POLITECNICO DI MILANO
Master of Science in Computer Science and Engineering
Department of Electronics, Information and Bioengineering

Non-Cooperative Configurable Markov Decision Processes

AI & R Lab
The Artificial Intelligence and Robotics Lab
Politecnico di Milano

Supervisor: Prof. Marcello Restelli
Co-supervisors: Dott. Alberto Metelli,
Dott.sa Giorgia Ramponi

Author:
Alessandro Concetti, 914267

Academic year 2019-2020

Alla mia famiglia...

Abstract

A *Markov Decision Process* (MDP) is a mathematical framework for modelling sequential decision making problems. The decision maker is called *agent*. That interacts with the *environment*, which represents everything outside the agent. They interacts continuously: the agent selects an action to perform and the environment responds to that action presenting a new situation to the agent. In addition, the environment provides rewards, special numerical values that the agent tries to maximize through the selection of actions. In most applications, the environment is considered a fixed entity out of the agent's control. However, there are many real world problems where we have the possibility to set some environmental parameters. Configurable Markov Decision Processes (Conf-MDPs) are able to model these *configurable* environments in order to find simultaneously the policy and the environment's configuration that maximize the agent's performance. From an abstract point of view, Conf-MDPs are composed by two *cooperative* entities: a *learning agent*, whose aim is to learn the optimal behavior in the environment, and a *configurator*, which selects environment dynamics that best suit the agent's needs. The question that gives birth to the research topic presented in this thesis is: *"what if the agent and the configurator are no longer cooperative?"*. In that case, the configurator is meant to achieve its own goal possibly different from the one of the agent. In this thesis, we deeply study the non-cooperative interaction between the learning agent and the configurator. We introduce a new extension of Conf-MDPs called *Non-Cooperative Configurable Markov Decision Processes* (NConf-MDPs) in order to model scenarios where the goal of the configurator does not coincide with that of the agent. Indeed, in some cases it could be even the opposite. Solving a NConf-MDP means finding the configuration that maximizes the configurator's reward function knowing that the agent will act to maximize its own reward function. We propose two algorithms, called *Action-feedback Optimistic Configuration Learning* (AfOCL) and *Reward-feedback Optimistic Configuration Learning* (RfOCL), that are able to efficiently solve NConf-MDPs, leveraging the *structure* of the problem. We provide theoretical performance guarantees of these algorithms showing both theoretically and experimentally that they suffer only bounded regret. Moreover, we present an experimental evaluation on different application domains comparing our algorithms with UCB, that solve NConf-MDPs ignoring the structure of the problem.

Estratto in Lingua Italiana

L'Apprendimento per Rinforzo [39] rappresenta una delle tre branche principali dell'Apprendimento Automatico, assieme all'Apprendimento Supervisionato e Non Supervisionato. Gli algoritmi di Apprendimento per Rinforzo si basano sull'interazione tra un un decisore, chiamato *agente*, e l'ambiente. Questa interazione è spesso modellizzata usando i *Processi Decisionali di Markov* (MDP) [34]. L'iterazione tra agente e ambiente avviene in maniera sequenziale: l'agente osserva lo stato dell'ambiente, decide quale azione attuare e l'ambiente risponde fornendo l'osservazione del nuovo stato raggiunto. Inoltre, l'ambiente fornisce all'agente una *ricompensa*, ovvero una quantità numerica che l'agente cercherà di massimmizare durante la sua interazione con l'ambiente.
L'Apprendimento per Rinforzo prende ispirazione dal processo di apprendimento di esseri umani e animali. Infatti, possiamo trovare diverse analogie tra il funzionamento dei MDPs e l'addestramento di un cane: il padrone chiede la zampa e se il cane risponde al comando guadagnerà un biscotto (ricompensa). L'obiettivo implicito del cane è chiaramente massimizzare la quantità di biscotti mangiati. Per raggiungere questo obiettivo il cane dovrà imparare la *politica ottima*, ovvero alzare la zampa ogni volta che il padrone lo richiede.

Nella maggior parte delle applicazioni, l'ambiente è considerato un'entità fissa fuori dal controllo dell'agente. Tuttavia, ci sono molti problemi reali in cui abbiamo la possibilità di modificare alcuni parametri ambientali. Ad esempio, in un'applicazione di guida autonoma, dove lo scopo dell'agente è imparare a guidare una vettura, potremmo modificare diversi parametri della macchina come la reattività del motore, la stabilità del veicolo o la velocità massima. I *Processi Decisionali di Markov Configurabili* (Conf-MDPs) [29], sono un'estensione dei MDPs che possono modellizzare gli ambienti configurabili al fine di trovare simultaneamente la politica ottimale dell'agente e la configurazione ambientale che massimizza le sue prestazioni. Da un punto di vista astratto, i Conf-MDP sono composti da due entità cooperative: un *agente*, il cui scopo è apprendere il comportamento ottimale nell'ambiente, e un *configuratore*, che seleziona le dinamiche ambientali che meglio si adattano alle esigenze dell'agente.

La domanda che ha dato vita al tema di ricerca presentato in questa tesi è: *"cosa potrebbe succedere se l'agente e il configuratore non fossero cooperativi?"*

In tal caso, il configuratore deve raggiungere un proprio obiettivo potenzialmente diverso da quello dell'agente. Pensiamo, ad esempio, ad un supermercato: un cliente (ovvero l'agente) vuole comprare alcuni prodotti e il suo scopo è acquistare il necessario nel minor tempo possibile; il gestore del supermercato (ovvero il configuratore) deve sistemare i prodotti negli scaffali in modo da indurre il cliente a comprare altro rispetto a quello che aveva pianificato. In questo caso, gli obiettivi dell'agente e del configuratore sono chiaramente diversi: l'agente vuole comprare il necessario nella maniera più efficiente possibile, mentre il configuratore vuole massimizzare i ricavi.

In questa tesi, studiamo a fondo l'interazione tra l'agente e un configuratore non-cooperativo. Introduciamo una nuova estensione dei Conf-MDP denominata *Processi Decisionali di Markov Configurabili Non-Cooperativi* (NConf-MDPs) per descrivere scenari in cui l'obiettivo del configuratore non coincide con quello dell'agente. Risolvere un NConf-MDP significa trovare la configurazione che massimizza la funzione di ricompensa del configuratore sapendo che l'agente agirà per massimizzare la propria funzione di ricompensa. Presentiamo due algoritmi, chiamati *Action-feedback Optimistic Configuration Learning* (AfOCL) e *Reward-feedback Optimistic Configuration Learning* (RfOCL), per risolvere un generico NConf-MDP sfruttando la *struttura* del problema. Inoltre, forniamo delle garanzie teoriche sulle performance, mostrando sia teoricamente sia sperimentalmente come il *rimpianto* prodotto dai nostri algoritmi converge a una quantità costante. Infine, presentiamo una valutazione sperimentale su diversi domini applicativi confrontando i nostri algoritmi con UCB, che risolve NConf-MDP ignorando la struttura del problema.

Ringraziamenti

Vorrei ringraziare il Prof. Marcello Restelli per avermi trasmesso, grazie ai suoi insegnamenti, la passione per l'Intelligenza Artificiale e per aver reso possibile lo svolgimento di questa tesi. Un grande ringraziamento va alla Dott.ssa Giorgia Ramponi e al Dott. Alberto Metelli per avermi supervisionato e supportato durante questo anno. Sono davvero onorato di aver lavorato con voi, vi stimo molto e vi auguro il meglio per il vostro futuro.

Ringrazio la mia famiglia per avermi sempre sostenuto durante i miei studi. Avete sempre creduto in me e questo è il dono più grande che avrei potuto desiderare.

Ringrazio la mia ragazza per essermi stata vicino nei momenti più duri. Ti ringrazio per avermi dato la forza di superare ogni ostacolo non dubitando mai della mia riuscita.

Ringrazio tutte le magnifiche persone che ho conosciuto durante i miei studi. Un particolare ringraziamento va ai miei due compagni di avventure: Antonio, l'inguaribile sognatore, e Mattia, colui che sa rendere tutti i sogni realtà. Grazie per aver reso belli anche i momenti più difficili. Ricorderò le nottate passate a lavorare con voi con grande gioia. Un caloroso ringraziamento va a tutti gli amici "automatici" per aver reso la triennale uno dei periodi più belli della mia vita. Un ultimo ringraziamento va ai miei due fantastici coinquilini Jack e Lucone. Grazie "fioi" per essere stati la mia famiglia e per aver reso il nostro appartamento la *mia casa*.

Un ultimo ringraziamento va a tutti gli insegnanti che mi hanno ispirato durante il mio percorso scolastico e accademico, in particolar modo la Maestra Fabrizia, la Prof.ssa La Galla, la Prof.ssa Acciarri, la Prof.ssa D'Agostino e il Prof. Braga. Grazie alla vostra passione per l'insegnamento mi avete trasmesso l'amore per la matematica, per il ragionamento e per l'informatica. Avete acceso in me la curiosità che mi ha portato fino alla fine del mio percorso accademico.

Contents

List of Figures

List of Tables

List of Algorithms

Notation

Matrices and vectors are denoted by bold (e.g. $\mathbf{V}$) and when the components are not clear by the context they are indicated as subscripts (e.g. $\mathbf{R}_s$ or $\mathbf{P}_{sas'}$). Cursive uppercase letters (e.g. $\mathcal{A}$, $\mathcal{S}$, $\mathcal{P}$) are used to denote sets.

$\mathcal{S}$	state space
$\mathcal{A}$	action space
$\mathcal{P}$	configuration space
$\mathbb{P}(E)$	probability of event E
$x \sim d(\cdot)$	x is sampled from distribution $d(\cdot)$
$\widehat{x}$	estimated version of a generic variable x
$\Delta(\mathcal{X})$	set of probability distributions over the set $\mathcal{X}$
$[N]$	Set $\{1, 2, \dots \mathrm{N}\}$
$\langle \dots \rangle$	triangular brackets are used as delimiters for tuples
$\langle s_t, a_t, r_t \rangle_{t=0,\dots,T}$	the sequence $s_0, a_0, r_0, s_1, a_1, r_1, \dots s_T, a_T, r_T$

Chapter 1

Introduction

Everyday, even several times a day, humans are facing decisions. We decide what to eat, what to study or who we want to spend our time with. Sometimes, we act to receive an immediate satisfaction, like eating a candy, sometimes we ponder our decisions by reducing the short-term pleasure for an higher long-term satisfaction. We are *decision makers* that have to make a sequence of decisions to achieve our long-term goals. This kind of problems are known in Artificial Intelligence literature as *Sequential Decision Making.*

As humans, we continuously make mistakes, taking wrong decisions that bring us to deal unpleasant situations. However, our brain is meant to learn from mistakes in order to gradually understand the effect of our actions and reduce the probability of making same mistakes twice. Just think of a child in front of a lit candle. He is intrigued by the movement of the flame and decides to catch it with his hands. As expected, the child will get a small burn and he will learn to not do it again. Our brains learn by interacting with the environment in an endless sequence of trial and error. We are likely to not repeat actions causing a negative *reinforcement.* [1] On the other hand, we tend to redo actions that brought us positive reinforcement in the past. *Reinforcement Learning* (RL) [39] is a sub-field of Artificial Intelligence that takes inspiration from the learning process of humans and animals studying how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. In RL context, the environment is usually stated in term of *Markov Decision Processes* (MDPs). A Markov

[1] In behavioral psychology, reinforcement is defined as a consequence that follows an operant response that affects the likelihood of that response occurring in the future. Positive reinforcements, like the production of dopamine, increase the likelihood of that response happening again while negative reinforcements, like pain or punishment, reduce that likelihood.

Decision Process [34] is a mathematical framework for modelling sequential decision making problems and it will be widely discussed in Section 2.1.

In most RL applications, the environment is considered a fixed entity out of the agent's control. However, there are many real world problems where we have the possibility to set some environmental parameters. For instance, in a car driving task, where a RL agent has to learn to drive a car, we could tune some parameters like vehicle stability or engine boost. In order to model these *configurable* environments, a new framework called *Configurable Markov Decision Process* (Conf-MDP) has been proposed in [29]. Conf-MDPs are a generalization of the standard MDPs with the possibility of altering the environmental dynamics. We can think to a Conf-MDP as a system with two entities: a *learning agent* that has to learn the optimal behavior in the environment and a *configurator* that chooses the environmental dynamics that best suit the agent's needs. From an abstract point of view, you can think to a Conf-MDP as a fully-cooperative scenario with two entities sharing the same goal.

The question that gave birth to the research topic presented in this thesis is:

What if the agent and the configurator are no longer cooperative?

In that case, the configurator is meant to achieve its own goal different from the one of the agent. In other words, the configurator should choose the environmental dynamics that maximize its own performance. Think, for instance, to an e-commerce scenario: the configurator has to choose the position of products on the website and the learning agent is the customer who wants to buy some products. Clearly, the goals of the configurator and the customer are different. The customer wants to buy what he or she needs in the most efficient way while the configurator wants to maximize the e-commerce's revenue. In this thesis, we analyze in depth the non-cooperative interaction between these two agents and we present a new framework, called *Non-Cooperative Configurable Markov Decision Process* (NConf-MDP). The two entities interacts sequentially: the configurator selects an environmental configuration and then the agent will learn the behavior that maximize its own performance in the proposed configuration. We also present two algorithms to solve NConf-MDPs, i.e. finding the configuration that maximize the configurator's performance. The two algorithms differ by the way they handle information coming from agent trajectories and they are called respectively *Action-feedback Online Configuration Learning*

(Af-OCL) and *Reward-feedback Online Configuration Learning* (Rf-OCL). We provide theoretical performance guarantees of these algorithms showing both theoretically and experimentally that they suffer only bounded regret. In other words, while other approaches that ignores the *structure* of the problem (like UCB) generates regret that grows indefinitely over time, our algorithms are able to converge to a zero-regret strategy. Moreover, we present an experimental evaluation on different application domains comparing our algorithms with UCB, in order to highlight the advantages of using an ad-hoc framework for configurable environments.

1.1 Outline of the Thesis

The contents of this thesis are organized as follows. In Chapter 2, we discuss preliminary theoretical concepts on which the proposed framework is based. In particular, we provide a basic overview on Markov Decision Processes, Multi-armed Bandits and Stackelberg games. Chapter 3 is dedicated to the state of the art. In this chapter, we mainly present the Conf-MDP literature showing the results obtained in recent years in this field. Moreover, we provide some links between our works and other related works in the Online Learning literature. In Chapter 4, after the discussion of some basic preliminaries, we present the novel framework Non-Cooperative Configurable Markov Decision Process focusing on the theoretical aspects and providing a formal definition of the problem. Chapter 5 is completely dedicated to the algorithmic contributions. We present our two algorithms for solving NConf-MDPs called *Action-feedback Online Configuration Learning* (Af-OCL) and *Reward-feedback Online Configuration Learning* (Rf-OCL). Chapter 6 provides the experimental evaluation of our algorithms compared with UCB1 [2] showing how a specialized framework as NConf-MDP outperforms a Multi-armed Bandit approach that does not leverage the *structure* of the problem. Before showing the experimental results, we describe in details the three environments we used to run experiments. In Chapter 7, we present some final considerations and we discuss the future research directions of the presented work. In Appendix A, we report the proofs and derivations that have been omitted in the text. In Appendix B, we provide some additional details on the presented experiments.

Chapter 2

Preliminaries

In this chapter, we introduce the fundamental theoretical frameworks on which this work is based. The algorithms and the theoretical discussion proposed in this thesis include various topics in the field of Artificial Intelligence, in particular Reinforcement Learning, Multi-armed Bandit and Game Theory. For this reason, we explain these topics, underlining the concepts necessary to understand the proposed framework.

Artificial Intelligence (AI) is the discipline that studies the theory and the development of computer systems able to perform tasks requiring human intelligence. Machine learning (ML) is a subfield of Artificial Intelligence that leverages statistical techniques to develop algorithms able to learn from data. Reinforcement Learning (RL) [39] is one of the three main branches of Machine Learning (ML), alongside Supervised Learning and Unsupervised Learning. While the goals of supervised and unsupervised learning can be summarized respectively in *"learning a model"* and *"learning a better representation of data"*, the goal of RL is *"learning to control"*. Hence, RL studies how artificial agents ought to take actions in an environment in order to maximize a cumulative reward coherent with its goal.

In Section 2.1, we will introduce Markov Decision Processes (MDP), i.e. the theoretical framework on which most of Reinforcement Learning literature is based. Then, in Section 2.2, we will analyze a specific class of Markov Decision Processes, called Multi-armed Bandits. Finally on Section 2.3.2, we will introduce some basic concepts of Game Theory focusing on a particular framework called Stackelberg Games.

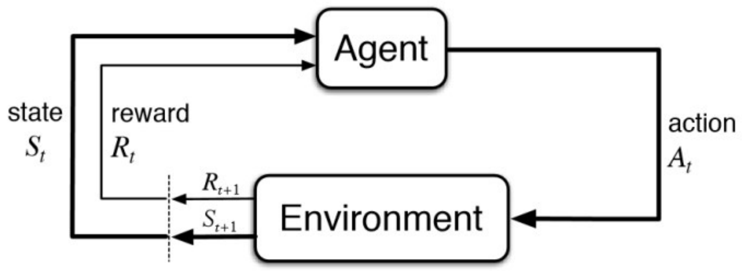

Figure 2.1: [39] Scheme describing the interaction between the agent and the environment

2.1 Markov Decision Processes

A *Markov Decision Process* (MDP) [34] is a mathematical framework for modelling sequential decision making problems.

The learner or decision-maker is called *agent*. The entity it interacts with is called the *environment*, and it represents everything outside the agent [39]. These interacts continuously, as described in Figure 2.3: the agent selects an action to perform and the environment responds to that action and presents a new situation to the agent. In addition, the environment provides rewards, special numerical values that the agent tries to maximize through the selection of actions.

In this manuscript, we will deal exclusively with *discrete-time MDPs*. This means that the agent and the environment interact at each step of a discrete time sequence, $t = 0, 1, 2, 3 \dots$. More precisely, at each time step t the agent receives a perception of the environment's *state* $s_t \in \mathcal{S}$, select an *action* $a_t \in \mathcal{A}$ and receives a *reward* $r_t \in \mathbb{R}$.

$$s_0, a_0, r_0, s_1, a_1, r_1, \dots$$

The goal of the agent is to collect as much reward as possible during the entire (possibly infinite) time horizon H. It is worth to notice that the agent is not interested in maximizing immediate rewards but the long-term cumulative sum of rewards; therefore, the agent may decide to give up immediate rewards in order to obtain higher rewards in the future.

In the following sections, we discuss the main aspects of MDPs by assuming that the time horizon is *infinite*. Instead, in Section 2.1.7, we present finite-horizon MDPs, discussing the main differences with the infinite-horizon setting.

2.1.1 Formal Definition

Formally, an *infinite-horizon MDP* is a tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma, \mu \rangle$ [39], where:

- $\mathcal{S}$ is a non-empty set of states, called *state space*;
- $\mathcal{A}$ is a non-empty set of actions, called *action space*;
- p is a function $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, called *transition model*, where $p(\cdot|s, a)$ represents the probability distribution over the next state given the current state s and the performed action a;
- r is a function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, called *reward function*, where $r(s, a)$ is the reward for performing action a in state s. Sometimes - as in this thesis - we can make the assumption that the reward function depends only on states $r : \mathcal{S} \to \mathbb{R}$.
- $\gamma \in [0, 1]$ is the *discount factor*, which models the interest of the agent in future rewards.
- $\mu(s) \in \Delta(\mathcal{S})$ is the *initial state distribution*

In this thesis, we will consider only MDPs with finite state and action spaces, i.e. $S = |\mathcal{S}| \in \mathbb{N}$ and $A = |\mathcal{A}| \in \mathbb{N}$. A *trajectory* is a sequence $\langle s_h, a_h, r_h \rangle_{h=0,\ldots,T-1}$, where s_h, a_h, r_h represent the current state, the action performed, the immediate reward at time step h and T is the length of the trajectories.

The dynamics of the environment, modeled by the transitional model, must satisfy two properties:

- *Markov property*: the next state s' depends only on the current state s and the chosen action a, and not on the history;
- *Stationarity*: the dynamics of the environment do not change over time.

While the former represents a fundamental feature of MDPs, the classic MDP can be extended to deal with non-stationary transition models [20, 17, 19, 10].

2.1.2 Policy

In a Reinforcement Learning scenario, the agent performs actions through a stochastic policy $\pi(a|s) \in \Pi$, which provides a probability distribution over A given the current state s. The set Π is called *policy space* and is the function space containing all the possible policies.

$$\pi : \mathcal{S} \to \Delta(\mathcal{A})$$

$$\sum_{a \in \mathcal{A}} \pi(a|s) = 1 \quad \forall s \in \mathcal{S}$$

A policy is said to be *deterministic* if for all the states s there exists an action a such that $\pi(a|s) = 1$. A deterministic policy can also be defined as:

$$\pi_d : \mathcal{S} \to \mathcal{A}$$

2.1.3 Performance of a policy

The goal of the agent is to find the optimal policy π^* w.r.t. an optimality criterion. The general idea is that a policy π_1 is better then a policy π_2 if an agent following policy π_1 collects on average more reward than another agent following π_2. Several approaches have been proposed in literature [34], the most common one is to measure the performance of an agent along a trajectory as the γ-*discounted cumulative return*:

$$v = \sum_{t=0}^{H-1} \gamma^t r_t$$

Hence, the performance of a policy π can be defined as the *expected return* V^π of trajectories induced by π.

$$V^\pi = \mathbb{E}_\pi[v]$$

Definition 2.1.1. *The expected return V^π induced by the policy π can be computed as:*

$$V^\pi = \frac{1}{1-\gamma} \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi(a|s) r(s,a),$$

where $d^\pi(s)$ is the γ-discounted state distribution [40].

We can recursively define $d^\pi(s)$ as:

$$d^\pi(s) = (1-\gamma)\mu(s) + \gamma \sum_{s \in \mathcal{S}} d^\pi(s') p^\pi(s'|s)$$

where $p^\pi(s'|s)$ is the *kernel function*, obtained by marginalizing the transition model over the action space:

$$p^\pi(s'|s) = \sum_{a \in \mathcal{A}} \pi(a|s)p(s'|s,a). \tag{2.1}$$

It is worth to notice that in *infinite-horizon MDP*, the role of the discount factor γ is crucial. Infact, in these scenarios, if we removed the effect of discount factor, setting $\gamma = 1$, the return of trajectories would diverge:

$$v = \lim_{H \to \infty} \sum_{t=0}^{H-1} r_t = \infty$$

On the other hand, in *finite-horizon MDP*, the discounted sum of rewards would be finite even if the discount factor γ is set to 1.

Definition 2.1.2. *A policy π^* is said to be optimal if it maximizes the expected return J^π.*

$$\pi^* = \arg\max_{\pi \in \Pi} V^\pi.$$

2.1.4 Value functions

In most cases, it is useful for the agent to measure the utility of a state, or a state-action pair, in order to compare them and derive an optimal policy. Value functions provide such utility measure. It is straightforward to realize that the value of a state (or a state-action pairs) depends on the agent's behavior from that state forward. In other words, value functions depends strictly on the current policy of the agent.

Definition 2.1.3. *We define state value function $V^\pi : \mathcal{S} \to \mathbb{R}$ under a policy π the following recursive equation, named Bellman expectation equation:*

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right) \tag{2.2}$$

Hence, given a state s we can consider $V^\pi(s)$ as the expected return obtained from state s by following the policy π. Based on this consideration we can rewrite the performance of a policy π (definition 2.1.1) as:

$$V^\pi = \sum_{s \in \mathcal{S}} \mu(s) V^\pi(s)$$

Equivalently, we can rewrite definition 2.1.3 in matrix notation. Let's define $\mathbf{R}_s^\pi$ as the vector of length $|S|$ containing $r^\pi(s)$, the reward induced by policy π in state s:

$$r^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) r(s,a).$$

Moreover, let's define $\mathbf{P}^{\pi}_{ss'}$ as the $|S| \times |S|$ matrix where in position s, s' there is the probability $p^{\pi}(s'|s)$ of ending up in state s' following policy π on state s (equation 2.1).

Named $\mathbf{V}^{\pi}$ the vector of length $|S|$ containing the state values, we can reformulate definition 2.1.3 as:

$$\mathbf{V}^{\pi} = \mathbf{R}^{\pi}_{s} + \gamma \mathbf{P}^{\pi}_{ss'} \mathbf{V}^{\pi} \tag{2.3}$$

Definition 2.1.4. *We can define the expected Bellman operator* $T^{\pi} : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ *associated with policy* π *as:*

$$T^{\pi}(\mathbf{V}^{\pi}) = \mathbf{R}^{\pi}_{s} + \gamma \mathbf{P}^{\pi}_{ss'} \mathbf{V}^{\pi} \tag{2.4}$$

Although the state value function is essential for the evaluation of a policy, it is not provide any information on the value of actions in a given state. For this reason, the state value function is not suitable for control purpose and the *state-action value function* is needed in order to support the agent in the decision problem.

Definition 2.1.5. *We define state-action value function* $Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *under a policy* π *the following recursive equation:*

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^{\pi}(s',a'), \tag{2.5}$$

or equivalently

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi}(s'). \tag{2.6}$$

2.1.5 Methods to solve a MDP

Solving a Markov Decision Process means finding the optimal policy. Many algorithms have been proposed in literature and they can be classified along four dimensions:

- *Value-based* vs *Policy-based.* The former estimates the value of states $V(s)$ or state-action pairs $Q(s,a)$ and uses these estimates to derive a greedy policy. Popular algorithms presented in literature are *Policy Iteration*, *Value Iteration* [39, 41] . On the other hand Policy-based algorithm finds a policy by searching directly in a space of policies. Algorithms in this category [12] can be classified in Policy Gradient, Expectation Maximization and Information-Theoretic.

- *Model-based* vs *Model-free.* Model-based methods leverages a model of the environment that could be estimated from sample or known a priori. Instead, model-free methods derive a policy without an explicit model of the environment.

- *On-line* vs *Off-line.* In on-line methods, the agent learns while the experience is collected. While in off-line algorithms, the agent starts learning when all the experience have been collected.

- *On-policy* vs *Off-policy.* In on-policy algorithms the agent tries to estimate the optimal policy and uses the cur rent estimate to interact with the environment. In off-policy methods, instead, the current estimate of the optimal policy is updated by the the experience collected by a different policy.

In the following sections, two of the main value-based algorithm, namely *Policy Iteration* and *Value Iteration*, will be explored in depth.

Policy Iteration

Policy Iteration [21] is an algorithm based on the alternation of two phases: *policy evaluation* and *policy improvement.* In the policy evaluation phase we evaluate the current policy computing the value function. While, in the policy improvement phase we leverage that value function to derive a new policy that is guaranteed to be better then the previous one. More specifi-

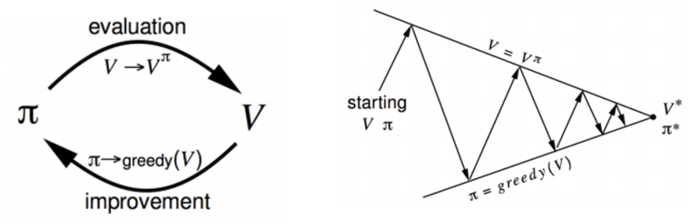

Figure 2.2: [39] A graphic representation of the policy iteration algorithm.

cally, in the policy evaluation phase we compute the state value function of the policy $\pi^{(t)}$, estimated at time t, applying multiple times the expected Bellman operator (equation 2.4) until convergence. Since the expected Bellman operator is a contraction (see appendix A.1 for more details), it can be proved that for any policy π and for any initial vector $\mathbf{V}$:

$$\lim_{k \to +\infty} (T^\pi)^k \mathbf{V} = \mathbf{V}^\pi. \tag{2.7}$$

Algorithm 1 Policy Iteration

```
1. Initialize policy $\pi^{(0)}$ arbitrarily

2. Policy Evaluation
repeat
  $\mathbf{V} \leftarrow T^{\pi^{(t)}}(\mathbf{V})$
until convergence

3. Policy Improvement
$\pi_{old} \leftarrow \pi$
$\pi(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s'), \quad \forall s$
if $\pi(s) = \pi_{old}(s), \ \forall s$ then stop and return $\pi$ else go to 2
```

This means that we are able to compute the value function associated with a given policy by applying the expected Bellman operator an infinite number of times starting from a random initial vector.

In the policy improvement phase, instead, we compute a new policy that is greedy w.r.t. the state-action value function, derived by equation 2.6. The complete procedure is formalized in the algorithm 1.

2.1.6 Value Iteration

One drawback of policy iteration is that each iteration requires a policy evaluation. Nevertheless, it can be proven that the policy evaluation step can be truncated in several ways without losing the convergence guarantees of policy iteration. In vanilla policy evaluation we apply the expected Bellman operator until convergence, i.e. when the new value function is almost equal to the previous one. However, the algorithm will converge even with a fixed finite number of application of that operator. One important special case is when the policy evaluation step is stopped after just one step. In that case the algorithm is called *value iteration.* In value iteration algorithm, the two phases of evaluation and improvement can be condensed in one step defining a single update rule, known as *Bellman optimality equation*:

$$V(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s') \right] \qquad \forall s. \tag{2.8}$$

Algorithm 2 Value Iteration

```
1: repeat
2:     $\mathbf{V} \leftarrow T^\star(\mathbf{V})$
3: until convergence
4: $\pi(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s'), \qquad \forall s$
5: return  optimal policy $\pi$
```

In most cases is convenient to write the Bellman optimality equation in matrix notation:

$$\mathbf{V} = \max_a \left[\mathbf{R}_{s,a} + \gamma \mathbf{P}_{s,a,s'}\mathbf{V}\right], \tag{2.9}$$

where $\mathbf{V}$ is the vector of state values, $\mathbf{R}$ and $\mathbf{P}$ are respectively the reward and the transition model matrices and the subscripts indicates the shape of the matrices.

Definition 2.1.6. *We can define the Bellman optimality operator as:*

$$T^\star(\mathbf{V}) = \max_a \left[\mathbf{R}_{s,a} + \gamma \mathbf{P}_{s,a,s'}\mathbf{V}\right], \tag{2.10}$$

Hence, Value Iteration algorithm finds the optimal value function by multiple applications of the Bellman optimality operator and then returns the greedy policy w.r.t the state-action value function, computed using equation 2.6. Value Iteration algorithm is formalized in Algorithm 2.

2.1.7 Finite Horizon MDPs

In this thesis, we mainly deal with *finite-horizon* MDPs therefore it is worth discussing what are the main differences w.r.t. the infinite-horizon setting. A finite-horizon Markov Decision Process is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \mu, H \rangle$ where $\langle \mathcal{S}, \mathcal{A}, p, r, \mu \rangle$ is a classic MDP and $H \in \mathbb{N}_{\geq 1}$ is the finite time horizon, i.e. the maximum number of time steps that the agent has available to interact with the environment. Moreover, we suppose finite state and action spaces, i.e. $S = |\mathcal{S}|$ and $A = |\mathcal{A}|$.

The first difference with infinite-horizon MDPs concerns the definition of *policy*. Let's define a deterministic decision rule $\pi_h : \mathcal{S} \to \mathcal{A}$ which prescribes in a given time step $h \in [H]$ an action $\pi_h(s) \in \mathcal{A}$ for every state $s \in \mathcal{S}$. In finite-horizon MDPs, a deterministic policy $\pi = \langle \pi_1, \pi_2, \ldots, \pi_H \rangle \in \Pi_D^H$ is defined as a sequence of decision rules, where Π_D^H is the set of deterministic policies.

Algorithm 3 Backward Value Iteration

```
1: $V_H(s) = \max_{a\in\mathcal{A}} [r(s,a)] \quad \forall s \in \mathcal{S}$
2: for $h = H-1, H-2 \dots 1$ do
3:     $\mathbf{V}_h \leftarrow T^\star(\mathbf{V}_{h+1})$
4: end for
5: $\pi_h(s) \leftarrow \arg\max_a [r(s) + \gamma \sum_{s'} p(s'|s,a) V_{h+1}(s')] \quad \forall s \in \mathcal{S}, \forall h \in [H-1]$
6: $\pi_H(s) \leftarrow \arg\max_a [r(s,a)], \quad \forall s \in \mathcal{S}$
7: $\pi \leftarrow (\pi_1, \pi_2, \dots, \pi_H)$
8: return optimal policy $\pi$
```

As for the policy, even the definition of value functions is adapted for the finite-horizon setting. Indeed, in finite-horizon MDPs, the value function $V_h^\pi(s)$ represents the expected return for following policy π starting from state $s \in \mathcal{S}$ at time step $h \in [H]$. In this context, we no longer have to find a single optimal value function $V^\star(s)$ as in classic MDP but we need a value function $V_h^\star(s)$ for each time step $h \in [H]$. The value function $V_h^\star(s)$ represents the average reward that the agent can gather following the optimal policy starting from state $s \in \mathcal{S}$ on time step $h \in [H]$. The value function $V_h^\star(s)$ for a generic time instant $h \in [H]$ can be defined as

$$V_h^\star(s) = \max_{a\in\mathcal{A}} \left[r(s,a) + \sum_{s'} p(s'|s,a) V_{h+1}^\star(s') \right], \tag{2.11}$$

or equivalently, using the the Bellman optimality operator:

$$\mathbf{V}_h^\star = T^\star(\mathbf{V}_{h+1}^\star) \tag{2.12}$$

It is straightforward to realize that on the last time step $h = H$, since the time budget is over, the optimal value function is equal to the reward function:

$$V_H^\star(s) = \max_{a\in\mathcal{A}} [r(s,a)] \quad \forall s \in \mathcal{S}. \tag{2.13}$$

Hence, a finite-horizon MDP can be easily solved in a backward manner starting from the last time instant H (equation 2.13) and going backward applying equation 2.11. This procedure represents the adaption of the value iteration algorithm for the finite-horizon setting, namely *Backward value iteration* (Algorithm 3).

For the same reasons, even the state-action value functions are modified so to characterized the values of state-action pairs for each time step. Hence we can define $Q_h^\pi(s,a)$ as the expected return for playing action $a \in \mathcal{A}$ is state $s \in \mathcal{S}$ at time step $h \in [H]$ following policy π. Analogously, $Q_h^\star(s, a)$ is the state-action functions associated with the optimal policy $\pi^\star$.

2.2 Multi-armed Bandits

In this section, we will focus on a specific class of Markov Decision Processes, called Multi-armed Bandits.
Imagine having access to k slot machines. Sometimes, slot machines are also called one-armed bandits because of the mechanical lever on the side and their ability to empty players' pockets. We know that, among those k machines, there is one that provides more money than the others, but we do not know which one it is. So we start playing almost randomly all the machines keeping track of gains and losses until we are quite sure to have identified the best machine. From then on, it is reasonable to keep playing on that machine most of the time to maximize our gains (or rather minimize our losses). However, we cannot be sure to have identified exactly the machine which yields more money. Hence, we keep exploring other options although playing on suboptimal machines could cause us lose money. This conflict between *exploring* new possibilities and *exploiting* current information has been extensively studied in the reinforcement learning literature and it is known as the *exploration-exploitation dilemma.* Algorithms in the family of Multi-armed Bandit (MAB) can solve this kind of problem balancing the exploration-exploitation dilemma. MAB is a simple but very powerful framework to make decisions over time under uncertainty [38]. Thanks to its simplicity and effectiveness it has been applied in several applications domains, such as:

- **A/B testing** A gaming company is testing a new user interface. The designers are undecided about the color of the button for buying additional coins. Once the user enters the game, the algorithm chooses the color of the button. The user can play as usual and possibly buy new coins. The algorithm will learn which is the color of the button which induces more gamers to buy.

- **Healthcare** The goal is to evaluate k experimental treatments for disease. For each patient, the algorithm chooses which treatment to administer and receives feedback based on the effect of the treatment on the patient. The algorithm will tend to give to patients treatments that proved to have better effects.

- **Dynamic pricing** A digital store is selling a given product online. Every time a user enters the web page, the algorithm proposes a price for the product to the user. The customer can buy (or not) and then leaves the page forever. The goal of the algorithm is to maximize store revenues.

2.2.1 Formal Definition

The multi-armed bandit (MAB) can be seen as a set of real distribution $B = \{\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_k\}$, where $k \in \mathbb{N}^+$ is the number of arms and distribution $\mathcal{R}_i$ describes the behavior of the stochastic reward $r_i \sim \mathcal{R}_i$ associated with arm i. At each iteration, the agent selects an arm to pull and observes the returned reward. The goal of the agent is to maximize the accumulated reward over a time horizon H, also called *budget* in this context. It is easy to show that multi-armed bandits can be modeled using one-state Markov Decision Processes, where all the actions are associated with different arms.

Performances and regret

In a k-armed bandit problem [39], each of the k actions has an expected reward, also called *value*. We denote by a_h and r_h respectively the selected action and the reward at time $h \in [H]$. The value of an arbitrary arm i, namely V_i, is the expected return when that arm is pulled and it can be defined as:

$$V_i \doteq \mathbb{E}[r_h | a_h = a].$$

As already explained, the goal of the agent is to maximize the accumulated reward over an episode of length H. To achieve this goal the agent should play as much as possible the arm $i^\star$ with higher value $V_{i^\star}$. More formally, the optimal strategy is playing always arm $i^\star$, where $i^\star = \arg\max_{i \in [k]} V_i$. In order to formalize the agent's objective we introduce a new quantity called *regret*. Informally, the regret Δ is the cost of playing a suboptimal strategy and can be defined as the difference between the reward that could have been obtained by playing the optimal strategy and the accumulated reward by playing the current strategy.

$$\Delta = \underbrace{H \cdot V_{i^\star}}_{\substack{\textit{performance} \\ \textit{optimal strategy}}} - \underbrace{\sum_{t=1}^{H} r_t}_{\substack{\textit{performance} \\ \textit{current strategy}}} .$$

Analogously, the regret can also be expressed as follow:

$$\Delta = \sum_{i=1}^{k} \mathbb{E}[N_i] \cdot \Delta_i,$$

where $\mathbb{E}[N_i]$ is the expected number of time arm i is pulled over the time horizon and $\Delta_i = V_{i^\star} - V_i$ is the suboptimality gap, i.e. the difference between the expected return of the optimal arm and a generic arm i.

Hence, the goal of the agent is to find a strategy that minimize the regret over the time horizon H. A *zero-regret strategy* is a strategy whose average regret per round Δ/H tends to zero with probability 1 when the number of played rounds tends to infinity [43].

Bandit strategies

If we knew the value of actions, the k-armed bandit problem would be trivially solved by playing always the actions with the higher value. Therefore we assume that we do not know exactly actions' value but we have to estimate them from samples. We denote the estimated value for arm i at time h as $\widehat{V}_{i,h}$. We would like that the estimated value $\widehat{V}_{i,h}$ is close to the real value V_i for each action $i \in [k]$. If we have access to a good approximation of the real values of actions it could be reasonable to play a so called *greedy* strategy, i.e. pulling always the arm $\hat{i}^\star$ with the higher estimated value:

$$\hat{i}^\star = \arg\max_{i \in [k]} \widehat{V}_{i,h}.$$

However, in most of real cases, it is rare to have a good estimate of the values without interacting with the environment. It is straightforward to conclude that we should spend some rounds exploring all the arms to build a reliable estimate of the values of actions. One natural way to estimate them is by averaging the reward actually received [39, p. 27]:

$$\widehat{V}_{i,h} \doteq \frac{\text{sum of rewards when } i \text{ is selected}}{\text{number of times } i \text{ is selected}} = \frac{\sum_{h=1}^{H} r_h \cdot \mathbb{I}_{a_h=i}}{\sum_{h=1}^{H} \mathbb{I}_{a_h=i}},$$

where $\mathbb{I}_{a_h=i}$ denotes the *indicator function* that it is 1 when arm i has been pulled at time h and 0 otherwise. Once estimates become more and more precise we can exploit them to play a greedy strategy. Many variants [9] of the greedy strategy have been proposed in literature, including:

- **Epsilon-greedy strategy**: This is the easiest variant of the greedy strategy and consists in playing the best arm with probability $1 - \epsilon$. On the other hand, with probability ϵ, the agent pulls a random arm. The value of ϵ is usually small (e.g. $\epsilon = 0.1$).

- **Epsilon-decreasing strategy**: Very similar to the previous strategy but the probability ϵ to select a random actions decreases as the number of rounds increases. This is reasonable since the estimates become more reliable as the number of collected sample increases.

- **Epsilon-first strategy**: Unlike the previous strategies, here there is a clear distinction between an *exploration phase* and an *exploitation phase*. Defined as H the total number of rounds, the agent spends the first ϵH rounds exploring the environment by selecting randomly the arms and the last $(1-\epsilon)H$ rounds exploiting the information collected in the first phase by playing a greedy strategy. This strategy is generally used to solve 2-armed bandits and is widely known as *A/B testing*.

Although the presented strategies have been extensively used in practice, they suffer from a problem: they try the non-greedy actions indiscriminately, with no preference for those that are nearly greedy or particularly uncertain. In order to address this issue an algorithm called *Upper Confidence Bound* (UCB) has been presented in the literature by *Auer et al.*[2]. This algorithm will be extensively explained in the next section.

2.2.2 Upper-Confidence-Bound Action Selection

The Upper Confidence Bound (UCB) [2] is an action selection criterion based on the OFU principle. OFU stands for "Optimism in Face of Uncertainty" and, as the name suggests, this principle aims to be optimistic in the evaluation of the actions pretending that the actions have the maximum value they can aspire to. For this purpose, in addition to estimating the values of each arm, we need to build and update confidence intervals on these values. Having confidence intervals allows us to compute an "optimistic value" for each arm. The arm selected by the UCB criterion will be the one with the highest optimistic value.

A prerequisite for building a confidence interval is to know the distribution from which the samples are drawn or at least to have enough sample to assume a Gaussian distribution, applying the central limit theorem. However, we cannot satisfy none of these prerequisites. In order to overcome this issue, confidence intervals can be constructed using *Hoeffding's concentration inequality*:

$$P(|\mathbb{E}(x) - \widehat{x}| \geq \epsilon) \leq 2e^{-2N\epsilon^2},$$

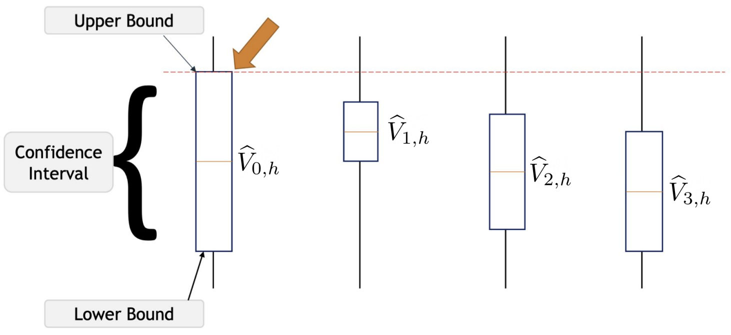

Figure 2.3: In this picture, we show an instance of UCB algorithm in a 4-arm setting. Each arm i is associated with a confidence interval centered in $\widehat{V}_{i,h}$. In each iteration the arm with the highest upper bound will be selected.

where x is a generic random variable, $\widehat{x}$ is the sample mean, N is the number of sample and ϵ is an lower bound for the deviation of the sample mean from the actual mean value.

As shown in [2], we can leverage this inequality to build a confidence interval on the action values, obtaining:

$$\widehat{V}_{i,h} - c\sqrt{\frac{\ln h}{N_{i,h}}} \leq V_i \leq \widehat{V}_{i,h} + c\sqrt{\frac{\ln h}{N_{i,h}}},$$

where i is a generic arm, h is the number of rounds played so far, $N_{i,h}$ is the number of time arm i has been pulled until time h and $c > 0$ controls the degree of exploration.
As already mentioned, the UCB criterion selects the arm with the highest optimistic value or, more formally, the arm with the highest upper bound of the confidence interval. Hence, the action selection strategy at time h can be formalized as follow:

$$a_h \doteq \arg\max_{i \in [k]} \left[\widehat{V}_{i,h} + c\sqrt{\frac{\ln h}{N_{i,h}}} \right].$$

2.3 Game theory concepts

Tools from Game Theory are playing an increasingly important role in Machine Learning. This can be credited to the possibility to formulate the learning process as an interaction between non-cooperative algorithms or *players*. In this section, we present two important classes of games, namely *Simultaneous games* and *Stackelberg games*, and we formally discuss their solution concepts.

2.3.1 Simultaneous games

Most game theory literature focuses on the study of *Simultaneous games.* In Simultaneous game, each player reveals its selected strategy concurrently. A famous example of Simultaneous game is the *prisoner dilemma* in which two members of a criminal gang are arrested and imprisoned. The two criminals must decide (simultaneously and without communicating with other) to collaborate with justice *betraying* their partner or to remain *silent.* If both remain silent they both will stay in prison for 1 year while if both betray they will remain in prison 2 years. If criminal A betrays his partner while criminal B remains silent, criminal A will be free while criminal B will stay in prison for 3 years. This scenario can be summarized by the payoff matrix in Table 2.1. The solution of a non-cooperative simultaneous game is often

B / A	B stays silent	B betrays
A stays silent	-1 / -1	0 / -3
A betrays	-3 / 0	-2 / -2

Table 2.1: Payoff matrix of prisoner dilemma

framed in term of *Nash Equilibrium.* In a Nash Equilibrium, each player plays the best response to the joint strategy of the competitors so that no player can benefit from unilaterally deviating from this strategy [13]. In the prisoner dilemma example, ⟨*A betrays, B betrays*⟩ represent a Nash Equilibrium strategy since none of the two prisoners are willing to change his choice if the choice of the other prisoner remains fixed. Indeed, if prisoner A knows that B will betray he has no interest on changing his choice remaining silent since he will stay in prison for 3 years rather then 2. Clearly, this reasoning is the same for prisoner B who is not interest on unilaterally deviates his strategy.

Formal definition

In a Nash equilibrium strategy, each player has no advantage in changing its strategy if the opponents' joint strategy remains the same. This concept can be formalized as follow. Let us first introduce some typical game-theoretic notation. We call $\mathcal{I}$ the player index set, $\mathcal{A}_i$ represents the action space of player $i \in \mathcal{I}$ and $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times, \dots, \mathcal{A}_{|\mathcal{I}|}$ is the joint action space. Moreover, $a_{-i} = (a_j)_{j \in \mathcal{I}/\{i\}}$ denotes the joint action profile of all the agents excluding agent i and $r_i : \mathcal{A} \to \mathbb{R}$ is the reward function of player $i \in \mathcal{I}$ representing the cost of the player i given an action profile $a \in \mathcal{A}$. A Nash equilibrium strategy can be formalized with the following defintion.

Definition 2.3.1 (Nash equilibrium)**.** *The joint strategy $a^\star \in \mathcal{A}$ is a Nash equilibrium if for each $i \in \mathcal{I}$,*

$$r_i(a^\star) \geq r_i(a_i, a^\star_{-i}), \quad \forall a_i \in \mathcal{A}_i. \tag{2.14}$$

2.3.2 Stackelberg games

Although Simultaneous games can describe a large variety of learning scenarios, there are many problems exhibiting a hierarchical order of play between agents in several fields such as human-robot interaction [30, 26], economics [1, 7] and autonomous vehicles [14, 36]. In game theory, this problem is known as Stackelberg game and the solution concept studied is called a Stackelberg equilibrium. The simplest formulation of Stackelberg game is characterized by two players, a *leader* and a *follower*, that interact in a hierarchical structure, i.e. the follower plays the selected strategy first and then the follower plays its best response. The leader usually can benefit from the advantage of moving as first player and this makes the Stackelberg Equilibrium more convenient for the leader than the Nash Equilibrium obtained by the analogous simultaneous game. For instance, we can model a duopoly market as a Stackelberg games. Suppose that one of the two firms is the market leader - namely *Firm* 1 - and it can be modeled as a Stackelberg leader. Let a_i be the production volume of Firm i and $P(a_1, a_2)$ be the demand function providing the good price as a function of the volumes a_1, a_2 produced by the two firms. Naming $C_i(a_i)$ the production cost of Firm i for producing a_i pieces of the good, we can define the reward functions of the two firms as:

$$r_1(a_1, a_2) = P(a_1, a_2)a_1 - C_1(a_1)$$
$$r_2(a_1, a_2) = P(a_1, a_2)a_2 - C_2(a_2)$$

Hence, we can easily derive the reaction functions $BR_1(a_2)$ and $BR_2(a_1)$ as

$$\frac{\partial r_1(a_1, a_2)}{\partial a_1} = 0 \rightarrow a_1^\star = BR_1(a_2)$$
$$\frac{\partial r_2(a_1, a_2)}{\partial a_2} = 0 \rightarrow a_2^\star = BR_2(a_1)$$

where $BR_i(a_j)$ represents the best response volume a_i^* of Firm i if Firm j produced the volume a_j. Clearly, being the leader represents an advantage since Firm 1 can substitute Firm 2's reaction function in its own reward equation, which it will then maximize as if it were a monopolist:

$$r_1(a_1) = P(a_1, BR_2(a_1))a_1 - C_1(a_1) \tag{2.15}$$

Therefore, the final Stackelberg Equilibrium will be:

$$(a_1^\star, \widetilde{a}_2)$$

where $a_1^\star$ can be derived maximizing Equation 2.15 with the first-order condition $\frac{\partial r_1(a_1)}{\partial a_1} = 0$ and $\widetilde{a}_2$ is the best response volume, i.e. $\widetilde{a}_2 = BR_2(a_1^\star)$.

In summary, Stackelberg games represent a powerful tool to model a hierarchical interaction between agents. In our work, we used this framework to describe relationship between the configurator and the learning agent. In this scenario, the configurator is modeled as the leader who acts as first player choosing an environment configuration, while the learning agent is modeled as the follower who learns the optimal policy in the environment selected by the configurator.

Formal definition

Consider a game between two agents where one is deemed the *leader* and the other the *follower*. Let us adopt the typical game theoretic notation in which the player index set is $\mathcal{I} = \{1, 2\}$, where player 1 is the leader and player 2 the follower. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be respectively the action space of the two players. The leader and the follower' reward function are respectively

$r_1 : \mathcal{A} \to \mathbb{R}$ and $r_2 : \mathcal{A} \to \mathbb{R}$, where $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$. Solving a Stackelberg game in practice means solving two distinct optimization problems. The leader aims to solve the optimization problem given by

$$\max_{a_1 \in \mathcal{A}_1} \left\{ r_1(a_1, a_2) | a_2 \in \arg\max_{a \in A_2} r_2(a_1, a) \right\} \tag{2.16}$$

while the follower, given the leader's strategy $a_1 \in \mathcal{A}_i$, aims to solve the optimization problem

$$\max_{a_2 \in \mathcal{A}_2} r_2(a_1, a_2). \tag{2.17}$$

Definition 2.3.2 (Stackelberg Equilibrium). *In a two-player game with player 1 as the leader, a strategy $a_1^\star \in \mathcal{A}_1$ is called a Stackelberg equilibrium strategy for the leader if*

$$\min_{a_2 \in BR(a_1^\star)} r_1(a_1^\star, a_2) \geq \min_{a_2 \in BR(a_1)} r_1(a_1, a_2), \quad \forall a_1 \in \mathcal{A}_1, \tag{2.18}$$

where $BR(a_1) = \{a \in A_2 | r_2(a_1, a) \geq r_2(a_1, a_2), \forall a_2 \in \mathcal{A}_2\}$.

Clearly, this definition can be extended even for n-follower setting when $BR(a_1)$ is replaced with the set of Nash equilibria NE(a_1), given that player 1 is playing a_1 so that the followers' reaction strategies a_{-1} is a Nash equilibrium.

Chapter 3

State of the art

In this chapter, we mainly focus of the state of the art of *Configurable Markov Decision Processes* (Conf-MDPs) and Online Learning in general. In Section 3.1, we introduce in details the literature of Conf-MDPs focusing both on theoretic and algorithmic aspects; while, in Section 3.2 we present some interesting connections between our works and other related works on Online Learning literature.

3.1 Configurable Markov Decision Processes

In this section, we extensively explained an extension of classic Markov Decision Processes called Configurable Markov Decision Processes (Conf-MDPs). The Conf-MDP framework was introduced in [29] in order to deal with *configurable environments*, i.e. environments characterized by tunable parameters. For example, in a car racing task, a Reinforcement Learning agent has to learn to drive a car and there is the possibility to modify the car setup to suit the driver's need. From a logical point of view, we can consider Conf-MDP as a fully-cooperative scenario with two entities acting in the environment: an agent who learns the optimal policy and a supervisor (also called *configurator*) whose aim is to configure parameters in order to optimize the agent's learning process. Solving a Conf-MDP, in practice, means finding simultaneously a policy and a configuration that maximize the expected return of the agent.

3.1.1 Formal Definition

As already mentioned in the previous section, the transition model of a MDP describes the dynamics of the environment. Moreover, tuning environmental parameters means altering in some way environmental dynamics. Hence, it is straightforward to realize that tuning environment's parameters means changing the transition model of a MDP. For this reasons, in Conf-MDP we have no longer a single transition model but a set (possibly infinite) of possible transition models associated with different configurations of the environment.

Formally, a Conf-MDP is a tuple $(S, A, r, \gamma, \mu, \mathcal{P}, \Pi)$ where (S, A, r, γ, μ) is an MDP without the transition model and $\mathcal{P}$ and Π are respectively the model and policy spaces. Solving a Conf-MDP means finding the optimal model-policy pair (p, π) that maximizes the agent's expected return $J^{p,\pi}$, defined as:

$$V^{p,\pi} = \frac{1}{1-\gamma} \int_S d^{p,\pi}(s) \int_A \pi(a|s) r(s,a) da ds, \tag{3.1}$$

where $d^{p,\pi}$ is the γ-discounted state distribution parametrized by the transition model p and the policy π:

$$d^{p,\pi} = (1-\gamma)\mu(s) + \gamma \int_S d^{\pi}(s') p^{\pi}(s'|s) ds' \tag{3.2}$$

3.1.2 Algorithms to Solve Conf-MDP

In this section, we go into details of the two state-of-the-art algorithms for solving Conf-MDPs. The first algorithm, presented in [29], is called *Safe Model-Policy Improvement* (SMPI) and represents the first approach to learn simultaneously the model and the policy in discrete environments with known dynamics. The second algorithm is called *Relative Entropy Model Policy Search* (REMPS) and it has been presented in [27] in order to overcome the limitations of SMPI, dealing with continuous environments without requiring the knowledge of the true model of the environment.

Safe Model-Policy Improvement

Safe Model-Policy Improvement (SMPI) is the first algorithm, proposed in [29], to solve a Conf-MDP. The SMPI algorithm jointly optimize the policy and the environment configuration. The proposed "safe" update rule guarantees that the new model-policy pair p', π' provides higher performance than the previous pair p, π. The difference between the performance $V_\mu^{p',\pi'}$ of the new pair p', π' and the performance $V_\mu^{p,\pi}$ of the current model-policy pair p, π can be bounded by the following lower-bound, as proved in [29]:

$$\underbrace{V_\mu^{p',\pi'} - V_\mu^{p,\pi}}_{\substack{\textit{performance}\\ \textit{improvement}}} \geq B(p', \pi') = \underbrace{\frac{\mathbb{A}_{p,\pi,\mu}^{p',\pi} + \mathbb{A}_{p,\pi,\mu}^{p,\pi'}}{1-\gamma}}_{\textit{adventage}} - \underbrace{\frac{\gamma \Delta Q^{p,\pi} D}{2(1-\gamma)^2}}_{\substack{\textit{dissimilarity}\\ \textit{penalization}}}. \tag{3.3}$$

The meaning of each symbol that composes the lower-bound $B(p', \pi')$ is beyond the scope of this section but it is worth notice that the lower-bound is composed by two terms, like in traditional performance improvement bounds in RL [33, 23]: the first term, *adventage*, represents how much gain in performance can be locally obtained by moving from (p, π) to (p', π'), whereas the second term, *dissimilarity penalization*, discourages updates towards model-policy pairs that are too far away. The idea of the algorithm is to select at each iteration the model-policy pair (p', π') that maximize the aforementioned lower-bound:

$$p', \pi' = \arg\max_{p,\pi} B(p, \pi). \tag{3.4}$$

Following the approach proposed in [33], the new model policy pair (p', π') can be formulated as linear combination of the current model-policy pair (p, π) and a target model-policy pair $(\bar{p}, \bar{\pi})$:

$$\pi' = \alpha\bar{\pi} + (1 - \alpha)\pi, \qquad p' = \beta\bar{p} + (1 - \beta)p, \tag{3.5}$$

where $\alpha, \beta \in [0, 1]$, $\bar{\pi} \in \Pi$ and $\bar{p} \in \mathcal{P}$. Therefore, we can rewrite equation 3.4 using the new symbols introduced in 3.5 and searching for values of α and β that maximizes the lowerbound $B(\alpha, \beta)$. However, it can be proven that the unique admissible solution is a saddle point, that is uninteresting for optimization purpose. Nevertheless, $B(\alpha, \beta)$ is continuous on the compact set $[0, 1]^2$ and so, for Weierstrass theorem, it admits a global maximum. Notice that such point is not a stationary point so it must lie on the boundary of $[0, 1]^2$. By setting to zero the equations $\frac{\partial B}{\partial \alpha}|_{\beta=0}$, $\frac{\partial B}{\partial \alpha}|_{\beta=1}$, $\frac{\partial B}{\partial \beta}|_{\alpha=0}$, $\frac{\partial B}{\partial \beta}|_{\alpha=1}$ we can the optimal values $\alpha_0^\star, \alpha_1^\star, \beta_0^\star, \beta_1^\star$. Hence, we can rewrite equation 3.4 as follow:

$$\alpha^\star, \beta^\star = \arg\max_{\alpha,\beta} \left\{B(\alpha, \beta) : (\alpha, \beta) \in \mathcal{V}\right\}, \tag{3.6}$$

where $\mathcal{V} \in \{(\alpha_0^\star, 0), (\alpha_1^\star, 1), (0, \beta_0^\star), (1, \beta_1^\star)\}$. The values $\alpha_0^\star$, $\alpha_1^\star$, $\beta_0^\star$ and $\beta_1^\star$ (shown in [29]) are not relevant in this section. The SMPI algorithm can be formalized by Algorithm 4, assuming that the *PolicyChooser* and *ModelChooser* procedures return respectively a policy and transition model following a greedy criterion.

Relative Entropy Model Policy Search

Although SMPI succeeded in showing the benefits of configuring the environment in some illustrative examples, it is quite far from being applicable to real-world scenarios. It suffers from two significant limitations. First of all, it is only applicable to problems with a finite state-action space,

Algorithm 4 Safe Model Policy Improvement

```
Initialize $\pi_0$, $p_0$
for i = 0,1,2 ... until $\epsilon$-convergence do
    $\bar{\pi}_i = PolicyChooser(\pi_i)$
    $\bar{p}_i = ModelChooser(p_i)$
    $\mathcal{V} \in \left\{(\alpha^\star_{0,i}, 0), (\alpha^\star_{1,i}, 1), (0, \beta^\star_{0,i}), (1, \beta^\star_{1,i})\right\}$
    $\alpha^\star_i, \beta^\star_i = \arg\max_{\alpha,\beta} \{B(\alpha, \beta) : (\alpha, \beta) \in \mathcal{V}\}$
    $\pi_{i+1} = \alpha^\star_i \bar{\pi}_i + (1 - \alpha^\star_i)\pi_i$
    $p_{i+1} = \beta^\star_i \bar{p}_i + (1 - \beta^\star_i)p_i$
end for
```

while the most interesting Conf-MDP examples have, at least, a continuous state space (e.g., the car configuration problem). Second, it requires full knowledge of the environment dynamics. This latter limitation is the most relevant since, in reality, we almost never known the true environment dynamics, and even if a model is available it could be too approximate or too complex and computationally expensive (e.g., the fluid-dynamic model of a car). To overcome these issues a new trust-region method called *Relative Entropy Model Policy Search* (REMPS) has been proposed in [27]. REMPS belongs to the trust-region class of methods [37] and takes inspiration from REPS [32].

REMPS is able to optimize simultaneously the policy and the configuration of the Conf-MDP by searching in the space of stationary distribution $d_{\pi_{\boldsymbol{\theta}}, p_{\boldsymbol{\omega}}}$ induced by a given policy $\pi_{\boldsymbol{\theta}}$ and transition model $p_{\boldsymbol{\omega}}$, where $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$ are respectively the parametrization of the policy and the transition model. Notice that $d_{\pi_{\boldsymbol{\theta}}, p_{\boldsymbol{\omega}}}(s, a, s')$ is the probability of ending up in state s' after having performed action a in state s. Let's define $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^p\}$ as the parametric policy space and $\mathcal{P}_\Omega = \{p_{\boldsymbol{\omega}} : \boldsymbol{\omega} \in \Omega \subseteq \mathbb{R}^q\}$ as the parametric model space. Given the policy space Π_Θ and the model space $\mathcal{P}_\Omega$, we can define $\mathcal{D}_{\Pi,\Omega}$ as the set of possible stationary distribution induced by a generic model-policy pair $(p, \pi) \in \mathcal{P}_\Omega \times \Pi_\Theta$.

The algorithm is divided in two main phases: *optimization* and *projection*. In the first phase, the algorithm searches for a new stationary distribution $d^\star$ that maximizes the performance of the agent, in a neighborhood of the current stationary distribution d. Notice that the constraint on the distance of the new distribution from the current one is expressed in terms of a threshold $k > 0$ on the KL-divergence. The optimization phase can be formalized by the following optimization problem:

$$d' = \underset{d' \in \Delta(S,A,S)}{\arg\max} \; J_{d'} \quad \text{s.t.} \quad D_{KL}(d'||d) \leq k. \tag{3.7}$$

Once we have found the new stationary distribution d', we want to find the policy $\pi_{\boldsymbol{\theta}}$ and the transition model $p_{\boldsymbol{\omega}}$ that will induce that stationary distribution d'. However, the stationary distribution d' could fall outside the space of the representable stationary distribution $\mathcal{D}_{\Pi,\Omega}$. Therefore in the *projection* phase we retrieve a policy $\pi_{\boldsymbol{\theta}}$ and a configuration $p_{\boldsymbol{\omega}}$ inducing a stationary distribution $d_{\pi_{\boldsymbol{\theta}},p_{\boldsymbol{\omega}}}$ as close as possible to d'. More formally, the aim of projection phase is to solve the following optimization problem (PROJ$_d$):

$$\boldsymbol{\theta}', \boldsymbol{\omega}' = \underset{\theta \in \Theta, \boldsymbol{\omega} \in \Omega}{\arg\min} \, D_{KL}(d'||d_{\pi_\theta, P_\omega}). \tag{3.8}$$

However, solving this problem requires the knowledge of $d_{\pi_{\boldsymbol{\theta}},p_{\boldsymbol{\omega}}}$ in functional form. This limitation makes the optimization problem unfeasible since in most cases $d_{\pi_{\boldsymbol{\theta}},p_{\boldsymbol{\omega}}}$ cannot be computed in closed form. In [27], two relaxations are proposed in order to perform the optimization phase. The first relaxation consists in finding an approximation of the transition kernel $p'^{\pi'}$ induced by d' (PROJ$_{p^\pi}$):

$$\boldsymbol{\theta}', \boldsymbol{\omega}' = \underset{\theta \in \Theta, \boldsymbol{\omega} \in \Omega}{\arg\min} \, \mathbb{E}_{s \sim d'} \left[D_{KL}(p'^{\pi'}(\cdot|s)||p_{\boldsymbol{\omega}}^{\pi_{\boldsymbol{\theta}}}(\cdot|s)) \right]. \tag{3.9}$$

By the way, as before, we can perform this optimization problem only if we are able to compute $p_{\boldsymbol{\omega}}^{\pi_{\boldsymbol{\theta}}}$ in functional form. This is possible only if the action space is finite as, in that case, we can marginalize over the action space the transition model: $p_{\boldsymbol{\omega}}^{\pi_{\boldsymbol{\theta}}}(s'|s) = \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) p_{\boldsymbol{\omega}}(s'|s)$. The second relaxation overcomes this issue separating projections for the policy and the transition model (PROJ$_{\pi,p}$):

$$\boldsymbol{\theta}' = \underset{\theta \in \Theta}{\arg\min} \, \mathbb{E}_{s \sim d'} \left[D_{KL}(\pi'(\cdot|s)||\pi_{\boldsymbol{\theta}}(\cdot|s) \right] \tag{3.10}$$

$$\boldsymbol{\omega}' = \underset{\omega \in \Omega}{\arg\min} \, \mathbb{E}_{s,a \sim d'} \left[D_{KL}(p'(\cdot|s,a)||p_{\boldsymbol{\omega}}(\cdot|s,a) \right]. \tag{3.11}$$

The REMPS method can be formalized by the Algorithm 5.

3.1.3 Applications of Conf-MDP

In many real-world problems, there is the possibility to configure some environmental parameters to improve the performance of a learning agent. Some examples could be car racing tasks, as already discussed, or student-teacher

Algorithm 5 Relative Entropy Model Policy Search

```
Initialize $\boldsymbol{\theta}_0$, $\boldsymbol{\omega}_0$ arbitrarly
for i = 0,1,2 ... until $\epsilon$-convergence do
    Collect N samples $\{(s_i, a_i, s'_i, r_i)\}_{i=1}^{N}$ with $d_{\pi_{\boldsymbol{\theta}_t}, p_{\boldsymbol{\omega}_t}}$
    (Optimization) Compute $d'$
    (Projection) Project $d'$ and obtain $\boldsymbol{\theta}_{t+1}$ and $\boldsymbol{\omega}_{t+1}$
end for
```

domain where a teacher (modeled as the configurator) has to propose exercises, characterized by different value of difficulty, to a student (the learning agent). The goal of both entities is to maximize the notions learned by the student.

Configurable Markov Decision Processes have proved to be useful also in policy space identification tasks [28] where the configurability of the environment can be leveraged to let the agent reveal its real potential. How Conf-MDPs can enhance the policy space identification task will be explained in the following section.

Policy Space Identification in Configurable Environment

We analyze the problem of identifying the agent's policy space in a Conf-MDP [28], by observing the agent's behavior and, possibly, exploiting the configuration opportunities of the environment. Let's suppose that the policy space of the agent is a subset of a known super-policy space Π_Θ induced by a parameter space $\Theta \subseteq \mathbb{R}^m$. Therefore, any policy $\pi_{\boldsymbol{\theta}}$ is induced by a m-dimensional parameter vector $\boldsymbol{\theta} \in \Theta$. However, the agent is able to control only a smaller number $m^\star < m$ of parameters (which are unknown), while the remaining ones are set to zero. Our goal is to identify what are the parameters that the agent can control, given a set of demonstration of the optimal policy $\pi^\star$. It is important to notice that there could be controllable parameters that, given the peculiarities of the environment, are useless for achieving the goal and its optimal value is actually zero. This makes the problem of policy identification much harder because it becomes challenging to distinguish uncontrollable parameters from useless controllable parameters since the optimal value in both cases will be zero. In [28], it has been proved that we can leverage the configurability of the environment in order to let the agent reveal what is its real potential, clarifying the distinction between uncontrollable parameters and useless controllable ones. Intuitively, a controllable parameters may be useless in an environment configuration but crucial in another one.

3.1.4 Open Questions in Conf-MDPs

In this section we have explored in depth the details of Configurable Markov Decision Processes. All the considerations brought forward so far are based on the following fundamental hypothesis:

The learning agent and the configurator share the same objective.

In other words, the learning agent and the configurator optimize the same reward function. As already discussed in the introduction of this section, from an abstract point of view we can consider the agent and the configurator as two separate entities acting in a fully-cooperative scenario. However, from a practical point of view, it could be misleading to adopt a cooperative multi-agent approach. The supervisor acts externally, at a different level and could be, possibly, totally transparent to the learning agent. But what if the configurator did not not have the same intentions of the agent? Would Conf-MDPs be able to model that situations? Would the framework need to me extend? In section 4, we will answer all these questions in depth.

3.2 Other Related Works

In this section we provide some connections between the work presented in this thesis and the Online Learning literature. As already briefly described in the Section 1, in the next section we present an extension of Conf-MDPs called *Non-Cooperative Configurable Markov Decision Process* (NConf-MDP), in order to deal with scenarios where the configurator and the learning agent are not cooperative. The role of the configurator is altering the environment dynamics in order to optimize its own reward function; while, the aim of the agent is to learn the optimal policy in the environment configuration chosen by the configurator.

Many works in the Online Learning literature have exploited the idea of altering the environment dynamics to improve the learning process of the agent. For instance, in *Curriculum Learning* [6], the agent learns in a sequence of environments of increasingly difficulty. This method takes inspiration by the way humans are used to learn. When we want to learn a new skill, like playing guitar, we start doing very simple exercises and then move on to more complex ones. Similarly, in Curriculum learning, we alter the environment dynamics increasing the level of difficulty shaping the learning process of the agent with possible benefits on the learning speed, e.g. [11, 15]. By the way, while the configuration of the environment is an *intrinsic* property of the NConf-MDP, in Curriculum Learning the configuration phase is performed in simulation only.

Moreover, there are many works in literature treating the problem of environment configuration in a non-cooperative manner. For instance, *robust control* literature [31, 22] studies the interaction between the agent and the non-cooperative configurator, in order to compute a robust policy for the agent. The underlying idea is that if the agent is able to behave optimally even in the worst possible environments, we end up with a policy that is resilient to unpleasant situations. Hence, while the agent is interested in maximizing its expected return, the configurator tries to challenge the agent presenting uncomfortable environments.

In [16], an extension of MDP has been proposed to deal with situation where an adversary tries to interfere with the reward generating process altering the transition probabilities. In addition, the field of planning, environment configuration carried out by an external entity has been studied as a form of *environment design* [45].

In Section 5, we introduce two algorithms that are able to solve NConf-MDPs, i.e. finding the configuration that maximizes the configurator's reward function. In particular, in our framework, the agent and the configurator interacts sequentially: the configurator selects a configuration among a set of possible configurations and the agent learns the optimal policy in the selected configuration. The configurator observes trajectories of the agent's optimal policy and leverages the gathered information to select next configurations.

Our algorithms is inspired by the principle of optimism in face of uncertainty for stochastic multi-armed bandits e.g. [24, 2, 18, 25] and MDPs e.g. [3, 5, 4]. Moreover, the problem of configuration learning can be cast as a Multi-armed Bandit problem [25], where the configurator can pull different arms, i.e. configurations, and receives a stochastic return based on the policy learned by the agent in the selected configuration. Hence our algorithms are related to structured bandits or bandits with correlated arms.[1]

[1]In our case, playing a configuration provides information about the agent's reward, which in turns provides information about the value of all configurations.

Chapter 4

Non-Cooperative Configurable Markov Decision Processes

In this chapter, the novel framework *Non-Cooperative Markov Decision Processes* (NConf-MDP) will be introduced mainly focusing on the theoretical aspects. The algorithmic and experimental details will be analysed in the next chapters. In Section 4.1, we will introduce the main reasons behind the formulation of this novel framework providing some application domains. Finally, in Section 4.2, the formal definition of NConf-MDP will be presented.

In this thesis, we will consider finite-horizon MDPs with finite state and actions only. In particular, a *finite-horizon Configurable Markov Decision Process* (Conf-MDP) is defined as $\mathcal{CM} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r, H \rangle$ and extends finite-horizon MDPs (Section 2.1.7) considering a configuration space $\mathcal{P}$ instead of a configuration model p.

From now on, we will consider the reward as a state-only function. Note that a state-only reward function $r : \mathcal{S} \to [0, 1]$ is always equivalent to a state-action reward function $\tilde{r} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that satisfies the following condition:

$$\tilde{r}(s, a) = r(s) \quad \forall s \in \mathcal{S} \quad \forall a \in \mathcal{A}.$$

Hence, all the considerations made so far using state-action reward functions are still valid also in the state-only setting.

4.1 Non-Cooperative Configurable Markov Decision Process

As already discussed in Section 3.1, Conf-MDP can model scenarios where an agent has to learn in a configurable environment. Thanks to this framework we are able to enhance the learning process of the agent and reach higher performance finding the model-policy pair that maximize the agent's reward function. However, as anticipated in Section 3.1.4, Conf-MDPs cannot deal with situations where the configurator and the agent have different interests. In this section, we present a new framework, called *Non-Cooperative Configurable Markov Decision Process* (NConf-MDP), that has been introduced in order to model a non-cooperative interaction between the agent and the configurator [35].

For instance, in a supermarket a customer has to buy some groceries, possibly spending as little time as possible; while the supermarket's owner has to arrange products on shelves in order to maximize the supermarket's revenue. This setting is clearly non-cooperative since the customer (the agent) and the supermarket's owner (the configurator) do not share the same interest and a classic Conf-MDP is not able to model this kind of interactions. Since the supermarket's owner does not know the customer's shopping needs - namely, the reward function that the agent is optimizing - he can try almost randomly different configurations and evaluate the customer reaction. However, if the owner knew the reward function that the customer is optimizing, he could infer what will be the customer's behavior in all the configurations and select configurations that maximize supermarket's revenues, for instance, placing complementary products near the products needed by the customer. Below, we present our novel framework *Non-Cooperative Configurable Markov Decision Process* (NConf-MDP) as an extension of Conf-MDP in order to deal with non-cooperative scenarios.

Definition 4.1.1. *A Non-Cooperative Configurable Markov Decision Process (NConf-MDP) is defined by a tuple* $\mathcal{NCM} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r_c, r_o, H \rangle$, *where* $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, H \rangle$ *is a Conf-MDP without reward and* r_c, $r_o : \mathcal{S} \times \mathcal{A} \to [0, 1]$ *are the configurator and the agent (opponent) reward functions, respectively.*

Before diving into the details of this novel framework, let's formalize the definition of policy, Q-function and V-function in a NConf-MDP. As in classic finite-horizon MDP, named H the horizon, a policy $\pi = \langle \pi_1, \pi_2 \dots, \pi_H \rangle \in \Pi_D^H$ is defined as a sequence of *deterministic* decision rules π_i. Each decision rule $\pi_i : \mathcal{S} \to \mathcal{A}$ is a deterministic mapping between states and actions. Given a policy $\pi = \langle \pi_h \rangle_{h \in [H]} \in \Pi_D^H$ and a configuration $p \in \mathcal{P}$, we can define the configurator and the agent (opponent) Q-functions, for every $(s, a) \in \mathcal{S} \times \mathcal{A}$ and $h \in [H]$:

$$Q_{c,h}^{\pi,p}(s,a) = \mathop{\mathbb{E}}_{s_{h'+1} \sim p, \pi} \left[\sum_{h'=h}^{H} r_c(s_{h'}) | s_h = s, a_h = a \right],$$
$$Q_{o,h}^{\pi,p}(s,a) = \mathop{\mathbb{E}}_{s_{h'+1} \sim p, \pi} \left[\sum_{h'=h}^{H} r_o(s_{h'}) | s_h = s, a_h = a \right].$$

Now, we can define the value functions as $V_{c,h}^{\pi,p} = Q_{c,h}^{\pi,p}(s, \pi_h(s))$ and $V_{o,h}^{\pi,p} = Q_{o,h}^{\pi,p}(s, \pi_h(s))$, which represent the expected return for playing policy π in a model p starting in state s at time instant h. Moreover, we denote with $V_c^{\pi,p} = \mathbb{E}_{s \sim \mu}[V_{c,1}^{\pi,p}(s)]$ and $V_o^{\pi,p} = \mathbb{E}_{s \sim \mu}[V_{o,1}^{\pi,p}(s)]$ respectively the expected returns for the configurator and the agent.

4.2 Problem Formulation

In this section, we go into details of the formulation of the problem of learning a configuration in a non-cooperative scenario. Let us start introducing the concept of optimality in a NConf-MDP. In classic Conf-MDP [29], it is easy to express an optimality criterion since the configurator and the agent share the same reward function therefore a model-policy pair is optimal in a Conf-MDP if it maximizes the agent's return (that is the same for the configurator). In a NConf-MDP, where the configurator and the agent present different reward functions, the definition of an optimality criterion is not that straightforward. Indeed, a given model-policy pair could be profitable for the agent but very bad for the configurator and vice versa.

Think to the supermarket example introduced at the beginning of this chapter: the supermarket owner (configurator) decides how to arrange products on shelves and waits for the customer (agent) to adapt to changes. In this context, for instance, there exist configurations that are beneficial for customers - e.g. if the products they need most are close to each other - and others successful for the supermarket owner. It is crucial to underline that the supermarket owner is privileged as it has a fundamental advantage: acting first. This is considered an advantage since if the owner knew (or were able to infer) the behavior of the customer in all the configurations then he would choose directly the configuration that maximize the supermarket revenue. This kind of sequential interaction between agents has been largely studied in *Stackelberg games* literature and it is known as *leader-follower* protocol [8], as discussed in Section 2.3.2.

Hence, we can assume a sequential interaction between the configurator and the agent: first, the configurator (leader) chooses a transition model $p \in \mathcal{P}$ among the transition space $\mathcal{P}$ and then the agent (follower) plays its best response policy $\pi^\star_p \in \Pi^H_D$, that is assumed to be a *deterministic* optimal policy for the MDP $(\mathcal{S}, \mathcal{A}, p, \mu, r_o, H)$. More formally, the agent and the configurator solve at each iteration two different optimization problem. While the agent searches for an optimal policy $\pi^\star_p \in \Pi^H_D$ in the current configuration $p \in \mathcal{P}$:

$$\pi^\star_p \in \arg\max_{\pi \in \Pi^H_D} V_o^{\pi,p}, \tag{4.1}$$

the configurator searches for the transition model $p^\star \in \mathcal{P}$ that maximizes the configurator expected return assuming that the agent will play the optimal policy corresponding to that transition model:

$$p^\star \in \arg\max_{p \in \mathcal{P}} V_c^{\pi^\star_p,p}. \tag{4.2}$$

Or equivalently, using a game theoretic formulation, the pair $(p^\star, \pi^\star_{p^\star})$ represents a *Stackelberg equilibrium* of this game [8].

Remark 4.2.1 (Agent's policy is optimal)**.** *We assume that the policy played by the agent at every episode is optimal. In practice, this means that the configurator has to wait until the agent learns the optimal behavior in the chosen configuration before evaluating its own performance.*

The configurator knows everything about the NConf-MDP, except for the agent reward function r_o. We assume that, at each episode $k \in [K]$, the configurator selects a configuration $p_k \in \mathcal{P}$ and then observes a trajectory of H steps generated by the agent's best response policy $\pi^\star_{p_k}$. The goal of the configurator is to minimize the expected regret all over the episodes:

$$\mathbb{E}[\text{Regret}(K)] = \mathbb{E}\left[\sum_{k=1}^{K} \max_{p \in \mathcal{P}} V_c^{\pi^\star_p,p} - V_c^{\pi^\star_{p_k},p_k}\right] \tag{4.3}$$

In this work, we assume that the configuration space $\mathcal{P} = \{p_1, p_2, \dots, p_M\}$ is composed by a finite number $M \in \mathbb{N}$ of *stochastic* transition models. In order to streamline the notation, we will call π_i, instead of $\pi^\star_{p_i}$, the best response policy of the agent in configuration π and $\pi_{i,h}$ indicates the decision rule associated with policy π_i at time step h . Moreover, we denote with V_i, instead of $V_c^{\pi^\star_{p_i}, p_i}$, the configurator's expected return when transition model p_i is selected and policy π_i is played by the agent.

In the next chapter, we will understand how the configurator can leverage the information provided by trajectories sampled from the optimal policy π_i in configuration p_i in order to identify the optimal configuration.

Chapter 5

Optimistic Configuration Learning

In this chapter, we will discuss the details of the two algorithms proposed in this manuscript: *Action-feedback Optimistic Configuration Learning* (Af-OCL) and *Reward-feedback Optimistic Configuration Learning* (Rf-OCL). The two algorithms differ by the way they handle the information provided by demonstrations of best response policies in each configuration. We discuss two types of feedback:

- *Action-feedback* (Af). The agent's trajectories observed by the configurator are composed by state and actions only.
- *Reward-feedback* (Rf). The agent's trajectories observed by the configurator are composed by states, actions and a noisy version reward.

It is worth to notice that Rf is more demanding than Af since it requires a noisy version of reward sample. On the other hand, Rf uses the additional information to transfer information across different configurations with possible performance benefits. Both algorithms are based on the following assumption.

Assumption 1. *The agent's best response policy π_i is deterministic for all the configuration $i \in [M]$. Moreover, every time the configurator selects a given configuration i the agent will play always the same optimal deterministic policy π_i.*

While, in the Action-feedback setting the configurator observes only the actions performed by the agent in each state, in the Reward-feedback setting a noisy version of the immediate rewards are provided to the configurator.

The reward signal can be used to transfer the knowledge gathered from a configuration across all the other configurations. In other words, Rf-OCL leverages the *structure* of underlying MDPs.

5.1 Action-feedback Optimistic Configuration Learning

In this section we treat the action-feedback (Af) setting in which the configurator observes the agent's trajectories composed by states and actions only:

$$\langle s_1, a_1, \dots, s_{H-1}, a_{H-1}, s_H \rangle,$$

where $a_h = \pi_{i,h}(s_h)$.

The proposed algorithm, *Action-feedback Optimistic Configuration Learning* (Af-OCL), is based on the OFU principle introduced in Section 2.2.2. Indeed, at each episode the configurator selects the configuration that maximize an optimistic estimate of its expected return. To do so, the algorithm maintains, for each configuration, a set of *plausible* policies - that provably contains the optimal policy of the agent. Then, the configurator will select the configuration that maximize its expected return, considering an optimistic policy among the set of *plausible* policies.

More specifically, naming M the number of configurations, H the time horizon and K the number of episodes, we can denote with $\mathcal{A}^i_{k,h}(s) \subseteq \mathcal{A}$ the set of plausible actions in state s at step h for configuration p_i at the beginning of episode k. Once the agent visit state s before episode k at given time step h in a given configuration i we can set $\mathcal{A}^i_{k,h}(s) = \{\pi_i(s)\}$. This is due to the fact that the agent policy is deterministic and so when we observed the action performed in a given configuration i, state s and time step h we are sure that for all future episodes the action performed by the agent in analogous situations will be the same. In all the other cases we have no information of the actions that the agent will perform therefore we set $\mathcal{A}^i_{k,h}(s) = \mathcal{A}$.

Now that we have defined the set of plausible agent's actions, we can formalize the criterion through which the configurations are selected. At each episode $k \in [K]$, the configurator computes an optimistic approximation of its expected return $\widetilde{V}^i_k$ for each configuration $i \in [M]$. In order to compute these optimistic returns, we can adopt an extension of the value iteration algorithm described by Algorithm 6. In particular, the only difference with

Algorithm 6 Optimistic Value Iteration

```
1: $\widetilde{V}^i_{k,H}(s) = 0 \quad \forall s \in \mathcal{S}$
2: for $h = H-1, H-2, \dots, 1$ do
3:     $\widetilde{V}^i_{k,h}(s) = r_c(s) + \max_{a \in \mathcal{A}^i_{k,h}(s)} \sum_{s' \in \mathcal{S}} p_i(s'|s,a) \widetilde{V}^i_{k,h+1}(s')$
4: end for
5: return Expected return $\sum_{s \in \mathcal{S}} \widetilde{V}^i_{k,1}(s)\mu(s)$
```

the classical value iteration algorithm is the update step:

$$\widetilde{V}^i_{k,h}(s) = r_c(s) + \max_{a \in \mathcal{A}^i_{k,h}(s)} \sum_{s' \in \mathcal{S}} p_i(s'|s,a) \widetilde{V}^i_{k,h+1}(s'). \tag{5.1}$$

In fact, the support of maximization operator is set to $\mathcal{A}^i_{k,h}(s)$ rather than $\mathcal{A}$ (Algorithm 6).

Therefore, for visited pair (s, h) the maximization over the actions reduces to the evaluation of the transition model in the agent's action $\pi_{i,h}(s)$. On the other hand, in non-visited pairs (s, h), we will consider the action that maximizes the configurator return: that makes *optimistic* the approximation of the configurator's expected return. Hence, we have that $\widetilde{V}^i_{k,h}(s) \geq V^i_h(s)$ deterministically for all $s \in \mathcal{S}$, $h \in [H]$ and $i \in [M]$. Thus, at each episode $k \in [K]$ the configurator plays the transition model p_{I_k} maximizing the optimistic approximation of its expected return $\widetilde{V}^i_k$:

$$I_k \in \arg\max_{i \in [M]} \widetilde{V}^i_k.$$

The AfOCL procedure is reported in Algorithm 7. The time complexity of the algorithm is $\mathcal{O}(KMHS^2A)$, since at each episode $k \in [K]$ and for each configuration $i \in [M]$ we run Algorithm 6, whose complexity is bounded by that of value iteration $\mathcal{O}(HS^2A)$.

5.1.1 Regret guarantees

In this section, we provide an expected regret bound for the AfOCL algorithm. The detailed proof is discussed in Appendix A.2.1.

First of all, let's remind that the expected regret is defined as follow:

$$\mathbb{E}[\text{Regret}(K)] = \sum_{i \in [M]: \Delta_i > 0} \Delta_i \, \mathbb{E}[N_i], \tag{5.2}$$

Algorithm 7 Action-feedback Optimistic Configuration Learning (AfOCL).

```
1: Input: $\mathcal{S}$, $\mathcal{A}$, $H$, $\mathcal{P} = \{p_1, \ldots, p_M\}$
2: Initialize $\mathcal{A}^i_{1,h}(s) = \mathcal{A}$ for all $s \in \mathcal{S}$, $h \in [H]$, and $i \in [M]$
3: for episodes $1, 2, \ldots, K$ do
4:     Compute $\widetilde{V}^i_k$ for all $i \in [M]$
5:     Play $p_{I_k}$ with $I_k \in \arg\max_{i \in [M]} \widetilde{V}^i_k$
6:     Observe $(s_{k,1}, a_{k,1}, \ldots, s_{k,H-1}, a_{k,H-1}, s_{k,H})$
7:     Compute the plausible actions for all $s \in \mathcal{S}$ and $h \in [H]$:
           $$\mathcal{A}^i_{k+1,h}(s) = \begin{cases} \{a_{k,h}\} & \text{if } i = I_k \text{ and } s = s_{k,h} \\ \mathcal{A}^i_{k,h}(s) & \text{otherwise} \end{cases}$$
8: end for
```

where $\mathbb{E}[N_i]$ is the expected number of times a suboptimal configuration i is chosen and $\Delta_i = V^\star - V_i$ is the suboptimality gap, i.e. the difference between the expected return of the optimal configuration and a suboptimal configuration i.

The main challenge of AfOCL is estimating the agent's best response policy in every model. Although all the states could be visited by the agent - thanks to the stochasticity of the environment - the agent may end up in some states with low probability and this makes the task of the configurator more difficult since some configurations are selected a high number of times before being discarded. In particular, thanks to the determinism of the agent's policies, it can be proven that the expected number of times a suboptimal configuration i is selected is proportional to $\frac{1}{\Delta_i}$, ignoring dependencies on other quantities (in contrast to standard bandits where it is proportional to $\frac{1}{\Delta_i^2}$):

$$\mathbb{E}[N_i] \propto \frac{1}{\Delta_i} \tag{5.3}$$

This result combined with Equation 5.2, removes the dependency on suboptimality gaps of the expected regret bound.

The second surprising result is that the upper bound for the expected number of times a suboptimal configuration is selected does not depend on the number of episodes K. In particular, $\mathbb{E}[N_i]$ can be bounded as follow:

$$\mathbb{E}[N_i] \leq 2\frac{H^3 S^2}{\Delta_i - c}, \tag{5.4}$$

where H is the time horizon, S is the number of states and $c > 0$ is an arbitrary positive constant. Hence, we can conclude that the expected regret bound is constant and independent on the number of episodes K.

$$\mathbb{E}[\text{Regret}(K)] = \sum_{i \in [M]:\Delta_i > 0} \Delta_i \, \mathbb{E}[N_i] \leq \sum_{i \in [M]:\Delta_i > 0} \Delta_i 2 \frac{H^3 S^2}{\Delta_i - c} \leq 2MH^3S^2,$$

having taken the infimum over $c > 0$.

Theorem 5.1.1 (Regret of AfOCL)**.** *Let $\mathcal{NCM} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r_c, r_o, H)$ with $\mathcal{P} = \{p_1, \dots, p_M\}$ be the M finite-horizon MDPs of the problem. The expected regret of AfOCL at every episode $K > 0$ is bounded by:*

$$\mathbb{E}[\textit{Regret}(K)] \leq MH^3S^2. \tag{5.5}$$

5.2 Reward-feedback Optimistic Configuration Learning

One of the main limitations of AfOCL is that there is no way to transfer information across different configurations. In particular, the sample collected in a given configuration cannot be leveraged to update estimates associated with other configurations. In other words, AfOCL does not leverage the *structure* of the environment, represented by the agent reward function r_o. Since the agent reward function r_o remains the same in all the configurations, it represents a link between them. Indeed, if the configurator knew exactly the reward function of the agent it could compute the agent's optimal policy in all the configurations and select always the configuration that maximizes its own reward function r_c without interacting with the agent. Hence, the reward function of the agent represents a valuable information that could be used to speed up the identification of the best configuration. In this section, we present an algorithm called *Reward-feedback Optimistic Configuration Learning* (RfOCL) that leverages noisy samples of the agent's reward function coming from trajectories to transfer information across different configurations. In particular, the configurator observes trajectories composed by states, actions and rewards:

$$\langle s_1, \widetilde{r}_1, a_1, \dots, s_{H-1}, \widetilde{r}_{H-1}, a_{H-1}, s_H, \widetilde{r}_H \rangle,$$

where $a_h = \pi_{i,h}(s_h)$ and $\widetilde{r}_h$ is a noisy version of $r_o(s)$.

Based on the noisy reward sample, RfOCL is able to compute an estimate of the agent reward function maintaining a confidence interval $\mathcal{R}_k(s) = [\underline{r}_{o,k}(s), \overline{r}_{o,k}(s)]$ for every states $s \in \mathcal{S}$ using the samples collected up to episode $k-1$ in all the configurations. This confidence interval can easily be computed applying Höeffding's inequality:

$$\widehat{r}_{o,k}(s) \pm \sqrt{\frac{\log(SHk^3)}{\max\{N_k(s), 1\}}}, \tag{5.6}$$

where $N_k(s)$ is the number of times state s has been visited up to episode k and $\widehat{r}_{o,k}(s)$ is the sample mean of the noisy reward samples collected in the first $k-1$ episodes in state s.

The confidence interval on the agent's reward function represents a source of information for all the configurations. In order to leverage this information, we can derive the possible agent's behaviors in all the configurations. To do so, for each configuration, we can compute a confidence interval on Q-values $\mathcal{Q}^i_{k,h}(s,a) = [\underline{Q}^i_{o,k,h}(s,a), \overline{Q}^i_{o,k,h}(s,a)]$ induced by the reward confidence interval $\mathcal{R}_k$. The bounds $\underline{Q}^i_{o,k,h}$ and $\overline{Q}^i_{o,k,h}$ of the confidence interval can be computed performing two separate instances of value iteration algorithm with two different reward function respectively $\underline{r}_{o,k}(s)$ and $\overline{r}_{o,k}(s)$.

More specifically $\underline{Q}^i_{o,k,h}(s,a)$ is computed applying the following Bellman equation:

$$\underline{Q}^i_{o,k,h}(s,a) = \underline{r}_{o,k}(s) + \sum_{s' \in \mathcal{S}} p_i(s'|s,a) \max_{a' \in \mathcal{A}} \underline{Q}^i_{o,k,h+1}(s',a'),$$

and $\underline{Q}^i_{o,k,H}(s,a) = \underline{r}_{o,k}(s)$.

On the other hand, $\overline{Q}^i_{o,k,h}(s,a)$ is computed as:

$$\overline{Q}^i_{o,k,h}(s,a) = \overline{r}_{o,k}(s) + \sum_{s' \in \mathcal{S}} p_i(s'|s,a) \max_{a' \in \mathcal{A}} \overline{Q}^i_{o,k,h+1}(s',a'),$$

and $\overline{Q}^i_{o,k,H}(s,a) = \overline{r}_{o,k}(s)$.

The fact that the real agent reward function belongs to the confidence interval, i.e. $r_o \in \mathcal{R}_k$, implies that the Q-values induced by r_o belongs to the confidence interval, i.e. $Q^i_h \in \mathcal{Q}_{k,h}$.

Since the best response policy of the agent in each consideration is greedy w.r.t. the Q-values, we can use $\mathcal{Q}_{k,h}$ to restrict the set of plausible actions in a state *without* actually observing the agent playing the action in that state. For this purpose, we can say that, for a given triplet $(s, h, i) \in \mathcal{S} \times [H] \times [M]$, an action $a \in \mathcal{A}$ is *dominated* by an action $a' \in \mathcal{A}$ at episode $k \in [K]$ if $\overline{Q}^i_{o,k,h}(s, a) \leq \underline{Q}^i_{o,k,h}(s, a')$, i.e. the maximum value of $Q^i_{o,k,h}(s, a)$ is less then the minimum value of $Q^i_{o,k,h}(s, a')$ with high probability. Therefore, the actions that the agent may perform are those that are not dominated by any other actions. In other words, the plausible actions are those that have an upper Q-value larger than the maximum Q-value lower bound:

$$\widetilde{\mathcal{A}}^i_{k,h}(s) = \left\{ a \in \mathcal{A} : \overline{Q}^i_{o,k,h}(s, a) \geq \max_{a' \in \mathcal{A}} \underline{Q}^i_{o,k,h}(s, a') \right\}. \tag{5.7}$$

Clearly, the considerations made for the Action-feedback setting are still valid and every time we observe playing an action in $(s, h, i) \in \mathcal{S} \times [H] \times [M]$ we can set the plausible actions to the singleton $\{\pi_{i,h}(s)\}$. Based on this new definition of plausible actions, we can compute the optimistic estimate $\widetilde{V}^i_k$ of the configurator expected return as in Algorithm 6 and proceed playing the optimistic configuration. The pseudocode of RfOCL is reported in Algorithm 8. The computational complexity of an iteration of RfOCL is dominated by the value iteration (steps 5 and 9) leading, as for AfOCL, to $\mathcal{O}(KMHS^2A)$.

5.2.1 Regret guarantees

In this section, we provide an expected regret bound for RfOCL algoritm. The detailed proof is discussed in Appendix A.2.2.

Notice that RfOCL algorithms is an extension of AfOCL, therefore all the considerations presented in Section 5.1.1 are still valid. In particular, Theorem A.2.1 stands also for RfOCL. In addition, under centain conditions, we have proven that the expected regret bound of RfOCL is independent from the number of configurations M.

In order to prove this result we have to make the following assumption on the NConf-MDP.

Assumption 2. *There exists $\epsilon > 0$ such that:*

$$\min_{i \in [M]} \min_{s \in \mathcal{S}} \max_{h \in [H]} d^i_h(s) \geq \epsilon,$$

where $d^i_h(s)$ is the probability of visiting the state $s \in \mathcal{S}$ at time $h \in [H]$ in configuration p_i under the agent's best response policy π_i.

Algorithm 8 Reward-feedback Optimistic Configuration Learning (RfOCL)

```
1: Input: $\mathcal{S}$, $\mathcal{A}$, $H$, $\mathcal{P} = \{p_1, \dots, p_M\}$
2: Initialize $\mathcal{A}^i_{1,h}(s) = \mathcal{A}$ for all $s \in \mathcal{S}$, $h \in [H]$, and $i \in [M]$
3: Initialize $\overline{r}_{o,1}(s) = 1$, $\underline{r}_{o,1}(s) = 0$, and $N_{1,h}(s) = 0$ for all $s \in \mathcal{S}$ and
   $h \in [H]$
4: for episodes $1, 2, \dots, K$ do
5:     Compute $\widetilde{V}^i_k$ for all $i \in [M]$
6:     Play $p_{I_k}$ with $I_k \in \arg\max_{i \in [M]} \widetilde{V}^i_k$
7:     Observe
       $(s_{k,1}, \widetilde{r}_{k,1}, a_{k,1}, \dots, s_{k,H-1}, \widetilde{r}_{k,H-1}, a_{k,H-1}, s_{k,H}, \widetilde{r}_{k,H})$
8:     Compute $\overline{r}_{0,k+1}(s)$, $\underline{r}_{o,k+1}(s)$, and $N_{k+1,h}(s)$ for all $s \in \mathcal{S}$ and $h \in [H]$
       using $\widetilde{r}_{k,1} \cdots \widetilde{r}_{k,H}$ as in Equation (5.6)
9:     Compute $\underline{Q}^i_{o,k+1,h}(s,a)$, $\overline{Q}^i_{o,k+1,h}(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, $h \in [H]$,
       and $i \in [M]$
10:    Compute the plausible actions for all $s \in \mathcal{S}$ and $h \in [H]$:

       $$\mathcal{A}^i_{k+1,h}(s) = \begin{cases} \{a_{k,h}\} & \text{if } i = I_k \text{ and } s = s_{k,h} \\ \mathcal{A}^i_{k,h}(s) & \text{if } N_{k,h}(s) > 0 \\ \widetilde{\mathcal{A}}^i_{k+1,h}(s) & \text{otherwise} \end{cases}$$

       with $\widetilde{\mathcal{A}}^i_{k+1,h}(s)$ as in Equation (5.7).
11: end for
```

This means that the agent, in all the configurations, visits at least all the states once with non-zero probabilities. More formally, this assumption guarantees that for every model $i \in [M]$ and every state $s \in \mathcal{S}$ there exists a time instant $h \in [H]$ such that the probability of visiting state s at time h is strictly greater than 0. This assumption ensures that the confidence interval on the reward of *every* state shrinks at every episode. Assumption 2 is related to the concept of *ergodicity* of a MDP. Indeed, a MDP is said to be ergodic if each state is visited at least once under any policy [34]. Hence, Assumption 2 is less strict than the standard ergodicity since it requires the MDP to be ergodic only under the optimal policy. Under Assumption 2, we can prove the following regret guarantee.

Theorem 5.2.1 (Regret of RfOCL)**.** *Let $\mathcal{NCM} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r_c, r_o, H)$ with $\mathcal{P} = \{p_1, \dots, p_M\}$ be the M finite-horizon MDPs of the problem. Under Assumption 2, the expected regret of RfOCL at every episode $K > 0$ is* *bounded by:*

$$\mathbb{E}[\mathit{Regret}(K)] \leq \overline{K}\Delta + \frac{\pi^2}{3},$$

where $\overline{K}$ is the smallest integer solution of the inequality

$$K \geq \left(\frac{2(SH+1)^2 \log(SHK^3)}{\Delta_Q^2} + \sqrt{\frac{K}{2} \log(S\Delta K^2)} \right) \frac{1}{\epsilon},$$

$\Delta = \max_{i \in [M]} \Delta_i$*, i.e. the maximum suboptimality gap, and* Δ_Q *is the minimum positive gap of the agent's Q-values.*

As already anticipated, the regret bound does not depend on the number of configurations M. This result is crucial for practical applications as it allows the algorithm to scale when the number of configurations is large. However, as expected, the bound depends on the minimum visitation probability ϵ.

5.3 Discussion

In this chapter, we have presented two different types of feedback. Thanks to Assumption 1, stating that the agent's best response policies are deterministic, both algorithms are able to reach constant regret. Moreover, RfOCL allows eliminating the dependence on the number of configurations, assuming that the agent, for each configuration, visits all the states with a non-zero probability (Assumption 2). This leads RfOCL to achieve higher performance than AfOCL, especially when the number of configuration is large. On the other hand, RfOCL is more computational intensive than AfOCL (although the asymptotic complexity is the same) as it requires to compute, for each episode, the optimistic values of the agent Q functions for each model.

It is worth to notice that Assumption 2 must hold in order to exploit the advantages of RfOCL. Indeed, if we removed this assumption all theoretical guarantees of RfOCL are no longer valid and the two algorithms are likely to achieve the same results. However, both algorithms are based on the fundamental assumption on the determinism of the policy (Assumption 1). If we removed this assumption, we no longer have theoretic guarantees of the regret generated by the two algorithms. Anyway, it is reasonable to believe that the regret generated by AfOCL grows logarithmically with the number of episodes K, as in unstructured bandits. On the other hand, RfOCL still transfer information across different configurations leveraging the structure of the underlying MPD. Therefore, we conjecture that it will keep paying constant regret. The investigation of this case represents one of the main future directions of this work.

The problem of learning a configuration can be cast to a stochastic multi-armed bandit problem [25]. In fact, the configurator, at each episode, can pull different arms (configurations) and receives a random realization of its expected return. Hence, in general, we can learn a configuration using standard multi-armed bandit algorithms, like UCB1 [2]. Although this algorithms are less computational intensive, they suffer regret that grows logarithmically with the number of episodes K. Indeed, they do not exploit neither the fact that the agent's policy is deterministic nor the structure induced by the agent's reward function. In Chapter 6, we will compare our algorithms with UCB1 in different environments in order to show experimental evidences of the advantages provided by the proposed algorithms.

Chapter 6

Experimental Evaluation

"In God we trust, all others must bring data."

W. Edwards Deming

In this section, we evaluate our algorithms on three different domains comparing them with the standard implementation of UCB1 [2]. In Section 6.1, we will describe in details the domains used during the experimental phase. While, in Section 6.2, the results of the experiments will be presented, showing that our algorithms are able to achieve constant regret as discussed in Section 5.

6.1 Environmental domains

In this section we are going to describe in details three domains we used as environments in the experiments presented in this chapter: *Configurable Gridworld*, *Student-Teacher*, *Configurable Market*.

6.1.1 Configurable Gridworld

Configurable Gridworld is a variant of a classic 3×3 Gridworld. As shown in Figure 6.1, a Configurable Gridworld, in contrast to standard implementation of Gridworld, is characterized by an obstacle in the central cell. The agent starts in the cell $(0, 1)$ and its goal is to reach the final cell $(2, 1)$ with the minimum number of steps. If the agent is in the central cell $(1, 1)$ and performs action *"go right"*, it hits the obstacle and it is bounced back with probability p. The configurator can change the probability or *power* p of the obstacle and its goal is to maximize the number of time spent by the agent in the central cell.

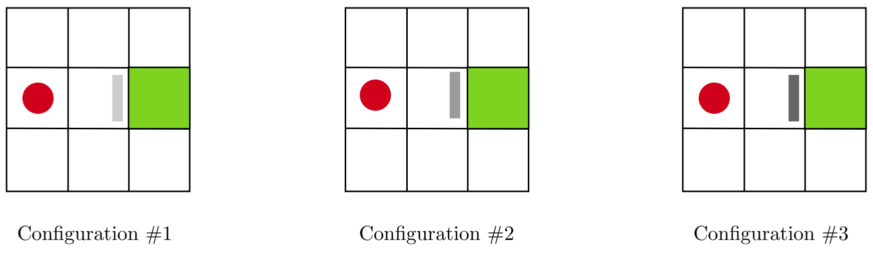

Figure 6.1: Configurable Gridworld: from left to right the 3 configurations represent increasing "power" of the obstacle.

In a classic Gridworld ($p = 0$), the optimal policy of the agent is trivially going straight to the final cell passing through the center of the Gridworld. On the other hand, in a Configurable Gridworld it is not that trivial. Indeed, if the power of the obstacle is small the agent keeps preferring the central path even if in some cases it will be bounced back by the obstacle. Instead, if the power of the obstacle is large then the agent will decide to avoid the interaction with the obstacle by passing close to the boundaries and this brings to very poor performances for the configurator. Hence, the role of the configurator is very sensitive: it is interested in increasing the power of the obstacle to let the agent remains in the central cell but not that much to let the agent prefer the “long path”. At implemetation level, the configurator selects a model among a set of M transition models with different values of p, obtained by a discretization of the interval $[0, 1]$.

6.1.2 Student-Teacher

Student-Teacher is an environment that models a basic interaction between a student and a teacher. The teacher has prepared a list of S exercises characterized by a different level of difficulty and its goal is to find the right sequence of exercises in order to optimize the learning process of the student. On the other hand, the student perceives the level of difficulties of the exercises in a different way and it can decide to not solve some exercises. The student’s goal is the same as that of the teacher, namely to start solving most difficult exercises as soon as possible. Nevertheless, the different perception of difficulty makes the environment non-cooperative.

We can model this scenario with a MDP with S states where each state represents an exercise - or equivalently a class of exercises with the same difficulty level. The states are ordered based on the difficulty assigned by

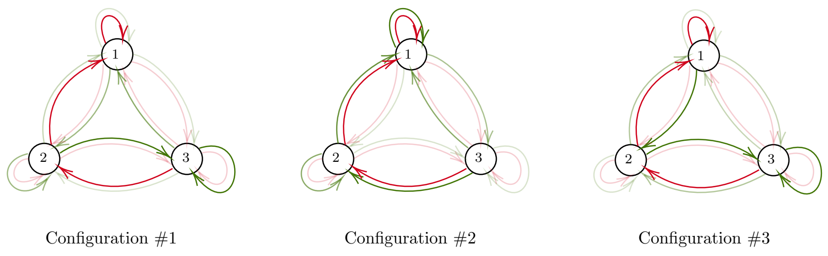

Figure 6.2: This picture shows 3 different configurations in a Teacher Student domain.

the teacher, i.e., the exercise associated with state s is more difficult (according to the teacher) than the one associated with s' if $s > s'$. In each state the student, i.e. the learning agent, can perform two actions *answer* or *not-answer*. If the agent decides to not answer in state s he will end up in state $s' = \max(s-1, 0)$ with probability 0.7 or in a random state with probability 0.3. Instead, if the agent decides to answer the teacher, i.e. the configurator, must decide which exercise should be proposed next. In practice, this can be implemented considering a set of M configurations that differ each other by the way they assign the probabilities to next states when the agent decides to answer. In the presented experiments, we have considered M random configurations.

An illustrative example of Student-Teacher domain is shown in Figure 6.7. Red arrows correspond to answer No, and green arrows to answer Yes. The transparency is due to the level of probability of every transition. The configurator can change the transition matrix for the answer Yes, instead the transition matrix for action No is fixed for all the configurations. The reward function of the configurator and the agent are the difficulty levels associated with the current state. Therefore, the configurator's immediate reward for each state are

$$r_c(s) = s$$

while the immediate rewards of the agent are a permutation of the one of the configurator since the student perceives differently the difficulty level of exercises.

6.1.3 Configurable Market

Configurable Market is a variant of a Gridworld for modelling the behavior of a customer in a marketplace. The customer (the agent) wants to buy a

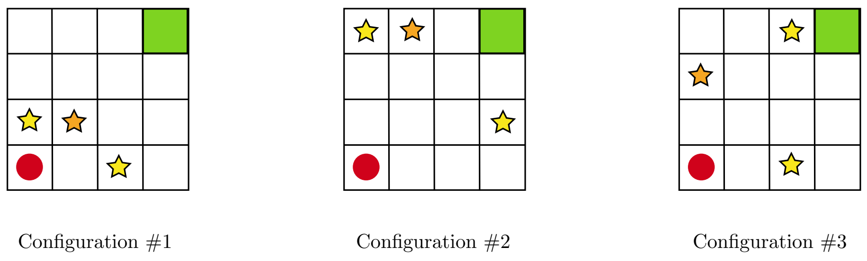

Figure 6.3: The figure shows a 4×4 configurable market. The red state is the starting state, instead the green state is the "end" state. The stars are the product and the orange star is the only product the agent is interested in.

set of products Q_A. On the other hand, the owner of the marketplace (the configurator) can change the location of all the products $Q \supset Q_A$. While the goal of the agent is minimizing the time spent in the marketplace, the configurator has to place products in the marketplace so to induce the agent to buy other products in addition to Q_A in order to maximize the marketplace's revenues. In Figure 6.3, we show an instance of a 4×4 Configurable Market where the configurator has to place 3 products ($|Q| = 3$) in the Gridworld but only one of this products is needed by the agent, i.e $|Q_A| = 1$. The reward of the agent is 0.9 in states with a product in Q_A and -1 in any other state. On the other hand, the reward of the configurator is 0 everywhere except in the states with any product in Q where it earns a bonus of 1.

At implementation level, the configurator cannot change the location of products in the Gridworld but it can select a transition model among a set of randomly-generated transition models. Anyway, from an abstract point of view, this is the same of shuffling the position of products in the gridworld.

6.2 Experiments

In the following section, our algorithms will be tested on the presented environmental domains. In particular, we have recorded the cumulative regret as a function of the episodes. We will show that our algorithms are able to converge to a zero-regret strategy, in contrast to UCB1 whose cumulative regret grows logarithmically. Additional details on the hyper-parameters used during the experiments are provided in Appendix B.

6.2.1 Configurable Gridworlds experiments

The results of the experiments in the Configurable Gridworld domain are shown in Figures 6.4, 6.5 and 6.6.

In the first experiment (Figure 6.4), we test the three algorithms in two settings that differs for the number of configurations, respectively 10 and 30. We have considered 3000 episodes and a time horizon $H = 10$. As expected from the theoretical guarantees, our algorithms are able to converge to a zero-regret strategy while UCB1 keep paying logarithmic regret. Moreover, RfOCL is able to converge in both cases in less of 500 episodes, while AfOCL needs 2000 episodes with $M = 10$ and 3000 episodes with $M = 30$. It is worth notice that increasing the number of configurations has a negative impact on the performance of AfOCL, while the performance of RfOCL remains quite stable. This is not surprising because, being Assumption 2 fulfilled - due to the stochasticity of the environment - the RfOCL regret bound does not depends on the number of configurations.

In Figure 6.5, we show a more extreme setting. We consider only 3 configurations, designed to not fulfilled Assumption 2. In this case, all the theoretical guarantees on RfOCL are no longer valid since the optimal policy of the agent generates a non-ergodic Markov chain. As can be seen in Figure 6.5, as expected, RfOCL does not bring any advantages compared to AfOCL. However, since Assumption 1 on the determinism of the agent's best response policies is still valid, both our algorithms converges to a constant regret.

Finally, in the last experiment, shown in Figure 6.6, we test our algorithm with a large number of configurations respect to the first experiment. As expected, being the ergodic assumption fulfilled, RfOCL is able to outperform AfOCL and UCB1 for higher number of configuration. This last experiment clearly highlights the difference between the regret bound of AfOCL and RfOCL. Indeed, since AfOCL regret upper-bound depends linearly on the number of configuration the algorithm always produces higher regret respect to RfOCL, whose regret upper-bound is independent on the number of configuration. Moreover, this gap gets larger as the number of configuration increases.

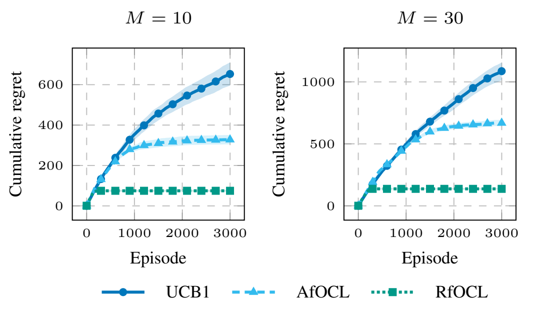

Figure 6.4: Cumulative regret as a function of the episodes for the Gridworld experiment. 50 runs, 98% c.i.

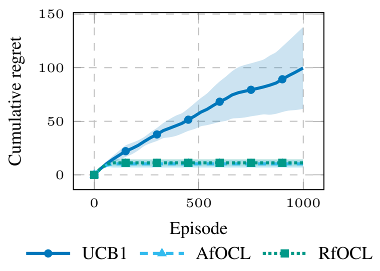

Figure 6.5: Cumulative regret as a function of the episodes for the Gridworld experiment in the extreme setting. 50 runs, 98% c.i.

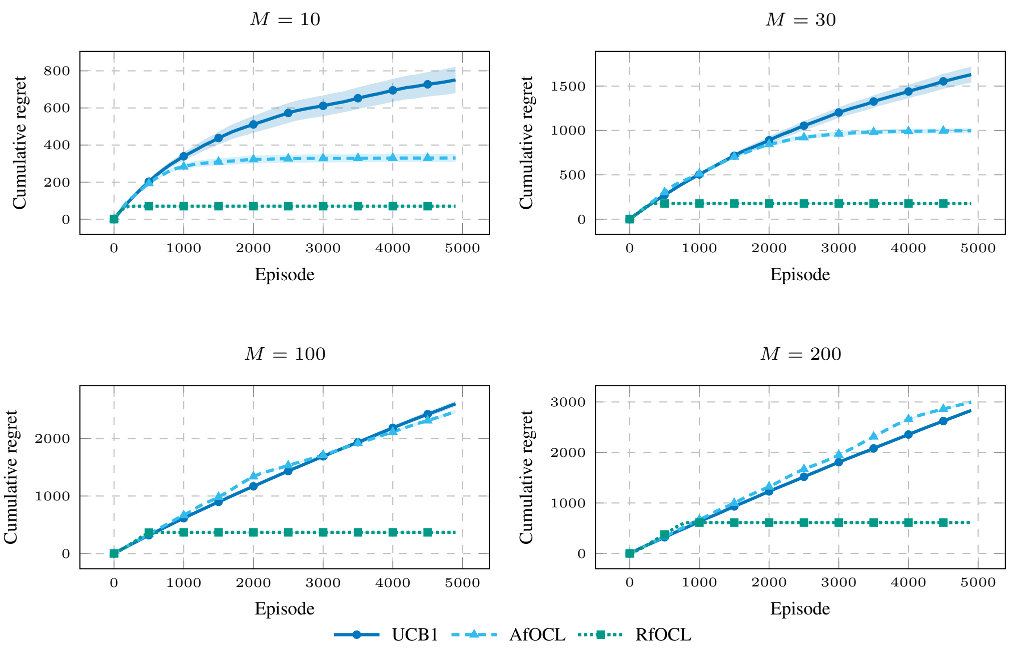

Figure 6.6: Cumulative regret as a function of the episodes for the Gridworld experiment with higher number of configuration. 50 runs, 98% c.i

6.2.2 Student-Teacher experiments

In this section, we present the experiment in the Student-Teacher domain, shown in Figure 6.7. In particular, we test the algorithms in three different settings characterized by a number of configurations $M \in \{40, 60, 100\}$ and horizon $H = 10$. Each curve in Figure 6.7, has been obtained averaging the outcomes of 50 independent runs. In each runs, we construct M different randomly-generated configurations and we change the *difficulty* of exercises from the agent's perspective, i.e. the agent's reward function.

As in the previous experiments, it is evident how the increasing use of structure positively affects performances. Indeed, RfOCL performs always better than AfOCL and UCB1 and the higher is the number of configurations the more this advantage is perceptible. Anyway, both AfOCL and RfOCL are able to converge to constant regret in all the cases under consideration.

6.2.3 Configurable Market experiments

In this section, we discuss the result obtained in the experiment in the Configurable Market domain, shown in Figure 6.8. The experiment has been performed in a 4×4 Configurable Market with number of configurations $M = 10$ and time horizon $H = 15$. The configurator can place 3 products on the gridworld, of which only 1 is needed by the agent. The 10 configurations are randomly generated at each run.

The result we obtained in this last experiment is coherent with the previous ones. In fact, as expected, UCB1 performs very poor with respect to our algorithms. On the other hand, RfOCL and AfOCL perform in a similar way. This is not surprising since the number of configuration is small and the advantages brought by the Reward-feedback setting are not very appreciable. However, RfOCL at the end of the available episodes is able to achieve quasi-constant regret.

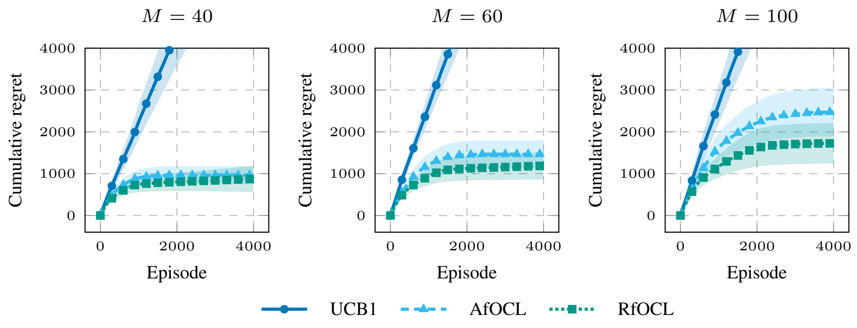

Figure 6.7: Cumulative regret as a function of the episodes for the Student-Teacher experiment. 50 runs, 98% c.i.

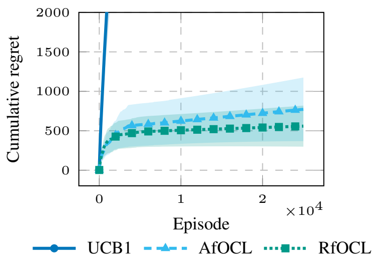

Figure 6.8: Cumulative regret as a function of the episodes for the Configurable Market experiment. 50 runs, 98% c.i.

Chapter 7

Conclusions and Future Research Directions

In this chapter, we discuss the main contribution of this thesis and the future research directions of the proposed topic.

The first contribution of this work is the proposal of the novel framework called *Non-Cooperative Configurable Markov Decision Process* (NConf-MDP) as an extension of the existing framework known as Configurable Markov Decision Process (Conf-MDP) introduced in [29]. Before the introduction of the Conf-MPDs, the environment was considered as a *fixed* entity out of the agent control. Indeed, when we firstly introduced Markov Decision Processes in Section 2.1, we defined the environment as *"everything outside the agent"* [39]. Conf-MDPs have overcome this limitation by providing the agent with partial control over the environment. In fact, thanks to learning algorithms like SMPI or REMPS proposed in [29, 27], the agent is able to learn simultaneously the optimal policy and the optimal environment configuration. Although Conf-MDPs have revealed its great potential in modeling configurable environments, they are not able to deal with those scenarios where the configurator does not share the same interests of the learning agent. This is a great lack of expressiveness since many real world applications are characterized by a non cooperative interaction between the configurator and the learning agent. NConf-MDPs overcome these issues allowing the configurator to optimize a reward function different from the one of the agent. We have discussed how the sequential interaction between the configurator and the agent can be modeled as a Stackelberg game where the leader is the configurator who firstly selects a configuration p_i and the follower is the learning agent who plays the optimal policy $\pi^{\star}_{p_i}$ in the chosen configuration p_i. Hence, solving a NConf-MDP means finding the Stackelberg equilibrium $(p^\star, \pi^\star_{p^\star})$, composed by the optimal configuration $p^\star$, i.e. the configuration that maximizes the configurator's performance, and the optimal policy $\pi^\star_{p^\star}$, i.e. the policy that maximizes the agent's performance in configuration $p^\star$.

The second contribution of this thesis is the proposal of two algorithms for solving NConf-MDPs: *Action-feedback Optimistic Configuration Learning* (Af-OCL) and *Reward-feedback Optimistic Configuration Learning* (Rf-OCL). The two algorithms differ by the way they handle information coming from agent's trajectories. While Af-OCL ignores immediate reward samples, Rf-OCL leverages these samples for transferring information across different configurations. This feature makes Rf-OCL achieve higher performance than Af-OCL. On the other hand, Rf-OCL can be used only if we can observe reward samples while Af-OCL only needs trajectories composed by states and actions. In Section 6, we have compared our algorithms with UCB1 showing how the increasing use of the *structure* of the problem positively affects performance. In particular, UCB1 makes no use of the structure treating the configuration problem as a classic Multi-armed Bandit problem where the configurator can pull different arms, i.e. different configurations. On the other hand, Af-OCL leverages the fact that each "arm" is associated with an underlying MDP. Finally, Rf-OCL is able to estimate the reward function of the agent providing a link to transfer information gathered in a given configuration in all the others. The different uses of the structure of the problems affect the regret trend. In fact, regret generated by UCB1 grows *indefinitely* with logarithmic speed. On the other hand, both our algorithms are able to reach a zero-regret strategy. In particular, the regret upper bound for Rf-OCL is independent from the number of configurations and this makes this algorithm significantly more efficient than the others when the number of configurations is large.

The work presented in this thesis can be further extended in the future following different research directions.

Stochastic policy The algorithms proposed in this work are based on Assumption 1 stating that the agent's best response policy π_i is deterministic in all the configuration $i \in [M]$ and every time the configurator selects a configuration $i \in [M]$ the agent will play always the same optimal policy π_i. It is worth to notice that the performance achieved by the configurator is not totally fixed in each configuration due to the stochasticity of the transition model. Breaking Assumption 1 gives rise to two possible scenarios representing the future research directions of this work. The first scenario is characterized by *fixed* stochastic agent's policies. This means that when the configurator chooses a configuration $i \in [M]$, the agent will respond always with the same stochastic policy π_i^s. On the other hand, the second scenario arises when there is no guarantee that the agent will always play the same optimal policies (which could be deterministic or stochastic). As it can be imagined, this second scenario is the most difficult since, even in case of deterministic policies, the configurator could achieve totally different performances in the same configuration. Indeed, the agent could converge to different optimal policies that are equivalents in term of the agent's reward function but very different in term of configurator's reward function.

Awareness The work presented in this thesis is based on the assumption that the agent is *unaware* of the presence of a non-cooperative configurator. Based on this assumption, the best response of the agent to the chosen configuration is trivially playing the optimal policy. On the other hand, if the agent were *aware* of the game it could try to deceive the configurator playing sub-optimal policies to influence the configuration selection strategy to his advantage. The awareness of the agent makes the game more difficult to solve for the arising of deeper strategic behaviors. This scenario represents an interesting research directions that is worth to be explored in the future.

Multiple Learning Agents In this thesis, we have considered only situations with only one learning agent. However, in several real world applications we have to deal with multiple agents. For instance, in the e-commerce scenario, presented in Section 1, the website may be visited by several customers simultaneously. In order to deal with these situations, we have to consider multiple learning agents that act in the environment optimizing different functions. Clearly, this makes the role of the configurator much harder since it must take into account the behaviors of all the agents. Moreover, this scenario combined with the awareness of the agents may lead to a collaborative behavior of agents against the configurator.

Inverse Reinforcement Learning As already discussed, one of the main limitations of RfOCL is that it requires reward samples in order to estimate a confidence interval over the agent's reward function. However, in practical applications, this is a great limitation. Indeed, if we think to the e-commerce example provided in Section 1, we can only observe how the customer is behaving on the website without observing reward samples.

Nevertheless, this limitation can be easily overcome using Inverse Reinforcement Learning for the estimation of the agent's reward function. While the goal of Reinforcement Learning is to induce an optimal behavior given a reward function, the goal of Inverse Reinforcement Learning (IRL) is to infer the reward function that the agent is optimizing given its optimal behavior. For instance, in the e-commerce example, we could analyze the customer's behavior to identify the reward function that he is optimizing. However, it is worth to notice that IRL is an ill-posed problem, i.e., there exists several reward functions making the agent's behavior optimal. To solve this problem, IRL algorithms presented in the literature optimize some additional objectives, like the entropy or the suboptimality gap, to discriminate between equivalent reward functions. However, in order to eliminate the dependencies on reward samples in RfOCL we need not only a reward function but a confidence interval on the reward of each state. A viable research direction is represented by the combination of an IRL algorithm with *bootstrap sampling*. In this way, we could leverage different subsets of the agent's trajectories to derive several reward functions acting as a proxy of the reward samples.

Bibliography

[1] Simon P Anderson and Maxim Engers. Stackelberg versus cournot oligopoly equilibrium. *International Journal of Industrial Organization*, 10(1):127–135, 1992.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[3] Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 89–96, 2009.

[4] Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J Kappen. Minimax pac bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning*, 91(3):325–349, 2013.

[5] Peter L. Bartlett and Ambuj Tewari. Regal: A regularization based algorithm for reinforcement learning in weakly communicating mdps. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 35â42, Arlington, Virginia, USA, 2009. AUAI Press.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 41–48. ACM, 2009.

[7] Timothy F Bresnahan. Duopoly models with consistent conjectures. *The American Economic Review*, 71(5):934–945, 1981.

[8] Michele Breton, Abderrahmane Alj, and Alain Haurie. Sequential stackelberg equilibria in two-person games. *Journal of Optimization Theory and Applications*, 59(1):71–97, 1988.

[9] Giuseppe Burtini, Jason Loeppky, and Ramon Lawrence. A survey of online experiment design with the stochastic multi-armed bandit. *arXiv preprint arXiv:1510.00757*, 2015.

[10] Torpong Cheevaprawatdomrong, Irwin E Schochetman, Robert L Smith, and Alfredo Garcia. Solution and forecast horizons for infinite-horizon nonhomogeneous markov decision processes. *Mathematics of Operations Research*, 32(1):51–72, 2007.

[11] Kamil Andrzej Ciosek and Shimon Whiteson. OFFER: off-environment reinforcement learning. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1819–1825. AAAI Press, 2017.

[12] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. *A survey on policy search for robotics.* now publishers, 2013.

[13] Tanner Fiez, Benjamin Chasnov, and Lillian Ratliff. Implicit learning dynamics in stackelberg games: Equilibria characterization, convergence analysis, and empirical study. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3133–3144. PMLR, 13–18 Jul 2020.

[14] Jaime F Fisac, Eli Bronstein, Elis Stefansson, Dorsa Sadigh, S Shankar Sastry, and Anca D Dragan. Hierarchical game-theoretic planning for autonomous vehicles. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9590–9596. IEEE, 2019.

[15] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pages 482–495. PMLR, 2017.

[16] Víctor Gallego, Roi Naveiro, and David Ríos Insua. Reinforcement learning under threats. In *The Thirty-Third AAAI Conference on Ar-*

tificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 9939–9940. AAAI Press, 2019.

[17] Alfredo Garcia and Robert L Smith. Solving nonstationary infinite horizon dynamic optimization problems. *Journal of Mathematical Analysis and Applications*, 244(2):304–317, 2000.

[18] Aurélien Garivier and Olivier Cappé. The kl-ucb algorithm for bounded stochastic bandits and beyond. In *Proceedings of the 24th annual conference on learning theory*, pages 359–376, 2011.

[19] Archis Ghate and Robert L Smith. A linear programming approach to nonstationary infinite-horizon markov decision processes. *Operations Research*, 61(2):413–425, 2013.

[20] Wallace J Hopp, James C Bean, and Robert L Smith. A new optimality criterion for nonhomogeneous markov decision processes. *Operations Research*, 35(6):875–883, 1987.

[21] Ronald A Howard. Dynamic programming and markov processes. 1960.

[22] Garud N. Iyengar. Robust dynamic programming. *Math. Oper. Res.*, 30(2):257–280, 2005.

[23] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, ICML '02, page 267â274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[24] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[25] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms.* Cambridge University Press, 2020.

[26] Chang Liu, Jessica B Hamrick, Jaime F Fisac, Anca D Dragan, J Karl Hedrick, S Shankar Sastry, and Thomas L Griffiths. Goal inference improves objective and perceived performance in human-robot collaboration. *arXiv preprint arXiv:1802.01780*, 2018.

[27] Alberto Maria Metelli, Emanuele Ghelfi, and Marcello Restelli. Reinforcement learning in configurable continuous environments. In *International Conference on Machine Learning*, pages 4546–4555, 2019.

[28] Alberto Maria Metelli, Guglielmo Manneschi, and Marcello Restelli. Policy space identification in configurable environments. *arXiv preprint arXiv:1909.03984*, 2019.

[29] Alberto Maria Metelli, Mirco Mutti, and Marcello Restelli. Configurable Markov decision processes. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3491–3500, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[30] Stefanos Nikolaidis, Swaprava Nath, Ariel D Procaccia, and Siddhartha Srinivasa. Game-theoretic modeling of human adaptation in human-robot collaboration. In *Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction*, pages 323–331, 2017.

[31] Arnab Nilim and Laurent El Ghaoui. Robustness in markov decision problems with uncertain transition matrices. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems*, pages 839–846. MIT Press, 2003.

[32] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[33] Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page IIIâ307âIIIâ315. JMLR.org, 2013.

[34] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

[35] Giorgia Ramponi, Alberto Maria Metelli, Alessandro Concetti, and Marcello Restelli. Online learning in non-cooperative configurable markov decision process. 2021.

[36] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions.

In *Robotics: Science and Systems*, volume 2. Ann Arbor, MI, USA, 2016.

[37] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, ICML'15, page 1889â1897. JMLR.org, 2015.

[38] Aleksandrs Slivkins. *Introduction to Multi-Armed Bandits.* 2019.

[39] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[40] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.

[41] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[42] Andrea Tirinzoni, Riccardo Poiani, and Marcello Restelli. Sequential transfer in reinforcement learning with a generative model. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9481–9492. PMLR, 2020.

[43] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer, 2005.

[44] Andrea Zanette, Mykel J Kochenderfer, and Emma Brunskill. Almost horizon-free structure-aware best policy identification with a generative model. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5625–5634. Curran Associates, Inc., 2019.

[45] Haoqi Zhang, Yiling Chen, and David C. Parkes. A general approach to environment design with one agent. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 2002–2014, 2009.

Appendix A

Proofs and Derivations

A.1 Bellman Operators

In this section, we provide some additional details on the *expected Bellman operator* and *Bellman optimality operators* and we discuss the convergence properties of policy iteration (Algorithm 1) and value iteration (Algorithm 2). Both algorithm are characterized by an iterative application of Bellman operators. In particular, policy iteration algorithms apply until convergence the expected Bellman operator T^π associated with the current policy π in the policy evaluation step. Similarly, value iterations applies at each step the Bellman optimality operator $T^\star$. An iterative application of a given operator T provably converges to a unique value if the operator T is a contraction. Hence it is sufficient to prove that the Bellman operators are contractions in order to prove the convergence of the algorithms.

Definition A.1.1 (Contraction)**.** *Given an operator $T : \mathbb{R}^n \to \mathbb{R}^n$ and a real number $\gamma \in (0, 1)$, T is a contraction if*

$$\|T(\mathbf{V}) - T(\mathbf{V}')\|_\infty \leq \gamma \|\mathbf{V} - \mathbf{V}'\|_\infty$$

for every vectors $\mathbf{V}, \mathbf{V}' \in \mathbb{R}^n$.

Theorem A.1.1. *The expected Bellman operator T^π, associated with any policy π, is a contraction.*

Proof. Let's remind that $T^\pi(\mathbf{V}) = \mathbf{R}^\pi_s + \gamma \mathbf{P}^\pi_{ss'} \mathbf{V}$. Equivalently, we can rewrite the expected operator, using the element-wise notation:

$$T^\pi(\mathbf{V})(s) = r^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s) V(s').$$

Thus, we have:

$$\begin{aligned}
\|T^{\pi}(\mathbf{V}) - T^{\pi}(\mathbf{V}')\|_{\infty} &= \max_{s\in\mathcal{S}} \left[T^{\pi}(\mathbf{V})(s) - T^{\pi}(\mathbf{V}')(s)\right] \\
&= \max_{s\in\mathcal{S}} \left[r^{\pi}(s) + \gamma \sum_{s'\in\mathcal{S}} p^{\pi}(s'|s)V(s') - r^{\pi}(s) - \gamma \sum_{s'\in\mathcal{S}} p^{\pi}(s'|s)V'(s')\right] \\
&= \gamma \max_{s\in\mathcal{S}} \left[\sum_{s'\in\mathcal{S}} p^{\pi}(s'|s)V(s') - \sum_{s'\in\mathcal{S}} p^{\pi}(s'|s)V'(s')\right] \\
&= \gamma \max_{s\in\mathcal{S}} \sum_{s'\in\mathcal{S}} p^{\pi}(s'|s)(V(s') - V'(s')) \\
&\leq \gamma \max_{s\in\mathcal{S}} \sum_{s'\in\mathcal{S}} p^{\pi}(s'|s)\|\mathbf{V} - \mathbf{V}'\|_{\infty} \\
&= \gamma\|\mathbf{V} - \mathbf{V}'\|_{\infty} \max_{s\in\mathcal{S}} \sum_{s'\in\mathcal{S}} p^{\pi}(s'|s) \\
&= \gamma\|\mathbf{V} - \mathbf{V}'\|_{\infty}
\end{aligned}$$

□

Theorem A.1.2. *The Bellman optimality operator $T^{\star}$ is a contraction.*

Proof. Let's remind that $T^{\star}(\mathbf{V}) = \max_a \left[\mathbf{R}_{s,a} + \gamma \mathbf{P}_{s,a,s'}\mathbf{V}\right]$ or, equivalently using element-wise notation

$$T^{\star}(\mathbf{V})(s) = \max_{a\in\mathcal{A}} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')\right]$$

□

Thus, we have:

$$
\begin{aligned}
\|T^\star(\mathbf{V}) - T^\star(\mathbf{V}')\|_\infty &= \max_{s\in\mathcal{S}} \left[T^\pi(\mathbf{V})(s) - T^\pi(\mathbf{V}')(s)\right] \\
&= \max_{s\in\mathcal{S}} \left[T^\pi(\mathbf{V})(s) - T^\pi(\mathbf{V}')(s)\right] \\
&= \max_{s\in\mathcal{S}} \left[\max_{a\in\mathcal{A}} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V(s') \right] \right. \\
&\qquad \left. - \max_{a\in\mathcal{A}} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V'(s') \right] \right] \\
&\le \max_{s,a} \left| r(s,a) + \gamma \sum_{s'} p(s'|s,a) V(s') - r(s,a) - \gamma \sum_{s'} p(s'|s,a) V'(s') \right| \\
&= \gamma \max_{s,a} \left| \sum_{s'} p(s'|s,a) V(s') - \sum_{s'} p(s'|s,a) V'(s') \right| \\
&= \gamma \max_{s,a} \sum_{s'} p(s'|s,a) \left| V(s') - V'(s') \right| \\
&\le \gamma \max_{s,a} \sum_{s'} p(s'|s,a) \|\mathbf{V} - \mathbf{V}'\|_\infty \\
&= \gamma \|\mathbf{V} - \mathbf{V}'\|_\infty \max_{s,a} \sum_{s'} p(s'|s,a) \\
&= \gamma \|\mathbf{V} - \mathbf{V}'\|_\infty
\end{aligned}
$$

A.2 Optimistic Configuration Learning

A.2.1 Regret bound of Af-OCL

Lemma A.2.1. *The distance between the optimistic value function $\widetilde{V}_1$ and the real value function V_1 is bounded by:*

$$
\widetilde{V}_1 - V_1 \le 2H \sum_{s\in\mathcal{S}} \sum_{h=1}^{H} \mathbb{1}\left\{s \text{ is not yet visited}\right\} d_h(s) \tag{A.1}
$$

Proof.

$$
\begin{aligned}
\widetilde{V}_1 - V_1 &= \sum_{s\in\mathcal{S}} \left[\mu(s)r(s) - \mu(s)r(s) + \sum_{h=2}^{H}(\widetilde{d}_h(s) - d_h(s))r(s)\right] && \text{(A.2)}\\
&= \sum_{s\in\mathcal{S}}\sum_{h=2}^{H}(\widetilde{d}_h(s) - d_h(s))r(s) && \text{(A.3)}\\
&= \sum_{s\in\mathcal{S}}\sum_{h=1}^{H-1}\sum_{s'\in\mathcal{S}} \widetilde{d}_h(s')\widetilde{p}(s|s')r(s) - d_h(s')p(s|s')r(s) && \text{(A.4)}\\
&= \sum_{s\in\mathcal{S}}\sum_{h=1}^{H-1}\sum_{s'\in\mathcal{S}}(\widetilde{d}_h(s') - d_h(s'))\widetilde{p}(s|s')r(s)+ && \text{(A.5)}\\
&\qquad + d_h(s')(\widetilde{p}(s|s') - p(s|s'))r(s)\\
&\le \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}}\sum_{h=1}^{H-1}\left\|(\widetilde{d}_h(s') - d_h(s'))r(s)\right\|\left\|\widetilde{p}(s|s')\right\| + && \text{(A.6)}\\
&\qquad + \left\|d_h(s')(\widetilde{p}(s|s') - p(s|s'))\right\|\left\|r(s)\right\|\\
&\le \widetilde{V}_1^{H-1} - V_1^{H-1} + \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}}\sum_{h=1}^{H-1}\left\|d_h(s')(\widetilde{p}(s|s') - p(s|s'))\right\| && \text{(A.7)}\\
&\le H\sum_{s'\in\mathcal{S}} d_h(s')\sum_{h=1}^{H-1}\sum_{s\in\mathcal{S}}\left\|(\widetilde{p}(s|s') - p(s|s'))\right\| && \text{(A.8)}\\
&\le 2H\sum_{s'\in\mathcal{S}}\sum_{h=1}^{H}\mathbb{1}\left\{s' \text{ is not yet visited}\right\}d_h(s'), && \text{(A.9)}
\end{aligned}
$$

where line (A.2) is the definition of value function, in line (A.4) we rewrite the visiting distribution, in line (A.5) we expanded the probability distribution of visiting states, in line(A.6) we multiply for the mixed terms, line (A.7) is due to upperbounding the reward with 1, line (A.8) is due to triangle inequality and recursion, line (A.9) is due to upperbounding the probability with 2. □

Lemma A.2.2. *A model is no longer played if every state with $d_h(s) \ge \frac{\Delta_i - c}{2H^2S}$ is visit at least one time, with $c > 0$ and where Δ_i is the gap between the best model i^* and the model i.*

Proof.

$$\widetilde{V}_i = \widetilde{V}_i - V_i + V_i \tag{A.10}$$

$$\leq V_i + 2H \sum_{s \in \mathcal{S}} \sum_{h=1}^{H} \mathbb{1}\{\text{s is not yet visited}\}\, d_h^i(s) \tag{A.11}$$

$$\leq V_i + 2H^2 S \frac{\Delta_i - c}{2H^2 S} \tag{A.12}$$

$$= V_i + \Delta_i - c < V^* \tag{A.13}$$

in line (A.11) we apply lemma A.2.1 and in the final inequality we use the fact that $V^* - V_i = \Delta_i$. □

Theorem A.2.1 (Regret of AfOCL)**.** *Let $\mathcal{NCM} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r_c, r_o, H)$ with $\mathcal{P} = \{p_1, \dots, p_M\}$ be the M finite-horizon MDPs of the problem. The expected regret of AfOCL at every episode $K > 0$ is bounded by:*

$$\mathbb{E}[\mathit{Regret}(K)] \leq MH^3S^2. \tag{A.14}$$

Proof. We define the regret as:

$$\mathbb{E}[\text{Regret}(K)] = \sum_{i \in [M]:\Delta_i > 0} \Delta_i \, \mathbb{E}[N_i],$$

where N_i is the expected number of time that the algorithm plays model i different from optimal model i^*. We bound the expected value of N_i:

$$\mathbb{E}[N_i] \leq \sum_{l=0}^{\infty} \Pr(N_i \geq l) \tag{A.15}$$

$$\leq \sum_{l=0}^{\infty} \Pr(\widetilde{V}_l^i - \widetilde{V}_l^* \geq 0) \tag{A.16}$$

$$\leq 2 + \sum_{l=2}^{\infty} \Pr(\widetilde{V}_l^i - \widetilde{V}_l^* \geq 0) \tag{A.17}$$

$$\leq 2 + \sum_{l=2}^{\infty} \Pr(\widetilde{V}_l^i - V_l^* \geq 0) \tag{A.18}$$

In line (A.16) we use the fact that a model is selected only if its optimistic expected return is more than the optimistic expected return of optimal expected return, and in line (A.17) we use optimism. We observe that for lemma A.2.2, if every state $s \in \mathcal{S}$ with $d_h(s) \geq \frac{\Delta_i - c}{2H^2 S}$ is visited at least one time then the model i will no play again. So:

$$\begin{aligned}\mathbb{E}[N_i] &\leq 2 + \sum_{l=2}^{\infty} \Pr(\widetilde{V}_l^i - \widetilde{V}_l^* \geq 0) && \text{(A.19)}\\
&\leq 2 + \sum_{l=2}^{\infty} \Pr\Bigg(\exists s \in \mathcal{S}, \exists h \in [H] \text{ s.t. } d_h^i(s) \geq \frac{\Delta_i - c}{2H^2S} \text{ s.t.} && \text{(A.20)}\\
&\qquad s \text{ is not yet visited after } l \text{ pulls}\Bigg)\\
&\leq 2 + \sum_{l=2}^{\infty} \sum_{s\in\mathcal{S},h\in[H]:d_h^i(s)\geq\frac{\Delta_i-c}{2H^2S}} \Pr(s \text{ is not yet visited} && \text{(A.21)}\\
&\qquad \text{after } l \text{ pulls})\\
&\leq 2 + SH\sum_{l=2}^{\infty}\left(1-\frac{\Delta_i - c}{2H^2S}\right)^{l-1} && \text{(A.22)}\\
&= 2 + SH\frac{1-\frac{\Delta_i-c}{2H^2S}}{\frac{\Delta_i-c}{2H^2S}} \leq 2\frac{H^3S^2}{\Delta_i - c}, && \text{(A.23)}\end{aligned}$$

where line (A.21) is due to union bound, in line (A.22) we use the fact that the probability is a geometric series and in line (A.23) we observed that $SH \geq 2$. So the expected regret is bounded by:

$$\mathbb{E}[\text{Regret}(K)] = \sum_{i\in[M]:\Delta_i>0} \Delta_i\, \mathbb{E}[N_i] \leq \sum_{i\in[M]:\Delta_i>0} \Delta_i 2\frac{H^3S^2}{\Delta_i - c} \leq 2MH^3S^2,$$

having taken the infimum over $c > 0$. □

A.2.2 Regret bound of Rf-OCL

In this section, we are going to prove the regret for the algorithm RfOCL. We start defining the events G_k for $k \in [K]$ such that:

$$G_k = \left\{\exists s \in \mathcal{S} \text{ s.t. } |\overline{r}(s) - r(s)| \leq \sqrt{\frac{\log(SHk^3)}{2N_k(s)}}\right\}$$

This event means that at step k the estimated reward for all states $s \in \mathcal{S}$ are inside the confidence intervals.

Lemma A.2.3 (Simulation lemma for finite-horizon)**.** *For every transition probabilities $p_i \in \mathcal{P}$, the distance between the optimistic state-action value function $\overline{Q}_{o,k,1}^i(s,a)$ and the real optimal state-action value function*

$Q^i_{o,1}(s,a)$ at episode k, if all G_k hold, for all $s \in \mathcal{S}$ and for all $a \in \mathcal{A}$ is bounded by:

$$\overline{Q}^i_{o,k,1}(s,a) - Q^i_{o,1}(s,a) \leq \sqrt{\frac{\log(SHk^3)}{2N_k(s)}} + \sum_{s' \in \mathcal{S}} \sum_{h \in [H]} \overline{d}^i_{k,h}(s') \sqrt{\frac{\log(SHk^3)}{2N_k(s')}},$$

where $\overline{d}^i_{k,h}$ is the visitation distribution induced by the greedy policy $\overline{\pi}_i$ w.r.t. $\overline{Q}^i_{o,k}$.

Proof. The proof is basically taken from [44, 4, 42].

$$\overline{Q}^i_{o,k,1}(s,a) - Q^i_{o,1}(s,a) \leq \overline{Q}^i_{o,k,1}(s,a) - Q^{\overline{\pi}_i}_{o,1}(s,a) \tag{A.24}$$

$$= \overline{r}(s) - r(s) + \sum_{s' \in \mathcal{S}} \sum_{h \in [H]} \overline{d}^i_{k,h}(s') \left(\overline{r}(s) - r(s)\right) \tag{A.25}$$

$$= \sqrt{\frac{\log(SHk^3)}{2N_k(s)}} + \sum_{s' \in \mathcal{S}} \sum_{h \in [H]} \overline{d}^i_{k,h}(s') \sqrt{\frac{\log(2SHk^3)}{2N_k(s')}}, \tag{A.26}$$

where line (A.24) is due to $Q^i_{o,1}(s,a) \geq Q^{\overline{\pi}_i}_{o,1}(s,a)$, recalling that $Q^i_{o,1}$ is the optimal Q-value under model p_i. $\square$

Lemma A.2.4. *Let $s \in \mathcal{S}$ be a state with minimum visitation probability $p(s) := \min_{i \in [M]} \max_{h \in [H]} d^i_h(s) > 0$. Then, at episode $k \in [K]$ it holds that:*

$$\mathbb{E}[N_k(s)] \geq (k-1)p(s).$$

Proof. We simply apply the definition of $N_k(s)$:

$$\begin{aligned}
\mathbb{E}[N_k(s)] &= \mathbb{E}\left[\sum_{i=1}^{k-1} \sum_{h \in [H]} \mathbb{1}\left\{s_{k,h} = s\right\}\right] \\
&= \sum_{i=1}^{k-1} \sum_{h \in [H]} \Pr(s_{k,h} = s | p_{I_i}) \\
&= \sum_{i=1}^{k-1} \sum_{h \in [H]} d^{I_i}_h(s) \\
&\geq \sum_{i=1}^{k-1} \max_{h \in [H]} d^{I_i}_h(s) \\
&\geq (k-1) \min_{i \in [M]} \max_{h \in [H]} d^i_h(s) = (k-1)p(s).
\end{aligned}$$

$\square$

Lemma A.2.5. *Let $s \in \mathcal{S}$ be a state with minimum visitation probability $p(s) := \min_{i\in[M]} \max_{h\in[H]} d^i_h(s) > 0$. Then, after k episodes with probability at least $1-\delta_k$ it holds that:*

$$N_k(s) \geq kp(s) - \sqrt{\frac{k-1}{2}\log\left(\frac{1}{\delta_k}\right)}$$

Proof.

$$\Pr\left(N_k(s) \leq \mathbb{E}[N_k(s)] - \sqrt{\tfrac{k-1}{2}\log\left(\tfrac{1}{\delta_k}\right)}\right) \leq \Pr\left(N_k(s) \leq (k-1)p(s) - \sqrt{\tfrac{k-1}{2}\log\left(\tfrac{1}{\delta_k}\right)}\right) \leq \delta_k$$

Using Höeffding inequality and Lemma A.2.4. □

Lemma A.2.6. *If for all $s \in \mathcal{S}$ and for all $i \in [M]$, $\sqrt{\frac{\log(SHk^3)}{2N_k(s)}} \leq \frac{\Delta^Q - c}{2(SH+1)}$ then the policies of all the MDPs are well estimated.*

Proof. Let Δ^Q be the minimum gap between the Q-function in the optimal action and a different action in all transition probabilities $p_i \in \mathcal{P}$:

$$\Delta^Q = \min_{i\in[M]} \min_{s\in\mathcal{S}} \min_{h\in[H]} \left\{ \max_{a\in\mathcal{A}} Q^i_{o,h}(s,a) - \max_{a'\in\mathcal{A}\setminus\arg\max_{a\in\mathcal{A}} Q^i_{o,h}(s,a)} Q^i_{o,h}(s,a') \right\}.$$

If $\max_{s\in\mathcal{S}} \sqrt{\frac{\log(SHk^3)}{2N_s}} \leq \frac{\Delta^Q - c}{2(SH+1)}$, with $c > 0$, then for all $s \in \mathcal{S}$ and $i \in [M]$, we denote with $a^* = \arg\max_{a\in\mathcal{A}} Q^i_{o,h}(s,a)$ and we have for all $a \in \mathcal{A} \setminus \{a^*\}$:

$$\begin{aligned}
\overline{Q}^i_{o,k,h}(s,a) - \underline{Q}^i_{o,k,h}(s,a^*) &= \overline{Q}^i_{o,k,h}(s,a) + Q^i_{o,h}(s,a) - \\
&\quad Q^i_{o,h}(s,a) - \underline{Q}^i_{o,k,h}(s,a^*) + \\
&\quad Q^i_{o,h}(s,a^*) - Q^i_{o,h}(s,a^*) \\
&\leq 2(SH+1)\max_{s\in\mathcal{S}} \sqrt{\frac{\log(SHk^3)}{2N_k(s)}} - \Delta^Q && \text{(A.27)} \\
&\leq 2(SH+1)\frac{\Delta^Q - c}{2(SH+1)} - \Delta^Q \leq -c && \text{(A.28)}
\end{aligned}$$

Where line (A.28) is due to Lemma A.2.4. So the policies are well estimated. □

Theorem A.2.2 (Regret of RfOCL)**.** *Let $\mathcal{NCM} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mu, r_c, r_o, H)$ with $\mathcal{P} = \{p_1, \dots, p_M\}$ be the M finite-horizon MDPs of the problem. Under Assumption 2, the expected regret of RfOCL at every episode $K > 0$ is bounded by:*

$$\mathbb{E}[\mathit{Regret(K)}] \leq \overline{K}\Delta + \frac{\pi^2}{3},$$

where $\overline{K}$ is the smallest integer solution of the inequality

$$K \geq \left(\frac{2(SH+1)^2 \log(SHK^3)}{\Delta_Q^2} + \sqrt{\frac{K}{2}\log(S\Delta K^2)} \right) \frac{1}{\epsilon},$$

$\Delta = \max_{i \in [M]} \Delta_i$*, i.e. the maximum suboptimality gap, and* Δ_Q *is the minimum positive gap of the agent's Q-values.*

Proof. So we rewrite the expected regret as:

$$\begin{aligned} \mathbb{E}[\text{Regret}(K)] &= \sum_{k=1}^{K} (\mathbb{E}[\Delta_{I_k} \mathbb{1}\{G_k\}] + \mathbb{E}[\Delta_{I_k} \mathbb{1}\{\neg G_k\}]) \\ &\leq \underbrace{\sum_{k=1}^{K} \mathbb{E}[\Delta_{I_k} | G_k]}_{A} + \underbrace{H \sum_{k=1}^{K} P(\neg G_k)}_{B} \end{aligned}$$

We start bounding the B term:

$$H \sum_{k=1}^{K} P(\neg G_k) = H \sum_{k=1}^{K} P(\exists s \in S \text{ s.t. } |\overline{r}(s) - r(s)| > \sqrt{\frac{\log(SHk^3)}{2N_k(s)}}) \tag{A.29}$$

$$\leq H \sum_{k=1}^{K} \sum_{s \in \mathcal{S}} P(|\overline{r}(s) - r(s)| > \sqrt{\frac{\log(SHk^3)}{2N_k(s)}}) \tag{A.30}$$

$$\leq H \sum_{k=1}^{K} \sum_{s \in \mathcal{S}} \sum_{j=0}^{k-1} P(|\overline{r}(s) - r(s)| > \sqrt{\frac{\log(SHk^3)}{2j}}) \tag{A.31}$$

$$\leq H \sum_{k=1}^{K} \sum_{s \in \mathcal{S}} \sum_{j=0}^{k-1} e^{-\log(SHk^3)} \tag{A.32}$$

$$= \sum_{k=1}^{K} SHk \frac{1}{SHk^3} \leq \frac{\pi^2}{6} \tag{A.33}$$

where line (A.30) is due to the union bound on the states, line (A.31) is due to the union bound on the possible values of the random variable N_s and in line (A.32) we use Hoeffding inequality.

For the first term (A) we define the event E_k $\forall k \in [K]$:

$$E_k = \left\{ \forall s \in \mathcal{S} \, : \, N_k(s) \geq kp(s) - \sqrt{\frac{k}{2} \log\left(\frac{S}{\delta_k}\right)} \right\}.$$

So we can rewrite the term (A) as:

$$\sum_{k=1}^{K} \mathbb{E}[\Delta_{I_k}|G_k] \leq \underbrace{\sum_{k=1}^{K} \mathbb{E}[\Delta_{I_k}|G_k, E_k]}_{C} + \underbrace{\Delta \sum_{k=1}^{K} P(\neg E_k)}_{D}$$

We start bounding the second term (D):

$$\Delta \sum_{k=1}^{K} P(\neg E_k) = \Delta \sum_{k=1}^{K} \delta_k$$

We set $\delta_k = \frac{1}{k^2\Delta}$. So:

$$\Delta \sum_{k=1}^{K} P(\neg E_k) = \Delta \sum_{k=1}^{K} \frac{1}{k^2\Delta} \leq \frac{\pi^2}{6}$$

Now it remains to bound the term (C) that, using lemma A.2.6, is zero:

$$\forall K > \overline{K} \geq \left(\frac{2(SH+1)^2 \log(SH\overline{K}^3)}{(\Delta_Q)^2} + \sqrt{\frac{\overline{K}}{2} \log(S\Delta\overline{K^2})} \right) \frac{1}{\epsilon}. \tag{A.34}$$

Then the total regret is:

$$\mathbb{E}[\text{Regret}(K)] = \min \left\{ 2MH^3S^2, \overline{K}\Delta + \frac{\pi^2}{3} \right\}.$$

□

Appendix B

Additional Experimental details

In this appendix, we report the hyperparameters employed in the experimental results, presented in Section 6.2.

B.1 Configurable Gridworld Experiments

In this section we provide the hyperparameters used for conducting the experiments on Configurable Gridword domain.

B.1.1 Experiment 1

The first experiment (Figure 6.4) presents the following hyperparameters:

- Number of states: 9
- Number of actions: 4
- Probability of action failure: 0.1
- Number of configurations: 10, 30
- Time horizon: 10
- Number of episodes: 3000
- Number of independent runs: 50

B.1.2 Experiment 2

The second experiment (Figure 6.5) presents the following hyperparameters:

- Number of states: 9
- Number of actions: 4
- Probability of action failure: 0
- Number of configurations: 3
- Time horizon: 10
- Number of episodes: 1000
- Number of independent runs: 50

B.1.3 Experiment 3

The third experiment (Figure 6.6) presents the following hyperparameters:

- Number of states: 9
- Number of actions: 4
- Probability of action failure: 0.1
- Number of configurations: 10, 30, 100, 200
- Time horizon: 10
- Number of episodes: 5000
- Number of independent runs: 50

B.2 Student-Teacher Experiment

In this section we provide the hyperparamenters of the experiment in the Student-Teacher domain:

- Number of states: 10
- Number of actions: 2
- Probability of action failure: 0.1
- Number of configurations: 40, 60, 100

- Time horizon: 10
- Number of episodes: 4000
- Number of independent runs: 50

B.3 Configurable Marketplace Experiment

In this section we provide the hyperparamenters of the experiment in the Configurable Marketplace domain:

- Number of states: 16
- Number of actions: 4
- All products: $\{0, 1, 2\}$
- Products needed by the agent: $\{1\}$
- Probability of action failure: 0.1
- Number of configurations: 10
- Time horizon: 15
- Number of episodes: 30000
- Number of independent runs: 50