}^d_{\widehat{I}^d,b(d)} \geq Z^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \cup \hat{Z}^d_{i^*_d,b(d)} \leq Z^d_{i^*_d,b(d)} - \frac{\Delta^d_{\widehat{I}^d}}{2} \,\middle|\, b(d)\right)\\ &\leq \mathbf{Pr}\left(\hat{Z}^d_{\widehat{I}^d,b(d)} \geq Z^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \,\middle|\, b(d)\right) + \mathbf{Pr}\left(\hat{Z}^d_{i^*_d,b(d)} \leq Z^d_{i^*_d,b(d)} - \frac{\Delta^d_{\widehat{I}^d}}{2} \,\middle|\, b(d)\right)\end{aligned}$$

From now on, the proof breaks to the analysis of one of the two terms on the right of the last inequality since both can be considered the same way. Let us consider the first term:

$$\begin{aligned}
&\mathbf{Pr}\left(\hat{Z}^d_{\widehat{I}^d,b(d)} \geq Z^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \,\middle|\, b(d)\right) \\
&= \sum_{t=1}^{b(d)} \mathbf{Pr}\left(\hat{Z}^d_{\widehat{I}^d,b(d)} \geq Z^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \,\middle|\, T^d_{\widehat{I}^d,b(d)} = t\right) \mathbf{Pr}\left(T^d_{\widehat{I}^d,b(d)} = t \,\middle|\, b(d)\right) \\
&\leq \sum_{t=1}^{b(d)} \exp\left(-\frac{1}{2}\left(\Delta^d_{\widehat{I}^d}\right)^2 t\right) \mathbf{Pr}\left(T^d_{\widehat{I}^d,b(d)} = t \,\middle|\, b(d)\right) \\
&\leq \sum_{t=\lceil \rho \ln(b(d)) \rceil}^{b(d)} \exp\left(-\frac{1}{2}\left(\Delta^d_{\widehat{I}^d}\right)^2 t\right) \mathbf{Pr}\left(T^d_{\widehat{I}^d,b(d)} = t \,\middle|\, b(d)\right) \\
&\leq \exp\left(-\frac{1}{2}\left(\Delta^d_{\widehat{I}^d}\right)^2 \lceil \rho \ln(b(d)) \rceil\right) \\
&\leq b(d)^{-\frac{\rho}{2}(\Delta^d_{\widehat{I}^d})^2} \\
&\leq b(d)^{-\frac{\rho}{2}(\delta^d)^2}
\end{aligned}$$

Where we first write the joint probability, then apply Hoeffding's inequality, followed by Lemma 3.1, page 51 and the fact that a convex combination is upper bounded by its higher element. Similarly to Kocsis and Szepesvári (2006), we shall assume that the UCT constant C_p is appropriately chosen for the tail inequalities to be verified. □

Proof of Theorem 3.1, page 52. We write the joint probability distribution:

$$\begin{aligned}
\mathbf{Pr}\left(\widehat{I}^d \neq i^*_d \,\middle|\, B\right) &= \sum_{t=1}^{B} \mathbf{Pr}\left(\widehat{I}^d \neq i^*_d \,\middle|\, B, b(d) = t\right) \mathbf{Pr}\left(b(d) = t \mid B\right) \\
&\leq \sum_{t=1}^{B} 2t^{-\frac{\rho}{2}(\delta^d)^2} \mathbf{Pr}\left(b(d) = t \mid B\right) \\
&\leq 2 \sum_{t=\lceil \rho \ln(b(d-1)) \rceil}^{B} t^{-\frac{\rho}{2}(\delta^d)^2} \mathbf{Pr}\left(b(d) = t \mid B\right) \\
&\leq 2\lceil \rho \ln(b(d-1)) \rceil^{-\frac{\rho}{2}(\delta^d)^2}
\end{aligned}$$

Where we first applied Lemma 3.2, page 51 and then used the fact that $b(d) = T^{d-1}_{\widehat{I}^{d-1},b(d-1)}$ onto which we apply Lemma 3.1, page 51. Finally, we use the fact that a convex combination is upper bounded by its higher element. The last result comes from the sequence of lower bounds in Lemma 3.1, page 51. □

A.3 Proofs of Chapter 4

Proof of Theorem 4.1, page 72. Consider an (L_T, L_r)-LC-NSMDP, a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ and two decision epochs $(t, \hat{t}) \in \mathcal{T}^2$. By definition of the expected reward function, the following holds:

$$\begin{aligned}
R_t(s,a) - R_{\hat{t}}(s,a) &= \int_{\mathcal{S}} \left(T_t(s' \mid s,a)\, r_t(s,a,s') - T_{\hat{t}}(s' \mid s,a)\, r_{\hat{t}}(s,a,s')\right) ds' \\
&= \int_{\mathcal{S}} r_t(s,a,s') \left(T_t(s' \mid s,a) - T_{\hat{t}}(s' \mid s,a)\right) ds' \\
&+ \int_{\mathcal{S}} T_{\hat{t}}(s' \mid s,a) \left(r_t(s,a,s') - r_{\hat{t}}(s,a,s')\right) ds' \\
&\leq \sup_{f \in \mathrm{Lip}_1(\mathcal{S})} \int_{\mathcal{S}} f(s') \left(T_t(s' \mid s,a) - T_{\hat{t}}(s' \mid s,a)\right) ds' \\
&+ \int_{\mathcal{S}} T_{\hat{t}}(s' \mid s,a)\, L_r \left|t - \hat{t}\right| ds' \\
&\leq W_1(p(\cdot \mid s,t,a), p(\cdot \mid s,\hat{t},a)) + L_r \left|t - \hat{t}\right| \\
&\leq (L_T + L_r) \left|t - \hat{t}\right|
\end{aligned}$$

Where, similarly to the proof of Lemma 2 in Rachelson and Lagoudakis (2010), we used the triangle inequality, the fact that r is a bounded function — which makes it upper bounded by a function of $\mathrm{Lip}_1(\mathcal{S})$ — and the dual formulation of the 1-Wasserstein distance (see Definition 4.7, page 71). The same inequality can be derived with the opposite terms which concludes the proof by taking the absolute value. □

Proof of Theorem 4.2, page 74. Straightforwardly, using the Lipschitz property of Definition 4.6, page 71 and Theorem 4.1, page 72, we have the following result for all $(s, t, a) \in \mathcal{S} \times \mathcal{T} \times \mathcal{A}$:

$$\begin{aligned}
W_1\left(T_t(\cdot \mid s,a), T_{t-1}(\cdot \mid s,a)\right) &\leq L_T\,, \\
\left|R_t(s,a) - R_{t-1}(s,a)\right| &\leq L_T + L_r\,,
\end{aligned}$$

which concludes the proof. □

Proof of Lemma 4.1, page 81. We use the dual representation of the 1-Wasserstein distance of Definition 4.7, page 71.

$$\begin{aligned}
W_1(w_0,\, \lambda w_1 + (1-\lambda)w_2) &= \sup_{f \in \mathrm{Lip}_1(X)} \int_X f(x)(w_0(x) - \lambda w_1(x) - (1-\lambda)w_2(x))dx \\
&= \sup_{f \in \mathrm{Lip}_1(X)} \int_X \left(\lambda f(x)(w_0(x) - w_1(x)) + (1-\lambda) f(x)(w_0(x) - w_2(x))\right) dx \\
&\leq \lambda \sup_{f \in \mathrm{Lip}_1(X)} \int_X f(x)(w_0(x) - w_1(x))dx \\
&+ (1-\lambda) \sup_{f \in \mathrm{Lip}_1(X)} \int_X f(x)(w_0(x) - w_2(x))dx \\
&\leq \lambda W_1(w_0, w_1) + (1-\lambda) W_1(w_0, w_2)
\end{aligned}$$

Where we used the linearity of the integral and the triangle inequality on the sup operator. □

Proof of Theorem 4.3, page 81. We are looking for a closed-form expression of the value of a chance node $\nu^{s,t,a}$ as defined in Equation 4.10, page 79. We look for a snapshot model minimizing the equation, *i.e.*,

$$\left(\tilde{T}, \tilde{R}\right) = \operatorname*{argmin}_{(T,R) \in \Delta_{t_0,t}} R\left(s,a\right) + \gamma \mathbb{E}_{s' \sim T(\cdot \mid s,a)} \left(V(\nu^{s',t+1}) \right) .$$

Obviously, we have that $\tilde{R}(s,a) = R_{t_0}\left(s,a\right) - L_R \left|t - t_0\right|, \forall \left(s,a\right) \in \mathcal{S} \times \mathcal{A}$ and $\tilde{T}$ is given by:

$$\tilde{T} = \operatorname*{argmin}_{T \in \mathcal{B}_{W_1}\left(T_{t_0}(\cdot \mid s,a), L_T|t-t_0|\right)} \sum_{s' \in \mathcal{S}} T\left(s' \mid s,a\right) V(\nu^{s',t+1})$$

Since we are in the discrete case, we enumerate through the elements of $\mathcal{S}$ and write the vectors $T \equiv \left(T\left(s' \mid s,a\right)\right)_{s'}$, $T_0 \equiv \left(T_{t_0}\left(s' \mid s,a\right)\right)_{s'}$ and $v \equiv (V(\nu^{s',t+1}))_{s'}$. The problem can then be re-written as follows:

$$\begin{aligned}
\tilde{T} = \operatorname*{argmin}_{T} \quad & T^\top v, && \text{(A.1)}\\
\text{such that } & T^\top \mathbf{1} = 1, && \text{(A.2)}\\
& T \geq 0, && \text{(A.3)}\\
& W_1(T, T_0) \leq C\,, && \text{(A.4)}
\end{aligned}$$

where we have $\mathbf{1} \in \mathbb{R}^S$ a vector of ones of size S, $C = L_T \left|t - t_0\right|$. Additionally, we wrote the 1-Wasserstein metric between two discrete distributions in dual form following Lemma 4.7, page 71 as:

$$\begin{aligned}
W_1\left(u,v\right) = \max_f f^\top (u - v), && \text{(A.5)}\\
\text{such that } Af \leq b\,,
\end{aligned}$$

where the matrix A and vector b are defined such that for any indexes i,j we have $|f_i - f_j| \leq d_{i,j}$ with $d_{i,j}$ the metric defined over the measured space, in our case the state space $\mathcal{S}$. Hence, we propose to solve the program A.1 under constraints A.2, A.3 and A.4.

Let us first show that this problem is convex. Clearly, the objective function in Equation A.1, page 131 is linear, hence convex, and the constraints A.2, page 131 and A.3, page 131 define a convex set. We prove that the 1-Wasserstein distance is convex in Lemma 4.1, page 81. The program A.1, page 131 is thus convex.

One can observe that the gradient of the objective function is constant, equal to $+v$. Furthermore, T_0 is an admissible initial point that we could use for a gradient descent method. However, given T_0, following the descent direction $-v$ may break the constraints A.2, page 131 and A.3, page 131. One would have to project this gradient onto a certain, unknown, set of hyperplanes in order to apply the gradient method descent. Let us write $\mathrm{proj}(v)$ the resulting projected gradient, that is unknown.

We remark that the vector $T_{\text{sat}} = (0, \cdots, 0, 1, 0, \cdots, 0)$ with 1 at the index $\operatorname{argmin}_i v_i$ where v_i denotes the ith coefficient of v, is the optimal solution of the program A.1, page 131 when we remove the Wasserstein constraint A.4, page 131. One can observe that the optimal solution with the constraint A.4, page 131 would as well be T_{sat} if the constant C is *large enough*. As a result, the descent direction $\nabla = T_{\text{sat}} - T_0$ is the one to be followed in this setting when applying the gradient descent method. Furthermore, following ∇ from T_0 until T_{sat} never breaks the constraints A.2, page 131 and A.3, page 131. Since the gradient of the objective function is constant, there can exist only one $\mathrm{proj}(v)$. ∇ fulfills the requirements, hence we have $\mathrm{proj}(v) = \nabla$.

We can thus apply the gradient method descent with the following 1-shot rule since the gradient is constant:

$$\tilde{T} := T_0 + \lambda \nabla \text{ with, } \begin{cases} \lambda = 1 \text{ if } W_1\left(T_{t_0}, T_{\text{sat}}\right) \leq C \\ \lambda = C / W_1\left(T_{t_0}, T_{\text{sat}}\right) \text{ otherwise.} \end{cases}$$

Indeed, in the first case, we can follow ∇ until the extreme distribution T_{sat} without breaking the constraint A.4, page 131. Going further is trivially unfeasible. In the second case, we have to stop *in between* so that the constraint A.4, page 131 is saturated. In such a case, we cannot go further without breaking this constraint. Hence we have the following:

$$\begin{aligned} W_1\left(T_0 + \lambda \nabla, T_0\right) &= C \\ \max_{Af \leq b} f^\top (T_0 + \lambda \nabla - T_0) &= C \\ \lambda \max_{Af \leq b} f^\top \nabla &= C \\ \lambda &= C / W_1\left(T_0, T_{\text{sat}}\right) \end{aligned}$$

Where we used the fact that $\nabla = T_{\text{sat}} - T_0$. The latter result concludes the proof. □

Proof of Theorem 4.4, page 82. The proof is made by induction, starting at depth $d_{\max}$ and reversely ending at depth 0. At $d_{\max}$, the nodes are leaf nodes, their values is estimated with the heuristic function, *i.e.*, $V(\nu^{s,t}) = \mathcal{H}(s,t)$. Hence the result is directly proven by hypothesis in Equation 4.11, page 82. We will now start by proving the result for the chance nodes which come as the first parents of the decision node for which we initialized the induction proof. Then we extend it to the parents decision nodes which will complete the proof.

Chance nodes case. Consider any chance node $\nu^{s,t,a}$ at depth $d \in [0, d_{\max})$. We suppose that the property is true for depth $d+1$, thus, if we write $\nu^{s',t'}$ a decision node at depth $d+1$, we have the following:

$$\left| V(\nu^{s',t'}) - \hat{V}^*_{t_0,t'}(s') \right| \leq \gamma^{(d_{\max}-(d+1))} \delta_{\mathcal{H}}$$

Following Equation 4.10, page 79, we have by construction:

$$Q(\nu^{s,t,a}) = \tilde{R}_t\,(s,a) + \gamma \sum_{s'} \tilde{T}_t\,(s' \mid s,a)\, V(\nu^{s',t'})\,,$$

with $\tilde{T}$ and $\tilde{R}$ computed with Theorem 4.3, page 81. By definition, the true Q-value function defined by the Bellman Equation 4.1, page 69 gives the true target value:

$$\hat{Q}^*_{t_0,t}(s,a) = \tilde{R}_t\,(s,a) + \gamma \sum_{s'} \tilde{T}_t\,(s' \mid s,a)\, \hat{V}^*_{t_0,t'}(s')$$

Hence, using the induction hypothesis, we have the following inequalities proving the result of Equation 4.13, page 82:

$$\begin{aligned}
\left| Q(\nu^{s,t,a}) - \hat{Q}^*_{t_0,t}(s,a) \right| &= \gamma \left| \sum_{s'} \tilde{T}_t\,(s' \mid s,a)\, V(\nu^{s',t'}) - \sum_{s'} \tilde{T}_t\,(s' \mid s,a)\, \hat{V}^*_{t_0,t'}(s') \right| \\
&\leq \gamma \sum_{s'} \tilde{T}_t\,(s' \mid s,a) \left| V(\nu^{s'}) - \hat{V}^*_{t_0,t'}(s') \right| \\
&\leq \gamma \sum_{s'} \tilde{T}_t\,(s' \mid s,a)\, \gamma^{(d_{\max}-(d+1))} \delta_{\mathcal{H}} \\
&\leq \gamma^{(d_{\max}-d)} \delta_{\mathcal{H}}
\end{aligned}$$

Decision nodes case. Consider now any decision node $\nu^{s,t}$ at depth $d \in \{0, \ldots, d_{\max}-1\}$. The value of such a node is given by Equation 4.9, page 79 and the following holds:

$$V(\nu^{s,t}) = V(\nu^{s,t,\bar{a}}), \text{ with, } \bar{a} = \operatorname*{argmax}_{a \in \mathcal{A}} V(\nu^{s,t,a})\,.$$

Similarly, we define $a^* \in \mathcal{A}$ as follows:

$$\hat{V}^*_{t_0,t}(s) = \hat{Q}^*_{t_0,t}(s,a^*), \text{ with, } a^* = \operatorname*{argmax}_{a \in \mathcal{A}} \hat{Q}^*_{t_0,t}(s,a)$$

We distinguish two cases: (i) if $\bar{a} = a^*$ and (ii) if $\bar{a} \neq a^*$. In case (i), the result is trivial by writing the value of the decision node as the value of the chance node with the action a^* and using the — already proven for depth d — result of Equation 4.13, page 82.

$$\begin{aligned}\left|V(\nu^{s,t}) - \hat{V}^*_{t_0,t}(s)\right| &= \left|V(\nu^{s,t,a^*}) - \hat{Q}^*_{t_0,t}(s,a^*)\right| \\ &\leq \gamma^{(d_{\max}-d)}\delta_{\mathcal{H}}\end{aligned}$$

In case (ii), page 133, the maximizing actions are different. Still following Equation 4.13, page 82, we have that $Q\left(\nu^{s,t,a^*}\right) \geq \hat{Q}^*_{t_0,t}(s,a^*) - \gamma^{(d_{\max}-d)}\delta_{\mathcal{H}}$. Yet, since $\bar{a}$ is the maximizing action in the tree, we have that $Q\left(\nu^{s,t,\bar{a}}\right) \geq Q\left(\nu^{s,t,a^*}\right)$. By transitivity, we can thus write the following:

$$\begin{aligned} & Q\left(\nu^{s,t,\bar{a}}\right) \geq \hat{Q}^*_{t_0,t}(s,a^*) - \gamma^{(d_{\max}-d)}\delta_{\mathcal{H}} \\ \Rightarrow \quad & \hat{Q}^*_{t_0,t}(s,a^*) - Q(\nu^{s,t,\bar{a}}) \leq \gamma^{(d_{\max}-d)}\delta_{\mathcal{H}} \end{aligned} \tag{A.6}$$

Furthermore, still following Equation 4.13, page 82, we have that $\hat{Q}^*_{t_0,t}(s,\bar{a}) \geq Q\left(\nu^{s,t,\bar{a}}\right) - \gamma^{(d_{\max}-d)}\delta_{\mathcal{H}}$. Yet, since a^* is the maximizing action in the worst case snapshot MDP$_t$, we have that $\hat{Q}^*_{t_0,t}(s,a^*) \geq \hat{Q}^*_{t_0,t}(s,\bar{a})$. By transitivity, we can thus write the following:

$$\begin{aligned} & \hat{Q}^*_{t_0,t}(s,a^*) \geq Q\left(\nu^{s,t,\bar{a}}\right) - \gamma^{(d_{\max}-d)}\delta_{\mathcal{H}} \\ \Rightarrow \quad & Q\left(\nu^{s,t,\bar{a}}\right) - \hat{Q}^*_{t_0,t}(s,a^*) \leq \gamma^{(d_{\max}-d)}\delta_{\mathcal{H}} \end{aligned} \tag{A.7}$$

By assembling equations A.6 and A.7, we prove equation 4.12, page 82 and the proof by induction is complete. □

Proof of Theorem 4.5, page 82. Let us first calculate the cost of constructing a tree with the minimax procedure. Following Algorithm 8, page 80, a tree is composed of at most n_l leaf nodes, n_d non-leaf decision nodes and n_c chance nodes, with the following values for the integers n_l, n_d and n_c:

$$n_l = (SA)^{d_{\max}}\,,\; n_d = \sum_{i=0}^{d_{\max}-1} (SA)^i\,,\text{ and } n_c = An_d\,.$$

As a result, we have that n_l is $\mathcal{O}((SA)^{d_{\max}})$, n_d is $\mathcal{O}((SA)^{d_{\max}})$ and n_c is $\mathcal{O}(A\,(SA)^{d_{\max}})$. We write respectively c_l, c_d and c_c the number of operations required to compute the values of a leaf node, a non-leaf decision node and a chance node. To compute the whole tree we need to build *and* evaluate all the nodes, resulting in at most the following number of operations:

$$n_l c_l + n_d c_d + n_c c_c\,. \tag{A.8}$$

We will assume that c_l is $\mathcal{O}(1)$ without further details on the nature of the heuristic function. As the value of a non-leaf decision node is computed by finding the maximum value among the A children, we have that c_d is $\mathcal{O}(A)$. From Theorem 4.3, page 81, the evaluation of a chance node is equivalent to computing a 1-Wasserstein distance, which is a linear program. Following Vaidya's algorithm (Vaidya, 1989), the cost in the worst case is $\mathcal{O}(S^{2.5})$ where S is the dimension of the problem in our case. As a result, c_c is $\mathcal{O}(S^{2.5})$. Replacing all the values in Equation A.8, we deduce that the total number of operations of computing a tree is

$$\mathcal{O}\left(S^{1.5}\left(SA\right)^{d_{\max}}\right) .$$

After computing a tree, the action maximizing the value should be selected which has complexity $\mathcal{O}(A)$. The operation being repeated for every time steps, one should multiply everything by τ, the total number of time steps for which the algorithm is run. As a result, the total computational complexity of RATS is

$$\mathcal{O}\left(\tau S^{1.5} A\left(SA\right)^{d_{\max}}\right) .$$

□

Proof of Lemma 4.2, page 84. The proof is made by induction. Let us first consider $n = 0$. By definition, we have:

$$\begin{aligned}\left|V^{\pi,0}_{\mathrm{MDP}_{t_0}}(s) - V^{\pi,0}_{\mathrm{MDP}_t}(s)\right| &= \left|\int_{\mathcal{A}} \pi(a \mid s)\left(R_{t_0}(s,a) - R_t(s,a)\right) da\right| \\ &\leq \int_{\mathcal{A}} \pi(a \mid s) L_R \left|t - t_0\right| da \\ &\leq L_R \left|t - t_0\right|\end{aligned}$$

Which verifies the property for $n = 0$ with $L_{V_0} = L_R$. Let us now consider $n \in \mathbb{N}$ and suppose the property true for rank $n-1$. By writing the Bellman equation for the two value functions, we obtain the following:

$$\begin{aligned}V^{\pi,n}_{\mathrm{MDP}_{t_0}}(s) - V^{\pi,n}_{\mathrm{MDP}_t}(s) = \int_{\mathcal{S}\times\mathcal{A}} \pi(a|s)\Big[&T_{t_0}\left(s' \mid s,a\right)\left(r_{t_0}(s,a,s') + \gamma V^{\pi,n-1}_{\mathrm{MDP}_{t_0}}(s')\right) \\ &- T_t\left(s' \mid s,a\right)\left(r_t\left(s,a,s'\right) + \gamma V^{\pi,n-1}_{\mathrm{MDP}_t}(s')\right)\Big] ds' da ,\end{aligned}$$

which yields

$$V^{\pi,n}_{\mathrm{MDP}_{t_0}}(s) - V^{\pi,n}_{\mathrm{MDP}_t}(s) = \int_{\mathcal{A}} \pi(a|s)\Big[A(s,a) + B(s,a)\Big] da . \tag{A.9}$$

With the following values for $A(s,a)$ and $B(s,a)$:

$$\begin{aligned}A(s,a) &= \int_{\mathcal{S}} \left(r_{t_0}(s,a,s') + \gamma V^{\pi,n-1}_{\mathrm{MDP}_{t_0}}(s')\right)\Big[T_{t_0}\left(s' \mid s,a\right) - T_t\left(s' \mid s,a\right)\Big] ds' \\ B(s,a) &= \int_{\mathcal{S}} T_t\left(s' \mid s,a\right)\Big[r_{t_0}(s,a,s') - r_t\left(s,a,s'\right) + \gamma\left(V^{\pi,n-1}_{\mathrm{MDP}_{t_0}}(s') - V^{\pi,n-1}_{\mathrm{MDP}_t}(s')\right)\Big] ds'\end{aligned}$$

Let us first bound $A(s,a)$ by noticing that $s' \mapsto r_{t_0}(s,a,s') + \gamma V^{\pi,n-1}_{\mathrm{MDP}_{t_0}}(s')$ is bounded by $\frac{R_{\max}}{1-\gamma}$.

Since the function $s' \mapsto \frac{1}{1-\gamma}$ belongs to $\mathrm{Lip}_1(\mathcal{S})$, we can write the following:

$$\begin{aligned} A(s,a) &\leq \sup_{f \in \mathrm{Lip}_1(\mathcal{S})} \int_{\mathcal{S}} f(s') \Big[T_{t_0}(s' \mid s,a) - T_t(s' \mid s,a) \Big] ds' \\ &\leq W_1(T_{t_0}, T_t) \\ &\leq L_T \left| t - t_0 \right| . \end{aligned}$$

B is straightforwardly bounded using the induction hypothesis:

$$\begin{aligned} B(s,a) \leq & \int_{\mathcal{S}} T_t(s' \mid s,a) \Big[L_r \left| t - t_0 \right| + \gamma \sum_{i=0}^{n-1} \gamma^i L_R \left| t - t_0 \right| \Big] ds' \\ \leq & L_r \left| t - t_0 \right| + \sum_{i=1}^{n} \gamma^i L_R \left| t - t_0 \right| \end{aligned}$$

We inject the result in Equation A.9, page 135:

$$\begin{aligned} V^{\pi,n}_{\mathrm{MDP}_{t_0}}(s) - V^{\pi,n}_{\mathrm{MDP}_t}(s) &\leq \int_{\mathcal{A}} \pi(a|s) \Big[L_T \left| t - t_0 \right| + L_r \left| t - t_0 \right| + \sum_{i=1}^{n} \gamma^i L_R \left| t - t_0 \right| \Big] da \\ &= (L_T + L_r) \left| t - t_0 \right| + \sum_{i=1}^{n} \gamma^i L_R \left| t - t_0 \right| \\ &\leq L_R \left| t - t_0 \right| + \sum_{i=1}^{n} \gamma^i L_R \left| t - t_0 \right| \\ &\leq \sum_{i=0}^{n} \gamma^i L_R \left| t - t_0 \right| \end{aligned}$$

The same result can be derived with the opposite expression. Hence, taking the absolute value, we prove the property at rank n, *i.e.*

$$\left| V^{\pi,n}_{\mathrm{MDP}_{t_0}}(s) - V^{\pi,n}_{\mathrm{MDP}_t}(s) \right| \leq \sum_{i=0}^{n} \gamma^i L_R \left| t - t_0 \right| \tag{A.10}$$

which concludes the proof by induction. □

Proof of Theorem 4.6, page 84. Consider $(s, t_0, t, n) \in \mathcal{S} \times \mathcal{T}^2 \times \mathbb{N}$. We examine the two snapshots MDP_{t_0} and MDP_t and are interested in the values of the policy π at s within those two snapshots. The result follows easily by remarking that the sequence L_{V_n} of Lemma 4.2, page 84 is geometric and converges towards $\frac{L_R}{1-\gamma}$ when n goes to infinity. □

A.4 Proofs of Chapter 5

Notation A.1. Given two sets X and Y, we write $\mathcal{F}(X,Y)$ the set of functions defined on the domain X with codomain Y.

Notation A.2. We denote by $\mathcal{V}_n$ the set of probability vectors of size $n \in \mathbb{N}$, defined by

$$\mathcal{V}_n \triangleq \left\{ v = \{v_1 \dots v_n\} \in \mathbb{R}^n \;\middle|\; v_i \geq 0,\, \forall i \in [1,n],\, \sum_{i=1}^{n} v_i = 1 \right\} .$$

Proof of Lemma 5.1, page 102. The proof follows closely that in Puterman, 2014 that proves that the Bellman operator over value functions is a contraction mapping. Let L be the functional operator that maps any function $d \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$ to

$$\begin{array}{rlll} Ld: & \mathcal{S} \times \mathcal{A} & \to & \mathbb{R} \\ & s, a & \mapsto & D_{sa}(M \| \bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \max_{a' \in \mathcal{A}} d_{s'a'} . \end{array}$$

Then for f and g, two functions from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$ and $(s,a) \in \mathcal{S} \times \mathcal{A}$, we have that

$$\begin{aligned} Lf_{sa} - Lg_{sa} &= \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \left(\max_{a' \in \mathcal{A}} f_{s'a'} - \max_{a' \in \mathcal{A}} g_{s'a'} \right) \\ &\leq \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \max_{a' \in \mathcal{A}} \left(f_{s'a'} - g_{s'a'} \right) \\ &\leq \gamma \left\| f - g \right\|_\infty . \end{aligned}$$

Since this is true for any pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, we have that

$$\left\| Lf - Lg \right\|_\infty \leq \gamma \left\| f - g \right\|_\infty .$$

Since $\gamma < 1$, L is a contraction mapping in the metric space $(\mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R}), \|\cdot\|_\infty)$. This metric space being complete and non-empty, it follows by direct application of Banach fixed-point theorem that the equation $d = Ld$ admits a unique solution. □

Proof of Theorem 5.1, page 101. The proof is by induction. The value iteration sequence of iterates $(Q^n_M)_{n \in \mathbb{N}}$ of the optimal Q-function of any MDP $M \in \mathcal{M}$ is defined for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ by:

$$\begin{aligned} Q^0_M(s,a) &= 0 , \\ Q^{n+1}_M(s,a) &= R^a_s + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \max_{a' \in \mathcal{A}} Q^n_M(s',a'),\, \forall n \in \mathbb{N} . \end{aligned}$$

Consider two MDPs $M, \bar{M} \in \mathcal{M}$. It is obvious that $\left| Q^0_M(s,a) - Q^0_{\bar{M}}(s,a) \right| \leq d_{sa}(M \| \bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. Suppose the property $\left| Q^n_M(s,a) - Q^n_{\bar{M}}(s,a) \right| \leq d_{sa}(M \| \bar{M})$ true at rank

$n \in \mathbb{N}$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. Consider now the rank $n+1$ and a pair $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$\begin{aligned}
\left|Q^{n+1}_M(s,a) - Q^{n+1}_{\bar{M}}(s,a)\right| &= \left|R^a_s - \bar{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \left[T^a_{ss'} \max_{a' \in \mathcal{A}} Q^n_M(s',a') - \bar{T}^a_{ss'} \max_{a' \in \mathcal{A}} Q^n_{\bar{M}}(s',a')\right]\right| \\
&\leq \left|R^a_s - \bar{R}^a_s\right| + \gamma \sum_{s' \in \mathcal{S}} \left|T^a_{ss'} \max_{a' \in \mathcal{A}} Q^n_M(s',a') - \bar{T}^a_{ss'} \max_{a' \in \mathcal{A}} Q^n_{\bar{M}}(s',a')\right| \\
&\leq \left|R^a_s - \bar{R}^a_s\right| + \gamma \sum_{s' \in \mathcal{S}} \max_{a' \in \mathcal{A}} Q^n_{\bar{M}}(s',a') \left|T^a_{ss'} - \bar{T}^a_{ss'}\right| \\
&\quad + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \left|\max_{a' \in \mathcal{A}} Q^n_M(s',a') - \max_{a' \in \mathcal{A}} Q^n_{\bar{M}}(s',a')\right| \\
&\leq \left|R^a_s - \bar{R}^a_s\right| + \sum_{s' \in \mathcal{S}} \gamma V^*_{\bar{M}}(s') \left|T^a_{ss'} - \bar{T}^a_{ss'}\right| + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \max_{a' \in \mathcal{A}} \left|Q^n_M(s',a') - Q^n_{\bar{M}}(s',a')\right| \\
&\leq D_{sa}(M \| \bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \max_{a'} d_{s'a'}(M \| \bar{M}) \\
&= d_{sa}(M \| \bar{M}) \,,
\end{aligned}$$

where we used Lemma 5.1, page 102 in the last inequality. Since Q^*_M and $Q^*_{\bar{M}}$ are respectively the limits of the sequences $(Q^n_M)_{n \in \mathbb{N}}$ and $\left(Q^n_{\bar{M}}\right)_{n \in \mathbb{N}}$, it results from passage to the limit that

$$\left|Q^*_M(s,a) - Q^*_{\bar{M}}(s,a)\right| \leq d_{sa}(M \| \bar{M}) \,.$$

By symmetry, we also have $\left|Q^*_M(s,a) - Q^*_{\bar{M}}(s,a)\right| \leq d_{sa}(M \| \bar{M})$ and we can take the minimum of the two valid upper bounds, yielding:

$$\left|Q^*_M(s,a) - Q^*_{\bar{M}}(s,a)\right| \leq \min\left\{d_{sa}(M \| \bar{M}), d_{sa}(\bar{M} \| M)\right\} \,,$$

which concludes the proof. □

Proof of Theorem 5.2, page 103. The proof follows exactly the same steps as the proof of Theorem 5.1, page 101, *i.e.*, by first constructing the value iteration sequence of iterates of the optimal value function, showing the result by induction for rank $n \in \mathbb{N}$ and then concluding with a passage to the limit. □

Proof of Lemma 5.2, page 103. Let L be the functional operator that maps any function $d \in \mathcal{F}(\mathcal{S}, \mathbb{R})$ to

$$\begin{aligned}
Ld: \quad \mathcal{S} &\to \mathbb{R} \\
s &\mapsto \textstyle\sum_{a \in \mathcal{A}} \pi(a \mid s) \left(D^{\gamma V^\pi_{\bar{M}}}_{sa}(M, \bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} d_{s'}\right) \,.
\end{aligned}$$

Then for f and g, two functions from $\mathcal{S}$ to $\mathbb{R}$, we have that

$$\begin{aligned}
Lf_s - Lg_s &= \gamma \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s'\in\mathcal{S}} T^a_{ss'} \left(f_{s'} - g_{s'}\right) \\
&\leq \gamma \left\| f - g \right\|_\infty .
\end{aligned}$$

Hence we have that $\left\| Lf - Lg \right\|_\infty \leq \gamma \left\| f - g \right\|_\infty$. Since $\gamma < 1$, L is a contraction mapping in the metric space $\left(\mathcal{F}\left(\mathcal{S}, \mathbb{R}\right), \left\|\cdot\right\|_\infty\right)$. This metric space being complete and non-empty, it follows by direct application of Banach fixed-point theorem that the equation $d = Ld$ admits a unique solution. □

Proof of Theorem 5.3, page 103. Consider a stochastic stationary policy π. The value iteration sequence of iterates $(V^{\pi,n}_M)_{n\in\mathbb{N}}$ of the value function of any MDP $M \in \mathcal{M}$ is defined for all $s \in \mathcal{S}$ by:

$$\begin{aligned}
V^{\pi,0}_M(s) &= 0 , \\
V^{\pi,n+1}_M(s) &= \sum_{a\in\mathcal{A}} \pi(a \mid s) \left(R^a_s + \gamma \sum_{s'\in\mathcal{S}} T^a_{ss'} V^{\pi,n}_M(s') \right)
\end{aligned}$$

Consider two MDPs $M, \bar{M} \in \mathcal{M}$. It is obvious that $\left| V^{\pi,0}_M(s) - V^{\pi,0}_{\bar{M}}(s) \right| \leq d^\pi_s(M \| \bar{M})$ for all $s \in \mathcal{S}$. Suppose the property $\left| V^{\pi,n}_M(s) - V^{\pi,n}_{\bar{M}}(s) \right| \leq d^\pi_s(M \| \bar{M})$ true at rank $n \in \mathbb{N}$ for all $s \in \mathcal{S}$. Consider now the rank $n+1$ and the state $s \in \mathcal{S}$:

$$\begin{aligned}
\left| V^{\pi,n+1}_M(s) - V^{\pi,n+1}_{\bar{M}}(s) \right| &\leq \sum_{a\in\mathcal{A}} \pi(a \mid s) \left| R^a_s - \bar{R}^a_s + \gamma \sum_{s'\in\mathcal{S}} \left(T^a_{ss'} V^{\pi,n}_M(s') - \bar{T}^a_{ss'} V^{\pi,n}_{\bar{M}}(s') \right) \right| \\
&\leq \sum_{a\in\mathcal{A}} \pi(a \mid s) \Bigg(\left| R^a_s - \bar{R}^a_s \right| + \gamma \sum_{s'\in\mathcal{S}} V^{\pi,n}_{\bar{M}}(s') \left| T^a_{ss'} - \bar{T}^a_{ss'} \right| \\
&\quad + \gamma \sum_{s'\in\mathcal{S}} T^a_{ss'} \left| V^{\pi,n}_M(s') - V^{\pi,n}_{\bar{M}}(s') \right| \Bigg) \\
&\leq \sum_{a\in\mathcal{A}} \pi(a \mid s) \left(D^{\gamma V^\pi_{\bar{M}}}_{sa}(M, \bar{M}) + \gamma \sum_{s'\in\mathcal{S}} T^a_{ss'} d^\pi_{s'}(M \| \bar{M}) \right) \\
&\leq d^\pi_s(M \| \bar{M}) ,
\end{aligned}$$

where we used Lemma 5.2, page 103 in the last inequality. Since V^π_M and $V^\pi_{\bar{M}}$ are respectively the limits of the sequences $(V^{\pi,n}_M)_{n\in\mathbb{N}}$ and $\left(V^{\pi,n}_{\bar{M}}\right)_{n\in\mathbb{N}}$, it results from passage to the limit that

$$\left| V^\pi_M(s) - V^\pi_{\bar{M}}(s) \right| \leq d^\pi_s(M \| \bar{M}) .$$

By symmetry, we also have $\left| V^\pi_M(s) - V^\pi_{\bar{M}}(s) \right| \leq d^\pi_s(\bar{M} \| M)$ and we can take the minimum of the two valid upper bounds, yielding:

$$\left|V_M^\pi(s) - V_{\bar{M}}^\pi(s)\right| \leq \min\left\{d_s^\pi(M\|\bar{M}), d_s^\pi(\bar{M}\|M)\right\} ,$$

which concludes the proof. □

Proof of Lemma 5.3, page 104. Let L be the functional operator that maps any function $d \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$ to

$$\begin{array}{rccl} Ld: & \mathcal{S} \times \mathcal{A} & \to & \mathbb{R} \\ & (s,a) & \mapsto & D_{sa}^{\gamma V_{\bar{M}}^\pi}(M,\bar{M}) + \gamma \sum_{(s',a') \in \mathcal{S}\times\mathcal{A}} T_{ss'}^a \pi(a' \mid s') d_{s',a'} \,. \end{array}$$

Then for f and g, two functions from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$, we have for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ that

$$\begin{aligned} Lf_{sa} - Lg_{sa} &= \gamma \sum_{(s',a') \in \mathcal{S}\times\mathcal{A}} T_{ss'}^a \pi(a' \mid s') \left(Lf_{s'a'} - Lg_{s'a'}\right) \\ &\leq \gamma \left\|f - g\right\|_\infty . \end{aligned}$$

Hence we have that $\left\|Lf - Lg\right\|_\infty \leq \gamma \left\|f - g\right\|_\infty$. Since $\gamma < 1$, L is a contraction mapping in the metric space $\left(\mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R}), \left\|\cdot\right\|_\infty\right)$. This metric space being complete and non-empty, it follows by direct application of Banach fixed-point theorem that the equation $d = Ld$ admits a unique solution. □

Proof of Theorem 5.4, page 104. Consider a stochastic stationary policy π. The value iteration sequence of iterates $(Q_M^{\pi,n})_{n\in\mathbb{N}}$ of the Q function for the policy π and MDP $M \in \mathcal{M}$ is defined for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ by:

$$\begin{aligned} Q_M^{\pi,0}(s,a) &= 0 \,, \\ Q_M^{\pi,n+1}(s,a) &= R_s^a + \gamma \sum_{(s',a') \in \mathcal{S}\times\mathcal{A}} T_{ss'}^a \pi(a' \mid s') Q_M^{\pi,n}(s',a') \end{aligned}$$

Consider two MDPs $M, \bar{M} \in \mathcal{M}$. It is obvious that $\left|Q_M^{\pi,0}(s,a) - Q_{\bar{M}}^{\pi,0}(s,a)\right| \leq d_{sa}^\pi(M\|\bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. Suppose the property $\left|Q_M^{\pi,n}(s,a) - Q_{\bar{M}}^{\pi,n}(s,a)\right| \leq d_{sa}^\pi(M\|\bar{M})$ true at rank $n \in \mathbb{N}$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. Consider now the rank $n+1$ and the state-action pair

$(s, a) \in \mathcal{S} \times \mathcal{A}$:

$$\begin{aligned}
\left|Q_M^{\pi,n+1}(s,a) - Q_{\bar{M}}^{\pi,n+1}(s,a)\right| &\leq \left|R_s^a - \bar{R}_s^a\right| \\
&\quad + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} \pi(a' \mid s') \left|T_{ss'}^a Q_M^{\pi,n}(s',a') - \bar{T}_{ss'}^a Q_{\bar{M}}^{\pi,n}(s',a')\right| \\
&\leq \left|R_s^a - \bar{R}_s^a\right| + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} \pi(a' \mid s') Q_{\bar{M}}^{\pi,n}(s',a') \left|T_{ss'}^a - \bar{T}_{ss'}^a\right| \\
&\quad + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} \pi(a' \mid s') T_{ss'}^a \left|Q_M^{\pi,n}(s',a') - Q_{\bar{M}}^{\pi,n}(s',a')\right| \\
&\leq \left|R_s^a - \bar{R}_s^a\right| + \sum_{s' \in \mathcal{S}} \gamma V_{\bar{M}}^{\pi}(s') \left|T_{ss'}^a - \bar{T}_{ss'}^a\right| \\
&\quad + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} \pi(a' \mid s') T_{ss'}^a d_\pi^{M,\bar{M}}(s',a') \\
&\leq D_{sa}^{\gamma V_{\bar{M}}^{\pi}}(M, \bar{M}) + \gamma \sum_{(s',a') \in \mathcal{S} \times \mathcal{A}} T_{ss'}^a \pi(a' \mid s') d_{s'a'}^{\pi}(M \| \bar{M}) \\
&= d_{sa}^{\pi}(M \| \bar{M}) \,,
\end{aligned}$$

where we used Lemma 5.3, page 104 in the last inequality. Since Q_M^π and $Q_{\bar{M}}^\pi$ are respectively the limits of the sequences $(Q_M^{\pi,n})_{n \in \mathbb{N}}$ and $\left(Q_{\bar{M}}^{\pi,n}\right)_{n \in \mathbb{N}}$, it results from passage to the limit that

$$\left|Q_M^\pi(s,a) - Q_{\bar{M}}^\pi(s,a)\right| \leq d_{sa}^\pi(M \| \bar{M}) \,.$$

By symmetry, we also have $\left|Q_M^\pi(s,a) - Q_{\bar{M}}^\pi(s,a)\right| \leq d_{sa}^\pi(\bar{M} \| M)$ and we can take the minimum of the two valid upper bounds, yielding for all $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$\left|Q_M^\pi(s,a) - Q_{\bar{M}}^\pi(s,a)\right| \leq \min\left\{d_{sa}^\pi(M \| \bar{M}), d_{sa}^\pi(\bar{M} \| M)\right\} \,,$$

which concludes the proof. □

Proof of Theorem 5.5, page 104. The proof is by induction. We consider the sequence of value iteration iterates defined for any MDP $M \in \mathcal{M}$ for $(s,a) \in \mathcal{S} \times \mathcal{A}$ by

$$\begin{aligned}
Q_M^0(s,a) &= 0 \,, \\
Q_M^{n+1}(s,a) &= R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a \max_{a' \in \mathcal{A}} Q_M^n(s',a'), \; \forall n \in \mathbb{N} \,.
\end{aligned}$$

Consider two MDPs $M, \bar{M} \in \mathcal{M}$. It is immediate for any $(s,a) \in \mathcal{S} \times \mathcal{A}$ that

$$\left|Q_M^0(s,a) - Q_{\bar{M}}^0(s,a)\right| \leq d(M \| \bar{M}) \,,$$

and, by symmetry, the result holds as well for $d(\bar{M} \| M)$. Suppose that it is true at rank

$n \in \mathbb{N}$. Consider rank $n+1$ and $(s,a) \in \mathcal{S} \times \mathcal{A}$, we have that:

$$\begin{aligned}
\left|Q_M^{n+1}(s,a) - Q_{\bar{M}}^{n+1}(s,a)\right| &\leq D_{sa}(M\|\bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a \max_{a' \in \mathcal{A}} \left|Q_M^n(s',a') - Q_{\bar{M}}^n(s',a')\right| \\
&\leq \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} D_{sa}(M\|\bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a \frac{1}{1-\gamma} \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} D_{sa}(M\|\bar{M}) \\
&\leq \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} D_{sa}(M\|\bar{M}) \left(1 + \frac{\gamma}{1-\gamma}\right) \\
&= d(M\|\bar{M})\,.
\end{aligned}$$

By symmetry, the results holds as well for $d(\bar{M}\|M)$ which concludes the proof by induction.

□

Proof of Theorem 5.6, page 106. The result is clear for all $(s,a) \notin K$ since the induced Lipschitz upper bounds are provably greater than Q_M^* (see Definition 5.2, page 105). For $(s,a) \in K$, the result is shown by induction. Let us consider the DP sequences of iterates $(Q_M^n)_{n\in\mathbb{N}}$ and $(Q^n)_{n\in\mathbb{N}}$, respectively converging to Q_M^* and Q, whose definitions follow for all $(s,a) \in K$ and for $n \in \mathbb{N}$:

$$\begin{cases} Q_M^0(s,a) = 0 \\ Q_M^{n+1}(s,a) = R_s^a + \gamma \sum_{s'\in\mathcal{S}} T_{ss'}^a \max\limits_{a'\in\mathcal{A}} Q_M^n(s',a') \end{cases},$$

$$\begin{cases} Q^0(s,a) = 0 \\ Q^{n+1}(s,a) = \hat{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \hat{T}_{ss'}^a \max\limits_{a'\in\mathcal{A}} Q^n(s',a') + \frac{\epsilon}{1-\gamma} \end{cases}.$$

Obviously, we have at rank $n = 0$ that $Q_M^0(s,a) \leq Q^0(s,a)$ for all $(s,a) \in K$. Suppose the property true at rank $n \in \mathbb{N}$ and consider rank $n+1$. For $(s,a) \in K$, we have that

$$\begin{aligned}
&Q_M^{n+1}(s,a) - Q^{n+1}(s,a) \\
&= R_s^a - \hat{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \left(T_{ss'}^a \max_{a'\in\mathcal{A}} Q_M^n(s',a') - \hat{T}_{ss'}^a \max_{a'\in\mathcal{A}} Q^n(s',a')\right) - \frac{\epsilon}{1-\gamma} \\
&\leq \epsilon + \gamma \sum_{s'\in\mathcal{S}} \max_{a'\in\mathcal{A}} Q_M^n(s',a') \left(T_{ss'}^a - \hat{T}_{ss'}^a\right) \\
&\quad + \gamma \sum_{s'\in\mathcal{S}} \hat{T}_{ss'}^a \left(\max_{a'\in\mathcal{A}} Q_M^n(s',a') - \max_{a'\in\mathcal{A}} Q^n(s',a')\right) - \frac{\epsilon}{1-\gamma} \\
&\leq \epsilon + \frac{\gamma}{1-\gamma}\epsilon + \gamma \sum_{s'\in\mathcal{S}} \hat{T}_{ss'}^a \left(\max_{a'\in\mathcal{A}} Q_M^n(s',a') - \max_{a'\in\mathcal{A}} Q^n(s',a')\right) - \frac{\epsilon}{1-\gamma} \\
&\leq \epsilon + \frac{\gamma}{1-\gamma}\epsilon - \frac{\epsilon}{1-\gamma} \\
&= 0\,.
\end{aligned}$$

Which concludes the proof by induction. The result holds by passage to the limit since the considered DP sequences converge to the true functions. □

Proof of Theorem 5.7, page 107. Consider two tasks $M = (T, R)$ and $\bar{M} = (\bar{T}, \bar{R})$, with K and $\bar{K}$ the respective sets of state-action pairs where their learned models $\hat{M} = (\hat{T}, \hat{R})$ and $\hat{\bar{M}} = (\hat{\bar{T}}, \hat{\bar{R}})$ are known with accuracy ϵ in $\mathcal{L}_1$-norm with probability at least $1 - \delta$, *i.e.*, we have that,

$$\mathbf{Pr}\begin{pmatrix} \left|R^a_s - \hat{R}^a_s\right| & \leq & \epsilon, & \forall\,(s,a) \in K & \text{and} \\ \left\|T^a_{ss'} - \hat{T}^a_{ss'}\right\|_1 & \leq & \epsilon, & \forall\,(s,a) \in K & \text{and} \\ \left|\bar{R}^a_s - \hat{\bar{R}}^a_s\right| & \leq & \epsilon, & \forall\,(s,a) \in \bar{K} & \text{and} \\ \left\|\bar{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'}\right\|_1 & \leq & \epsilon, & \forall\,(s,a) \in \bar{K} & \end{pmatrix} \leq 1 - \delta. \tag{A.11}$$

Importantly, notice that the probabilistic event of Inequality A.11 is the intersection of at most $4SA$ individual events of estimating either R^a_s, $T^a_{ss'}$, $\bar{R}^a_s$ or $\bar{T}^a_{ss'}$ with precision ϵ. Each one of those individual events is itself true with probability at least $1 - \delta'$, where $\delta' \in (0, 1]$ is a parameter, as described in Theorem 2.8, page 30. For *all* the individual events to be true at the same time, *i.e.* for Inequality A.11 to be verified, one must apply Boole's inequality and set $\delta' = \delta/(4SA)$ to ensure a total probability — *i.e.*, probability of the intersection of all the individual events — of at least $1 - \delta$.

We demonstrate now the result for each one of the three cases (i) $(s, a) \in K \cap \bar{K}$, (ii) $(s, a) \in K \cap \bar{K}^c$ and (iii) $(s, a) \in K^c \cap \bar{K}^c$, the case $(s, a) \in K^c \cap \bar{K}$ being the symmetric of case (ii).

(i) If $(s, a) \in K \cap \bar{K}$, then we have ϵ-close estimates of both models with high probability, as described by Inequality A.11. By definition:

$$D^{\gamma V^*_{\bar{M}}}_{sa}(M, \bar{M}) = \left|R^a_s - \bar{R}^a_s\right| + \gamma \sum_{s' \in \mathcal{S}} V^*_{\bar{M}}(s') \left|T^a_{ss'} - \bar{T}^a_{ss'}\right| . \tag{A.12}$$

The first term of the right hand side of Equation A.12 respects the following sequence of inequalities with probability at least $1 - \delta$:

$$\begin{aligned} \left|R^a_s - \bar{R}^a_s\right| &\leq \left|R^a_s - \hat{R}^a_s\right| + \left|\hat{R}^a_s - \hat{\bar{R}}^a_s\right| + \left|\bar{R}^a_s - \hat{\bar{R}}^a_s\right| \\ &\leq \left|\hat{R}^a_s - \hat{\bar{R}}^a_s\right| + 2\epsilon\,. \end{aligned} \tag{A.13}$$

The second term of the right hand side of Equation A.12 respects the following sequence of inequalities with probability at least $1-\delta$:

$$\begin{aligned}
\gamma \sum_{s' \in \mathcal{S}} V^*_{\bar{M}}(s') \left| T^a_{ss'} - \bar{T}^a_{ss'} \right| &\leq \gamma \sum_{s' \in \mathcal{S}} \bar{V}(s') \left(\left| T^a_{ss'} - \hat{T}^a_{ss'} \right| + \left| \hat{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'} \right| + \left| \bar{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'} \right| \right) \\
&\leq \gamma \max_{s'} \bar{V}(s') \sum_{s' \in \mathcal{S}} \left| T^a_{ss'} - \hat{T}^a_{ss'} \right| + \gamma \sum_{s' \in \mathcal{S}} \bar{V}(s') \left| \hat{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'} \right| + \\
&\quad \gamma \max_{s'} \bar{V}(s') \sum_{s' \in \mathcal{S}} \left| \bar{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'} \right| \\
&\leq \gamma \sum_{s' \in \mathcal{S}} \bar{V}(s') \left| \hat{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'} \right| + 2\epsilon\gamma \max_{s' \in \mathcal{S}} \bar{V}(s'). \qquad \text{(A.14)}
\end{aligned}$$

Replacing the Inequalities A.13 and A.14 in Equation A.12 yields

$$\begin{aligned}
D_{sa}(M \| \bar{M}) &\leq \left| \hat{R}^a_s - \hat{\bar{R}}^a_s \right| + \gamma \sum_{s' \in \mathcal{S}} \bar{V}(s') \left| \hat{T}^a_{ss'} - \hat{\bar{T}}^a_{ss'} \right| + 2\epsilon + 2\epsilon\gamma \max_{s' \in \mathcal{S}} \bar{V}(s') \\
&\leq D^{\gamma \bar{V}}_{sa}(\hat{M}, \hat{\bar{M}}) + 2\epsilon \left(1 + \gamma \max_{s' \in \mathcal{S}} \bar{V}(s') \right),
\end{aligned}$$

which holds with probability at least $1-\delta$ and proves the Theorem for case (i).

(ii) If $(s,a) \in K \cap \bar{K}^c$, then we do not have an ϵ-close estimate of $\bar{T}^a_{s\cdot}$ and $\bar{R}^a_s$. Similarly to the proof of case (i), we upper bound sequentially the two terms of the right hand side of Equation A.12. With probability at least $1-\delta$, we have the following:

$$\begin{aligned}
\left| R^a_s - \bar{R}^a_s \right| &\leq \left| R^a_s - \hat{R}^a_s \right| + \left| \hat{R}^a_s - \bar{R}^a_s \right| \\
&\leq \epsilon + \max_{\bar{R} \in [0,1]} \left| \hat{R}^a_s - \bar{R} \right|. \qquad \text{(A.15)}
\end{aligned}$$

Similarly, with probability at least $1-\delta$, we have:

$$\begin{aligned}
\gamma \sum_{s' \in \mathcal{S}} V^*_{\bar{M}}(s') \left| T^a_{ss'} - \bar{T}^a_{ss'} \right| &\leq \gamma \sum_{s' \in \mathcal{S}} \bar{V}(s') \left(\left| T^a_{ss'} - \hat{T}^a_{ss'} \right| + \left| \hat{T}^a_{ss'} - \bar{T}^a_{ss'} \right| \right) \\
&\leq \gamma \max_{s' \in \mathcal{S}} \bar{V}(s') \epsilon + \gamma \max_{\bar{T} \in \mathcal{V}_S} \sum_{s' \in \mathcal{S}} \bar{V}(s') \left| \hat{T}^a_{ss'} - \bar{T}_{s'} \right|, \qquad \text{(A.16)}
\end{aligned}$$

where $\mathcal{V}_S$ is the set of probability vectors of size S, see Notation A.2, page 137. Combining inequalities A.15 and A.16, we get the following with probability at least $1-\delta$, by noticing $D^{\gamma V^*_{\bar{M}}}_{sa}(M, \bar{M})$ on the left hand side:

$$D_{sa}(M \| \bar{M}) \leq \max_{\bar{m} \in \mathcal{M}} D^{\gamma \bar{V}}_{sa}(\hat{M}, \bar{m}) + \epsilon \left(1 + \gamma \max_{s'} \bar{V}(s') \right),$$

which is the expected result for case (ii).

(iii) If $(s,a) \in K^c \cap \bar{K}^c$, then we do not have ϵ-close estimates of both tasks. In such a case, the result

$$D_{sa}(M \| \bar{M}) \leq \max_{m, \bar{m} \in \mathcal{M}^2} D^{\gamma \bar{V}}_{sa}(m, \bar{m})$$

is straightforward by remarking that, as a consequence of Inequality A.11, we have that $V^*_{\bar{M}}(s) \leq \bar{V}(s)$ with probability at least $1 - \delta$. □

Proof of Lemma 5.4, page 110. Let L be the functional operator that maps any function $d \in \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$ to

$$\begin{aligned} Ld: \ \mathcal{S} \times \mathcal{A} &\to \mathbb{R} \\ (s,a) &\mapsto \begin{cases} \hat{D}_{sa}(M\|\bar{M}) + \gamma \left(\sum_{s' \in \mathcal{S}} \hat{T}^a_{ss'} \max_{a' \in \mathcal{A}} d(s',a') + \epsilon \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} d(s',a') \right) \\ \hfill \text{if } (s,a) \in K, \\ \hat{D}_{sa}(M\|\bar{M}) + \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} d(s',a') \text{ otherwise.} \end{cases} \end{aligned}$$

Let f and g be two functions from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$. If $(s,a) \in K$, we have

$$\begin{aligned} Lf_{sa} - Lg_{sa} &= \gamma \sum_{s' \in \mathcal{S}} T^a_{ss'} \left(\max_{a' \in \mathcal{A}} f_{s'a'} - \max_{a' \in \mathcal{A}} g_{s'a'} \right) + \gamma\epsilon \left(\max_{(s',a') \in \mathcal{S} \times \mathcal{A}} f_{s'a'} - \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} g_{s'a'} \right) \\ &\leq (\gamma + \gamma\epsilon) \left(\max_{(s',a') \in \mathcal{S} \times \mathcal{A}} f_{s'a'} - \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} g_{s'a'} \right) \\ &\leq \gamma(1+\epsilon) \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} (f_{s'a'} - g_{s'a'}) \\ &\leq \gamma(1+\epsilon) \|f - g\|_\infty . \end{aligned}$$

If $(s,a) \notin K$, we have

$$\begin{aligned} Lf_{sa} - Lg_{sa} &= \gamma \left(\max_{(s',a') \in \mathcal{S} \times \mathcal{A}} f_{s'a'} - \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} g_{s'a'} \right) \\ &\leq \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} (f_{s'a'} - g_{s'a'}) \\ &= \gamma(1+\epsilon) \|f - g\|_\infty . \end{aligned}$$

In both cases, $\|Lf - Lg\|_\infty \leq \gamma(1+\epsilon)\|f - g\|_\infty$. If $\gamma(1+\epsilon) < 1$, L is a contraction mapping in the metric space $(\mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R}), \|\cdot\|_\infty)$. This metric space being complete and non-empty, it follows from Banach fixed-point theorem that $d = Ld$ admits a single solution. □

Proof of Theorem 5.8, page 110. Consider two MDPs $M, \bar{M} \in \mathcal{M}$. Before proving the result, notice that we shall put ourselves in the case of Theorem 5.7, page 107, for the upper bound on the pseudometric between models $\hat{D}_{sa}(M\|\bar{M})$ to be true upper bounds with probability at least $1 - \delta$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. As seen in the proof of Theorem 5.7, this implies learning any reward or transition function with precision ϵ in $\mathcal{L}_1$-norm with probability at least $1 - \delta/(4SA)$.

The proof is done by induction, by calculating the values of $d_{sa}(M\|\bar{M})$ and $\hat{d}_{sa}(M\|\bar{M})$ following the value iteration algorithm. Those values can respectively be computed via the sequences of iterates $(d^n)_{n\in\mathbb{N}}$ and $(\hat{d}^n)_{n\in\mathbb{N}}$ defined as follows for all $(s,a)\in\mathcal{S}\times\mathcal{A}$:

$$\begin{aligned}
d^0_{sa}(M\|\bar{M}) &= 0\\
d^{n+1}_{sa}(M\|\bar{M}) &= D_{sa}(M\|\bar{M}) + \gamma\sum_{s'\in\mathcal{S}} T^a_{ss'}\max_{a'\in\mathcal{A}} d^n_{s'a'}(M\|\bar{M})\,,
\end{aligned}$$

and,

$$\hat{d}^0_{sa}(M\|\bar{M}) = 0,$$

$$\hat{d}^{n+1}_{sa}(M\|\bar{M}) = \begin{cases} \hat{D}_{sa}(M\|\bar{M}) + \gamma\left(\sum_{s'\in\mathcal{S}} \hat{T}^a_{ss'}\max_{a'\in\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M}) + \epsilon\max_{(s',a')\in\mathcal{S}\times\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\right) & \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{if } (s,a)\in K, \\ \hat{D}_{sa}(M\|\bar{M}) + \gamma\max_{(s',a')\in\mathcal{S}\times\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M}) \text{ otherwise.} \end{cases}$$

The proof at rank $n=0$ is trivial. Let us assume the proposition $d^n_{sa}(M\|\bar{M})\leq\hat{d}^n_{sa}(M\|\bar{M})$ true at rank $n\in\mathbb{N}$ for all $(s,a)\in\mathcal{S}\times\mathcal{A}$ and consider rank $n+1$. There are two cases, depending on the fact that (s,a) is in K or not.

If $(s,a)\in K$, we have

$$\begin{aligned}
d^{n+1}_{sa}(M\|\bar{M}) - \hat{d}^{n+1}_{sa}(M\|\bar{M}) = {} & D_{sa}(M\|\bar{M}) - \hat{D}_{sa}(M\|\bar{M})\\
& + \gamma\sum_{s'\in\mathcal{S}}\left(T^a_{ss'}\max_{a'\in\mathcal{A}} d^n_{s'a'}(M\|\bar{M}) - \hat{T}^a_{ss'}\max_{a'\in\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\right)\\
& - \gamma\epsilon\max_{(s',a')\in\mathcal{S}\times\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\,.
\end{aligned}$$

Using Theorem 5.7, page 107, we have that $\hat{D}_{sa}(M\|\bar{M})$ is an upper bound on $D_{sa}(M\|\bar{M})$ with probability at least $1-\delta$. Hence

$$\mathbf{Pr}\left(D_{sa}(M\|\bar{M}) - \hat{D}_{sa}(M\|\bar{M}) \leq 0\right) \geq 1-\delta.$$

This plus the fact that $d^n_{sa}(M\|\bar{M})\leq\hat{d}^n_{sa}(M\|\bar{M})$, $\forall\,(s,a)\in\mathcal{S}\times\mathcal{A}$ by induction hypothesis, we have with probability at least $1-\delta$,

$$\begin{aligned}
d^{n+1}_{sa}(M\|\bar{M}) - \hat{d}^{n+1}_{sa}(M\|\bar{M}) &\leq \gamma\sum_{s'\in\mathcal{S}}\max_{a'\in\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\left(T^a_{ss'} - \hat{T}^a_{ss'}\right)\\
&\quad - \gamma\epsilon\max_{(s',a')\in\mathcal{S}\times\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\\
&\leq \gamma\max_{(s',a')\in\mathcal{S}\times\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\sum_{s'\in\mathcal{S}}\left(T^a_{ss'} - \hat{T}^a_{ss'}\right)\\
&\quad - \gamma\epsilon\max_{(s',a')\in\mathcal{S}\times\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\\
&\leq \gamma\max_{(s',a')\in\mathcal{S}\times\mathcal{A}}\hat{d}^n_{s'a'}(M\|\bar{M})\left(\left\|T-\hat{T}\right\|_1 - \epsilon\right)\,.
\end{aligned}$$

Since $\mathbf{Pr}\left(\left\|T - \hat{T}\right\|_1 \leq \epsilon\right) \geq 1 - \delta$, we have with probability at least $1 - \delta$,

$$d_{sa}^{n+1}(M\|\bar{M}) - \hat{d}_{sa}^{n+1}(M\|\bar{M}) \leq \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'}^{n}(M\|\bar{M})\,(\epsilon - \epsilon) = 0,$$

which concludes the proof in the first case case.

Conversely, if $(s, a) \notin K$, we have

$$\begin{aligned} d_{sa}^{n+1}(M\|\bar{M}) - \hat{d}_{sa}^{n+1}(M\|\bar{M}) = D_{sa}(M\|\bar{M}) - \hat{D}_{sa}(M\|\bar{M}) + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^{a} \max_{a' \in \mathcal{A}} d_{s'a'}^{n}(M\|\bar{M}) \\ - \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'}^{n}(M\|\bar{M})\,. \end{aligned}$$

Using the same reasoning than in case $(s, a) \in K$, we have with probability higher than $1 - \delta$,

$$\begin{aligned} d_{sa}^{n+1}(M\|\bar{M}) - \hat{d}_{sa}^{n+1}(M\|\bar{M}) &\leq \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^{a} \max_{a' \in \mathcal{A}} \hat{d}_{s'a'}^{n}(M\|\bar{M}) - \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'}^{n}(M\|\bar{M}) \\ &\leq \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'}^{n}(M\|\bar{M}) - \gamma \max_{(s',a') \in \mathcal{S} \times \mathcal{A}} \hat{d}_{s'a'}^{n}(M\|\bar{M}) \\ &= 0, \end{aligned}$$

which concludes the proof in the second case. □

Proof of Theorem 5.10, page 114. The cost of LRMAX is constant on most time steps since the action is greedily chosen with respect to the upper bound on the optimal Q-function, which is a lookup table. Let $N \in \mathbb{N}$ be the number of source tasks that have been learned by LRMAX during a lifelong RL experiment. When updating a new state-action pair, *i.e.*, labeling it as a known pair, the algorithm performs $2N$ DP (Dynamic Programming) computations to update the induced Lipschitz bounds (Equation 5.14, page 110) plus one DP computation to update the total-bound (Equation 5.11, page 106). In total, we apply $(2N + 1)$ DP computations for each state-action pair update. As at most SA state-action pairs are updated during the exploration of the current MDP, the total number of DP computations is at most $SA(2N+1)$, for which we use the value iteration algorithm.

In Theorem 2.5, page 19, we reported the minimum number of iterations needed by the value iteration algorithm to estimate a value function (or Q-function in our case) that is ϵ_Q-close to the optimum in maximum norm. This minimum number is given by

$$\left\lceil \frac{1}{\ln(\gamma)} \ln\left(\frac{\epsilon_Q(1-\gamma)}{R_{\max}}\right) \right\rceil .$$

As the complexity should be expressed as a polynomial in relevant quantities for the result to be PAC-MDP, we will consider a slightly larger number of iterations that verifies this constraint, namely,

$$\left\lceil \frac{1}{1-\gamma} \ln\left(\frac{1}{\epsilon_Q(1-\gamma)}\right) \right\rceil .$$

As stated in Theorem 2.6, page 20, each iteration has a cost S^2A. Overall, the cost of all the DP computations in a complete run of LRMAX is

$$\tilde{\mathcal{O}}\left(\frac{S^3A^2N}{1-\gamma}\ln\left(\frac{1}{\epsilon_Q(1-\gamma)}\right)\right).$$

This, plus the constant cost $\mathcal{O}(1)$ applied on each one of the τ decision epochs concludes the proof. □

Proof of Theorem 5.11, page 114. Consider $\left(M, \bar{M}, s, a\right) \in \mathcal{M}^2 \times \mathcal{S} \times \mathcal{A}$. By definition of the pseudometric between models, we have that:

$$\begin{aligned} D_{sa}(M \| \bar{M}) &= D_{sa}^{\gamma V^*_{\bar{M}}}(M, \bar{M}) \\ &= \left|R^a_s - \bar{R}^a_s\right| + \gamma \sum_{s' \in \mathcal{S}} V^*_{\bar{M}}(s') \left|T^a_{ss'} - \bar{T}^a_{ss'}\right| . \end{aligned}$$

The remainder of the proof consists in upper bounding the two terms of the right hand side of the previous definition, by considering the MDPs maximizing those quantities. For the reward model, we obviously have that

$$\left|R^a_s - \bar{R}^a_s\right| \leq 1 . \tag{A.17}$$

For the transition model, we have that

$$\begin{aligned} \sum_{s' \in \mathcal{S}} V^*_{\bar{M}}(s') \left|T^a_{ss'} - \bar{T}^a_{ss'}\right| &\leq \frac{1}{1-\gamma} \sum_{s' \in \mathcal{S}} \left|T^a_{ss'} - \bar{T}^a_{ss'}\right| \\ &\leq \frac{1}{1-\gamma} \cdot 2 , \end{aligned} \tag{A.18}$$

given that the maximum $\mathcal{L}_1$-norm between any two probability distributions is equal to 2. By combining Equations A.17 and A.18, we get the following inequalities:

$$\begin{aligned} D_{sa}(M \| \bar{M}) &\leq 1 + \gamma \frac{2}{1-\gamma} \\ &\leq \frac{1+\gamma}{1-\gamma} , \end{aligned}$$

which concludes the proof. □

Proof of Theorem 5.12, page 116. Consider an algorithm producing ϵ-accurate model estimates $\hat{D}_{sa}(M \| \bar{M})$ for a subset K of $\mathcal{S} \times \mathcal{A}$ after interacting with any two MDPs $M, \bar{M} \in \mathcal{M}$. Assume $\hat{D}_{sa}(M \| \bar{M})$ to be an upper bound of $D_{sa}(M \| \bar{M})$ for any $(s, a) \notin K$. These assumptions are guaranteed with high probability by Theorem 5.7, page 107 while running Algorithm 10, page 113 in the lifelong RL setting. Then, for any $(s, a) \in \mathcal{S} \times \mathcal{A}$ and any two

MDPs $M, \bar{M} \in \mathcal{M}$, we have that

$$\begin{array}{rcll}\hat{D}_{sa}(M\|\bar{M}) & = & D_{sa}(M\|\bar{M}) \pm \epsilon & \text{if } (s,a) \in K \\ \hat{D}_{sa}(M\|\bar{M}) & \geq & D_{sa}(M\|\bar{M}) & \text{else.}\end{array}$$

Particularly, $\hat{D}_{sa}(M\|\bar{M}) + \epsilon \geq D_{sa}(M\|\bar{M})$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ and any $M, \bar{M} \in \mathcal{M}$. By definition of $D_{\max}(s,a)$, this implies that, for all $(s,a) \in \mathcal{S} \times \mathcal{A}$,

$$\max_{M,\bar{M} \in \tilde{\mathcal{M}}} \hat{D}_{sa}(M\|\bar{M}) + \epsilon \geq D_{\max}(s,a)\,, \tag{A.19}$$

where $\tilde{M}$ is the set of possible tasks in the considered lifelong RL experiment. Consider $\hat{\mathcal{M}}$, the set of sampled MDPs which allows to define $\hat{D}_{\max}(s,a) = \max_{M,\bar{M} \in \hat{\mathcal{M}}} \hat{D}_{sa}(M\|\bar{M})$ as the maximum model distance for all the experienced MDPs at $(s,a) \in \mathcal{S} \times \mathcal{A}$. We have that

$$\hat{D}_{\max}(s,a) = \max_{M,\bar{M} \in \tilde{\mathcal{M}}} \hat{D}_{sa}(M\|\bar{M})\,,$$

only if two MDPs maximizing the right hand side of this equation belong to $\hat{M}$. If it is the case, then Equation A.19 imply that

$$\hat{D}_{\max}(s,a) + \epsilon \geq D_{\max}(s,a)\,. \tag{A.20}$$

Overall, we require the two MDPs maximizing $\max_{M,\bar{M} \in \tilde{\mathcal{M}}} \hat{D}_{sa}(M\|\bar{M})$ to be sampled for Equation A.20 to hold. Let us now derive the probability that those two MDPs have been sampled. We note them M_1 and M_2. There may exist more candidates for the maximization but, for the sake of generality, we put ourselves in the case where only two MDPs achieve the maximization. Let us consider drawing $m \in \mathbb{N}$ tasks. We note p_1 (respectively p_2) the probability of sampling M_1 (respectively M_2) each time a task is sampled. We note X_1 (respectively X_2) the random variable of the first occurrence of the task M_1 (respectively M_2) among the m trials. Hence, the probability of sampling M_1 for the first time at trial $k \in \{1, \dots, m\}$ is given by the geometric law and is equal to

$$\mathbf{Pr}\left(X_1 = k\right) = p_1 \left(1 - p_1\right)^{k-1}\,.$$

Additionally, the probability of sampling M_1 at least once in the first m trials is given by the cumulative distribution function:

$$\mathbf{Pr}\left(X_1 \leq m\right) = 1 - \left(1 - p_1\right)^{m}\,. \tag{A.21}$$

We are interested in the probability of the event that M_1 *and* M_2 have been sampled in the m first trials, *i.e.* $\mathbf{Pr}\left(X_1 \leq m \cap X_2 \leq m\right)$. Following the rule of addition for probabilities, we have that,

$$\mathbf{Pr}\left(X_1 \leq m \cap X_2 \leq m\right) = \mathbf{Pr}\left(X_1 \leq m\right) + \mathbf{Pr}\left(X_2 \leq m\right) - \mathbf{Pr}\left(X_1 \leq m \cup X_2 \leq m\right)\,.$$

Given that the event of sampling either M_1 or M_2 during a single trial happens with probability $p_1 + p_2$, we have by analogy with Equation A.21 that $\mathbf{Pr}\left(X_1 \leq m \cup X_2 \leq m\right) =$

$1-(1-(p_1+p_2))^m$. As a result, the following holds:

$$\begin{aligned}\mathbf{Pr}\left(X_1 \leq m \cap X_2 \leq m\right) &= 1-(1-p_1)^m + 1-(1-p_2)^m - (1-(1-(p_1+p_2))^m)\\ &= 1-(1-p_1)^m - (1-p_2)^m + (1-(p_1+p_2))^m\\ &\geq 1-2(1-p_{\min})^m + (1-2p_{\min})^m\,.\end{aligned}$$

As said earlier, Equation A.20 holds if M_1 and M_2 have been sampled during the first m trials, which imply that the probability for Equation A.20 to hold is at least equal to the probability of sampling both tasks. Formally,

$$\begin{aligned}\mathbf{Pr}\left(\hat{D}_{\max}(s,a)+\epsilon \geq D_{\max}(s,a)\right) &\geq \mathbf{Pr}\left(X_1 \leq m \cap X_2 \leq m\right)\\ &\geq 1-2(1-p_{\min})^m + (1-2p_{\min})^m\,.\end{aligned}$$

In turn, if m verifies $2(1-p_{\min})^m-(1-2p_{\min})^m \leq \delta$, then $1-2(1-p_{\min})^m+(1-2p_{\min})^m \geq 1-\delta$ and $\mathbf{Pr}\left(\hat{D}_{\max}(s,a)+\epsilon \geq D_{\max}(s,a)\right) \geq 1-\delta$, which concludes the proof. $\square$

Appendix B

Complete version of the MCTS algorithm

The MCTS procedure, presented in Algorithm 2, page 24, is a high level description and lacks implementation details. In this chapter, we provide a more thorough description of the algorithm. The main core of the procedure, mimicking Algorithm 2, page 24, is described in Algorithm 11. The remaining sub-processes, introduced in Section 2.2.3, page 20, are described in Algorithms 12, page 152, 13, page 152, 14, page 153 and 15, page 153, which respectively detail the selection, expansion, simulation and back-propagation steps.

Algorithm 11 MCTS (detailed procedure)

```
Set: MDP $\{\mathcal{S}, \mathcal{A}, T, r\}$; initial state distribution $\mathcal{T}_0$.
Input: generative model $\hat{M} = (\hat{T}, \hat{r})$; budget $B$; tree policy $\pi_{\text{tree}}$; default policy $\pi_{\text{default}}$;
discount factor $\gamma$; horizon $H$.
$s \sim \mathcal{T}_0$  # Set the initial state.
for $t \in \{1, \dots, H\}$ do
  $\nu_s \leftarrow$ decisionNode($s$)  # Initialize the search tree with the current state $s$.
  for $i \in \{1, \dots, B\}$ do
    initialize an empty list of collected rewards $\mathscr{R}$
    $\nu_{\tilde{s}}, a, s', \mathscr{R} \leftarrow$ selection($\nu_s, \hat{M}, \pi_{\text{tree}}, \mathscr{R}$)  # Apply $\pi_{\text{tree}}$ to select a leaf node.
    $\nu_{\tilde{s}'} \leftarrow$ expansion($\nu_s, \nu_{\tilde{s}}, a, s'$)  # Expand the tree with a new decision node.
    $Z \leftarrow$ simulation($\tilde{s}', \pi_{\text{default}}, \hat{M}, \gamma, H$)  # Apply $\pi_{\text{default}}$ to generate a trajectory and
    record the collected return.
    backPropagation($\nu_{\tilde{s}'}, \mathscr{R}, Z, \gamma$)  # Update the values of the chance nodes encountered
    along the trajectory with the collected rewards and simulation return.
  end for
  $s' \sim T^a_{s\cdot}$  # Sample the next state.
  if $s'$ is terminal then
    break the for loop  # Stop the episode.
  end if
  $s \leftarrow s'$
end for
```

Selection. The selection method is recursively exploring the tree until it finds a leaf node (a node where every actions have not been sampled). It applies the tree policy π_{tree} and uses the generative model $\hat{M}$ to perform this exploration. The procedure ends once a decision

Algorithm 12 selection

```
**Input:** decision node $\nu_s$; generative model $\hat{M}$; tree policy $\pi_{\text{tree}}$; list of collected rewards $\mathscr{R}$.
if $\nu_s$ is a leaf node then
  $a \leftarrow$ non-sampled action at $\nu_s$
  $s' \sim \hat{T}^a_{s\cdot}$
  $r \leftarrow \hat{r}^a_{ss'}$
  return $(\nu_s, a, s', \mathscr{R})$  # Candidate found, no need to extend the selection process.
else
  $a = \pi_{\text{tree}}(s)$
  $s' \sim \hat{T}^a_{s\cdot}$
  $r \leftarrow \hat{r}^a_{ss'}$
  add $r$ to $\mathscr{R}$
  if $\nu_{s'}$ in children of $\nu^a_s$ then
    return selection$(\nu_{s'}, \hat{M}, \pi_{\text{tree}}, \mathscr{R})$  # Explore further by recursive call.
  else
    return $(\nu_s, a, s', \mathscr{R})$  # Candidate found, no need to extend the selection process.
  end if
end if
```

node yields a new state, either because it had an unexplored action or because a new state was sampled with $\hat{M}$ by applying an already explored action.

Algorithm 13 expansion

```
**Input:** root node $\nu_{s_0}$; leaf node $\nu_s$; sampled action $a$; sampled state $s'$.
Create node $\nu^a_s$ in children of $\nu_s$ if non-existent
Create node $\nu_{s'}$ in children of $\nu^a_s$
return $\nu_{s'}$
```

Expansion. The expansion method creates a new chance-decision node pair given the selected leaf node. The selected leaf node, action and resulting state returned by the selection method are given as input.

Simulation. The simulation method uses the generative model to generate a trajectory of length H and return the collected discounted return Z. Special attention should be brought in case of a finite MDP horizon where the length of the simulation should be reduced from the depth of the node in the tree. Additionally, if the decision node from which the simulation begins has multiple parents with multiple depth, several discounted returns should be returned, each corresponding to a different simulation length.

Back-propagation. The back-propagation method updates the list of collected discounted returns of all the chance node encountered along the whole simulation (starting from the root node of the tree).

Algorithm 14 simulation

```
Input: initial state $s$; default policy $\pi_{\text{default}}$; generative model $\hat{M}$; discount factor $\gamma$; horizon $H$.
$Z = 0$
for $t \in \{0, \ldots, H\}$ do
    $a = \pi_{\text{default}}(s)$
    $s' \sim \hat{T}^{a}_{s\cdot}$
    $r = \hat{r}^{a}_{ss'}$
    $Z \leftarrow Z + \gamma^{t} r$
    if $s'$ is terminal then
        break the for loop # Stop the simulation if a terminal state is reached.
    end if
    $s \leftarrow s'$
end for
return $Z$
```

Algorithm 15 backPropagation

```
Input: leaf decision node $\nu_s$; collected rewards $\mathscr{R}$ to reach $\nu_s$; simulation return $Z$; discounted return $\gamma$.
for $\nu^{a}_{\tilde{s}}$ in parents of $\nu_s$ do
    $\mathscr{R}' \leftarrow \mathscr{R}$
    $r \leftarrow$ last element of $\mathscr{R}'$ (remove that element from $\mathscr{R}'$)
    $Z \leftarrow r + \gamma Z$
    Append $Z$ to the list of sampled returns of $\nu^{a}_{\tilde{s}}$
    backPropagation($\nu_{\tilde{s}}, \mathscr{R}', Z, \gamma$) # Recursive call to the parent of the current chance node
end for
```

Appendix C

Additional experimental results for the OLTA algorithm

We provide the readers with several additional experiments in similar settings as presented in Chapter 3, page 33, Section 3.6, page 53. The distinction is essentially based on the transition from discrete to continuous state spaces and vice-versa. In Chapter 3, page 33, we chose to mainly present the two extreme cases of the 1D track and the continuous PTSP for the theoretical interest of the first one and the complexity of the second one.

Continuous 1D track

In order to test the algorithm on a more complex setting than the discrete 1D track, we extended the latter to the continuous case. A comprehensive illustration of the environment is provided in Figure C.1. The width of the track is 50 and the state of the agent is its position

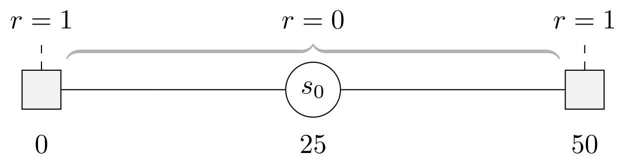


Figure C.1: Illustration of the continuous 1D track environment.

along that track, hence $\mathcal{S} \equiv [0, 50]$. The action space is kept unchanged, $\mathcal{A} = \{\text{Left, Right}\}$. Each action increases or decreases the position of the agent along the track with a magnitude of 1. The agent starts at the middle state $s_0 = 25$. The reward is zero everywhere except for the two terminal states 0 and 50 where it is +1. To the transition misstep probability presented in Chapter 3, page 33, we added a Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_{\text{noise}})$ to the resulting state after each transition. The simulations are performed with the following settings: $q \in \{0.0, 0.05, \cdots, 0.5\}$; $\sigma_{\text{noise}} = 0.1$; $B = 100$ (initial tree budget); $\pi_{\text{default}} = \pi_{\text{optimal}}$; $H = 50$ (simulation horizon for π_{default}); $C_p = .7$; $\gamma = 0.9$; and the generative model is the true model. The different decision criteria parameters were selected empirically and set to the following values: $\tau_{\text{SDV}} = 0.4$; $\tau_{\text{SDSD}} = 1$; $\tau_{\text{RDV}} = 5 \cdot 10^{-4}$. As in the continuous PTSP case, we reserve the development of SDM-OLTA in the continuous case for future work. We generated 1000 episodes for each transition misstep probability.

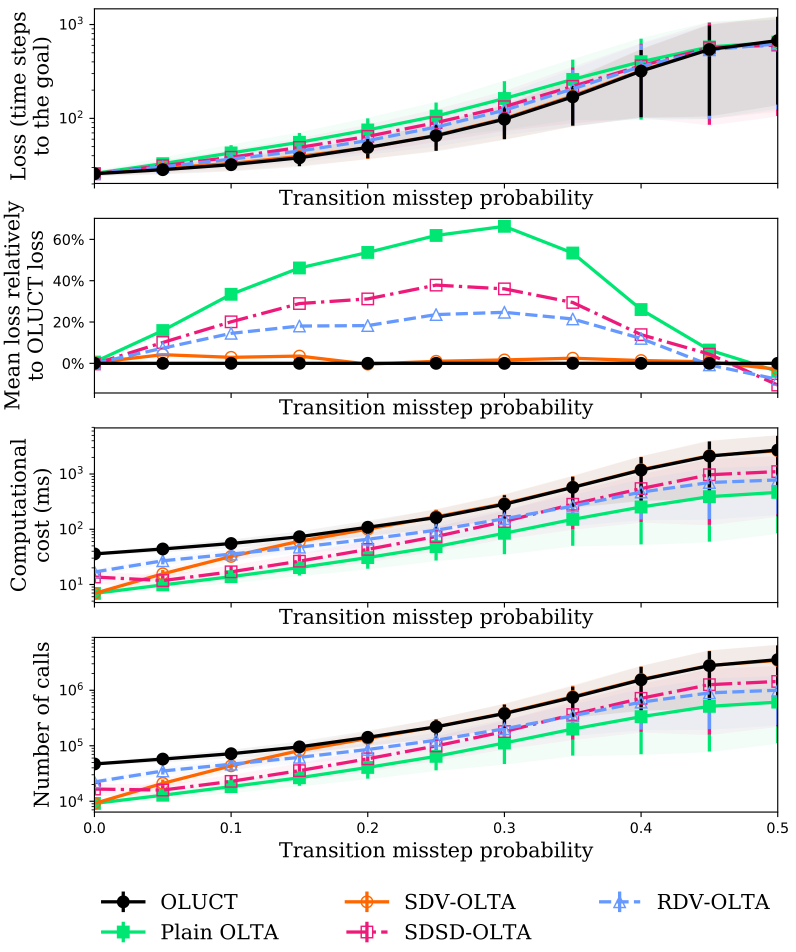


Figure C.2: Comparison between OLUCT and OLTA on the continuous 1D track environment for varying values of the misstep probability q.

The results are presented in Figure C.2, page 156. Logarithmic scales are used for display purposes. As in the discrete case, OLTA achieves comparable loss as OLUCT. Particularly, SDV-OLTA performs as well as OLUCT on the whole range of misstep probabilities. In this setting, SDSD-OLTA and RDV-OLTA achieved an intermediate loss between Plain OLTA and OLUCT. In terms of computational cost, two behaviors are observed. In the case of SDV-OLTA, the computational gain is relevant for low transition misstep probabilities and catches up with OLUCT as the latter increases. This allows the algorithm to achieve the same score as OLUCT. In the case of SDSD-OLTA and RDV-OLTA, the computational gain seems to be constant on the whole range of transition misstep probabilities. However, the reached lower performance accounts for the fact that, as for Plain OLTA, the decision criteria do not adapt well to the stochasticity increase, causing the algorithms to discard less trees than needed. Notice that the computational cost achieved by the SDSD-OLTA algorithm is greater for the transition misstep probability $q = 0$ than for $q = 0.05$. This is due to the fact that its criterion computes the distance between the current state of the agent and the empirical mean of the state distribution normalized by the variance. This difference comes

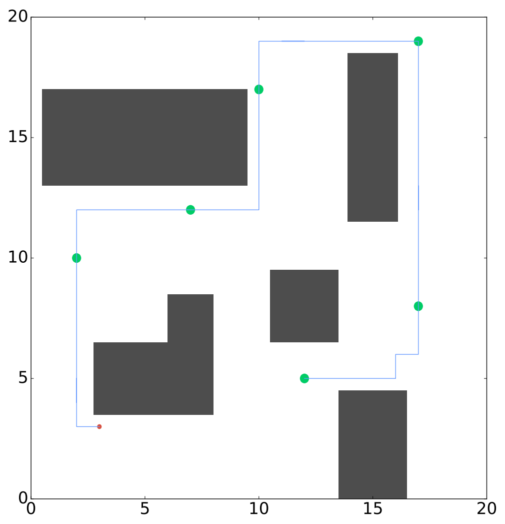


Figure C.3: Illustration of the discrete Physical Traveling Salesman Problem. A trajectory derived by an OLUCT algorithm is displayed in green. The starting point is displayed in red, the waypoints in green and the walls in gray.

from the fact that the distribution is mono-modal for $q = 0$ and bi-modal otherwise. Indeed, in the latter case, the variance increases, causing the normalization to decrease the value of the computed distance. Additionally, the empirical mean does not correspond to the mean of a mode but a point between the two mode means, which interacts in the opposite way: increasing the computed distance. In this setting, the interaction of the two mechanisms results in less sub-trees approvals for $q = 0$. In the discrete case, it does not occur since the current state lies exactly on the mean for $q = 0$ because no Gaussian noise is added to the state transition. As a result, the distance is always zero.

Discrete Physical Traveling Salesman Problem

We restricted the PTSP environment to the discrete case. The resulting problem is a grid-world navigation problem as illustrated in Figure C.3. As in the continuous case, the state of the agent is characterized by $s = (x, y, \theta, v) \in \mathbb{R}^4$, respectively the position in the 2D grid-world, the orientation and the velocity. In our case, we set the velocity to 1 so that the agent only has access to adjacent cells. The action space is $\mathcal{A} = \{\text{Right, Up, Left, Down}\}$, each action being the direction of the next adjacent cell reached by the agent. The reward is set to $+1$ when a waypoint is reached for the first time and to 0 elsewhere. We introduce the same misstep probability $q \in [0, 1]$ as in the continuous PTSP which is the probability for another action to be undertaken instead of the current one. The simulations are performed with the following settings: $s_0 = (3, 3, 0, 1)$; $q \in \{0.0, 0.05, \cdots, 0.5\}$; $B = 200$ (initial tree budget); $\pi_{\text{default}} = \pi_{\text{go-straight}}$ that applies no orientation variation; $H = 20$ (simulation horizon for

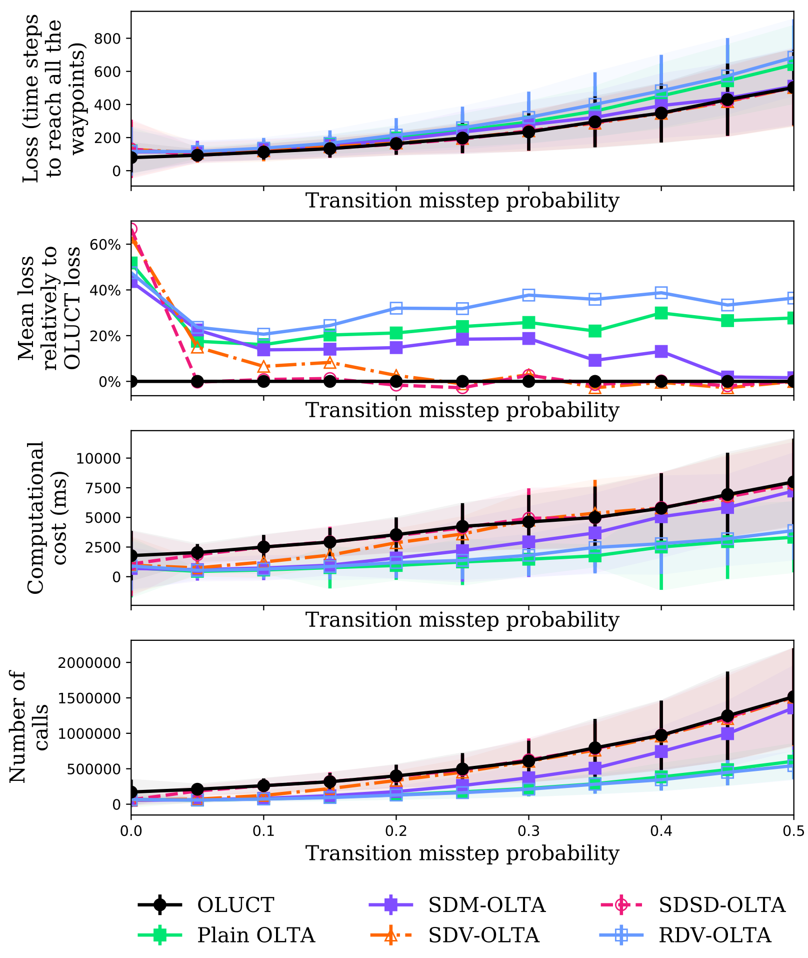


Figure C.4: Comparison between OLUCT and OLTA on the discrete PTSP for varying values of the misstep probability q.

π_{default}); $C_p = 0.7$; $\gamma = 0.99$. The provided map is the one depicted in Figure C.3 with six waypoints. The different decision criteria parameters were selected empirically and set as follows: $\tau_{\text{SDM}} = 0.7$, $\tau_{\text{SDV}} = 0.2$, $\tau_{\text{SDSD}} = 1.5$, and $\tau_{\text{RDV}} = 0.1$. Additionally, we provided Plain OLTA with the ability to discard a sub-tree if the recommended action was not available, *i.e.*, leading to a wall. We generated 1000 episodes for each transition misstep probability.

The results are presented in Figure C.4. As in the continuous case, OLTA achieves comparable loss as OLUCT. Particularly, SDV-OLTA and SDSD-OLTA had a very similar performance on most of the range of misstep probabilities. SDM-OLTA, Plain OLTA and RDV-OLTA achieved poorer performance but still comparable given the high variance of the losses. In terms of computational cost, all the variations of OLTA outperform OLUCT with an approximately constant gain. For each one of them, the consequence of this gain was the increasing of the achieved loss, so that each algorithm attained a different compromise between performance and computational cost gain.

Appendix D

Additional information on the experiments of Chapter 4, Section 4.5

We here provide additional technical information about the experiments run for the RATS algorithm in Section 4.5, page 85. The computing infrastructure used was a laptop using four 64-bit CPU (model: Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz). The collected samples sizes and number of evaluation runs for each experiment is summarized in Table D.1.

Experiment	Number of experiment repetitions	Number of episodes	Maximum length of an episode	Upper bound on the number of computed transition samples (s, a, r, s')
Non-stationary Bridge Figure 4.3, page 85	3 (one per agent)	96	10	89,579,520

Table D.1: Summary of the number of experiment repetition, number of sampled tasks, number of episodes, maximum length of episodes and upper bounds on the number of collected samples.

The displayed confidence intervals in Figure 4.4, page 87 is 50% of the estimated confidence interval $\bar{\sigma}$ computed with respect to the following formula:

$$\bar{\sigma} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \quad \text{where,} \quad \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

with $D = \{x_i\}_{i=1}^{N}$ the set of the collected data (discounted return in this case). No data were excluded neither pre-computed. Hyper-parameters were determined to our appreciation, they may be sub-optimal but we found the results convincing enough to display interesting behaviors.

Appendix E

Additional information on the experiments of Chapter 5, Section 5.6

For the experiments run in Chapter 5, page 91, Section 5.6, page 117, the computing infrastructure used was a laptop using a single 64-bit CPU (model: Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz). The collected samples sizes and number of evaluation runs for each experiment is summarized in Table E.1.

Task	Number of experiment repetitions	Number of sampled tasks	Number of episodes	Maximum length of episodes	Total number of samples (s, a, r, s')
"Tight" task of Figures, 5.5a, 5.5b, and 5.5c, page 118	10	15	2000	10	3,000,000
"Tight" task of Figure 5.5d, page 118	100	2	2000	10	4,000,000

Table E.1: Summary of the number of experiment repetition, number of sampled tasks, number of episodes, maximum length of episodes and number of collected samples.

The displayed confidence intervals for any curve presented in the paper is the 95% confidence interval (Neyman, 1937) on the displayed mean. No data were excluded neither precomputed. Hyper-parameters were determined to our appreciation, they may be sub-optimal but we found the results convincing enough to display interesting behaviors.

Bibliography

Abbasi, Yasin, Peter L. Bartlett, Varun Kanade, Yevgeny Seldin, and Csaba Szepesvári (2013). "Online Learning in Markov Decision Processes with Adversarially Chosen Transition Probability Distributions". In: *Advances in Neural Information Processing Systems 27 (NeurIPS 2013)*, pp. 2508–2516 (cit. on p. 64).

Abdallah, Sherief and Michael Kaisers (2016). "Addressing Environment Non-stationarity by Repeating Q-learning Updates". In: *Journal of Machine Learning Research* 17.1, pp. 1582–1612 (cit. on p. 64).

Abel, David, Yuu Jinnai, Sophie Yue Guo, George Konidaris, and Michael L. Littman (2018). "Policy and Value Transfer in Lifelong Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pp. 20–29 (cit. on pp. 66, 95, 96, 99, 116–118).

Ammar, Haitham Bou, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls (2014). "An Automated Measure of MDP Similarity for Transfer in Reinforcement Learning". In: *Workshops at the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)* (cit. on pp. 94, 96).

Araya-López, Mauricio, Vincent Thomas, Olivier Buffet, and François Charpillet (2010). "A Closer Look at MOMDPs". In: *Proceedings of the 22nd International Conference on Tools with Artificial Intelligence (ICTAI 2010)*. Vol. 2. IEEE, pp. 197–204 (cit. on p. 65).

Asadi, Kavosh, Dipendra Misra, and Michael L. Littman (2018). "Lipschitz Continuity in Model-Based Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)* (cit. on p. 70).

Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer (2002). "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2-3, pp. 235–256 (cit. on pp. 25, 43).

Auger, David, Adrien Couëtoux, and Olivier Teytaud (2013). "Continuous Upper Confidence Trees with Polynomial Exploration – Consistency". In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2013)*. Springer, pp. 194–209 (cit. on p. 36).

Baker, Chris, Gopal Ramchurn, Luke Teacy, and Nicholas Jennings (2016). "Factored Monte-Carlo Tree Search for Coordinating UAVs in Disaster Response". In: (cit. on p. 36).

Banerjee, Taposh, Miao Liu, and Jonathan P. How (2017). "Quickest Change Detection Approach to Optimal Control in Markov Decision Processes with Model Changes". In: *2017 American Control Conference (ACC)*. IEEE, pp. 399–405 (cit. on p. 66).

Bellemare, Marc G., Will Dabney, and Rémi Munos (2017). "A Distributional Perspective on Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)* (cit. on p. 47).

Bellemare, Marc G., Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47, pp. 253–279 (cit. on p. 115).

Bellman, Richard (1957). *Dynamic Programming*. Princeton, USA: Princeton University Press (cit. on pp. 18, 75, 78).

Boutin, Victor, Angelo Franciosini, Franck Ruffier, and Laurent Perrinet (2019). *Meaningful representations emerge from Sparse Deep Predictive Coding.* Tech. rep. (cit. on p. 126).

Bouzy, Bruno, Marc Métivier, and Damien Pellier (2011). "MCTS Experiments on the Voronoi Game". In: *Advances in Computer Games: 13th International Conference (ACG 2011).* Springer, pp. 96–107 (cit. on p. 36).

Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton (2012). "A survey of Monte Carlo Tree Search methods". In: *IEEE Transactions on Computational Intelligence and AI in Games (T-CIAIG)* 4.1, pp. 1–43 (cit. on pp. 24, 35).

Brunskill, Emma and Lihong Li (2013). "Sample Complexity of Multi-task Reinforcement Learning". In: *Proceedings of the 29th conference on Uncertainty in Artificial Intelligence (UAI 2013)* (cit. on pp. 94, 121).

— (2014). "PAC-inspired Option Discovery in Lifelong Reinforcement Learning". In: *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pp. 316–324 (cit. on pp. 66, 95, 96).

Bubeck, Sébastien and Rémi Munos (2010). "Open Loop Optimistic Planning". In: *Proceedings of the 23rd Conference on Learning Theory (COLT 2010)* (cit. on pp. 22, 36, 42, 49).

Carroll, James L. (2005). "Task Localization, Similarity, and Transfer; Towards a Reinforcement Learning Task Library System". MA thesis (cit. on p. 93).

Carroll, James L. and Kevin Seppi (2005). "Task Similarity Measures for Transfer in Reinforcement Learning Task Libraries". In: *Proceedings of the 5th International Joint Conference on Neural Networks (IJCNN 2005).* Vol. 2. IEEE, pp. 803–808 (cit. on p. 93).

Chanel, Caroline Ponzoni Carvalho (2013). "Planification de perception et de mission en environnement incertain: Application à la détection et à la reconnaissance de cibles par un hélicoptère autonome". PhD thesis. Université de Toulouse (cit. on p. 65).

Chaslot, G., M. Winands, J. Uiterwijk, H. Van Den Herik, and B. Bouzy (2007). "Progressive Strategies for Monte-Carlo Tree Search". In: *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pp. 655–661 (cit. on p. 41).

Choi, Samuel P. M., Dit-yan Yeung, and Nevin L. Zhang (1999). "Hidden-Mode Markov Decision Processes". In: *IJCAI Workshop on Neural, Symbolic, and Reinforcement Methods for Sequence Learning* (cit. on p. 64).

Choi, Samuel P. M., Dit-Yan Yeung, and Nevin L. Zhang (2000). "Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making". In: *Sequence Learning.* Springer, pp. 264–287 (cit. on p. 64).

Choi, Samuel Ping-Man, Nevin Lianwen Zhang, and Dit-Yan Yeung (2001). "Solving Hidden-Mode Markov Decision Problems". In: *Proceedings of the 8th International Conference on Artificial Intelligence and Statistics (AISTATS 2001).* Citeseer (cit. on p. 65).

Chung, Jen Jen, Nicholas R. J. Lawrance, and Salah Sukkarieh (2015). "Learning to soar: Resource-constrained exploration in reinforcement learning". In: *International Journal of Robotics Research* 34.2, pp. 158–172 (cit. on p. 69).

Couëtoux, Adrien, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard (2011). "Continuous Upper Confidence Trees". In: *Proceedings of the 4th In-*

ternational Conference on Learning and Intelligent Optimization (LION 2011). Springer, pp. 433–445 (cit. on pp. 35, 41).

Coulom, Rémi (2006). “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search”. In: *Proceedings of the 5th International Conference on Computer and Games (ICCG 2006)*. Springer, pp. 72–83 (cit. on p. 35).

— (2007). “Computing Elo Ratings of Move Patterns in the Game of Go”. In: *Computer Games Workshop (CGW 2007)* (cit. on p. 41).

Csáji, Balázs Csanád and László Monostori (2008). “Value Function Based Reinforcement Learning in Changing Markovian Environments”. In: *Journal of Machine Learning Research* 9.Aug, pp. 1679–1709 (cit. on p. 63).

Da Silva, Bruno C., Eduardo W. Basso, Ana L. C. Bazzan, and Paulo M. Engel (2006). “Dealing with Non-Stationary Environments using Context Detection”. In: *Proceedings of the 23rd International Conference on Machine Learning (ICML 2006)*. ACM, pp. 217–224 (cit. on pp. 65, 66).

Dabney, Will, Mark Rowland, Marc G. Bellemare, and Rémi Munos (2018). “Distributional Reinforcement Learning with Quantile Regression”. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)* (cit. on p. 70).

Dann, Christoph, Tor Lattimore, and Emma Brunskill (2017). “Unifying PAC and Regret: Uniform PAC Bounds for Episodic Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pp. 5713–5723 (cit. on p. 121).

De Maesschalck, Roy, Delphine Jouan-Rimbaud, and Désiré L. Massart (2000). “The Mahalanobis distance”. In: *Chemometrics and Intelligent Laboratory Systems* 50.1, pp. 1–18 (cit. on p. 53).

Dennett, Daniel C. (1975). “Why the Law of Effect will not Go Away”. In: *Journal for the Theory of Social Behaviour* 5.2, pp. 169–188 (cit. on p. 17).

Dick, Travis, Andras Gyorgy, and Csaba Szepesvari (2014). “Online Learning in Markov Decision Processes with Changing Cost Sequences”. In: *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pp. 512–520 (cit. on p. 63).

Doya, Kenji, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato (2002). “Multiple Model-based Reinforcement Learning”. In: *Neural Computation* 14.6, pp. 1347–1369 (cit. on p. 65).

Enzenberger, Markus, Martin Muller, Broderick Arneson, and Richard Segal (2010). “Fuego - an Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search”. In: *IEEE Transactions on Computational Intelligence and AI in Games (T-CIAIG)* 2.4, pp. 259–270 (cit. on p. 35).

Even-Dar, Eyal, Sham M. Kakade, and Yishay Mansour (2009). “Online Markov Decision Processes”. In: *Mathematics of Operations Research* 34.3, pp. 726–736 (cit. on p. 63).

Feldman, Zohar and Carmel Domshlak (2014). “Monte-Carlo Tree Search: To MC or to DP?” In: *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pp. 321–326 (cit. on p. 36).

Fern, Alan and Paul Lewis (2011). “Ensemble Monte-Carlo Planning: An Empirical Study”. In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)* (cit. on p. 37).

Fernández, Fernando and Manuela Veloso (2006). “Probabilistic Policy Reuse in a Reinforcement Learning Agent”. In: *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006).* ACM, pp. 720–727 (cit. on p. 93).

Ferns, Norm, Prakash Panangaden, and Doina Precup (2004). “Metrics for Finite Markov Decision Processes”. In: *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence (UAI 2004).* AUAI Press, pp. 162–169 (cit. on pp. 95, 97).

Fudenberg, Drew and Jean Tirole (1991). “Game Theory”. In: *Cambridge, Massachusetts* 393.12, p. 80 (cit. on p. 77).

Gelly, Sylvain and David Silver (2007). “Combining Online and Offline Knowledge in UCT”. In: *Proceedings of the 24th International Conference on Machine Learning (ICML 2007).* ACM, pp. 273–280 (cit. on p. 35).

— (2008). “Achieving Master Level Play in 9 x 9 Computer Go”. In: *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008).* Vol. 8, pp. 1537–1540 (cit. on p. 35).

— (2011). “Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go”. In: *Artificial Intelligence* 175.11, pp. 1856–1875 (cit. on p. 36).

Ghallab, Malik, Dana Nau, and Paolo Traverso (2014). “The Actor’s View of Automated Planning and Acting: A Position Paper”. In: *Artificial Intelligence* 208, pp. 1–17 (cit. on p. 126).

— (2016). *Automated Planning and Acting.* Cambridge University Press (cit. on pp. 2, 5, 6).

Hadoux, Emmanuel (2015). “Markovian sequential decision-making in non-stationary environments: application to argumentative debates”. PhD thesis. UPMC, Sorbonne Universités CNRS (cit. on p. 66).

Hadoux, Emmanuel, Aurélie Beynier, and Paul Weng (2014a). “Sequential Decision-Making under Non-stationary Environments via Sequential Change-point Detection”. In: *Learning over Multiple Contexts (LMCE): workshop at the the 13th European Conference on Machine Learning (ECML 2014)* (cit. on p. 66).

— (2014b). “Solving Hidden-Semi-Markov-Mode Markov Decision Problems”. In: *Proceedings of the 8th International Conference on Scalable Uncertainty Management (SUM 2014).* Springer, pp. 176–189 (cit. on p. 65).

Hallak, Assaf, Dotan Di Castro, and Shie Mannor (2015). “Contextual Markov Decision Processes”. In: *Proceedings of the 12th European Workshop on Reinforcement Learning (EWRL 2015)* (cit. on p. 96).

Heusner, Manuel (2011). “UCT for Pac-Man”. Bachelor thesis. University of Basel (cit. on pp. 37, 39).

Hostetler, Jesse, Alan Fern, and Tom Dietterich (2014). “State Aggregation in Monte Carlo Tree Search”. In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)* (cit. on p. 36).

Howard, R. A. (1963). “Semi-Markovian Decision Processes”. In: *Proceedings of International Statistical Institute, Ottawa, Canada* (cit. on p. 68).

Iyengar, Garud N. (2005). “Robust Dynamic Programming”. In: *Mathematics of Operations Research* 30.2, pp. 257–280 (cit. on pp. 62, 75, 77, 78, 89).

Jaulmes, Robin, Joelle Pineau, and Doina Precup (2005). “Learning in non-stationary Partially Observable Markov Decision Processes”. In: *ECML Workshop on Reinforcement Learning in non-stationary environments.* Vol. 25, pp. 26–32 (cit. on p. 65).

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). “Planning and acting in partially observable stochastic domains”. In: *Artificial Intelligence* 101.1, pp. 99–134 (cit. on pp. 46, 48, 65).

Kalmár, Zsolt, Csaba Szepesvári, and András Lőrincz (1998). “Module-Based Reinforcement Learning: Experiments with a Real Robot”. In: *Autonomous Robots* 5.3-4, pp. 273–295 (cit. on pp. 63, 88).

Katehakis, Michael N. and Arthur F. Veinott Jr. (1987). “The Multi-Armed Bandit Problem: Decomposition and Computation”. In: *Mathematics of Operations Research* 12.2, pp. 262–268 (cit. on p. 25).

Keller, Thomas and Patrick Eyerich (2012). “PROST: Probabilistic planning based on UCT”. In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)* (cit. on pp. 35, 36).

Keller, Thomas and Malte Helmert (2013). “Trial-Based Heuristic Tree Search for Finite Horizon MDPs”. In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)* (cit. on pp. 22, 35, 36, 42, 76).

Kleinberg, Robert, Aleksandrs Slivkins, and Eli Upfal (2008). “Multi-Armed Bandits in Metric Spaces”. In: *Proceedings of the 40th annual ACM symposium on Theory of computing.* ACM, pp. 681–690 (cit. on p. 70).

Kocsis, Levente and Csaba Szepesvári (2006). “Bandit Based Monte Carlo Planning”. In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2006).* Vol. 6. Springer, pp. 282–293 (cit. on pp. 24, 26, 50, 128, 129).

Kok, Adrianus Leendert, Erwin W. Hans, and Johannes M.J. Schutten (2012). “Vehicle routing under time-dependent travel times: The impact of congestion avoidance”. In: *Computers & Operations Research* 39.5, pp. 910–918 (cit. on p. 69).

Konidaris, George, Ilya Scheidwasser, and Andrew Barto (2012). “Transfer in Reinforcement Learning via Shared Features”. In: *Journal of Machine Learning Research* 13.May, pp. 1333–1371 (cit. on p. 94).

Lazaric, Alessandro (2012). “Transfer in Reinforcement Learning: a Framework and a Survey”. In: *Reinforcement Learning.* Springer, pp. 143–173 (cit. on p. 93).

Lazaric, Alessandro, Marcello Restelli, and Andrea Bonarini (2008). “Transfer of Samples in Batch Reinforcement Learning”. In: *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pp. 544–551 (cit. on pp. 93, 96).

Lecarpentier, Erwan, David Abel, Kavosh Asadi, Yuu Jinnai, Emmanuel Rachelson, and Michael L Littman (2020). *Lipschitz Lifelong Reinforcement Learning.* Tech. rep. (cit. on p. 3).

Lecarpentier, Erwan, Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson (2018). “Open Loop Execution of Tree Search Algorithms”. In: *Proceedings of the 27th International Joint Conferences on Artificial Intelligence (IJCAI 2018)* (cit. on p. 2).

Lecarpentier, Erwan and Emmanuel Rachelson (2019). “Non-Stationary Markov Decision Processes, a Worst-Case Approach using Model-Based Reinforcement Learning”. In: *Ad-*

vances in Neural Information Processing Systems 32 (NeurIPS 2019), pp. 7214–7223 (cit. on p. 2).

Lecarpentier, Erwan, Sebastian Rapp, Marc Melo, and Emmanuel Rachelson (2017). "Empirical evaluation of a Q-Learning Algorithm for Model-free Autonomous Soaring". In: *12èmes Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA 2017)* (cit. on pp. 64, 69).

Levine, John, Clare Bates Congdon, Marc Ebner, Graham Kendall, Simon M. Lucas, Risto Miikkulainen, Tom Schaul, and Tommy Thompson (2013). "General Video Game Playing". In: *Artificial and Computational Intelligence in Games* 6, pp. 77–83 (cit. on p. 36).

Lim, Shiau Hong, Huan Xu, and Shie Mannor (2013). "Reinforcement Learning in Robust Markov Decision Processes". In: *Advances in Neural Information Processing Systems 27 (NeurIPS 2013)*, pp. 701–709 (cit. on pp. 63, 88).

Lotter, William, Gabriel Kreiman, and David Cox (2016). *Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning.* Tech. rep. (cit. on p. 126).

Mahmud, M. M., Majd Hawasly, Benjamin Rosman, and Subramanian Ramamoorthy (2013). *Clustering Markov Decision Processes for Continual Transfer.* Tech. rep. (cit. on pp. 94, 121).

Mansley, Chris, Ari Weinstein, and Michael L. Littman (2011). "Sample-Based Planning for Continuous Action Markov Decision Processes". In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)* (cit. on p. 35).

Munos, Rémi (2014). "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning". In: *Foundations and Trends in Machine Learning* 7.1, pp. 1–129 (cit. on p. 70).

Neyman, Jerzy (1937). "X—outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability". In: *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 236.767, pp. 333–380 (cit. on p. 161).

Ong, Sylvie C.W., Shao Wei Png, David Hsu, and Wee Sun Lee (2009). "POMDPs for Robotic Tasks with Mixed Observability". In: *Proceedings of the 5th Robotics: Science and Systems (RSS 2009).* Vol. 5, p. 4 (cit. on p. 65).

Pazis, Jason and Ronald Parr (2013). "PAC Optimal Exploration in Continuous Space Markov Decision Processes". In: *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013)* (cit. on p. 70).

Pazis, Jason, Ronald E. Parr, and Jonathan P. How (2016). "Improving PAC Exploration using the Median of Means". In: *Advances in Neural Information Processing Systems 29 (NeurIPS 2016)*, pp. 3898–3906 (cit. on p. 121).

Perez, Diego, Philipp Rohlfshagen, and Simon M. Lucas (2012a). "Monte Carlo Tree Search: Long-term Versus Short-term Planning". In: *Proceedings of the 8th Computational Intelligence and Games conference (CIG 2012).* IEEE, pp. 219–226 (cit. on pp. 36, 37, 39).

— (2012b). "The Physical Travelling Salesman Problem: WCCI 2012 Competition". In: *Proceedings of the 14th Congress on Evolutionary Computation (CEC 2012).* IEEE, pp. 1–8 (cit. on p. 53).

Perez Liebana, Diego, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas (2015). "Open Loop Search for General Video Game Playing". In: *Proceedings of the 17th*

Genetic and Evolutionary Computation Conference (GECCO 2015). ACM, pp. 337–344 (cit. on pp. 36, 37, 39).

Perez Liebana, Diego, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson (2015). “The 2014 General Video Game Playing Competition”. In: *IEEE Transactions on Computational Intelligence and AI in Games (T-CIAIG)* 8.3, pp. 229–243 (cit. on p. 36).

Pirotta, Matteo, Marcello Restelli, and Luca Bascetta (2015). “Policy gradient in Lipschitz Markov Decision Processes”. In: *Machine Learning* 100.2-3, pp. 255–283 (cit. on p. 70).

Powley, Edward J., Peter I. Cowling, and Daniel Whitehouse (2017). “Memory Bounded Monte-Carlo Tree Search”. In: *Proceedings of the 13th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2017)* (cit. on p. 37).

Prabuchandran, K. J., Nitin Singh, Pankaj Dayama, and Vinayaka Pandit (2019). *Change Point Detection for Compositional Multivariate Data.* Tech. rep. (cit. on p. 66).

Puterman, Martin L. (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons (cit. on pp. 2, 5, 8, 11, 12, 14, 15, 67–69, 72, 137).

Rachelson, Emmanuel and Michail G. Lagoudakis (2010). “On the Locality of Action Domination in Sequential Decision Making”. In: *Proceedings of the 11th International Symposium on Artificial Intelligence and Mathematics (ISAIM 2010)* (cit. on pp. 48, 70, 130).

Rao, Karun and Shimon Whiteson (2012). “V-MAX: Tempered Optimism for Better PAC Reinforcement Learning”. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pp. 375–382 (cit. on p. 121).

Rimmel, Arpad and Fabien Teytaud (2010). “Multiple overlapping tiles for contextual Monte Carlo tree search”. In: *Proceedings of the 1st International Conference on the Applications of Evolutionary Computation (EvoApplications 2010).* Springer, pp. 201–210 (cit. on p. 35).

Schultz, Wolfram (2015). “Neuronal Reward and Decision Signals: from Theories to Data”. In: *Physiological Reviews* 95.3, pp. 853–951 (cit. on p. 7).

Sigaud, Olivier and Freek Stulp (2019). “Policy search in continuous action domains: An overview”. In: *Neural Networks* (cit. on p. 28).

Silver, Daniel L., Qiang Yang, and Lianghao Li (2013). “Lifelong Machine Learning Systems: Beyond Learning Algorithms”. In: *AAAI Spring Symposium: Lifelong Machine Learning.* Vol. 13, p. 05 (cit. on pp. 66, 95).

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529.7587, p. 484 (cit. on p. 35).

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). “Mastering the Game of Go without Human Knowledge”. In: *Nature* 550.7676, p. 354 (cit. on pp. 37, 39).

Silver, David and Gerald Tesauro (2009). “Monte-Carlo Simulation Balancing”. In: *Proceedings of the 26th International Conference on Machine Learning (ICML 2009).* ACM, pp. 945–952 (cit. on p. 35).

Silver, David and Joel Veness (2010). “Monte Carlo Planning in Large POMDPs”. In: *Advances in Neural Information Processing Systems 24 (NeurIPS 2010)*, pp. 2164–2172 (cit. on pp. 36, 46, 66).

Sindhu, Padakandla, Prabuchandran K. J., and Bhatnagar Shalabh (2019). *Reinforcement Learning for Non-Stationary Environments.* Tech. rep. (cit. on p. 66).

Soemers, Dennis J.N.J., Chiara F. Sironi, Torsten Schuster, and Mark H.M. Winands (2016). “Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing”. In: *Proceedings of the 12th Computational Intelligence and Games conference (CIG 2016).* IEEE, pp. 1–8 (cit. on pp. 37, 39).

Song, Jinhua, Yang Gao, Hao Wang, and Bo An (2016). “Measuring the Distance Between Finite Markov Decision Processes”. In: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, pp. 468–476 (cit. on pp. 95, 97, 98).

Sorg, Jonathan and Satinder Singh (2009). “Transfer via Soft Homomorphisms”. In: *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009).* International Foundation for Autonomous Agents and Multiagent Systems, pp. 741–748 (cit. on p. 94).

Spratling, Michael W. (2017). “A Hierarchical Predictive Coding Model of Object Recognition in Natural Images”. In: *Cognitive Computation* 9.2, pp. 151–167 (cit. on p. 126).

Strehl, Alexander L., Lihong Li, and Michael L. Littman (2009). “Reinforcement Learning in Finite MDPs: PAC Analysis”. In: *Journal of Machine Learning Research* 10.Nov, pp. 2413–2444 (cit. on pp. 19, 30, 32, 105, 114, 127).

Sutton, Richard S. (1991). “Dyna, an Integrated Architecture for Learning, Planning, and Reacting”. In: *ACM SIGART Bulletin* 2.4, pp. 160–163 (cit. on p. 17).

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction.* MIT press, Cambridge (cit. on pp. 2, 5, 26).

Sutton, Richard S., Doina Precup, and Satinder Singh (1999). “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial Intelligence* 112.1-2, pp. 181–211 (cit. on p. 68).

Szepesvári, Csaba and Michael L. Littman (1996). “Generalized Markov Decision Processes: Dynamic-Programming and Reinforcement Learning Algorithms”. In: *Proceedings of the 13th International Conference on Machine Learning (ICML 1996).* Vol. 96 (cit. on p. 63).

Szita, István and Csaba Szepesvári (2010). “Model-Based Reinforcement Learning with Nearly Tight Exploration Complexity Bounds”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pp. 1031–1038 (cit. on p. 121).

Szita, István, Bálint Takács, and András Lőrincz (2002). “ε-MDPs: Learning in Varying Environments”. In: *Journal of Machine Learning Research* 3.Aug, pp. 145–174 (cit. on p. 63).

Taylor, Matthew E. and Peter Stone (2009). “Transfer Learning for Reinforcement Learning Domains: A Survey”. In: *Journal of Machine Learning Research* 10.Jul, pp. 1633–1685 (cit. on pp. 93, 97).

Vaidya, Pravin M. (1989). “Speeding-up linear programming using fast matrix multiplication”. In: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS 1989).* IEEE, pp. 332–337 (cit. on p. 134).

Van Woensel, Tom, Laoucine Kerbache, Herbert Peremans, and Nico Vandaele (2008). “Vehicle routing with dynamic travel times: A queueing approach”. In: *European Journal of Operational Research* 186.3, pp. 990–1007 (cit. on p. 69).

Villani, Cédric (2008). *Optimal Transport: Old and New.* Vol. 338. Springer Science & Business Media (cit. on pp. 70, 95).

Watkins, Christopher J. C. H. and Peter Dayan (1992). “Q-learning”. In: *Machine Learning* 8.3-4, pp. 279–292 (cit. on pp. 66, 97).

Weinstein, Ari and Michael L. Littman (2012). “Bandit-Based Planning and Learning in Continuous-Action Markov Decision Processes”. In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)* (cit. on pp. 22, 36, 42, 49).

Wiering, Marco A. (2001). “Reinforcement Learning in Dynamic Environments using Instantiated Information”. In: *Proceedings of the 18th International Conference on Machine Learning (ICML 2001)*, pp. 585–592 (cit. on p. 65).

Williams, Ronald J. and Leemon C. Baird (1993). *Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions.* Tech. rep. Northeastern University, College of Computer Science, Boston (cit. on p. 19).

Wilson, Aaron, Alan Fern, Soumya Ray, and Prasad Tadepalli (2007). “Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach”. In: *Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*, pp. 1015–1022 (cit. on p. 96).

Xie, Fan and Zhiqing Liu (2009). “Backpropagation Modification in Monte-Carlo Game Tree Search”. In: *Proceedings of the 3rd International Symposium on Intelligent Information Technology Application (IITA 2009).* Vol. 2. IEEE, pp. 125–128 (cit. on p. 36).

Abstract —

How should an agent act in the face of uncertainty on the evolution of its environment? In this dissertation, we give a Reinforcement Learning perspective on the resolution of non-stationary problems. The question is seen from three different aspects. First, we study the planning *vs.* re-planning trade-off of tree search algorithms in stationary Markov Decision Processes. We propose a method to lower the computational requirements of such an algorithm while keeping theoretical guarantees on the performance. Secondly, we study the case of environments evolving gradually over time. This hypothesis is expressed through a mathematical framework called Lipschitz Non-Stationary Markov Decision Processes. We derive a risk averse planning algorithm provably converging to the minimax policy in this setting. Thirdly, we consider abrupt temporal evolution in the setting of lifelong Reinforcement Learning. We propose a non-negative transfer method based on the theoretical study of the optimal Q-function's Lipschitz continuity with respect to the task space. The approach allows to accelerate learning in new tasks. Overall, this dissertation proposes answers to the question of solving Non-Stationary Markov Decision Processes under three different settings.

Keywords: Reinforcement Learning; Planning; Markov Decision Process; Non-Stationary Markov Decision Process; Lifelong Learning.

Résumé —

Comment un agent doit-il agir étant donné que son environnement évolue de manière incertaine ? Dans cette thèse, nous fournissons une réponse à cette question du point de vue de l'apprentissage par renforcement. Le problème est vu sous trois aspects différents. Premièrement, nous étudions le compromis planification *vs.* re-planification des algorithmes de recherche arborescente dans les Processus Décisionnels Markoviens. Nous proposons une méthode pour réduire la complexité de calcul d'un tel algorithme, tout en conservant des guaranties théoriques sur la performance. Deuxièmement, nous étudions le cas des environnements évoluant graduellement au cours du temps. Cette hypothèse est formulée dans un cadre mathématique appelé Processus de Décision Markoviens Non-Stationnaires Lipschitziens. Dans ce cadre, nous proposons un algorithme de planification robuste aux évolutions possibles, dont nous montrons qu'il converge vers la politique minmax. Troisièmement, nous considérons le cas de l'évolution temporelle abrupte dans le cadre du “lifelong learning” (apprentissage tout au long de la vie). Nous proposons une méthode de transfert non-négatif basée sur l'étude théorique de la continuité de Lipschitz de la Q-fonction optimale par rapport à l'espace des tâches. L'approche permet d'accélérer l'apprentissage dans de nouvelles tâches. Dans l'ensemble, cette dissertation propose des réponses à la question de la résolution des Processus de Décision Markoviens Non-Stationnaires dans trois cadres d'hypothèses.

Mots clés : planification ; apprentissage par renforcement ; Processus Décisionnel de Markov ; Processus Décisionnel de Markov Non-Stationnaire ; apprentissage tout au long de la vie.