POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information, Bio-engineering
Master of Science Degree in Computer Science and Engineering

REWARD-FREE POLICY SPACE COMPRESSION FOR LEARNING IN MARKOV DECISION PROCESSES

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

Supervisor: Prof. Marcello Restelli
Co-supervisor: Dott. Mirco Mutti

Author:
Stefano Del Col, 920559

Academic Year 2020-2021

E la coscienza farà appunto esperienza di ciò che lo Spirito è:
Io che è Noi, e Noi che è Io.

G.W.F. Hegel, Fenomenologia dello Spirito

Ringraziamenti

I miei primi ringraziamenti vanno al professor Marcello Restelli e al dottorando Mirco Mutti: con la loro guida sono stati parte fondamentale di questo lavoro di ricerca, che altrimenti non avrebbe visto la luce. Ringrazio il Politecnico, per avermi dato la possibilità di esprimere la mia passione per l'informatica e per avermi offerto un ambiente universitario ricco di opportunità e persone in gamba. Un grazie speciale a tutti i miei amici e ai compagni di corso: avete reso il mio periodo a Milano felice e sereno. Infine ringrazio la mia famiglia per avermi sostenuto in tutti questi anni, permettendomi di vivere appieno l'università senza troppi pensieri.

A tutti, un grazie di cuore!

Stefano
Milano, 28 aprile 2021

Contents

List of Figures

List of Algorithms

Abstract

A *Markov decision process* is a widespread mathematical framework to formalize *sequential decision-making problems.* In this context, an artificial learning agent interacts with a potentially unknown environment, aiming to reach a previously defined long-term goal. Usually, the performance of the agent is evaluated through a reward function: a numerical feedback that indicates to the agent how well it is performing with respect to its task. Solving a Markov decision process means finding an interaction strategy that maximizes the cumulative sum of rewards, commonly called *optimal policy.* To find such a policy, a suitable way is to explore the set of all the strategies, looking for an optimal one. Unfortunately, because of the enormous size of the policy space and its complex relation with the environment dynamics, performing this search is an arduous challenge.

The goal of this thesis is to devise a method to substantially ease the policy search problem, by performing a *compression* of the policy space into a *finite* set of *representative* elements. These elements are representative in the sense that, via off-policy estimation techniques, they allow to approximately evaluate the performance of any policy. In this document we propose and analyze a viable solution procedure to such problem, and we consequently evaluate the compression quality with some practical examples. Moreover, we provide theoretical guarantees on the obtained results and we suggest interesting directions for future works.

Estratto in lingua italiana

L'*intelligenza artificiale* è il settore dell'informatica che studia come costruire sistemi informatici *intelligenti*, che sono cioè in grado di risolvere problemi e compiti complessi che normalmente richiederebbero l'intervento umano. In questa prospettiva, l'*apprendimento automatico* si concentra su come costruire programmi che *imparino attraverso l'esperienza.* Più specificatamente, l'*apprendimento per rinforzo* sviluppa agenti in grado di risolvere *problemi decisionali sequenziali.* Questi agenti artificiali possono raggiungere efficacemente un obiettivo a lungo termine definito dall'utente, attraverso l'interazione con l'ambiente in cui si trovano. Grazie alla sua grande flessibilità ed espandibilità, il *processo decisionale di Markov* è una struttura matematica molto usata per formalizzare tale interazione. In questo contesto, l'ambiente nel quale si trova l'agente può essere descritto attraverso un insieme di stati, un insieme di azioni e un modello di transizione che definisce come il comportamento dell'agente influenzi l'ambiente. Per quanto riguarda l'agente, il suo obiettivo a lungo termine è di solito codificato attraverso una funzione di ricompensa che, ad ogni interazione, valuta numericamente come il decisore stia operando rispetto al suo compito. L'obiettivo dell'agente è quello di trovare una *politica ottimale*, cioè una strategia di azione che sia in grado di massimizzare la somma delle ricompense ottenute.

Purtroppo, la ricerca di tale politica non è affatto facile. Prima di tutto lo spazio di ricerca è enorme. Il numero di politiche deterministiche, cioè che non possono rendere casuale la scelta dell'azione, è *esponenziale* rispetto al numero di stati dell'ambiente, numero che può facilmente superare i milioni in scenari reali. Inoltre, se non limitiamo la nostra attenzione a quelle deterministiche, la dimensione dello spazio delle politiche è *infinita.* In aggiunta, la relazione tra una politica e la sua capacità di raccogliere ricompense è strettamente dipendente dal modello di transizione dell'ambiente e introduce così un ulteriore livello di complessità. Nonostante queste difficoltà, se il segnale di ricompensa è sufficientemente *frequente* e *informativo*, la letteratura fornisce molti algoritmi che sono in grado di trovare una politica ottimale. Sfortunatamente le ricompense così ben definite sono una rarità e, quando esse non sono disponibili, gli algoritmi standard non riescono a convergere in modo efficiente su nessuna soluzione significativa.

In risposta a questo problema, la branca appena nata dell'*apprendimento per rinforzo senza ricompensa* ha esplorato diverse possibilità per ottenere il meglio dall'agente anche in una situazione così difficile. L'idea è di preparare l'agente alla ricerca efficace delle ricompense, *qualsiasi sia il compito definito dall'utente.* A tal fine possiamo trovare in letteratura due tipi principali di approccio. La prima categoria comprende i metodi che affidano all'agente un compito esplorativo, con l'obiettivo di motivarlo a esplorare l'ambiente prima che il compito definito dall'utente gli sia reso noto. Il secondo approccio consiste invece nel raccogliere una quantità sufficiente di dati dall'ambiente, in modo da rendere l'agente capace di affrontare qualsiasi compito utilizzando soltanto i dati raccolti.

Obiettivo e Motivazione della Tesi

Lo scopo di questa tesi è la definizione di un nuovo approccio all'apprendimento per rinforzo senza ricompensa nel quale, al fine di facilitare la successiva ottimizzazione dell'agente, si effettua una *compressione dello spazio delle politiche.* L'idea che motiva lo sviluppo di questa tesi è duplice. In primo luogo, sfruttando la struttura del problema, è possibile stimare il valore di una politica (cioè, quanta ricompensa essa è in grado di raccogliere) utilizzando i dati provenienti da un'altra politica. In secondo luogo, è possibile quantificare quanto sarà accurata e affidabile questa stima sfruttando una quantità propria della teoria dell'informazione, la divergenza di Rényi. Con tali strumenti è possibile costruire un insieme di elementi *rappresentativi* dello spazio delle politiche, dove per rappresentativo si intende che, utilizzando tali elementi, è possibile stimare in modo affidabile il valore di qualsiasi politica. Trovare una rappresentazione dello spazio delle politiche così compressa non è certamente un obiettivo facile, ma sicuramente vale la pena di intraprendere una ricerca al riguardo.

Contributi Principali

Il primo contributo di questa tesi è la definizione del problema di compressione dello spazio delle politiche. Per quanto ne sappiamo, una prospettiva così interessante non è mai stata considerata in letteratura e noi intendiamo colmare la lacuna con questa tesi. Forniamo due formalizzazioni equivalenti della questione: la prima come problema di copertura di un insieme, la seconda come gioco non convesso a due giocatori. Inoltre presentiamo una discussione della loro rispettiva complessità computazionale e trattabilità.

Il secondo contributo è una procedura basata sulla discesa del gradiente per risolvere approssimativamente la formulazione del gioco a due giocatori. L'algoritmo sfrutta la nozione di equilibrio differenziale di Stackelberg e la procedura GDA per trovare soluzioni localmente ottimali al gioco, corrispondenti a politiche di copertura localmente ottime.

Il terzo contributo è una garanzia teorica sui risultati ottenuti dall'algoritmo, sotto forma di un limite superiore alla copertura effettiva, che non richieda di trovare una soluzione globale del problema.

Il quarto e ultimo contributo è una validazione numerica della soluzione proposta attraverso alcuni esempi pratici.

Chapter 1

Introduction

Artificial intelligence is the field of computer science that studies how to build *intelligent* computer systems, meaning that they are able to solve complex problems and tasks that would normally require a human to accomplish. In this perspective, *machine learning* focuses on how to build software that *learns through experience*. More specifically, *reinforcement learning* [1] addresses intelligent *sequential decision-making* agents. These artificial agents can effectively reach a user-defined long-term goal by interacting with the environment they find themselves in. Thanks to its great flexibility and expandability, the *Markov Decision Process* [3] is a convenient mathematical framework to formalize such interaction. In this context, the environment can be described through a collection of *states*, a set of *actions*, and a *transition model* that defines how the behavior of the agent influences the environment. On the agent's side, its long-term goal is usually encoded through a *reward function*, that, at each interaction, numerically assesses how well the decision-maker is performing with respect to its task. The goal of the agent is to find an *optimal policy*, that is, an action strategy that is able to maximize the cumulative sum of the rewards.

Unfortunately, searching for an optimal policy is not an easy task. First of all the search space is enormous. The number of deterministic policies, meaning that they are not allowed to randomize the action choice, is *exponential* in the number of states of the environment, number that can easily reach *millions* in real-case scenarios. Moreover, if we do not restrict our attention to the deterministic ones, the size of the policy space becomes *infinite*. Additionally, the relation between a policy and its ability to collect rewards is strictly dependent on the transition model of the environment and introduces a further layer of complexity. Nonetheless, if the reward signal is sufficiently *frequent* and *informative*, the literature provides many algorithms that can find an optimal policy. Regrettably, well-defined rewards are a rarity in real-world scenarios and, when they are not available, the standard algorithms fail to efficiently converge to any meaningful solution [4].

In response to this problem, the newly born field of *reward-free reinforcement learning* [5] has been exploring different possibilities to get the best out of the agent even in such a difficult situation. The idea is to prepare the agent for effective reward seeking, whatever the user-defined task. To this end we can currently find in the literature two main approaches: *pre-training for online learning* and *reward-free sampling*. The first category comprises the methods that address an *exploratory task* to optimize the agent's policy [6, 7, 8, 9, 10, 11, 12], with the objective of motivating it to explore the environment before the user-defined task is revealed. On the other hand, the second approach is to collect a sufficient amount of data from the environment, so that the agent is able to tackle any task using only the collected data [13, 14, 15].

1.1 Thesis Goal and Motivation

The aim of this thesis is the definition of a novel approach for reward-free reinforcement learning in which, in order to facilitate subsequent agent optimization, a *compression* of the *policy space* is performed. The idea motivating the development of this thesis is twofold. First, by exploiting the structure of the problem, it is possible to estimate the value of a policy (i.e., how much reward it is able to collect) by using the data collected by another policy [16, 17]. Secondly, it is possible to quantify how accurate and reliable this estimation will be, by exploiting an information-theoretic quantity, the *Rényi divergence* [18]. With such tools it is possible to construct a set of *representative* elements of the policy space, where by representative we mean that, by using the elements in the set, it is possible to reliably estimate the value of any other policy. Coming up with such a condensed representation is certainly not an easy goal, but surely worth undertaking.

1.2 Main Contributions

The first contribution of this thesis is the definition of the *policy space compression problem*. To the best of our knowledge, such an interesting perspective has never been considered before in the literature and we plan to fill the gap with this thesis. Especially, we provide two equivalent formalizations of the problem: the first as a *set-covering*, the second as a *non-convex two-player game*. Moreover, we present a discussion of their respective computational complexity and tractability.

The second contribution is a *gradient-based* procedure (Algorithm 3) to approximately solve the two-player game formulation. The algorithm exploits the notion of *differential Stackelberg Equilibrium* [19] and the ∞-*GDA procedure* [20] to find locally optimal solutions to the game, corresponding to locally optimal covering policies.

The third contribution is a *theoretical guarantee* on the results obtained by the algorithm, in the form of an upper bound to the actual covering that does not require a global solution to the problem.

The fourth, and last, contribution is a *numerical validation* of the algorithm through some practical examples.

1.3 Thesis Structure

The thesis is structured as follows. We start in Chapter 2, where we present all the fundamental concepts and algorithms that we will use in the remaining of the writing. In Chapter 3, we formally present the policy space compression as a set-covering problem, and analyze some of its peculiar properties. In Chapter 4 we reformulate the problem into a more practical, non-convex two-player game, and we propose a gradient-based method to locally find a solution. In Chapters 5, 6 and 7 respectively, we analyze in-depth each of the algorithm's components: the optimization of the follower, the optimization of the leader, and the coverage guarantees on the results. In Chapter 8, we validate the solution proposed with some illustrative experiments and, lastly, in Chapter 9 we summarize our findings and propose different directions for future works.

Chapter 2

Background and State of the Art

In this chapter, we will present all the fundamental topics and concepts that are needed for the assimilation of the following discussion, together with some significant results of the current state-of-the-art. In Section 2.1, we will present the main mathematical framework used nowadays to formalize sequential decision-making problems and outline some of its basic properties. In Section 2.2, we will switch context and outline some statistical methodologies to estimate the properties of a distribution, namely Monte Carlo and importance sampling techniques. In Section 2.3, we will put everything together and present the reinforcement learning field. Lastly, in Section 2.4 we will present some basic concepts of game theory.

2.1 Markov Decision Processes

Many engineering and practical problems can be formulated as a *sequential decision-making problem*. In this type of problem, a decision-maker sequentially interacts with an environment with the objective of maximizing its long-term performance with respect to a pre-specified goal. The *agent-environment interface* provides a nice visualization for this interaction process.

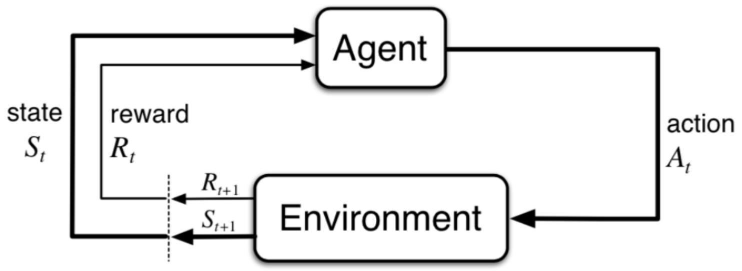


Figure 2.1: The agent-environment interface (from [1]).

At each time step, the agent executes an action on the environment with the intention of reaching its specified goal. As a consequence, it receives a reward that indicates how appropriate the action was, and observes to which state the environment has transitioned to. This setting seems very simplistic, but in practice it has proven very effective as a modeling tool for problem-solving. Nowadays, the *Markov Decision Process* (MDP) [3] is the most common framework used to formalize such types of problems.

2.1.1 Mathematical Formulation

Formally we can define the Markov Decision Process as a tuple of seven elements $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, d_0, \gamma)$:

- *State space* $\mathcal{S}$, a non-empty and possibly infinite set of states.
- *Action space* $\mathcal{A}$, a non-empty and possibly infinite set of actions.
- *Transition model* $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$, a function that, for each state-action pair, specifies the probability distribution over the state space at the next step. We denote as $P(s'|s, a)$ the probability of reaching state s' after performing action a in state s.
- *Reward function* $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, a function that, for each state-action pair, specifies the scalar reward obtained by performing that action in that state. Without loss of generality, we can assume $R(s, a) \in [0, 1]$.
- *Initial state distribution* $d_0 : \Delta(\mathcal{S})$, is a probability distribution over the initial state of the process.
- *Discount factor* $\gamma \in [0, 1)$, the probability that the process will <u>not</u> terminate at the next time-step. It is used to model the inherent risk linked to the uncertainty of the future.

Additionally, we define the state-action space as the Cartesian product between the state space and the action space.

The Markov Property

We say that this process is *Markovian* [3] because both the transition model and the reward function satisfy the Markov property, which can be informally stated as *the future is independent of the past given the present.* This means that all the information that the environment needs to determine the immediate reward and the next state is all contained in the present state of the process and the current action of the agent.

The Stationary Property

Analogously, we say that the process is *stationary* because the transition model and reward function do not change over time.

2.1.2 Policies and Induced State-Action Distributions

A *policy* is a function that defines the behavior of the agent in the environment

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

For each state, the policy defines a conditional probability distribution over the actions to perform in that state. Especially, we denote with $\pi(a|s)$ the probability of choosing action a in state s. If such probability distribution collapses on only one action for each state, the policy is said to be *deterministic*, otherwise it is referred to as *stochastic.*

We can combine the policy π and the transition model to understand how the agent actually navigates the environment, by taking into account both the individual decisions of the agent and the underlying structure of the problem. This combination induces a probability distribution over the state space $\mathcal{S}$, referred in the literature as *state distribution* [3] and denoted with the symbol d_π^s. In the γ-discounted setting, the state distribution is defined as

$$d_\pi^s(s) = (1-\gamma) d_0^s(s) + \gamma \int_{\mathcal{S}\mathcal{A}} d_\pi^s(s')\pi(a'|s')P(s|s',a')\,\mathrm{d}s'\,\mathrm{d}a', \quad \forall s \in \mathcal{S}. \tag{2.1}$$

Starting from the state distribution and the policy, it is possible to define a probability distribution on all the possible state-action pairs, called *state-action distribution* and denoted with the symbol d_π^{sa}

$$d_\pi^{sa}(s,a) = \pi(a|s)\,d_\pi^s(s), \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}.$$

If we are given a state-action distribution that is induced by some unknown policy, it is possible to recover the originating policy in a simple way, namely

$$\pi(s,a) = \frac{d_\pi^{sa}(s,a)}{\sum_{a' \in \mathcal{A}} d_\pi^{sa}(s,a')}. \tag{2.2}$$

Note that this relation *holds only for distributions that are induced by some policy*! If one is given a distribution that does not have a policy counterpart and (wrongly) applies this relation, they will nonetheless obtain a policy, but that policy will not induce the same distribution they had at the beginning. Fortunately, we can enforce a state-action distribution to be induced by some policy with the following constraint

$$\begin{aligned}\int_{\mathcal{A}} d^{sa}(s,a)\,\mathrm{d}a = (1-\gamma)d_0^s(s)& \\ + \gamma \int_{\mathcal{SA}} d^{sa}(s',a')P(s|s',a')\,\mathrm{d}s'\,\mathrm{d}a',\ \forall s \in \mathcal{S}.&\end{aligned} \tag{2.3}$$

2.1.3 Reward and Performance

Having defined how the agent *behaves* in the environment, let us define how to *evaluate* such behavior in mathematical terms. Intuitively, the objective of the agent is to maximize the long-term cumulative reward, also by keeping into consideration the uncertainty of the future with the discount factor. Several performance metrics have been developed in the literature, but the most common one is the *expected γ-discounted cumulative reward*[3] criterion. According to the former, we can define the *performance* (denoted as $J(\pi)$) in terms of expectation of the discounted summation of rewards collected by following a policy π, i.e.,

$$J(\pi) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\pi, d_0}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right].$$

Note that the expectation is needed because even with a deterministic policy, the stochasticity of the transition model and/or the reward function can make the rewards possibly different at each execution of the policy. Alternatively, we can define the performance in terms of the state distribution as

$$J(\pi) = \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\pi^s(s) \int_{\mathcal{A}} \pi(a|s)\,R(s,a)\,\mathrm{d}a\,\mathrm{d}s,$$

or in terms of the state-action distribution as

$$J(\pi) = \frac{1}{1-\gamma} \int_{\mathcal{SA}} d_\pi^{sa}(s,a)\,R(s,a)\,\mathrm{d}a\,\mathrm{d}s.$$

Solving an MDP means to find a particular policy, called *optimal policy* and denoted as π^*, that is able to achieve a performance greater or equal than any other policy in the policy space Π (i.e., the set of all the possible policies). Put in mathematical terms, we want to find a policy π^* such that

$$J(\pi^*) \geq J(\pi), \forall \pi \in \Pi.$$

2.1.4 MDP Exact Solution Methods

In this section, we will see some methods [1] that can be used to find such an optimal policy. These methods have the advantage of always finding the exact solution, but they require a lot of computational effort and, more importantly, the knowledge of the transition model and the reward function, which may not be available in practice. As we will see in the following sections, it is possible to relax this assumption and find good solutions even in a *unknown* environment.

Linear Programming

The first exact solution method we present exploits a *linear programming formulation* [21] of the optimal policy problem, which immediately derives from the above definition of the performance. Let $\boldsymbol{d} = \big(d(s,a)\big)_{s\in\mathcal{S},a\in\mathcal{A}}$ be the vector that represents the γ-discounted state-action distribution induced by a policy

$$
\begin{aligned}
\underset{\boldsymbol{d}}{\text{maximize:}} \quad & \int_{\mathcal{SA}} d(s,a)R(s,a) \\
\text{subject to:} \quad & d(s,a) \geq 0, \qquad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} && (1) \\
& \int_{\mathcal{SA}} d(s,a) = 1 && (2) \\
& \int_{\mathcal{A}} d(s,a)\,\mathrm{d}a = (1-\gamma)d_0^s(s) \\
& \qquad + \gamma \int_{\mathcal{SA}} d(s',a')P(s|s',a')\,\mathrm{d}s'\,\mathrm{d}a', \ \forall s \in \mathcal{S} && (3)
\end{aligned}
$$

The solution $\boldsymbol{d}^*$ of the above problem is the state-action distribution that maximizes the discounted cumulative sum of the rewards (i.e., the performance) on the MDP. Since we enforced such distribution to be induced by some policy through the third constraint, we can directly extract an optimal policy π^* from it with Equation (2.2). The linear program formulation presents $|\mathcal{S}| \times |\mathcal{A}|$ variables and $|\mathcal{S}| + |\mathcal{S}| \times |\mathcal{A}|$ constraints, and therefore its complexity strongly depends on the dimension of the MDP.

Dynamic Programming

The second possibility we have for exactly computing the solution of an MDP is to exploit a *dynamic programming* [1] formulation. To this end, we first have to define two additional quantities, the *state value function V* and the *state-action value function Q*.

The state value function of the policy π in state s is the *expected discounted cumulative reward* we get starting from state s and following policy π, i.e.,

$$V_\pi(s) = \mathop{\mathbb{E}}_{\pi,d_0}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,\middle|\, s_t = s\right], \quad \forall s \in \mathcal{S}. \tag{2.4}$$

The state-action value function of policy π in state s, action a, is the expected discounted cumulative reward we get starting from state s, performing action a and then following policy π, i.e.,

$$Q_\pi(s,a) = \mathop{\mathbb{E}}_{\pi,d_0}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}.$$

The two quantities are linked by the relation

$$V_\pi(s) = \int_{\mathcal{A}} \pi(a|s)\, Q_\pi(s,a), \quad \forall s \in \mathcal{S}.$$

Note that we can now define the performance J with these quantities as

$$J(\pi) = \int_{\mathcal{S}} d_0^s(s) V_\pi(s)\,\mathrm{d}s.$$

The *optimal* state value and state-action value functions (denoted respectively $V^*(s)$ and $Q^*(s,a)$), are the value functions associated with an optimal policy π^*, namely

$$\begin{aligned} V^{\pi^*}(s) &= V^*(s) = \max_\pi V_\pi(s), && \forall s \in \mathcal{S} \\ Q^{\pi^*}(s,a) &= Q^*(s,a) = \max_\pi Q_\pi(s,a), && \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \end{aligned}$$

Value Iteration

Expanding the definition in Equation (2.4), we can rewrite the state value function in a recursive formulation, named *Bellman expectation equation* [1]

$$\begin{aligned} V_\pi(s) &= \mathop{\mathbb{E}}_{\pi,d_0}\left[r_{t+1} + \gamma V_\pi(s_{t+1}) \,\middle|\, s_t = s\right] \\ &= \int_{\mathcal{A}} \pi(a|s)\Big(R(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) V_\pi(s')\,\mathrm{d}s'\Big), \quad \forall s \in \mathcal{S}. \end{aligned} \tag{2.5}$$

Note that the above formulation is linear in the variables $V_\pi(s)$, therefore it can be easily solved in closed form with a matrix inversion. We can combine the definition of optimal state value function and the Bellman equation to derive the *Bellman optimality equation*, that recursively defines the *optimal* state value function

$$V^*(s) = \max_{a\in\mathcal{A}}\left\{R(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) V^*(s')\,\mathrm{d}s'\right\}, \quad \forall s \in \mathcal{S} \tag{2.6}$$

Note that, by knowing the transition model and the optimal state value function, it is possible to compute the optimal state-action value function, and from it the optimal policy. Unfortunately, because of the maximization over the actions, the system of equations in (2.6) is not linear and therefore can't be solved in closed form.

The idea of value iteration is to *approximately* solve the Bellman optimality equation in (2.6), and then use it to extract the optimal policy. Starting from an arbitrary state value function, it is possible to approach V^* by iteratively applying Equation (2.6) turned into an operator, i.e., *the Bellman optimality operator*. This procedure converges at the limit to the true V^*, and the magnitude of the update is proportional to the distance from the unique fixed point V^* of such operator [3]. In practice, this procedure is applied until the resulting variation of the state value function is smaller than a threshold. From this approximate (but neighboring) solution, we will extract a policy very close, or equal to, an optimal one. Unfortunately, we have to be aware that since the optimization is performed in the state value space, intermediate value functions may not correspond to any policy.

Policy Iteration

The idea of policy iteration is to decompose the value iteration update in two separate procedures: *policy evaluation* and *policy improvement*.

The *policy evaluation* step consists of computing the state value function of a given policy. We can perform this calculation in closed form by solving the system of $|\mathcal{S}|$ linear equations in (2.5).

The *policy improvement* step is a procedure that, given a policy π, produces a policy π' improving its state value function, i.e.,

$$V_{\pi'}(s) \geq V_{\pi}(s), \ \forall s \in \mathcal{S}.$$

From the *policy improvement theorem* [1], we know that such procedure can be implemented simply by acting greedily on the policy π

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_{\pi}(s, a), \ \forall s \in \mathcal{S}.$$

Starting from a random policy π_0, policy iteration iteratively applies the evaluation and improvement steps, until the latter does not change the policy anymore (i.e., $\pi' = \pi$). From Equation (2.5), the only policy for which the policy improvement step yields no changes is the optimal one. If the state-action space is finite, policy iteration is guaranteed to converge to the optimal policy in a finite number of steps [3].

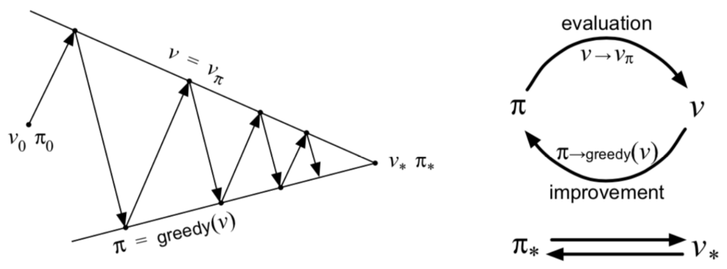


Figure 2.2: Policy iteration structure (from [1]).

2.2 Sample-Based Estimation

The methods presented in the previous section are all capable of finding the exact solution (i.e., the optimal policy) for any given MDP. Nonetheless, they are rarely used in practice. Why is it so?

Setting aside for a moment the computational complexity, the main limitation of linear/dynamic programming techniques is that *they require the knowledge of the MDP*, meaning that the transition model and reward function must be available to the agent during the optimization. In real-world situations, these two quantities are usually unknown to the agent, which therefore has to try to find a solution in a different way.

To this end, we now present a seemingly unrelated topic: how to estimate the properties of a statistical distribution by repeated random sampling, namely *Monte Carlo techniques.*

2.2.1 Monte Carlo Estimation

Monte Carlo techniques are a class of computational methods used to estimate the parameters (usually mean and/or variance) of a distribution. There are many problem settings in probability theory for which computing an analytical solution is challenging, either because finding the exact solution may be too computationally demanding, or because the underlying dynamics of the process are not known. A possible, empirical solution to this problem is to *randomly draw samples* from the target distribution and use them to *estimate* the desired quantities.

Suppose, for example, that we want to compute the expected value of a real-valued function $f(x)$ defined on the domain $\mathcal{D}$, where $p(x)$ is a probability density function on $\mathcal{D}$

$$\mu_p = \mathop{\mathbb{E}}_{x \sim p} \big[f(x)\big] = \int_{\mathcal{D}} f(x)p(x)\,\mathrm{d}x.$$

If finding the exact expected value by analytically computing the above integral is not a viable option, we can resort to Monte Carlo techniques. By using the probability function $p(x)$, we sample points on the domain $\mathcal{D}$. For each sample, we compute the corresponding value of $f(x)$ and then average the results. What we obtain is a *sample-based estimate* of the expected value μ_p denoted as $\widehat{\mu_p}$, i.e.,

$$\mu_p = \int_{\mathcal{D}} f(x)p(x)\,\mathrm{d}x \quad \Longrightarrow \quad \widehat{\mu_p} = \frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

From the *law of large numbers*, as the number of samples n grows to infinity, the empirical mean approaches the theoretical one.

2.2.2 Importance Sampling Estimator

What happens to the Monte Carlo estimation if $f(x)$ is zero everywhere, except in a region of $\mathcal{D}$ where the probability $p(x)$ is very small? Clearly, since the probability of drawing a sample with a non-zero value is minimal, it is very likely that the empirical mean will be zero! A possible solution to mitigate this problem is to intentionally take more samples from the critical region of $\mathcal{D}$, but, since we altered the sampling procedure, the new estimator must be adjusted accordingly. *Importance sampling* [22] is a statistical technique that does exactly so: it is used to estimate the parameters of a distribution, called *target*, by using samples taken by a different distribution, called *behavioral*.

Let $q(x)$ be a *positive* probability density function defined on the same domain $\mathcal{D}$, satisfying the condition that $q(x) > 0$ whenever $p(x)f(x) \neq 0$, we have that

$$\begin{aligned}
\mu_p &= \mathop{\mathbb{E}}_{x \sim p} \big[f(x)\big] = \int_{\mathcal{D}} f(x)p(x)\,\mathrm{d}x \\
&= \int_{\mathcal{D}} \frac{f(x)p(x)}{q(x)} q(x)\,\mathrm{d}x = \mathop{\mathbb{E}}_{x \sim q} \left[\frac{f(x)p(x)}{q(x)}\right] \\
&= \mu_q.
\end{aligned}$$

The term $w_{p/q}(x) = p(x)/q(x)$ is called *likelihood ratio* or *importance weight* in the literature, and it is the adjustment term needed to compensate the fact that samples are drawn from $q(x)$ instead of $p(x)$. The sample-based, importance sampling estimator of the expected value of $f(x)$ is

$$\mu_q = \mathop{\mathbb{E}}_{x \sim q}\left[\frac{f(x)p(x)}{q(x)}\right] \quad \Longrightarrow \quad \widehat{\mu_q} = \frac{1}{n}\sum_{i=1}^{n} w_{p/q}(x_i) f(x_i).$$

The importance sampling estimator has the desirable property of being *unbiased*, meaning that the expected value of the estimator equals the expected value of the estimated quantity, i.e.,

$$\mathop{\mathbb{E}}_{x \sim q}\left[\widehat{\mu_q}(x)\right] = \mathop{\mathbb{E}}_{x \sim p}\left[f(x)\right].$$

Unfortunately, it may suffer from exceptionally high *variance* (potentially infinite) of the importance weights, making the estimator of little use in such cases. Intuitively, the more the two distributions p and q differ from one another, the more the variance of the estimation gets bigger. In the following, we will see how to define such a notion of difference (i.e., divergence) between two probability distributions.

2.2.3 Rényi Divergence

The *Rényi divergence* is an information-theoretic measure of the difference between two probability distributions. Let P and Q be two σ-finite probability distributions on the measurable space $(\mathcal{X}, \mathcal{F})$ with p and q their respective probability density functions. If P is *absolutely continuous* with respect to Q (denoted as $P \ll Q$ and meaning that $P(A) = 0$ whenever $Q(A) = 0$ for all events $A \in \mathcal{F}$), we can define the Rényi divergence between P and Q of order α ($\alpha > 0 \wedge \alpha \neq 1$) as

$$d_\alpha(P \parallel Q) = \frac{1}{\alpha - 1} \log \int_{\mathcal{X}} q(x) \left(\frac{p(x)}{q(x)}\right)^{\alpha} \mathrm{d}x.$$

Note that d_α is a *divergence*, not a *distance*, as its definition is not symmetric (i.e., in general $d_\alpha(P \parallel Q) \neq d_\alpha(Q \parallel P)$). The Rényi divergence is a *non-negative quantity* for any $\alpha > 0$ and it is zero if and only if the two distributions are the same, i.e.,

$$d_\alpha(P \parallel Q) = 0 \iff P = Q.$$

In the limit case of $\alpha = 1$, the Rényi divergence converges to the *Kullback-Leibler* (KL) divergence

$$d_1(P \parallel Q) = d_{KL}(P \parallel Q) = \int_{\mathcal{X}} p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x.$$

Finally, we can define the exponentiated α-Rényi divergence as

$$D_\alpha(P \parallel Q) = e^{d_\alpha(P \parallel Q)} = \left[\int_{\mathcal{X}} q(x) \left(\frac{p(x)}{q(x)}\right)^{\alpha} \mathrm{d}x\right]^{\frac{1}{\alpha-1}}.$$

Of particular interest to us are the results presented in [23], where the authors show a tight connection between the importance weights and the Rényi divergence, especially

$$\underset{x \sim q}{\mathbb{E}} \left[w_{P/Q}(x) \right] = 1, \qquad \underset{x \sim q}{\mathbb{V}\mathrm{ar}} \left[w_{P/Q}(x) \right] = D_2(P \parallel Q) - 1.$$

Thanks to this last result, by computing the second order Rényi divergence between any two distributions, we can know in advance how much variance the importance weights will have. More precisely, we can upper bound the variance of the importance sampling estimator [18] as

$$\underset{x \sim q}{\mathbb{V}\mathrm{ar}} \left[\widehat{\mu_q} \right] \leq \frac{\|f\|_\infty^2 \, D_2(P \parallel Q)}{N}$$

where $\|\cdot\|_\infty$ is the infinity norm and N the number of samples taken from q.

2.3 Reinforcement Learning

We are finally ready to tackle the difficult problem of reinforcement learning: finding the optimal policy through sample-based interactions with an *unknown* MDP. We say that the MDP is unknown, meaning that the transition model and the reward function are not explicitly available to the agent, but can only be *observed* by interacting with the environment. It is maybe surprising how, with enough samples, it is possible to find very good solutions even in such a challenging scenario.

2.3.1 Approximate Dynamic Programming

The first techniques we can use to tackle the reinforcement learning problem are a modified version of the dynamic programming algorithms previously outlined. Since we will not use these formulations in the remainder of the thesis, we will limit ourselves to provide references. We greatly encourage the curious reader to consult [24] for a detailed explanation of the first sample-based version of value iteration, *Q-Learning* [25], and for a discussion on the many sample-based versions of policy iteration, like *Monte Carlo Control* or *SARSA* [1]. Instead, we will now focus on the *policy gradient* approach.

2.3.2 Stochastic Policy Gradient

As the dimension of the state-action space grows, representing explicitly the policies becomes increasingly demanding. A possible solution is to choose a suitable parameterization (for example, neural networks [26]), and represent the policy with a parameter vector instead. Many different parameterizations are possible, and how to choose between them is a problem-dependent task left to the user. Differently from the previous sample based algorithms, in policy search methods [27] we do not evaluate a policy by first computing or estimating its value function, but we directly search the solution by navigating the *parametric policy space*

$$\Pi_\Theta = \left\{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subset \mathbb{R}^k\right\},$$

where $\boldsymbol{\theta} \in \Theta$ is a vector of parameters that defines the parametric policy $\pi_{\boldsymbol{\theta}}$, and Θ is the set of all the possible parameters. We require each parametric policy $\pi_{\boldsymbol{\theta}}$ to be stochastic and differentiable with respect to the parameters vector $\boldsymbol{\theta}$. Note that the stochasticity requirement can be relaxed and a *deterministic* policy gradient can be defined [28].

Policy Gradient [29] algorithms exploit the *gradient* (i.e., the local direction of maximum growth) of the performance measure to update the policy parameters closer to the optimal one. This type of approach turns out to be especially useful in scenarios where the state-space and/or the action-space are continuous, in which explicitly computing the state-value function is impractical.

Let us see how to derive $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, the gradient of the performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameters $\boldsymbol{\theta}$. This quantity, difficult to tackle at first glance, turns out to have a simple and concise expression, formalized in the *policy gradient theorem* [30]

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \int_{\mathcal{S}} d^s_{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q_{\pi_{\boldsymbol{\theta}}}(s,a) \,\mathrm{d}a \,\mathrm{d}s.$$

Several approaches have been proposed for estimating such gradient in a sample-based fashion, like [30, 31]. Having a way to compute or estimate the policy gradient, the algorithm is very simple:

Algorithm 1 Stochastic Policy Gradient

```
Input: Initial policy parameters $\boldsymbol{\theta}^{(0)}$, learning rate $\eta$
  for $t = 0, 1, 2, \dots$ until convergence do
      Estimate the policy gradient:  $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$
      Update the policy parameters: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$
  end for
Output: Policy parameters $\boldsymbol{\theta}$
```

Note that, like all gradient-based approaches, unless the objective function is convex or some strict conditions apply [32, 33] we are not guaranteed to reach the global optimum of the performance. The *learning rate* η is a hyper-parameter that determines how fast the policy moves in the policy space. From a theoretical perspective, since the gradient is a *local* measure, the learning rate should approach zero. However, for the sake of converging in a smaller number of iterations, we may choose larger learning rates, while still being careful not to exaggerate. The choice of the optimal learning rate is unfortunately problem-dependent and still an open question. Many works have explored ways to develop a sound *learning rate schedule* [34], we do not cover them here.

2.3.3 The Problem of Exploration

All the procedures presented so far work well as long as the agent is provided an *informative* and *frequent* reward signal. Unfortunately, designing reward functions is a challenging feat and it is not always possible to come up with such well-defined functions. If the reward is *sparse*, meaning that it is zero almost everywhere but in few state-action pairs, the agent will have a much more difficult problem to solve. Sometimes, the difficulty increases so steeply that naive algorithms fail to converge or produce very unsatisfying solutions [4]. This happens because the reward signal is used by the agent as a *guide* to steer towards regions of the MDP where more rewards could be collected. If the reward received by the agent is always zero, and no additional guidance is provided, it will end up randomly wandering the state space, probably never encountering a positive reward, failing its task. Because of this, nowadays *exploration is widely regarded as one of the most challenging problems in reinforcement learning* [5].

If the extrinsic reward function (i.e., the natural one encoded in the MDP) is not informative enough to perform a sufficient and meaningful exploration, we must then devise some methodologies that work also in the complete absence of any pre-determined task. In recent years, the field of *reward-free reinforcement learning* [5] is blossoming with many interesting works in this direction. We can divide the contributions of the field in two main classes: *pre-training for online learning* and *reward-free sampling.* The first category of approaches [6, 7, 8, 9, 10, 11, 12], is to optimize the agent with an explicit exploration objective, usually based on some information-theoretic quantity, like the entropy of the state distribution. The idea is to soft-start the agent with a good exploratory policy, in order to ease the reward-finding problem when the task will finally be revealed. The second category, like [13, 14, 15], is to collect a set of samples (called *batch*) from the environment, informative enough to be sufficient to compute the ϵ-optimal policy of any task, without having to interact with the environment anymore.

2.4 Game Theory

Game theory [2] is a field of mathematics concerned with describing the dynamics of the interaction between a group of *rational* and *selfish* decision-makers, that interact with each other for the sake of reaching some goal. In this setting, we say what an agent is rational and selfish, meaning that, based on its knowledge, it will always choose the most preferred option according to its preferences. Note that this formulation does not exclude the possibility for the players to exhibit *altruistic* behaviors, but only that such attitude has to be made explicit in the player's preferences. Many possible formalizations of strategic games exist in the literature [2], here we will only present the normal form, which will be useful in the remainder of the thesis.

2.4.1 Normal Form Strategic Game

A *normal form strategic game* consists of a collection of three elements $(\mathcal{I}, X, F)$:

- A set of players $\mathcal{I} = \{1, 2, \ldots, n\}$
- For each player, the set of actions available to it $X = \{X_1, X_2, \ldots, X_n\}$
- For each player, a function that encodes its preferences (called *utility function* or *payoff*) $F = \{f_1, f_2, \ldots, f_n\}$.

We can think of the functions in F either as utilities to be maximized or costs to be minimized. In the remainder of the thesis, we use the former convention. If the reader prefers the opposite, it is sufficient to change all $\geq$ with $\leq$ and *max* with *min* in the following definitions.

Normal form games between two players where both can only play a finite number of actions are called *bimatrix games*. Such type of game can easily be represented by two matrices, each representing the payoffs of the two players. A very simple and familiar example of bimatrix game is the *Matching Pennies game*.

	Head	*Tail*
Head	\$1, −\$1	−\$1, \$1
Tail	−\$1, \$1	\$1, −\$1

Figure 2.3: Matching Pennies game (from [2]).

2.4.2 Classification of games

Many classifications of strategic games exist in the literature, we will present in the following a few relevant to our subsequent discussion.

- **Cooperative** vs **Competitive** games.
  In cooperative games, players are allowed to forge alliances and coalitions, while in the competitive setting every player is for itself.
- **Simultaneous** vs **Sequential** games.
  In simultaneous games, all the players choose their actions (i.e., make a move) at the same time. In sequential games instead, there is a predetermined order of play between agents, which can therefore observe the behavior of the preceding players before making their move. In two-player settings, the first player is called *leader* and the second one *follower.*
- **Single-Shot** vs **Repeated** games.
  In single-shot games, the game is played only once, in repeated games the game is repeated a number of times. In this second scenario, the outcome of previous games can of course affect the choices of the players.
- **Constant-Sum** vs **Non-Constant-Sum** games.
  A game is said to be *constant-sum* if, in every possible outcome of the game, the payoffs of all the players sum to the same constant. If this constant is zero, the game is called a *zero-sum game.* If instead such a constant does not exist, the game is called *non-constant-sum.*

2.4.3 Game Strategies

A *strategy* is a complete specification of how the player will act throughout the game. We say that a strategy is a *pure strategy* if the action selection process is deterministic, meaning that given the state of the game the player will always play a particular action with certainty. Instead, if the player chooses how to behave according to some probability distribution over the actions available at the moment, we say that the strategy is a *mixed strategy.* Finally, a *strategy profile* contains a strategy for each player, and therefore it defines how the game will be played. In mathematical terms, we can denote these quantities as:

- The strategy of a single player: x_i
- The set of all possible strategies for player i: X_i
- The strategy profile (a strategy for each player): $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
- The strategy profile for all players except i: $\boldsymbol{x_{-i}} = (x_j)_{j \in \mathcal{I}/\{i\}}$.

2.4.4 Games Equilibria

In game theory, a *solution* to a game is an *equilibrium* between the players. We present in the following two equilibrium criteria: *Nash equilibrium* and *Stackelberg equilibrium.*

Nash Equilibrium

The *Nash Equilibrium* [35], proposed by the mathematician John Nash in 1951, is the most common way to define the solution of simultaneous, non-cooperative games. The equilibrium condition can be informally stated as *no player can benefit from changing his/her strategy, given that the other players do not change theirs.* Formally, we say that a strategy profile $\boldsymbol{x^*} = (x_1^*, x_2^*, \dots, x_n^*)$ is a Nash equilibrium if

$$\forall i \in \mathcal{I},\ f_i(x_i^*, x_{-i}^*) \geq f_i(x_i', x_{-i}^*), \quad \forall x_i' \in X_i.$$

If the number of actions available to each player is finite and mixed strategies are allowed, at least a Nash equilibrium exists [35]. Unfortunately, the computational complexity of finding such equilibrium is PPAD-complete even for a bimatrix game [36]. Therefore, it is impractical to look for such a strategy profile, and many approximated notions of Nash equilibria are used instead, like the *differential Nash equilibrium* [37].

Stackelberg Equilibrium

The *Stackelberg Equilibrium* [38] is a solution concept that applies to sequential, non-cooperative games. Even if the Stackelberg equilibrium concept applies to a generic number of players, we will restrain us here to the simpler two-player setting. The inherent order of play between players (the first one called *leader* and the second one *follower*) is not compatible with the definition of Nash equilibrium given earlier, as in a Nash equilibrium the order of play between players can be interchanged [39]. The leader aims to solve the problem given by

$$\max_{x_1 \in X_1} \{f_1(x_1, x_2) \mid x_2 \in \arg\max_{y \in X_2} f_2(x_1, y)\},$$

while the follower is left with the problem

$$\max_{x_2 \in X_2} f_2(x_1, x_2).$$

As the two-players actually deal with asymmetric problems, the order of play is crucial in a Stackelberg game. A strategy $x_1^* \in X_1$ is called a Stackelberg equilibrium for the leader if

$$\sup_{x_2 \in r(x_1^*)} f_1(x_1^*, x_2) \geq \sup_{x_2 \in r(x_1)} f_1(x_1, x_2), \quad \forall x_1 \in X_1,$$

where $r(x_1) = \{y \in X_2 \mid f_2(x_1, y) \geq f_2(x_1, x_2), \forall x_2 \in X_2\}$ is the *rational reaction set* (or *best response set*) of x_2. Moreover, for any $x_2^* \in r(x_1^*)$, the strategy set (x_1^*, x_2^*) is a Stackelberg equilibrium for the game.

Unfortunately, as for the Nash equilibrium, the computational complexity of finding a (even approximate) Stackelberg equilibrium is PPAD-complete [40]. As a result, there has been a significant interest in the study of approximated equilibrium notions such as *local* or *differential* Stackelberg equilibria [38].

Local Stackelberg Equilibrium

The definition of local Stackelberg equilibrium is quite similar to the standard definition, with the only addition of restricting the equilibrium condition to a subset of all the strategies.

Consider $U_i \subset X_i$ for $i = 1, 2$, the strategy $x_1^* \in U_1$ is a *local* Stackelberg solution for the leader if

$$\sup_{x_2 \in r_{U_2}(x_1^*)} f_1(x_1^*, x_2) \geq \sup_{x_2 \in r_{U_2}(x_1)} f_1(x_1, x_2), \quad \forall x_1 \in U_1,$$

where $r_{U_2}(x_1) = \{y \in U_2 \mid f_2(x_1, y) \geq f_2(x_1, x_2), \forall x_2 \in U_2\}$ is the best response set of $x_2 \in U_2$. Moreover, for any $x_2^* \in r_{U_2}(x_1^*)$, the strategy set (x_1^*, x_2^*) is a local Stackelberg equilibrium for the game.

Differential Stackelberg Equilibrium

The notion of differential Stackelberg equilibrium stems from the first and second-order conditions on the objective function that must hold for its extreme points. Let us denote the total derivative as ∇, the second-order total derivative as $\nabla^T \nabla$ and the second-order derivative with respect to x as $\nabla_x^T \nabla_x$. The joint strategy $x = (x_1^*, x_2^*) \in X$ is a *differential* Stackelberg equilibrium if

$$\begin{aligned} &\nabla_{x_1} f(x_1^*, x_2^*) = 0, && \nabla_{x_2} f(x_1^*, x_2^*) = 0, \\ &\nabla^T \nabla f(x_1^*, x_2^*) > 0, && \nabla_{x_2}^T \nabla_{x_2} f(x_1^*, x_2^*) > 0. \end{aligned}$$

Moreover, it is interesting to note that, in a generic zero-sum game, the notion of differential and local Stackelberg equilibrium *coincide* for sufficiently smooth functions f_1 and f_2 [38].

2.4.5 Gradient Descent Ascent

Having defined the approximated equilibrium conditions, let us now see how to find differential Stackelberg equilibria in practice. We will tackle the simpler *two-player*, *zero-sum* setting, where $f_1 = -f$ and $f_2 = f$. The *gradient descent ascent* algorithm stems directly from the well-known *gradient descent* algorithm, presented in Section 2.3.2. While the latter has only one function and one parameter vector to optimize, the gradient descent ascent algorithm applies the same idea to two functions and two parameters vectors at the same time. If the updates of the two players are done sequentially, the set of stable critical points of the GDA algorithm coincides with the set of local Stackelberg equilibrium points [20]. Nonetheless, in practice it is possible to relax the separation between the two players, by setting accordingly their learning rates. The intuitive idea is that, if the learning rate of the follower η_2 is much bigger than the one of the leader η_1, the former will move much faster in the parameter space, allowing it to reach a (local) best response before its adversary has moved significantly from its last position. The proportionality factor between the two learning rates is called *timescale separation* in the literature and usually denoted with τ. The resulting algorithm is called τ-GDA.

Algorithm 2 Gradient Descend Ascend (τ-GDA)

```
**Input:** Initial parameters $(x_1^{(0)}, x_2^{(0)})$, timescale separation $\tau = \eta_2/\eta_1$
  **for** $t = 0, 1, 2, \ldots$, until convergence **do**
      $x_1^{(t+1)} \leftarrow x_1^{(t)} - \eta_1 \nabla_{x_1} f(x_1^{(t)}, x_2^{(t)})$
      $x_2^{(t+1)} \leftarrow x_2^{(t)} + \underbrace{\tau\eta_1}_{\eta_2} \nabla_{x_2} f(x_1^{(t)}, x_2^{(t)})$
  **end for**
**Output:** Stable critical point $\boldsymbol{x} = (x_1, x_2)$
```

If $\tau \to \infty$, we obtain the vanilla, completely separated update. If $\tau = 1$, it is known that the algorithm is not guaranteed to converge to a game-theoretically meaningful point [41]. Interestingly, a recent work [19] filled the gap between these two extremes, by proving that there exists a *finite* τ^* such that $\forall\tau \in (\tau^*, \infty)$, $\boldsymbol{x}$ is a stable critical point of τ-GDA if and only if it is a local Stackelberg equilibrium.

Chapter 3

Policy Space Compression

In this chapter, we will present the main focus of this thesis, the *policy space compression* problem. In Section 3.1, we will contextualize the problem in the reinforcement learning setting. In Section 3.2, we will provide a mathematical formulation and analyze its computational complexity.

3.1 The Policy Space in Reinforcement Learning

The policy space is a notoriously difficult-to-handle component in reinforcement learning problems. First of all, it is huge: the number of deterministic policies is exponential in the number of states ($|\mathcal{A}|^{|\mathcal{S}|}$), and the number of stochastic policies is infinite. Moreover, as we have seen in Section 2.1.2, the relation between a policy and its induced state-action distribution is strongly problem-dependent, as it relies on the transition model of the MDP. Given the size and the complexity of such space, it should not come as a surprise that navigating it looking for an optimal policy is a demanding feat, especially if the state and/or action spaces are infinite. For example, policy gradient algorithms try to manage this complexity by exploiting local information on the growth direction of the performance (i.e., its gradient) in order to move the current policy closer to the optimal one. While policy search methods may still operate well in practice, they nonetheless have to handle the undiminished complexity of the policy space.

3.1.1 The OPTIMIST Approach

A different approach to the policy search problem comes from OPTIMIST [16] and subsequently RANDOMIST [17], where the policy search problem is formulated as a *structured Multi Armed Bandit* (MAB) [42, 43] problem. In the MAB setting, the agent is given a set of actions (called arms), each one associated with a stationary reward function, *unknown* to the agent. The goal is to promptly find the most rewarding arm, doing as few actions as possible. While in the classical setting, the rewards of the arms are all mutually independent (i.e., pulling one arm does not give any information on the other arms), in the *structured* MAB setting, the former is not true, and it is possible to *estimate* the reward that would have been obtained by pulling a different arm than the selected one. The idea of OPTIMIST is to treat the whole parameter space of the parametric policy space (defined in Section 2.3.2) as the set of *continuous* actions available to the MAB algorithm, where the reward for each arm is the *performance* of the corresponding policy. To exploit the underlying structure of the problem, OPTIMIST applies a fine-tuned version of the importance sampling estimator presented in Section 2.2.2, as a means to use the samples collected with the selected arm (i.e., the *behavioral* policy) to estimate the performance of a different arm. Of course, as we have seen in Section 2.2.3, the variance of the importance sampling estimator strictly depends on the *Rényi divergence* between the induced state-action distributions of the two policies, which means that they must be sufficiently *similar* in order to be used in such way. In order to construct its arm set and perform such estimation, the algorithm has to perform a discretization of the, otherwise infinite, policy space. Clearly, for the sake of estimating a larger number of arms, one would prefer to have a set of *representative* policies, instead of a collection of unrelated elements. Unfortunately, the complex relation between a policy and its induced state-action distribution makes this type of discretization very difficult and problem-dependent.

3.1.2 An Interesting Question

Let us digest all the considerations made so far. The policy space is a considerably complex, but fundamental, element in the reinforcement learning context. Its cardinality is infinite (or exponential if restricted to the deterministic subset), and it is related to the distribution space with a complex, problem-dependent relation. Since it represents the set of all the possible agent's strategies within the environment, we must deal with it in order to find the optimal policy of the given task. Different algorithms perform this search in different ways, each with its pros and cons. For example, it is possible to reduce the policy search problem to a structured MAB problem, exploiting the relation that exists between the performance of similar policies with the importance sampling estimator. For the sake of performing such evaluation, not all the policies have the same capabilities, and we can use the Rényi divergence between their induced state-action distributions to quantify the variance of the estimation. In light of all these considerations, one question arises naturally:

Is it possible to perform a ***compression*** *of the* ***infinite*** *policy space into a* ***finite, representative*** *set, by keeping all and only the components that are needed to reliably estimate the performance of any other policy in the policy space?*

In this work, we give a *positive* answer to the previous question and propose an algorithm to find such reduced policy space in an efficient way.

3.2 Problem Statement

Let us now formally present the problem statement of this thesis.

Policy Space Compression

Let $\mathcal{M}$ be an MDP defined by the tuple $(\mathcal{S}, \mathcal{A}, P, d_0)$ and ϵ a coverage requirement expressed in α-Rényi divergence. The goal of *Policy Space Compression* is to reduce the policy space Π to a *finite*, *representative*, *minimal* set of elements $\mathcal{C}$. These elements are such that for every policy $\mu \in \Pi$, there exist at least one element $\pi \in \mathcal{C}$ that is sufficiently close to μ, meaning that the α-Rényi divergence between the state-action distributions induced by μ and π is bounded by ϵ.

3.2.1 Problem Complexity

As the reader may have noticed, we are presenting a problem that is, in its essence, a *set-covering problem*, which is well known to be NP-hard [44]. Because of the problem complexity, finding the exact solution is highly impractical. Nonetheless, in the following chapter, we will provide a dual game-theoretic formulation that we plan to approximately solve with sample-based methods.

3.2.2 Coverage of Deterministic Policies

Elaborating further on the goal of the compression, it is important to know that for every reward function, there exists at least a *stationary*, *Markovian* and <u>*deterministic*</u> optimal policy [3]. Since our motivation is to ease the policy search problem (i.e., the search for the *optimal* policy of the given task), we can focus our attention solely on the coverage of the deterministic subset of the policy space. In this way, we restrict the goal of the coverage to a smaller set, and we avoid requiring the solution to have more components than needed.

3.2.3 Formal Problem Statement

Wanting to outline more formally the (deterministic) policy space compression problem, we report in the following the conditions that the covering set $\mathcal{C}$ must satisfy in order to be a viable solution.

Policy Space Compression

Let the policy space Π be the set of Markovian, stationary policies and $\Pi^D \subset \Pi$ be the set Markovian, stationary, *deterministic* policies. Given an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, d_0)$ and a coverage requirement ϵ expressed in α-Rényi divergence, the solution $\mathcal{C} \subset \Pi$ to the policy space compression problem is a *finite*, *representative*, *minimal* covering set of policies, i.e.,

- $|\,\mathcal{C}\,| < \infty$
- $\forall \mu \in \Pi^D,\ \exists\, \pi \in \mathcal{C} \mid D_\alpha(d_\mu^{sa} \parallel d_\pi^{sa}) \leq \epsilon$
- $\nexists\, \mathcal{C}' \mid \big(\forall \mu \in \Pi^D,\ \exists\, \pi \in \mathcal{C}' \mid D_\alpha(d_\mu^{sa} \parallel d_\pi^{sa}) \leq \epsilon\big) \wedge \big(|\mathcal{C}'| < |\mathcal{C}|\big)$

Chapter 4

An Algorithm for Policy Space Compression

In this chapter, we will provide a dual formulation of the policy space compression problem as a non-convex two-player game. Even though finding the global solution of such a game is still computationally impractical, it is possible to efficiently find locally optimal solutions by employing gradient-based techniques. By using these methods, we are able to develop an algorithm to find approximated solutions to the policy space compression problem. In Section 4.1, we will present the dual game-theoretic formulation and analyze the fundamental properties of the game. In Section 4.2, we will report the pseudocode of the approximated solution algorithm and explain its general line of reasoning.

4.1 Min-Max Game Formulation

Let us consider a set of parameterized covering policies and a generic parametric *deterministic* policy

$$\mathcal{C} = \{\pi_{\boldsymbol{\theta}_k} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^n\}_{k\in[K]}, \qquad \pi_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in M \subseteq \mathbb{R}^n \mid \pi_{\boldsymbol{\mu}} \in \Pi^D.$$

If we fix the number of covering policies to K, we can recast the policy space compression problem into a *non-convex min-max game*[45] of the form

$$\min_{\boldsymbol{\theta}\in\Theta^K} \max_{\boldsymbol{\mu}\in M} f(\boldsymbol{\theta}, \boldsymbol{\mu}), \tag{4.1}$$

where the objective function f is defined as

$$f(\boldsymbol{\theta}, \boldsymbol{\mu}) := \min_{k\in[K]} D_\alpha\big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha-1}. \tag{4.2}$$

The optimal solution of the game $(\boldsymbol{\theta}^*, \boldsymbol{\mu}^*)$, represents the covering set $\mathcal{C}$ with $\boldsymbol{\theta}^*$ and the most distant policy from the set with $\boldsymbol{\mu}^*$. Since the first player can be either seen as a single leader with multiple components or a fully coordinating coalition of K players, with some overloading of notation in the following we will use the term *leader* to refer to both notations interchangeably.

4.1.1 Game Properties

We can characterize the game defined in (4.1) by stating some of its core properties, which are listed below.

- The game is **zero-sum**. From the definition in (4.1), the utility of the first player is the opposite of the utility of the second player.
- The game is clearly **competitive** by construction.
- The game is **sequential**: the order of play greatly influences the resulting solution. For example, consider what would happen if the min player had its turn *after* the max player. By setting one of its policies equal to the one put forward by the max player, it could easily win the game and obtain $f(\boldsymbol{\theta}, \boldsymbol{\mu}) = 0$, with any configuration of the remaining $K - 1$ policies. Clearly, this is not a way of finding good covering policies, therefore the min player *must* play before the max player. Since we know from the minimax theorem [39], that in a Nash equilibrium the order of play can be interchanged, it is clear that the equilibrium criterion that we must adopt is the one defined by Stackelberg.
- The game is **single shot**, as it does not make sense to play more than once. As the reader may have noticed, the role of the player $\boldsymbol{\mu}$ is quite ancillary here, as it serves only as motivation for the leader to find good covering policies.

4.2 How to Solve the Min-Max Game

To solve the policy space compression problem, we can iteratively solve its dual game-theoretic formulation presented above. Starting the procedure with $K = 1$, we find the corresponding optimal solution of the two-player game in (4.1). With the optimal solution, we check if the obtained coverage $f(\boldsymbol{\theta}^*, \boldsymbol{\mu}^*)$ is smaller than the coverage requirement ϵ. If such evaluation is successful, it means that we have found the desired covering policies and we can stop the procedure. Otherwise, it means that the current dimension K is not enough to satisfy the requirements, and therefore we will need to repeat the procedure with $K + 1$ policies. In order to find the solution to the compression problem, we will continue to increment K until the corresponding optimal solution of the game is able to satisfy the coverage requirement.

4.2.1 Finding the Solution in Practice

The procedure we just presented is a viable way to exactly solve the policy space compression problem. Unfortunately, as we outlined in Section 2.4.4, finding the exact solution to a generic non-convex two-player game is computationally very challenging. As a consequence, in order to be able to find a set of covering policies also for big MDPs, we will have to employ an approximated procedure for finding a local solution to the two-player game, and therefore an approximated solution of the policy space compression problem. As we will see in Chapter 7, even if we are not able to efficiently find the global solution to the two-player game, it is possible to *upper bound* its value, and therefore to conservatively assess the quality of the covering policies found. Note that, by relaxing the optimality requirement on the solution of the game, we are implicitly relaxing the minimality requirement on the covering set.

4.2.2 Solution Algorithm

We are ready to present the algorithm employed in this thesis. The general structure is the same as the exact procedure outlined in the previous section, with the only difference that instead of the *global* optimal solution to the two-player game, we employ a *locally* optimal one. In particular, in order to find such a solution we will use the ∞-GDA algorithm presented in Section 2.4.5, or a slight modification that we will present in Section 6.3.2.

Algorithm 3 Policy Space Compression

```
Input: Coverage requirement $\epsilon$, MDP $\mathcal{M}$
  $\boldsymbol{\theta}^{(0)} \leftarrow$ INITIALIZE-LEADER-COALITION($K = 1$)
  repeat
      for $t = 0, 1, 2, \dots,$ until convergence do
          $\boldsymbol{\mu}_{br} \leftarrow$ BEST-RESPONSE($\boldsymbol{\theta}^{(t)}$)
          $\boldsymbol{\theta}^{(t+1)} \leftarrow$ UPDATE-COALITION($\boldsymbol{\mu}_{br}$)
      end for
      if CHECK-COVERAGE($\boldsymbol{\theta}$) $> \epsilon$ then
          $K \leftarrow K + 1$
          $\boldsymbol{\theta}^{(0)} \leftarrow$ UPDATE-LEADER-COALITION($K$)
      end if
  until CHECK-COVERAGE($\boldsymbol{\theta}$) $\leq \epsilon$
Output: Covering policies $\mathcal{C}$
```

In the next chapters, we will take a closer look at each of the three main components of the algorithm. We will start from the follower's perspective, then present the leader's optimization, and conclude with the coverage guarantees. *Bon appétit!*

Chapter 5

Finding the Best Response

In this chapter, we will tackle the game-theoretic formulation from the follower's perspective, and we will see different alternative methods to find a solution. In Section 5.1, we will provide a formalization of the problem amenable to global optimization with off-the-shelf solvers. In Section 5.2, we will outline an alternative gradient-based approach and analyze some of its peculiar properties. In Section 5.3, we will report an efficient way to find a surrogate solution to the problem, close enough to the true solution to be a worthy challenge for the leader.

5.1 Problem Formulation

In a minimax game, given the leader's strategy x_1, the follower's problem reduces to finding the best response x_2^* in the set of all possible follower's strategies X_2, i.e.,

$$x_2^* \in \arg\max_{x_2 \in X_2} f_2(x_1, x_2).$$

Let us take this generic problem definition in the context of the policy space compression problem.

5.1.1 Policy Space Formulation

The problem of finding the deterministic best response (denoted as $\boldsymbol{\mu}_{br}$) to the leader's strategy can be formalized as follows

$$\boldsymbol{\mu}_{br} \in \arg\max_{\boldsymbol{\mu} \in M} \Big(\min_{k \in [K]} D_\alpha \big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha-1} \Big). \tag{5.1}$$

Unfortunately, this problem formulation is difficult to solve in practice, as the policy parameter $\boldsymbol{\mu}$ and its induced state-action distribution $d^{sa}_{\boldsymbol{\mu}}$ are linked by the recursive relation in Equation (2.1), that is challenging to be expressed exactly in a non-recursive form.

5.1.2 From Primal to Dual Variables

Fortunately, it is possible to avoid this limitation by performing a proper change of variable, from *policies* to *distributions*, and look for the solution of the problem directly in the distribution space. Once we found the solution, we can revert the variable change and extract the corresponding policy with Equation (2.2). Notably, not all the distributions in the distributions space are induced by some policy. Therefore, in order to leave unaltered the nature of the problem, we will have to enforce the distribution found by the optimization to be a *valid* distribution (i.e., that it is induced by some policy on the MDP) with Equation (2.3).

5.1.3 Distribution Space Formulation

We present in the following the problem formalization in the distribution space, that is now amenable to be solved by standard optimization techniques

$$
\begin{aligned}
\underset{\boldsymbol{d} \in \mathbb{R}^{sa}}{\text{maximize}} \quad & \min_{k \in [K]} D_\alpha\big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha-1} \\
\text{subject to:} \quad & d(s,a) \geq 0, \qquad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} && (1) \\
& \int_{\mathcal{SA}} d(s,a) = 1 && (2) \\
& \int_{\mathcal{A}} d(s,a)\,\mathrm{d}a = (1-\gamma) d_0(s) \\
& \qquad + \gamma \int_{\mathcal{SA}} d(s',a') P(s|s',a')\,\mathrm{d}s'\,\mathrm{d}a', \ \forall s \in \mathcal{S} && (3) \\
& \|d(s,\cdot)\|_1 = \|d(s,\cdot)\|_2, \quad \forall s \in \mathcal{S} && (4)
\end{aligned}
$$

where $\boldsymbol{d} = \big(d(s,a)\big)_{s \in \mathcal{S}, a \in \mathcal{A}}$ is the vector that represents the state-action distribution of the follower and $\|.\|_p$ is the standard L^p-norm. Let us now analyze each constraint more in depth to showcase its purpose.

1–2. The first two constraints ensure that the *continuous* variables $d(s,a)$ are indeed specifying a probability distribution over the state-action space of the MDP.

3. The third constraint makes sure that the distribution specified in $\boldsymbol{d}$ is a *valid* distribution, permitting the change of variable.

4. The fourth constraint specifies that the state-action distribution $\boldsymbol{d}$ is induced by a *deterministic* policy.

To understand how the last constraint works, notice that by applying Equation (2.2), we can infer that a policy π is deterministic if and only if its induced state-action distribution satisfies the following

$$\forall s \in \mathcal{S}, \exists!\, a \in \mathcal{A} \mid \frac{d_\pi^{sa}(s,a)}{\sum_{a' \in \mathcal{A}} d_\pi^{sa}(s,a')} = 1,$$

which means that, for each state, the state-action distribution has only one non-zero entry. We can express this last condition by enforcing that, for each state, the summation of the elements in $d(s,\cdot)$ must be equal to *square root* of the summation of the *squares* of the same elements.

5.1.4 Deterministic vs Stochastic Best Response

Before analyzing the complexity of the proposed problem in the next section, it is useful to first make a small detour on the effect of the fourth constraint on the problem. One could ask:

> *Does the solution to the follower's problem change if we require the best response to be a **deterministic** policy?*

As it turns out, if the leader's coalition is composed of more than one element, the answer is positive. As we will see in Chapter 8, if the follower is allowed to pick non-deterministic policies, it can choose a policy that randomizes over strategies that are collectively averse to the leader, even if the single element is not. Such randomized policies are usually hard to cover and they might cause the algorithm to *underestimate* the coverage of the deterministic ones, forcing it to add more leaders to the coalition. On the bright side, the coverage obtained on the non-deterministic policies is a conservative measure of the coverage obtained by considering only deterministic best responses. It is easy to see why this is true, by noting that the deterministic best response problem is equivalent to the stochastic one with an additional constraint (i.e., the fourth). Clearly, any solution found by a *more restrictive* version of the same problem cannot exceed the solution found by the restricted version.

5.1.5 Problem Complexity

The problem outlined in Section 5.1.3, even in its dual formulation, still includes a number of significant challenges. While the first three constraints are linear in the decision variables $d(s,a)$, the fourth one is an equality quadratic constraint, which is non-convex. Moreover, the presence of the *min* in the objective function makes it non-convex as well.

Non-convex optimization is a much more challenging problem than its convex counterpart, as we are bound with the existence of local minima (or maxima), which makes global optimization harder. Indeed, it is well-known that the computational complexity for solving a generic non-convex problem is exponential in the size of the problem. The inherent hardness means that we are unfortunately unable to find the exact solution for big MDPs, making evident the need for an *approximated* approach.

5.2 Gradient-Based Approach

In this section, we present a scalable, gradient-based approach to optimize for a relaxed solution concept, i.e., local optimality. Since the purpose of the follower is to *challenge* the leader with difficult-to-cover policies to motivate it to present better covering policies, it is not strictly necessary to provide to the leader the *global* best response at each optimization step, but any challenging enough policy will make it.

5.2.1 Handling the Minimum

Before we can dive into the discussion, we must first solve one last issue with the objective function. Let us report here once again the definition of f:

$$f(\boldsymbol{\theta}, \boldsymbol{\mu}) := \min_{k \in [K]} D_\alpha\big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha-1}.$$

While it's possible to extract the gradient of the inner term, it is not trivial how to handle the *min* in the above equation, since it makes the whole equation *non-differentiable.* A possible solution is to consider only the *closest* component of the leader's coalition at each step and move the follower's policy away from it. Clearly, since the closest leader can *change* after some iterations, the direction will be always adjusted to consider the *currently* closest component.

5.2.2 Policy Gradient Derivation

Let us now derive the gradient of the inner term of the objective function with respect to the follower's parameters. We can apply the well-known *chain rule* formula to ease our computations

$$\nabla_{\boldsymbol{\mu}} D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1} = \frac{\partial D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}}{\partial \boldsymbol{\mu}} = \frac{\partial D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}}{\partial d^{sa}_{\boldsymbol{\mu}}} \frac{\partial d^{sa}_{\boldsymbol{\mu}}}{\partial \boldsymbol{\mu}}.$$

The derivative of the Rényi divergence with respect to its first argument is easy to obtain

$$\frac{\partial D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}}{\partial d^{sa}_{\boldsymbol{\mu}}} = \alpha \int_{\mathcal{SA}} \left(\frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{d^{sa}_{\boldsymbol{\theta}_k}(s,a)} \right)^{\alpha-1} \mathrm{d}s\, \mathrm{d}a.$$

The second term, i.e., the derivative of the γ-discounted state-action distribution with respect to the policy parameters $\boldsymbol{\mu}$, requires instead a bit more computations

$$\begin{aligned}\frac{\partial d^{sa}_{\boldsymbol{\mu}}(s,a)}{\partial \boldsymbol{\mu}} &= d^{sa}_{\boldsymbol{\mu}}(s,a)\,\nabla_{\boldsymbol{\mu}} \log d^{sa}_{\boldsymbol{\mu}}(s,a)\\ &= d^{sa}_{\boldsymbol{\mu}}(s,a)\Big(\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(s)\Big).\end{aligned}$$

Chaining the two terms we obtain

$$\begin{aligned}&\nabla_{\boldsymbol{\mu}} D_\alpha(d^{sa}_{\boldsymbol{\mu}}||d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1} =\\ &= \alpha \int_{\mathcal{SA}} d^{sa}_{\boldsymbol{\theta}_k}(s,a)\left(\frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{d^{sa}_{\boldsymbol{\theta}_k}(s,a)}\right)^{\alpha}\Big(\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(s)\Big)\,\mathrm{d}s\,\mathrm{d}a.\end{aligned}$$

The term $\nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(s)$ is particularly tricky to compute in a form that is suitable for model-free estimation in continuous environments, due to the recursive relation that links the policy parameter $\boldsymbol{\mu}$ and the corresponding state-distribution. Fortunately, many works in the *imitation learning* [46, 47, 48] area tackle this problem from various angles.
Taking inspiration from [46, 47], we can write

$$\begin{aligned}\nabla_{\boldsymbol{\mu}} d^{s}_{\boldsymbol{\mu}}(s) &= \nabla_{\boldsymbol{\mu}}\bigg((1-\gamma)d_0(s) + \gamma\int_{\mathcal{SA}} d^{s}_{\boldsymbol{\mu}}(\overline{s})\pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})P(s|\overline{s},\overline{a})\,\mathrm{d}\overline{s}\,\mathrm{d}\overline{a}\bigg)\\ d^{s}_{\boldsymbol{\mu}}(s)\nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(s) &= \gamma\int_{\mathcal{SA}} d^{s}_{\boldsymbol{\mu}}(\overline{s})\pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})P(s|\overline{s},\overline{a})\bigg(\nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(\overline{s})\\ &\qquad + \nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})\bigg)\,\mathrm{d}\overline{s}\,\mathrm{d}\overline{a}\\ \nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(s) &= \int_{\mathcal{SA}} \gamma\frac{d^{s}_{\boldsymbol{\mu}}(\overline{s})}{d^{s}_{\boldsymbol{\mu}}(s)}\pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})P(s|\overline{s},\overline{a})\bigg(\nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(\overline{s})\\ &\qquad + \nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})\bigg)\,\mathrm{d}\overline{s}\,\mathrm{d}\overline{a}\\ \nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(s) &= \int_{\mathcal{SA}} \gamma Q^{0}_{\boldsymbol{\mu}}(\overline{s},\overline{a}|s)\Big(\nabla_{\boldsymbol{\mu}} \log d^{s}_{\boldsymbol{\mu}}(\overline{s}) + \nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})\Big)\,\mathrm{d}\overline{s}\,\mathrm{d}\overline{a},\end{aligned}$$

where in the last equation we have substituted the *inverse dynamics model* $Q^{0}_{\pi}(\overline{s},\overline{a}|s)$, which represents for each state s the probability of having reached that state by performing action $\overline{a}$ in state $\overline{s}$ in the previous step

$$Q^{0}_{\pi}(\overline{s},\overline{a}|s) = \frac{d^{s}_{\pi}(\overline{s})}{d^{s}_{\pi}(s)}\pi(\overline{a}|\overline{s})P(s|\overline{s},\overline{a}).$$

5.2.3 First-Order Algorithm

We have all the ingredients to construct a gradient-based algorithm for finding an approximated best response, of which we report the pseudocode in the following.

Algorithm 4 Best Response Gradient

```
Input: Leader strategy $\boldsymbol{\theta} = (\boldsymbol{\theta}_k)_{k\in[K]}$, divergence order $\alpha$, learning rate $\eta$
  $\boldsymbol{\mu}^{(0)} \leftarrow$ Initialize-Follower-Policy($\boldsymbol{\theta}$)
  for $t = 0, 1, 2, \dots$ until convergence do
    $\overline{k} \leftarrow$ Closest-Leader-Index($\boldsymbol{\theta}, \boldsymbol{\mu}^{(t)}$)
    $\boldsymbol{\mu}^{(t+1)} \leftarrow \boldsymbol{\mu}^{(t)} + \eta\, \nabla_{\boldsymbol{\mu}} D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_{\overline{k}}})^{\alpha-1}$
  end for
Output: Follower's policy parameters $\boldsymbol{\mu}$
```

5.2.4 Second-Order Algorithm

Algorithm 4 that we presented in the previous section uses a first-order approximation of the objective function (i.e., the gradient) to update the parameter $\boldsymbol{\mu}$ in order to maximize the objective function. However, it is possible to define a *second-order* algorithm that is able to exploit not only the first-order approximation of the objective function, but also its curvature, which can provide a significant benefit in minimax optimization [49]. After computing the second-order derivative (i.e., the Hessian) of $D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}$, denoted as $\mathcal{H} = \nabla^\top_{\boldsymbol{\mu}} \nabla_{\boldsymbol{\mu}} D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}$, we can modify the update rule of Algorithm 4 and compute the curvature-aided policy update as follows

$$\boldsymbol{\mu}^{(t+1)} \leftarrow \boldsymbol{\mu}^{(t)} + \big(\mathcal{H}\big)^{-1} \nabla_{\boldsymbol{\mu}} D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}.$$

Moreover, in order to generally improve the stability of the second-order optimization, we can substitute the inverse of the Hessian $\mathcal{H}$ with its positive truncated inverse [50]. As we will see in Section 8.1, the addition of the information on the curvature can greatly help the optimization, enabling the follower to find the *global* best-response even in very challenging leader configurations. Unfortunately, calculating the Hessian matrix is a computationally demanding operation that will obviously slow down the whole algorithm. One interesting direction for future works could be to devise some heuristics to dynamically decide whether to switch to the second-order method as the leader provides more and more challenging covering policies during the optimization.

5.2.5 Second-Order Policy Gradient Derivation

Let us start by computing the Hessian of the Rényi divergence with respect to the parameter $\boldsymbol{\mu}$

$$\begin{aligned}
&\nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} D_{\alpha}(d_{\boldsymbol{\mu}}^{sa} || d_{\boldsymbol{\theta}_k}^{sa})^{\alpha-1} = \\
&= \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \int_{\mathcal{SA}} \big(d_{\boldsymbol{\mu}}^{sa}(s,a)\big)^{\alpha} \big(d_{\boldsymbol{\theta}_k}^{sa}(s,a)\big)^{1-\alpha} \, \mathrm{d}s \, \mathrm{d}a \\
&= \nabla_{\boldsymbol{\mu}} \int_{\mathcal{SA}} \alpha \big(d_{\boldsymbol{\mu}}^{sa}(s,a)\big)^{\alpha-1} \big(d_{\boldsymbol{\theta}_k}^{sa}(s,a)\big)^{1-\alpha} \nabla_{\boldsymbol{\mu}} d_{\boldsymbol{\mu}}^{sa}(s,a) \, \mathrm{d}s \, \mathrm{d}a \\
&= \alpha \int_{\mathcal{SA}} \left(\frac{d_{\boldsymbol{\mu}}^{sa}(s,a)}{d_{\boldsymbol{\theta}_k}^{sa}(s,a)} \right)^{\alpha-1} \Bigg(\nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} d_{\boldsymbol{\mu}}^{sa}(s,a) \\
&\qquad\qquad + (\alpha-1) \frac{\nabla_{\boldsymbol{\mu}} d_{\boldsymbol{\mu}}^{sa}(s,a) \; \nabla_{\boldsymbol{\mu}}^{\top} d_{\boldsymbol{\mu}}^{sa}(s,a)}{d_{\boldsymbol{\mu}}^{sa}(s,a)} \Bigg) \, \mathrm{d}s \, \mathrm{d}a.
\end{aligned}$$

Notably the second factor contains the Hessian of the state-action distribution $d_{\boldsymbol{\mu}}^{sa}$ and the square of its first order derivative. We can easily obtain the latter following similar steps as in the previous section

$$\begin{aligned}
&\frac{\nabla_{\boldsymbol{\mu}} d_{\boldsymbol{\mu}}^{sa}(s,a) \; \nabla_{\boldsymbol{\mu}}^{\top} d_{\boldsymbol{\mu}}^{sa}(s,a)}{d_{\boldsymbol{\mu}}^{sa}(s,a)} = d_{\boldsymbol{\mu}}^{sa}(s,a) \big(\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{sa}(s,a) \; \nabla_{\boldsymbol{\mu}}^{\top} \log d_{\boldsymbol{\mu}}^{sa}(s,a) \big) = \\
&= d_{\boldsymbol{\mu}}^{sa}(s,a) \Big[\big(\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(s) \big) \big(\nabla_{\boldsymbol{\mu}}^{\top} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}}^{\top} \log d_{\boldsymbol{\mu}}^{s}(s) \big) \Big].
\end{aligned}$$

It is less straightforward to obtain the Hessian of $d_{\boldsymbol{\mu}}^{sa}$, as it is

$$\begin{aligned}
&\nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} d_{\boldsymbol{\mu}}^{sa}(s,a) = \nabla_{\boldsymbol{\mu}} \Big(d_{\boldsymbol{\mu}}^{sa}(s,a) \nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{sa}(s,a) \Big) = \\
&= d_{\boldsymbol{\mu}}^{sa}(s,a) \Big(\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{sa}(s,a) \; \nabla_{\boldsymbol{\mu}}^{\top} \log d_{\boldsymbol{\mu}}^{sa}(s,a) + \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{sa}(s,a) \Big) \\
&= d_{\boldsymbol{\mu}}^{sa}(s,a) \Big(\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{sa}(s,a) \; \nabla_{\boldsymbol{\mu}}^{\top} \log d_{\boldsymbol{\mu}}^{sa}(s,a) \\
&\qquad\qquad + \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(s) \Big).
\end{aligned}$$

Going further expanding the last term, we get

$$
\begin{aligned}
&\nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(s) = \\
&= \int_{\mathcal{SA}} \gamma Q_{\boldsymbol{\mu}}^{0}(\overline{s},\overline{a}|s)\Big(\nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(\overline{s}) + \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})\Big) \\
&\qquad\qquad + \gamma \nabla_{\boldsymbol{\mu}}^{\top} Q_{\boldsymbol{\mu}}^{0}(\overline{s},\overline{a}|s)\Big(\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(\overline{s}) + \nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})\Big) \,\mathrm{d}\overline{s}\,\mathrm{d}\overline{a} \\
&= \int_{\mathcal{SA}} \gamma Q_{\boldsymbol{\mu}}^{0}(\overline{s},\overline{a}|s)\Bigg[\nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(\overline{s}) + \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s}) \\
&\qquad\qquad + \Big(\nabla_{\boldsymbol{\mu}}^{\top} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s}) + \nabla_{\boldsymbol{\mu}}^{\top} \log d_{\boldsymbol{\mu}}^{s}(\overline{s}) - \nabla_{\boldsymbol{\mu}}^{\top} \log d_{\boldsymbol{\mu}}^{s}(s)\Big) \\
&\qquad\qquad\qquad \times \Big(\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(\overline{s}) + \nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s})\Big)\Bigg] \,\mathrm{d}\overline{s}\,\mathrm{d}\overline{a},
\end{aligned}
$$

by noting that

$$
\nabla_{\boldsymbol{\mu}} Q_{\boldsymbol{\mu}}^{0}(\overline{s},\overline{a}|s) = Q_{\boldsymbol{\mu}}^{0}(\overline{s},\overline{a}|s)\Big(\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(\overline{a}|\overline{s}) + \nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(\overline{s}) - \nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(s)\Big).
$$

Finally, bringing it all together we have

$$
\begin{aligned}
&\nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} D_{\alpha}(d_{\boldsymbol{\mu}}^{sa}\|d_{\boldsymbol{\theta}_k}^{sa})^{\alpha-1} = \\
&= \alpha \int_{\mathcal{SA}} d_{\boldsymbol{\theta}_k}^{sa}(s,a)\left(\frac{d_{\boldsymbol{\mu}}^{sa}(s,a)}{d_{\boldsymbol{\theta}_k}^{sa}(s,a)}\right)^{\alpha} \\
&\quad \times \Big(\alpha\big(\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(s)\big)\big(\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(s)\big)^{\top} \\
&\qquad + \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log \pi_{\boldsymbol{\mu}}(a|s) + \nabla_{\boldsymbol{\mu}}^{\top}\nabla_{\boldsymbol{\mu}} \log d_{\boldsymbol{\mu}}^{s}(s)\Big) \,\mathrm{d}s\,\mathrm{d}a.
\end{aligned}
$$

5.2.6 Initialization of the Follower's Policy

After having defined the core of the gradient-based algorithm, we must tackle a minor but important aspect: the initialization of the follower's policy. Since the gradient-based algorithms rely on local information of the objective function's curvature to perform the optimization, the starting point can greatly influence the final result, whenever the objective function is non-convex like in our case. One possible solution to mitigate this problem is to repeat the process for multiple random initializations to keep the best result. Additionally, it is possible to devise some initialization heuristics that may help guide the algorithm to more favorable regions. In particular, we report in the following two of these initialization tactics that proved effective in practice.

Uniform Initialization

The first heuristic we devised is to initialize the follower to a uniform policy. The idea behind such initialization is to position the follower in a neutral starting point, enabling it to choose how to move without relying too much on its own action distribution.

Opposite Initialization

The second heuristic is orthogonal to the previous one, as the follower is initialized to a policy that chooses with high probability actions that are *not* chosen by the leaders. The idea is trying to push the follower away from the coalition in the policy space, in an attempt to reach a distant region also in the distribution space.

5.2.7 Entropy Regularization

As we have seen in Section 5.1.4, the value of the objective function $f(\boldsymbol{\theta}, \boldsymbol{\mu})$ might be higher when considering stochastic policies instead of just deterministic policies. However, since we are not interested in covering stochastic policies, we aim to incentivize the gradient-based algorithm to output deterministic policies, by adding to the objective function a penalization term for stochastic policies. A common way to measure how much a policy is stochastic is the *entropy* function [51]. While it is common to include an entropy incentive in RL [52, 53], in our case we want to *penalize* the policy entropy. Since the more stochastic a policy the greater its entropy, we will *subtract* the gradient of the policy entropy from the objective function.

5.3 Relaxed Linear Program Formulation

By employing a relaxed linear program formulation of the best response problem, it is possible to efficiently compute a surrogate solution. Since we will present such formalization as a means to evaluate the coverage obtained by the leader, we will delay the discussion of this alternative to Section 7.1.

Chapter 6

Leader Optimization

In this chapter, we will devote our attention to the leader's optimization scheme. In Section 6.1, we will define formally the problem and provide a visualization tool that will be used throughout the discussion. In Section 6.2, we will explain how to coordinate the leader's coalition. In Section 6.3, we will present a gradient-based algorithm to move the components closer to the best response in the policy space. In Section 6.4, we will illustrate how to detect that a new component should be added to the coalition, and how to properly initialize it.

6.1 Problem Formulation

Let us report here, for the reader's convenience, the problem formulation outlined in Section 4.1, in which we denote explicitly the best response that we have properly defined in the previous chapter. The leader aims to solve the problem

$$\min_{\boldsymbol{\theta} \in \Theta^K} f(\boldsymbol{\theta}, \boldsymbol{\mu} = \textit{best response}(\boldsymbol{\theta})),$$

where f is our usual objective function defined as

$$f(\boldsymbol{\theta}, \boldsymbol{\mu}) := \min_{k \in [K]} D_\alpha \big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha - 1}.$$

In order to be able to perform the optimization on the leader's side, we have to face three main tasks. First, we need to define how to *coordinate* the components of the leader coalition. Arguably, they have to operate as a single entity, rather than a disconnected group of individuals. As we will see, the problem of the coordination of the components can be reduced to the problem of choosing, at each iteration, which component to move in the direction of the best response. Secondly, having selected a component to move, we need to define *how to move* it. We will see a gradient-based algorithm to do so and some critical aspects of the optimization. Lastly, we need to define *how to detect* that the current members of the coalition are not sufficient to reach the desired coverage threshold, and how to initialize the newly added component to the coalition.

6.1.1 Visualizing the Problem

In order to ease the subsequent discussion, we would like to present some visual examples based on real problem configurations. Unfortunately, since the policies lie in a high-dimensional space, providing illustrative visualizations is quite challenging. Thus, for the sake of clarity, we will present our considerations in a much simpler, two-dimensional, Euclidean version of the problem at hand. In the following, we will represent the distribution space with a two-dimensional triangle, the leader components with red circles, the best response with a yellow star, and the Rényi divergence between them (i.e., the objective function) with a green line.

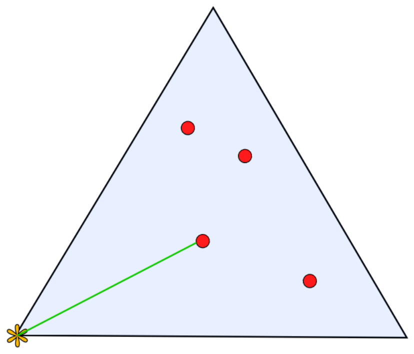


Figure 6.1: Four leaders and the corresponding best response.

Note that with the blue triangle we are representing the *distribution* space and not the *policy* space, as the objective function is computed between the state-action distributions of the policies, not the policies themselves. Recall that because of the complex, problem-dependent relation between the two, the *distance between policies is not necessarily representative of the distance between distributions.* For example, changing the policy in its less-visited states has little to no influence on the induced distribution. To give the reader a visual intuition, we represent here the policy space with a green triangle and we draw a black line to connect a policy to its corresponding induced distribution. As it is easy to see, we both have very distant policies that elicit the same behaviour and vice versa.

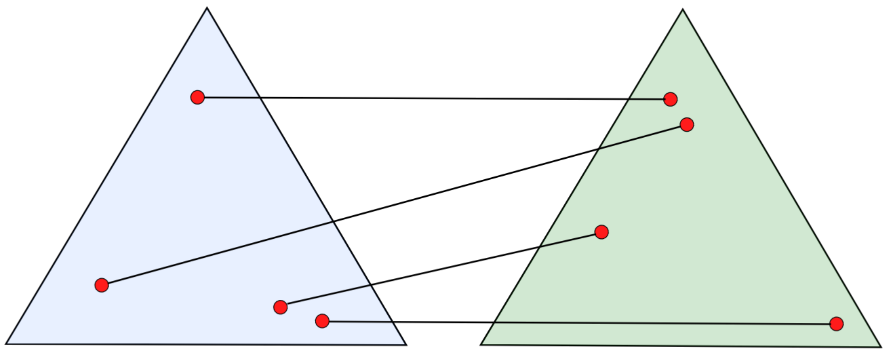


Figure 6.2: Distribution space (blue) vs policy space (green).

Since all the following considerations and criteria are based on the Rényi distance computed between induced state-action distributions, with a little abuse of notation we will use the term *leader* and *best response* to denote their respective state-action distribution (i.e., "the distance between the leader and the best response" translates to "the distance between the state-action distributions induced by the leader and the best response"). With all the needed definitions in place, we are ready to start tackling the first issue of the leader's optimization.

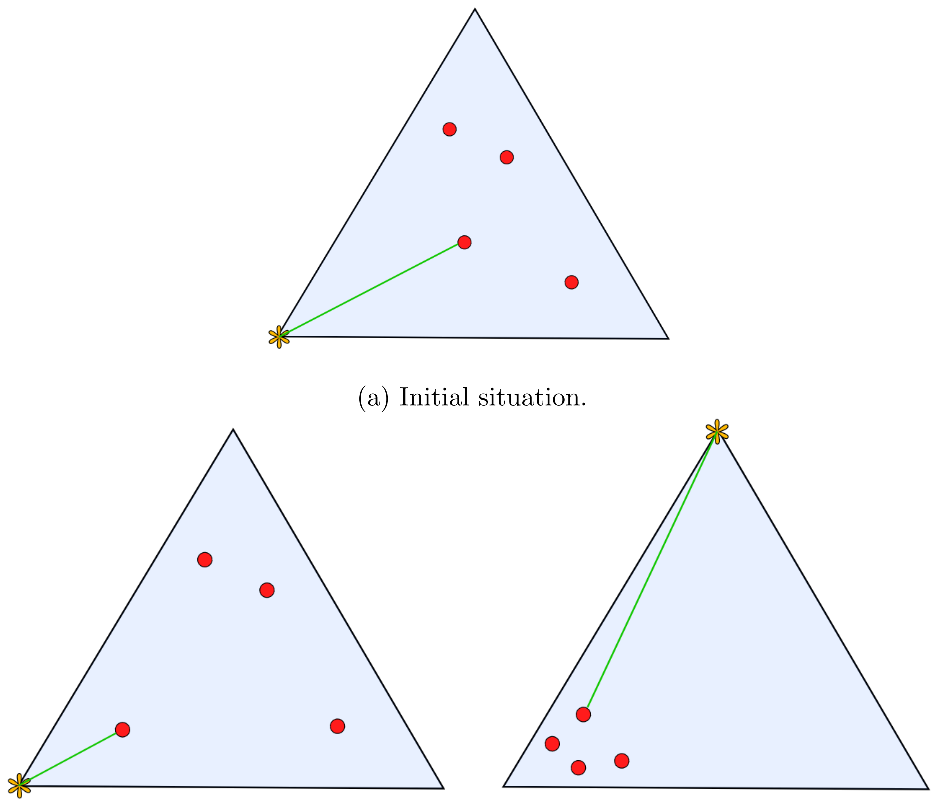
6.2 Coalition Coordination

The first problem we need to solve is deciding how to coordinate the leader's coalition. It is not trivial to define the behavior of a group of individual elements in such a way that they behave consistently with respect to a social objective. In this work, we solved the coordination problem by moving the components of the leader *one at a time*. As we will see in the following, by doing so we avoid making unnecessary (or even detrimental) modifications and we keep the leader's optimization as stable as possible.

6.2.1 Moving the Closest Component

The most immediate way for the leader to reduce the objective function f is to reduce the distance between the *closest* of its components and the best response. This fact follows directly from the definition of f, as every component that is not the minimizer with respect to k does not affect the objective function. While the selected component moves, it is wise to keep the other components of the leader still. Moving all the components at once towards the current best response would indeed leave unguarded previously covered regions of the distribution space, likely favoring the follower at the next iteration.

It is easy to convince oneself of the former by observing Figure 6.3: notice how myopically moving all components simultaneously significantly worsens the leader's covering capabilities.



(a) Initial situation.

(b) Moving only the closest component.

(c) Moving all components.

Figure 6.3: Moving one vs all components comparison.

6.2.2 Moving the Least Effective Component

Another strategy to minimize the objective function is to move the *least effective* component of the leader's coalition. Intuitively, since the goal of the leader is to cover the policy space with the minimum number of components, the distributions covered by each element of the coalition should not overlap. Therefore, we can identify the *least effective* element in the coalition as the element that is *closest* with respect to all the other components, as such element does not contribute much to the coverage objective. Recall that, differently from the Euclidean distance, the Rényi divergence is not symmetric. Therefore, the closest component with respect to the others in the coalition will almost always be only one element, not a pair. In Picture 6.4 we denote the least effective leader with a violet circle.

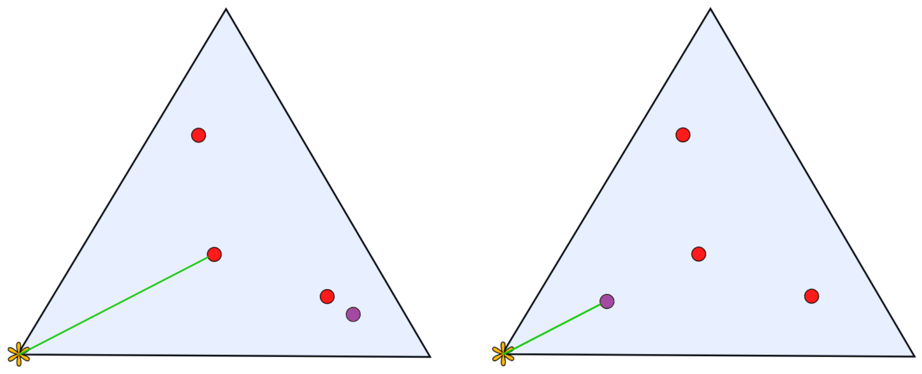

(a) Initial situation. (b) Final situation.

Figure 6.4: Moving the least effective component.

6.2.3 Comparison

What are the differences between the two strategies we outlined and what are the advantages and the disadvantages of using one with respect to the other?

Starting from the first strategy, moving the closest component is arguably the fastest and more direct way to reduce the value of the objective function. However, evolving the coalition according to this criterion may be inefficient with respect to the utilization of the components. As a matter of fact, we are not guaranteed to have a small overlapping between the components: the risk is to incur in a sub-optimal coverage of the distribution space.

On the other hand, moving the least effective component directly incentivizes the leader to make the best out of its coalition, by indeed avoiding unnecessary overlaps. Unfortunately, as the least effective component might be very distant from the current best response, it could take a long series of steps to cover it, therefore increasing the overall computational load. Moreover, individuating such component requires the computation of all the pair-wise divergences. It may be an interesting stimulus for future work to design even better coordination procedures for the solution of this problem, by ideating new strategies and/or combining the current ones.

6.3 Moving the Leader's Components

Having established a criterion to select which leader's components to move, we have to define how to perform such movement in the policy space. In the following, we will present the gradient-based algorithm that we use to do so and we will confront two possible ways of implementing it.

6.3.1 Policy Gradient Derivations

Similarly as in Section 5.2.3, we can compute the gradient of the Rényi divergence with respect to the parameters of one of the leader components $\boldsymbol{\theta}_k$

$$\nabla_{\boldsymbol{\theta}_k} D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1} = \frac{\partial D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}}{\partial \boldsymbol{\theta}_k} = \frac{\partial D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}}{\partial d^{sa}_{\boldsymbol{\theta}_k}} \frac{\partial d^{sa}_{\boldsymbol{\theta}_k}}{\partial \boldsymbol{\theta}_k}.$$

The derivative of the Rényi divergence with respect to its second argument is easy to obtain

$$\frac{\partial D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1}}{\partial d^{sa}_{\boldsymbol{\theta}_k}} = (1-\alpha) \int_{\mathcal{SA}} \left(\frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{d^{sa}_{\boldsymbol{\theta}_k}(s,a)} \right)^\alpha \mathrm{d}s \, \mathrm{d}a,$$

and for the second term we can apply the same derivations we did before

$$\begin{aligned} \frac{\partial d^{sa}_{\boldsymbol{\theta}_k}(s,a)}{\partial \boldsymbol{\theta}_k} &= d^{sa}_{\boldsymbol{\theta}_k}(s,a) \, \nabla_{\boldsymbol{\theta}_k} \log d^{sa}_{\boldsymbol{\theta}_k}(s,a) \\ &= d^{sa}_{\boldsymbol{\theta}_k}(s,a) \Big(\nabla_{\boldsymbol{\theta}_k} \log \pi_{\boldsymbol{\theta}_k}(a|s) + \nabla_{\boldsymbol{\theta}_k} \log d^{s}_{\boldsymbol{\theta}_k}(s) \Big). \end{aligned}$$

Chaining the two terms we obtain the full expression of the gradient

$$\begin{aligned} &\nabla_{\boldsymbol{\theta}_k} D_\alpha(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})^{\alpha-1} = \\ &= (1-\alpha) \int_{\mathcal{SA}} d^{sa}_{\boldsymbol{\theta}_k}(s,a) \left(\frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{d^{sa}_{\boldsymbol{\theta}_k}(s,a)} \right)^\alpha \Big(\nabla_{\boldsymbol{\theta}_k} \log \pi_{\boldsymbol{\theta}_k}(a|s) + \nabla_{\boldsymbol{\theta}_k} \log d^{s}_{\boldsymbol{\theta}_k}(s) \Big) \mathrm{d}s \, \mathrm{d}a, \end{aligned}$$

where the last term $\nabla_{\boldsymbol{\theta}_k} \log d^{s}_{\boldsymbol{\theta}_k}(s)$ is derived from [46, 47] as

$$\nabla_{\boldsymbol{\theta}_k} \log d^{s}_{\boldsymbol{\theta}_k}(s) = \int_{\mathcal{SA}} \gamma Q^0_{\boldsymbol{\theta}_k}(\overline{s}, \overline{a}|s) \Big(\nabla_{\boldsymbol{\theta}_k} \log d^{s}_{\boldsymbol{\theta}_k}(\overline{s}) + \nabla_{\boldsymbol{\theta}_k} \log \pi_{\boldsymbol{\theta}_k}(\overline{a}|\overline{s}) \Big) \mathrm{d}\overline{s} \, \mathrm{d}\overline{a}.$$

6.3.2 Update Methodology

Now that we have the expression of the gradient, we can move any component of the leader in the direction of the best response. We will discuss in the following *how* to perform such an update. In particular, we will see that we can both apply the vanilla ∞-GDA procedure (Section 2.4.5), and a modified version of the same process based on the specific properties of the objective function we are facing.

Computing the Best Response at each Iteration

The most natural way of finding a Stackelberg equilibrium is to apply the ∞-GDA algorithm. At each optimization step, we compute the best response for the current leader configuration and use the local information of the gradient to update the coalition accordingly. By doing so, we are guaranteed to converge to a differential Stackelberg equilibrium [20], which represents a locally optimal solution to the game. While this procedure is theoretically sound and it can be applied to any two-player game, it is not taking into consideration any special property of our specific game-theoretic formulation. As it turns out, computing the best response at each optimization step is quite expensive and, more importantly, not strictly necessary in the perspective of the covering problem. In the following, we will present a slight modification of the standard ∞-GDA procedure aimed at reducing the computational complexity of the overall algorithm, which proved very effective in practice.

Computing the Best Response when Covered

Consider the situation presented in Figure 6.5 where, along with the already defined visual conventions, we added a circle of radius ϵ to all the leader components.

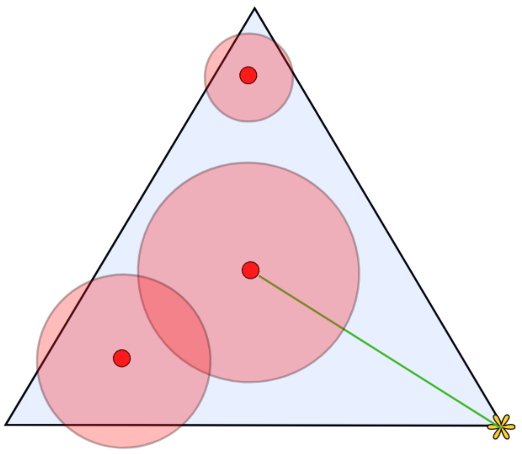


Figure 6.5: Distributions and corresponding ϵ-circles.

For each component, the intersection of its circle with the triangle represents the set of distributions that are covered by it. The reader may now find him/herself very puzzled by the dimensions of the circles reported in the figure, as they appear to shrink the closer they get to the border of the triangle. This counter-intuitive fact derives from the peculiar nature of the Rényi divergence. Since the distribution of the covering policy is at the denominator of the integrand function, it is easy to see that the more a distribution is peaked (i.e., closer to the border of the distribution space), the smaller the set of numerators that will keep the integral lower than epsilon. At the limit, a deterministic distribution is *infinitely* distant from all other distributions except itself. Thus, the closer is the distribution to a deterministic one, the smaller is its ϵ-circle in the distribution space. In fact, the ϵ-circle centered on the best response collapses in only one point and is not visible in the picture.

Let us continue our reasoning by visualizing what happens at each optimization step of our ∞-GDA procedure. As the reader could realize from Figure 6.6, *moving one leader component towards the best response does not guarantee that the best response will change at the next iteration*, as the leader update is usually small. In this case, it would have not been necessary to perform again the follower's optimization.

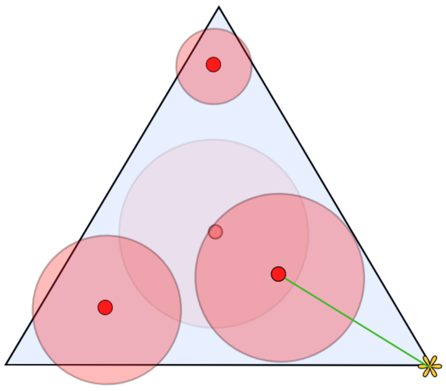


Figure 6.6: The best response did not change after one optimization step.

Moreover, even if the best response changed, since the goal of the leader is to ultimately cover the policy space, one could argue that until the distance from the old best response is less than ϵ, we could decide to temporarily ignore the fact that the follower's solution changed, and continue the optimization until the previous one is covered. Once it is done, we will compute again the best response and the optimization will go on with the new follower's challenge, as shown in Figure 6.7.

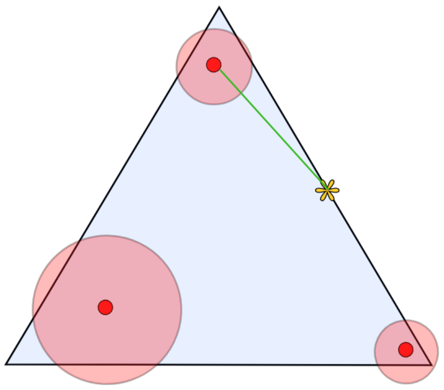


Figure 6.7: The old best response is covered.

By performing the optimization in this way, we avoid redundant computations of the best response and greatly reduce the complexity of the algorithm. Even though we lose the theoretical guarantees of converging to a local Stackelberg, it is not for us a matter of utmost importance, since we are interested in finding a good set of covering policies, not a game-theoretical equilibrium.

6.4 Adding a Component to the Coalition

The third and last task that remains for us to untangle in order to complete the discussion on the leader's optimization is defining how to detect that the current number of leaders is not sufficient to reach the desired coverage and we, therefore, need to *add a new component* to the coalition. Especially, the condition changes according to how we have chosen to perform the update (either the vanilla GDA or our modified version).

6.4.1 Computing the Best Response at each Iteration

Although more computationally demanding, computing the best response at each optimization step has the effect of stabilizing the leader's optimization. By renewing the follower's challenge at each step, we are guaranteed to converge to a local Stackelberg equilibrium, and therefore we avoid the risk of endlessly wandering around in the policy space with no hope of converging. Because of this, deciding when it is time to add another component to the coalition is a trivial decision: wait for the leader to converge and check if it has reached the desired coverage. We will see how to do so in the next chapter.

6.4.2 Computing the Best Response when Covered

If we want to apply the second optimization schema to speed up the leader's optimization, we are not guaranteed to converge to a stationary point during the optimization. In order to avoid aimlessly update the coalition when it has no chance of converging, we will have to come up with a criterion to detect such situations to promptly stop the optimization and to add a member to the coalition.

One option is to make a decision on the basis of the *distance between the leader and the previous best response.* If, after moving the components of the leader to cover the best response, the previous policy played by the follower is now very distant from the coalition, probably the coalition is under-sized. This happens because if the leader now moved again to cover the *previous* best response, most likely it would lose the coverage on the *current* one. Clearly, deciding how to select the threshold distance for which the criterion triggers is a hyper-parameter to tune. The general idea is that the bigger the threshold, the more the current coalition will try to do its best with the current number of components, potentially at the cost of computational efficiency.

6.4.3 New Component Initialization

Finally, let us complete the discussion on the leader's optimization with *how to initialize* a new component added to the coalition. The new members should strive to fit well in the current group of policies, compensating for the current shortcomings of the coalition.

Epsilon-Greedy Initialization

One possible way to do so is to initialize the newcomer to an *epsilon-greedy* version of the last best response. Since the best response is by definition the most distant policy from the coalition, adding a component close to it is an effective way to cover a new region of the distribution space.

Uniform Initialization

Another option is to initialize the leader with a uniform policy so that the algorithm will freely deploy it where it is most useful. Note that this second criterion is better suited if the leader moves its components according to the *least effective* criterion, since a uniformly distributed policy is very unlikely to be the *closest* to the best response.

Chapter 7

Coverage Guarantees

In the previous chapters, we defined all the main components of the proposed solution for the policy space compression problem. Although it would be possible to implement the algorithm from start to end with the elements outlined so far, one crucial element is still missing from the picture: the theoretical guarantees on the obtained results. Recall from Chapter 5 that the problem of computing the follower's best response has exponential computational complexity. As a consequence, we are unable to find the exact solution for big MDPs, and therefore to effectively evaluate the output of the algorithm at the end of the optimization. For this reason, it is crucial to have an *estimate* of the coverage obtained by the leader without having to find the best response. Such estimation must be *conservative*, so that to never overestimate the quality of the solution. Moreover, we would like the estimation to be *consistent* with the Rényi divergence, which means that the value of the estimator should approach the minimum of the Rényi divergence as we increase the number of policies in the leader's coalition towards infinity. In this chapter, we are going to fill this last void by providing three alternatives for computing such estimation efficiently.

In Section 7.1, we will present a surrogate best response problem that we can use to build the coverage guarantee we need. In Section 7.2, we will present an alternative upper bound based on the relaxed solution of a quadratic formulation of the best response problem. In Section 7.3, we will present a consistent, but difficult to optimize, coverage guarantee based on the Kullback-Leibler divergence.

7.1 Linear Program Formulation

We will start by tackling the best response problem from a different angle. As we will see, it is possible to devise a collection of reward functions, one for each leader's component, that we can use to compute a surrogate best response *and* a conservative estimate of the coverage obtained by the leader.

Let us start the discussion by reporting once again the best response problem

$$\boldsymbol{\mu}_{br} \in \operatorname*{arg\,max}_{\boldsymbol{\mu}\in\Theta} \big(\min_{k\in[K]} D_\alpha\big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha-1}\big).$$

We can rewrite the former as

$$\boldsymbol{\mu}_{br} \in \operatorname*{arg\,max}_{\boldsymbol{\mu}\in\Theta} \big(J_{BR}(\boldsymbol{\mu})\big),$$

where the performance metric J is defined as

$$J_{BR}(\boldsymbol{\mu}) := \min_{k\in[K]} D_\alpha\big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha-1}.$$

7.1.1 Single Leader Case

Let us begin by considering a leader's coalition with a single component. We can rearrange the performance metric J as

$$\begin{aligned}
J_{BR}(\boldsymbol{\mu}) &= \min_{k\in[K]} D_\alpha\big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}\big)^{\alpha-1} \\
&\Big\Downarrow \; K = 1 \\
J_{RL}(\boldsymbol{\mu}) &= D_\alpha\big(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}}\big)^{\alpha-1} \\
&= \int_{\mathcal{SA}} d^{sa}_{\boldsymbol{\mu}}(s,a) \left(\frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{d^{sa}_{\boldsymbol{\theta}}(s,a)}\right)^{\alpha-1} \mathrm{d}s\,\mathrm{d}a. \\
&= \int_{\mathcal{SA}} d^{sa}_{\boldsymbol{\mu}}(s,a)\, R(s,a)\,\mathrm{d}s\,\mathrm{d}a.
\end{aligned}$$

It is clear from the previous equation that, in order to find the best response, the follower aims to find a policy $\boldsymbol{\mu}$ that maximizes the discounted sum of the rewards

$$R(s,a) = \left(\frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{d^{sa}_{\boldsymbol{\theta}}(s,a)}\right)^{\alpha-1}.$$

Unfortunately, such a reward function cannot be optimized by standard reinforcement learning techniques, as it depends on the policy parameters $\boldsymbol{\mu}$. Therefore, in order to find an approximated solution to the follower's problem, we could look for a surrogate reward function $\overline{R}$ independent of $\boldsymbol{\mu}$ (i.e., that we can optimize), such that $J(\overline{R}, \boldsymbol{\mu})$ is a *lower bound* of the performance $J(R, \boldsymbol{\mu})$. In this way, by maximizing the performance with respect to the surrogate reward, we are implicitly lifting the performance with respect to the original reward. Moreover, if we can derive an *upper bound* of $J(R, \boldsymbol{\mu})$ with respect to $J(\overline{R}, \boldsymbol{\mu})$, we can use the latter to conservatively and efficiently estimate the distance between the leader and the true best response (i.e., the coverage).

7.1.2 Surrogate Reward

From now on, we will restrict our analysis to the second order Rényi divergence. Especially, starting from the reward $R(s,a) = (d^{sa}_{\boldsymbol{\mu}}(s,a)/d^{sa}_{\boldsymbol{\theta}}(s,a))^{\alpha-1}$ and assuming $\alpha = 2$, we can define the surrogate reward function $\overline{R}(s,a)$ as

$$R(s,a) = \frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{d^{sa}_{\boldsymbol{\theta}}(s,a)} \quad \Rightarrow \quad \overline{R}(s,a) := \frac{1}{\sqrt{d^{sa}_{\boldsymbol{\theta}}(s,a)}}.$$

We will show in the following the derivations of both an upper and a lower bound of $J(R,\boldsymbol{\mu})$ with respect to $J(\overline{R},\boldsymbol{\mu})$. Note that, since the state-action distributions of the leader $d^{sa}_{\boldsymbol{\theta}}$ is enforced to be positive in all the state-action pairs, $\overline{R}(s,a)$ is always finite.

Derivation of the Bounds

By exploiting common bounds on the L^p-norms (see [54]), we can derive the following lower bound

$$\begin{gathered} \int_{\mathcal{SA}} \frac{\left(d^{sa}_{\boldsymbol{\mu}}(s,a)\right)^2}{d^{sa}_{\boldsymbol{\theta}}(s,a)} \,\mathrm{d}s\,\mathrm{d}a \geq \frac{1}{|\mathcal{SA}|}\left(\int_{\mathcal{SA}} \frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{\sqrt{d^{sa}_{\boldsymbol{\theta}}(s,a)}} \,\mathrm{d}s\,\mathrm{d}a\right)^2, \\ J(R,\boldsymbol{\mu}) \geq \frac{\left(J(\overline{R},\boldsymbol{\mu})\right)^2}{|\mathcal{SA}|}, \end{gathered} \tag{7.1}$$

and the following upper bound

$$\begin{gathered} \left(\int_{\mathcal{SA}} \frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{\sqrt{d^{sa}_{\boldsymbol{\theta}}(s,a)}} \,\mathrm{d}s\,\mathrm{d}a\right)^2 \geq \int_{\mathcal{SA}} \frac{\left(d^{sa}_{\boldsymbol{\mu}}(s,a)\right)^2}{d^{sa}_{\boldsymbol{\theta}}(s,a)} \,\mathrm{d}s\,\mathrm{d}a, \\ \left(J(\overline{R},\boldsymbol{\mu})\right)^2 \geq J(R,\boldsymbol{\mu}). \end{gathered} \tag{7.2}$$

Putting everything together, we have

$$\left(J(\overline{R},\boldsymbol{\mu})\right)^2 \geq J(R,\boldsymbol{\mu}) \geq \frac{\left(J(\overline{R},\boldsymbol{\mu})\right)^2}{|\mathcal{SA}|}. \tag{7.3}$$

With the last inequality, we can say that for the single leader case, maximizing the discounted cumulative sum of the reward $\overline{R}(s,a)$ is a viable way to upper bound the coverage obtained by the leader and, additionally, to compute a surrogate best response.

7.1.3 Multiple Leader Case

Generalizing the single leader case to the multiple leader scenario is not a trivial feat. We have a collection of surrogate rewards $\overline{\mathcal{R}} = \{\overline{R}_k\}_{k\in[K]}$ to optimize instead of a single one and, at first glance, there is no direct way of performing the maximization by considering all the rewards at once. Our aim is to find a policy $\boldsymbol{\mu}$ that *maximizes the minimum performance* over all the rewards in $\overline{\mathcal{R}}$. Notably, some works in the *robust* MDPs [55] framework can help us solve the problem. In particular, taking inspiration from [56], we can rewrite the *minimax performance* problem with the following linear program

$$\begin{aligned}
\underset{z\in\mathbb{R},\,\boldsymbol{\mu}\in\mathbb{R}^{sa}}{\text{maximize}} \quad & z \\
\text{subject to:} \quad & z \le \int_{\mathcal{SA}} d(s,a)\overline{R}_k(s,a)\,\mathrm{d}s\,\mathrm{d}a, \quad \forall k \in [K] \\
& \int_{\mathcal{A}} d(s,a)\,\mathrm{d}a = (1-\gamma)d_0(s) \\
& \qquad\qquad + \gamma \int_{\mathcal{SA}} d(s',a')P(s|s',a')\,\mathrm{d}s'\,\mathrm{d}a', \quad \forall s \in \mathcal{S} \\
& d(s,a) \ge 0, \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}.
\end{aligned}$$

Moreover, by applying the upper and lower bounds in Equation (7.3), we can state that

$$z^* = \max_{\boldsymbol{\mu}\in\Theta}\min_{k\in[K]} \big(J(\overline{R}_k,\boldsymbol{\mu})\big)^2 \ge \max_{\boldsymbol{\mu}\in\Theta}\min_{k\in[K]} J(R,\boldsymbol{\mu}) \ge \max_{\boldsymbol{\mu}\in\Theta}\min_{k\in[K]} \frac{\big(J(\overline{R},\boldsymbol{\mu})\big)^2}{|\mathcal{SA}|}.$$

The computational complexity of finding the solution of any linear program is polynomial in the number of decision variables [57]. Therefore, we can efficiently solve the previous problem and use the optimal value z^* as an upper bound to the leader's coverage, and the resulting policy $\boldsymbol{\mu}^*$ as a surrogate best response. Let us now analyze the estimation's *consistency.* The optimal objective of the LP is

$$z^* = \max_{\boldsymbol{\mu}\in\Theta}\min_{k\in[K]} \big(J(\overline{R}_k,\boldsymbol{\mu})\big)^2 = \max_{\boldsymbol{\mu}\in\Theta}\min_{k\in[K]} \left(\int_{\mathcal{SA}} \frac{d^{sa}_{\boldsymbol{\mu}}(s,a)}{\sqrt{d^{sa}_{\boldsymbol{\theta}_k}(s,a)}}\,\mathrm{d}s\,\mathrm{d}a\right)^2.$$

With infinite leaders, for each possible policy $\boldsymbol{\mu}$ there exists at least a leader $\boldsymbol{\theta}_k$ that is equal to it. Therefore, we can simplify the previous as

$$z^* = \max_{\boldsymbol{\mu}\in\Theta} \left(\int_{\mathcal{SA}} \sqrt{d^{sa}_{\boldsymbol{\mu}}(s,a)}\,\mathrm{d}s\,\mathrm{d}a\right)^2, \tag{7.4}$$

which it is bounded by

$$|\mathcal{SA}| \ge z^* \ge 1.$$

Unfortunately, the previous upper bound is consistent with the Rényi divergence only when the MDP allows for a deterministic $d^{sa}_{\boldsymbol{\mu}}$, which is almost never the case. Therefore, if the coverage requirement is stricter than $|\mathcal{SA}|$, we are not in general able to assess the coverage obtained by the algorithm through this upper bound.

7.2 Quadratic Program Formulation

Notably, we can also frame the best response problem as a Quadratically Constrained Quadratic Program (QCQP), namely

$$\begin{aligned}
\underset{z\in\mathbb{R}, d\in\mathbb{R}^{sa}}{\text{maximize}} \quad & z \\
\text{subject to:} \quad & z \leq \int_{\mathcal{SA}} \frac{d(s,a)^2}{d^{sa}_{\boldsymbol{\theta}_k}(s,a)} \,\mathrm{d}s\,\mathrm{d}a, \quad \forall k \in [K] \\
& \int_{\mathcal{A}} d(s,a)\,\mathrm{d}a = (1-\gamma)d_0(s) \\
& \qquad\qquad + \gamma \int_{\mathcal{SA}} d(s',a')P(s|s',a')\,\mathrm{d}s'\,\mathrm{d}a', \quad \forall s \in \mathcal{S} \\
& d(s,a) \geq 0, \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}.
\end{aligned}$$

Unfortunately, due to the quadratic constraints, this problem is known to be NP-hard in general. However, it can be relaxed to a linear program through the *reformulation-linearization technique*, or into a semidefinite program through the *semidefinite relaxation*, for which the optimal values are an *upper bound* to the value of the original problem [58, 59, 60]. Unfortunately, such relaxation worsens as the number of leaders K increases. Indeed, the more quadratic constraints we remove from the problem, the more the solution will be distant from the original one. Thus, the estimation based on the previous relaxation is *not consistent*.

7.3 Kullback-Leibler Formulation

Luckily, it is possible to overcome the limitations of the two previous bounds and to devise a *conservative* and *consistent* estimator. Especially, given two probability measures p and q on the space $\mathcal{X}$, and being $d_{KL}(p||q)$ their Kullback-Leibler divergence, the following holds

$$\frac{d_{KL}(p \parallel q)}{\inf_{x\in\mathcal{X}} q(x)} + 1 \geq D_2(p \parallel q) \geq \exp\big(d_{KL}(p \parallel q)\big). \tag{7.5}$$

The right-hand side of the equation originates from the definition of Rényi divergence, which is *non-decreasing* in the order α [61]. The left-hand side can be obtained through the following derivations

$$\begin{aligned}
\int_{\mathcal{X}} \frac{p(x)^2}{q(x)}\,\mathrm{d}x - 1 &= \int_{\mathcal{X}} \frac{(p(x)-q(x))^2}{q(x)}\,\mathrm{d}x \leq \frac{1}{\inf_{x\in\mathcal{X}} q(x)} \int_{\mathcal{X}} (p(x)-q(x))^2\,\mathrm{d}x \\
&\leq \frac{\int_{\mathcal{X}} |p(x)-q(x)|\,\mathrm{d}x}{\inf_{x\in\mathcal{X}} q(x)} \sup_{x\in\mathcal{X}} |p(x)-q(x)| \leq \frac{\|p-q\|_1^2}{2\inf_{x\in\mathcal{X}} q(x)} \\
&\leq \frac{d_{KL}(p||q)}{\inf_{x\in\mathcal{X}} q(x)},
\end{aligned}$$

where we employed $\|p - q\|_1 \geq 2 \sup_{x \in \mathcal{X}} |p(x) - q(x)|$ and the Pinsker's inequality, as previously done in [62, 63]. As a consequence of the result in (7.5), we have that

$$\begin{aligned}
&\max_{\boldsymbol{\mu} \in \Theta} \min_{k \in [K]} \left(\frac{d_{KL}(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})}{\inf_{(s,a) \in \mathcal{SA}} d^{sa}_{\boldsymbol{\theta}_k}(s,a)} + 1 \right) \\
&\qquad\qquad \geq \\
&\max_{\boldsymbol{\mu} \in \Theta} \min_{k \in [K]} D_2(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}) \\
&\qquad\qquad \geq \\
&\max_{\boldsymbol{\mu} \in \Theta} \min_{k \in [K]} \exp\left(d_{KL}(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}) \right).
\end{aligned}$$

Moreover, we can further upper bound the first term as

$$\frac{\max_{\boldsymbol{\mu} \in \Theta} \min_{k \in [K]} d_{KL}(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})}{\min_{k \in [K]} \inf_{(s,a) \in \mathcal{SA}} d^{sa}_{\boldsymbol{\theta}_k}(s,a)} + 1 \geq \max_{\boldsymbol{\mu} \in \Theta} \min_{k \in [K]} \left(\frac{d_{KL}(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})}{\inf_{(s,a) \in \mathcal{SA}} d^{sa}_{\boldsymbol{\theta}_k}(s,a)} + 1 \right).$$

The term at the denominator of the left-hand side of the previous equation is constant given a leader coalition, and always greater than zero in discrete MDPs with stochastic leader policies. Since the upper bound and the lower bound on the performance of the best response depend on the same quantity, it follows that by solving the problem

$$z^* = \max_{\boldsymbol{\mu} \in \Theta} \min_{k \in [K]} \left(d_{KL}(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k}) \right), \tag{7.6}$$

we are able to compute a conservative upper bound of the leader's coverage and a surrogate best response. Moreover, the estimator is *consistent* with the Rényi divergence, as the value of z^* approaches zero as the number of leaders grows. Unfortunately, the difficulty of the previous formalization now lies in the optimization. In fact, in order to have the coverage guarantee, we should find the *global* optimum of Problem (7.6), and such a global solution is not trivial to obtain, as there is not a straightforward linear program formalization of the problem. Nonetheless, it is undeniably an interesting direction for future works.

Chapter 8

Experimental Results

In this chapter, we will report a numerical validation of the proposed algorithm and the claims we made in the previous chapters. In Section 8.1, we will illustrate the workings of the basic building blocks of the solution algorithm we presented, namely the follower's and leader's optimization. In Section 8.2, we will showcase the algorithm on the complete problem and report the results obtained.

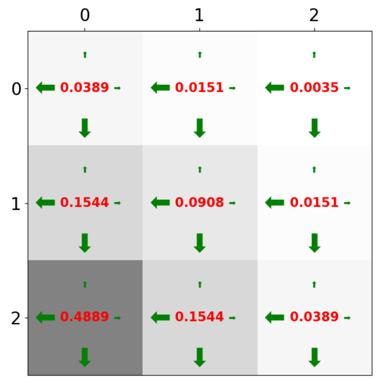
8.1 An Illustrative Domain

In this section, we present an illustrative domain for the policy space compression problem, in order to help the reader visualize the overall process. We conduct our experiments in a *grid world* environment. It represents a discrete, square room having nine discrete positions (i.e., $|\mathcal{S}| = 9$). The agent starts at the center of the room and it can move in the four cardinal directions (i.e., $|\mathcal{A}| = 4$). All the transitions of the environment are deterministic, which means that the agent's actions are always successful and they move the agent in the intended direction. For each leader's component and best response we provide a visualization of its policy and the induced state distribution. We render the former with green arrows (the bigger the arrow, the bigger the probability of choosing that action in that state) and the latter by overlaying a heatmap of the state distribution to the grid, along with a text annotation of the state visitation probability.

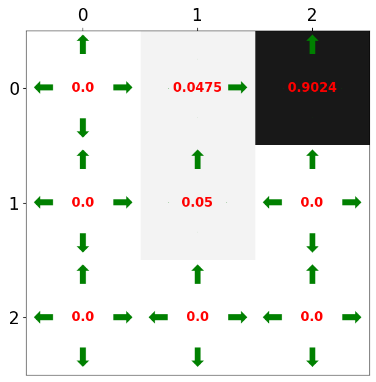
8.1.1 Leader with a Single Component

Let us start with a simple example. In this first case, the coalition of leaders is composed of just one element, as depicted in Figure 8.1a. The follower's solution for this specific single leader configuration (Figure 8.1b) can be guessed just by looking at the state distribution induced by its policy.

Clearly, in this case the leader's coverage is not very effective, as the reader can see from the heatmap of the state distribution it induces. For this first configuration, the follower manages to maximize its objective function $f(\boldsymbol{\theta}, \boldsymbol{\mu}) = D_2(d^{sa}_{\boldsymbol{\mu}} || d^{sa}_{\boldsymbol{\theta}_k})$ up to the value of 2300.



(a) Leader.



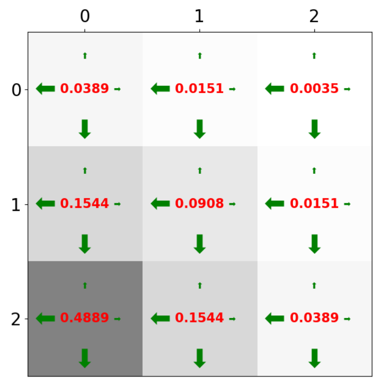
(b) Follower.

Figure 8.1: First leader coalition and its best response.

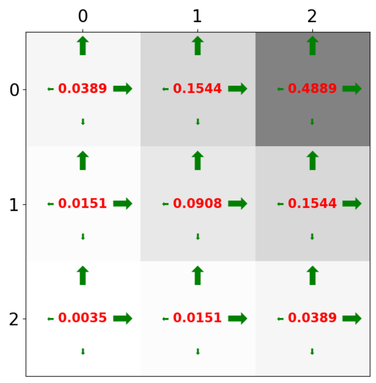
For the gradient counterpart, this first configuration is rather easy. Most random initializations and the simple first-order algorithm are sufficient to reach the global best response.

8.1.2 Leader with Two Components

Let us now add another leader to the coalition to make the follower's problem a bit more challenging. In Figure 8.2, we can see the updated leader coalition and the corresponding best responses in Figure 8.3. As expected, the deterministic follower is able to maximize its objective only up to 53, a lower value than the previous scenario. Since we now have more than one leader in the coalition, we can show the reader also the stochastic best response, which manages to obtain an even higher value for f, in this case 576, which is an order of magnitude larger than the optimal deterministic value.

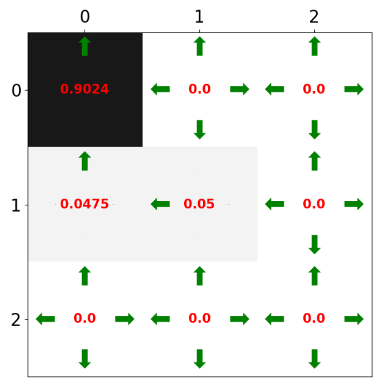


(a) First leader component.

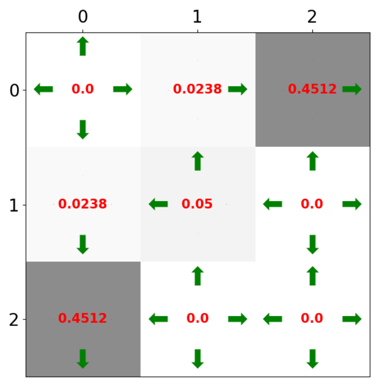


(b) Second leader component.

Figure 8.2: Leader coalition of two elements.



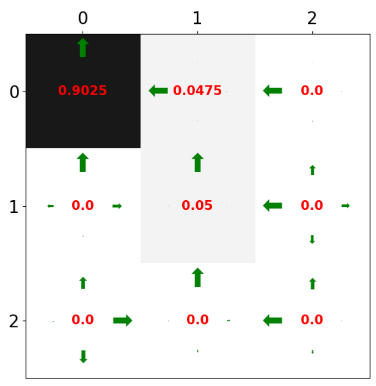
(a) Deterministic best response.



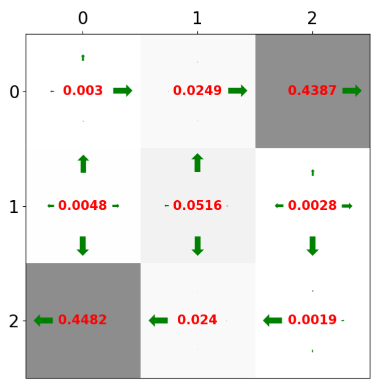
(b) Stochastic best response.

Figure 8.3: Best responses to the second leader coalition.

For what concerns the gradient algorithm, as we have previously mentioned, in the presence of more than one leader's component the follower is unsurprisingly attracted by stochastic policies. The reader can see in Figure 8.4b that the first-order algorithm tends to find policies similar to the stochastic best response in Figure 8.3b. Interestingly, by adding to the gradient the entropy-based incentive (Section 5.2.7) and repeating few random initializations, we are able to find the optimal deterministic policy. Of course, because of the random initialization, the algorithm converges also to policies that are neither the stochastic nor the deterministic best response but are in some middle ground between them, which we omit here for brevity. Although, we stress that it is not crucial for the follower to always converge to the global optimum, as any challenging enough policy will improve the leader's coverage.



(a) Deterministic solution.

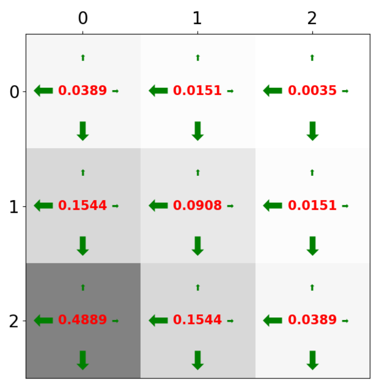


(b) Stochastic solution.

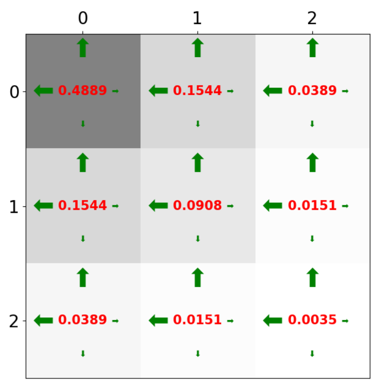
Figure 8.4: Gradient-based best responses.

8.1.3 Very Challenging Leader

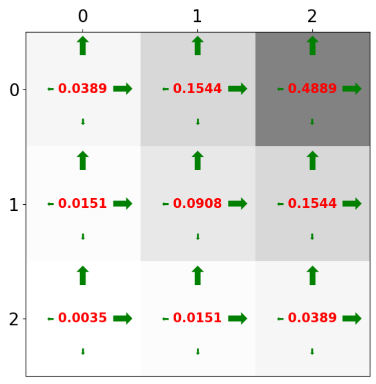
Let us conclude the set of examples for the best response with a very challenging configuration: four completely symmetric covering policies, each one effectively covering one corner of the environment (Figure 8.5). The deterministic follower is now able to maximize its objective only up to 23, while the stochastic policy is able to reach an objective of 160.



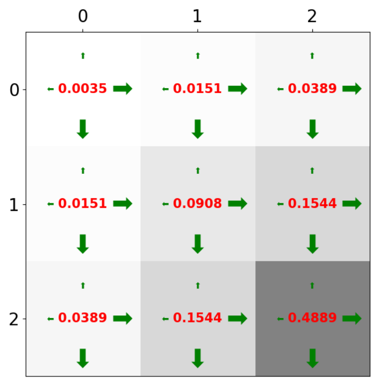
(a) First leader component.



(b) Second leader component.

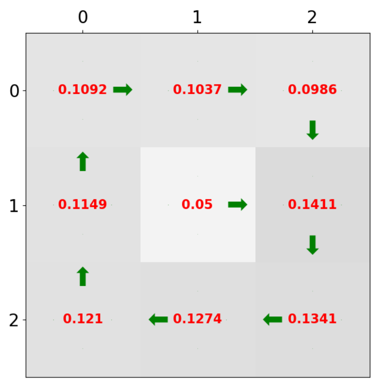


(c) Third leader component.

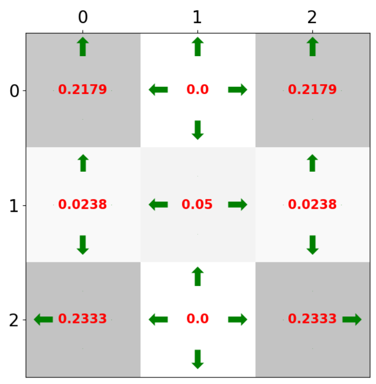


(d) Fourth leader component.

Figure 8.5: Leader coalition of four elements.



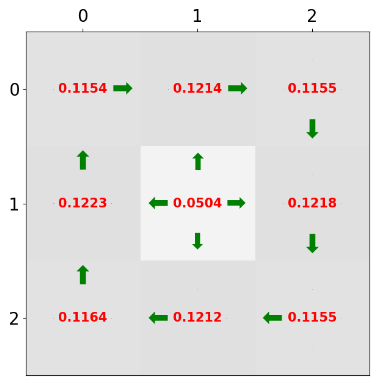
(a) Deterministic best response.



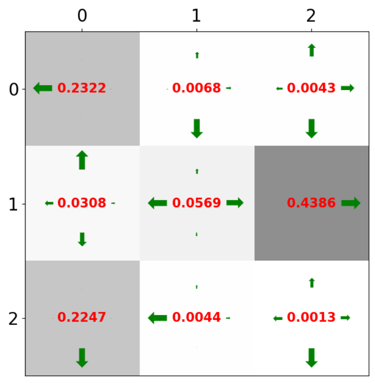
(b) Stochastic best response.

Figure 8.6: Best responses for the third leader configuration.

In this difficult setting the first-order algorithm, even though it finds many challenging stochastic policies (Figure 8.7b), is unable to converge to the deterministic *"Ring Around the Rosie"* policy depicted in Figure 8.6a. Interestingly, if we employ the second-order algorithm, coupled with a strong PT inverse normalization and the uniform initialization heuristic, we are able to converge to the policy in Figure 8.7a, which is very similar to the deterministic optimal one, although still being stochastic in the initial state. This example in particular sheds some light on the difference between first and second-order optimization, highlighting the fact that exploiting the knowledge on the curvature of the objective function enables the algorithm to reach otherwise out-of-reach solutions.



(a) Second-order best response.



(b) First-order best response.

Figure 8.7: Gradient-based best responses for the third leader configuration.

8.1.4 Optimal Single Leader Configuration

In the previous section, we showcased how the optimization of the follower is carried out, by providing three fixed leader coalitions and computing the best response. As we have seen, adding more and more leaders to the coalition, and placing them strategically in the MDP, incrementally lowers the value of the follower's best response, thus improving the leader's coverage. Let us now see what happens if we perform the optimization on the leader's side, starting from a uniformly distributed policy. The result is shown in Figure 8.8. Maybe surprisingly, the coverage obtained by this optimized leader's configuration outperforms even the four symmetric leader scenario, as the value obtained by the follower is even lower (i.e., 20). This example makes evident that hand-picking the covering policies is not easily done, and that the solution found by performing the automated optimization is more effective.

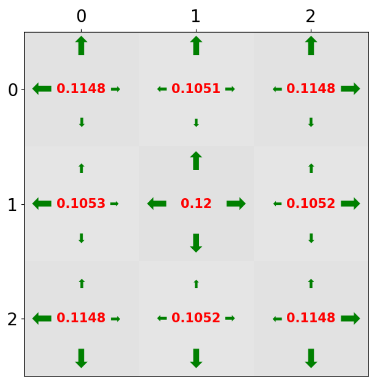


Figure 8.8: Optimal single leader configuration.

8.2 Approaching the Complete Problem

We will now empirically validate our proposed solution (Algorithm 3) of the policy space compression problem from start to end. In the following experiments, we will use an expanded version of the grid world we used in the previous chapter. In particular, we have two rooms composed of nine squares, which are connected by a single doorway. Clearly, to move from one room to the other, the player is forced to avoid the walls and go through the door. The number of states characterizing the environment is $|\mathcal{S}| = 9 + 9 + 1 = 19$, while the action space available to the agent remains the same as before ($|\mathcal{A}| = 4$). The agent starts at the center of the left room. Our goal is to reach the coverage threshold ϵ, which in this case we set at the value of 80. Since $\epsilon \geq |\mathcal{SA}|$, we are guaranteed to reach the desired coverage by using the upper bound presented in Section 7.1. Indeed, as we will see in the following, we can reach our goal with a coalition of four leaders.

8.2.1 Evaluating the Results

Let us start by reporting the progress of the leader coalition. In Figure 8.9, we denoted the value of the upper bound with a blue line, the value of the true distance from the best response (i.e., the coverage) with a red line, and the coverage objective with a green dashed line. As the reader can notice, the upper bound provides a reliable under-estimation of the coverage. Indeed, such estimation might be deemed even too conservative, as the coverage objective is actually achieved by the coalition with only two leaders. In Figure 8.10, we take a closer look at the progression of the optimization when $K = 2$. We can see that, approximately halfway through the 200 iterations allocated, the coalition reaches the coverage objective. As expected, due to the conservative nature of the upper bound, the algorithm is not able to immediately detect such attainment and therefore continues the optimization. However, since our main concern is to reach the coverage objective, the number of policies in the leader's coalition is of secondary importance. Moreover, by changing the hyper-parameters, it is possible to fine-tune the algorithm to obtain superior results. For example, by doubling the number of iterations for each round, we can reach the same objective with only three leader components, as shown in Figure 8.11.

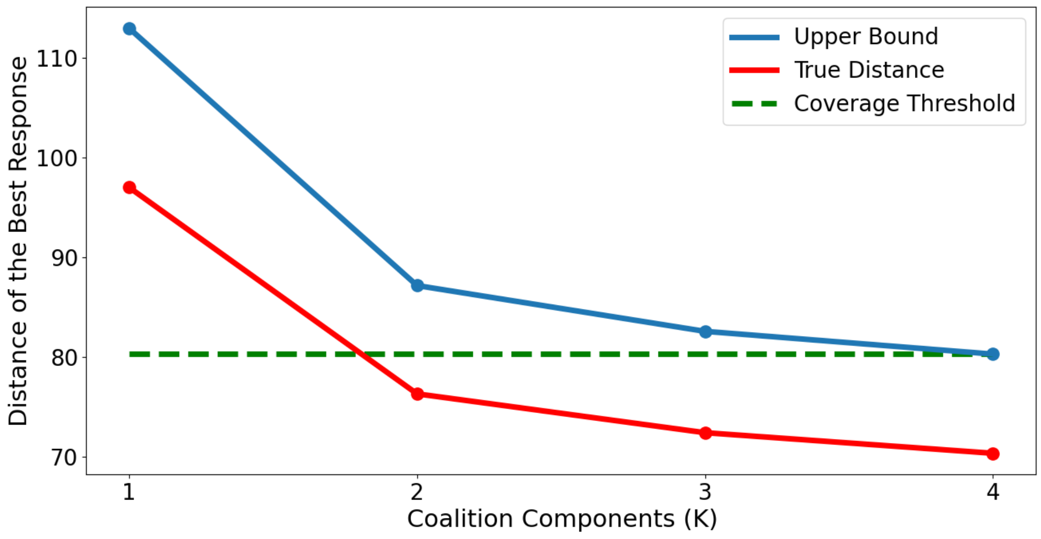


Figure 8.9: Trend of the coverage obtained by the leader's coalition as a function of the number of components.

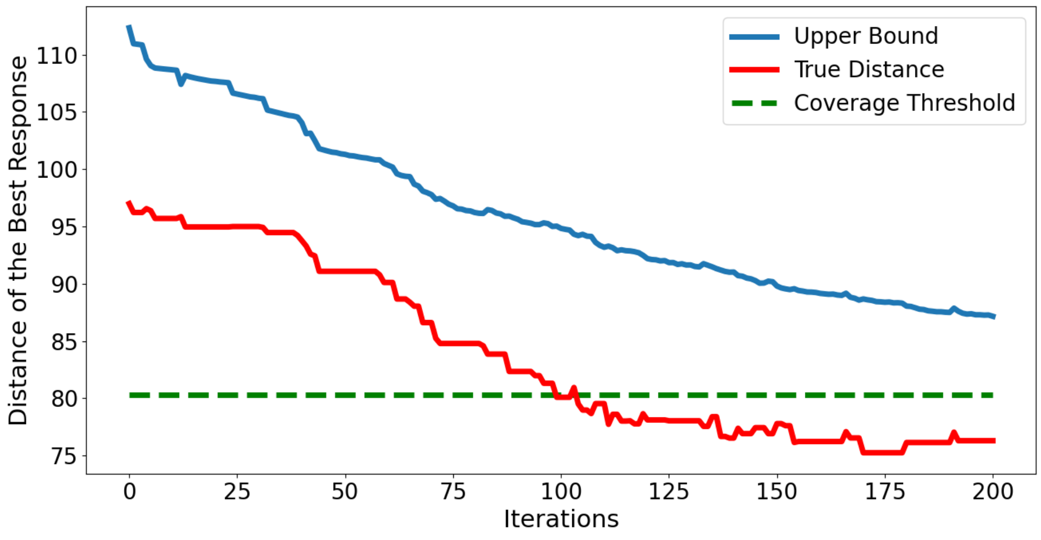


Figure 8.10: Trend of the coverage when the coalition is formed of two components.

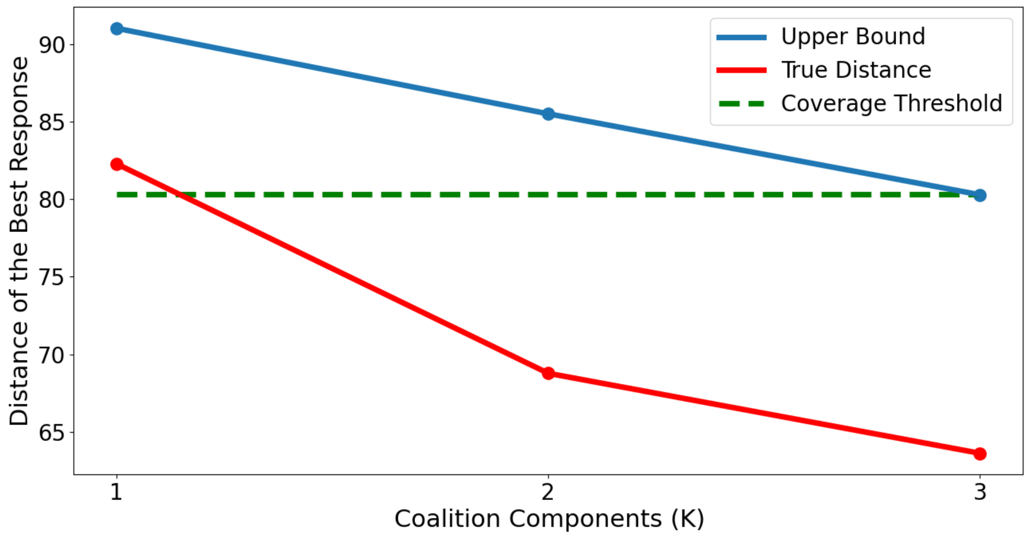


Figure 8.11: Trend of the coalition's coverage by allocating double iterations per round.

8.2.2 Confronting the Limitations of the Upper Bound

Unfortunately, as we anticipated in Section 7.1, the upper bound based on the linear program formalization is not consistent. As a result, if the coverage requirement ϵ is stricter than $|\mathcal{SA}|$, we are not able to evaluate it, even if the algorithm's optimization is able to reach such objective. To visualize this adversity, we report in Figure 8.12, with a purple dashed line, the limit of the upper bound for this specific environment, and we show that the *true* coverage obtained by the leader convincingly outperforms such limitation. For this reason, we strongly feel that going deeper into the consistent, Kullback-Leibler formulation presented in Section 7.3 is an interesting direction for future works, as it would allow reaching an arbitrarily ambitious coverage requirement.

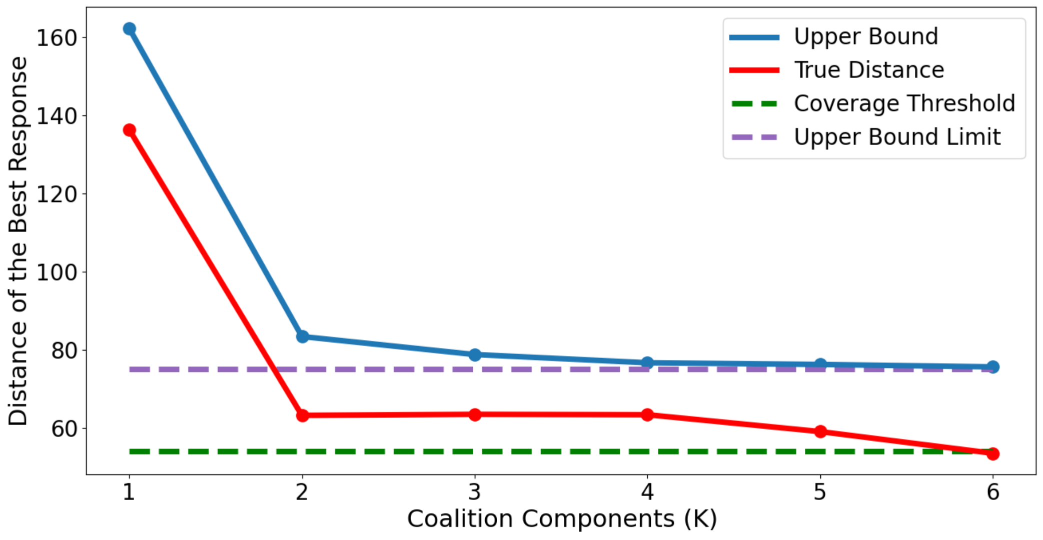
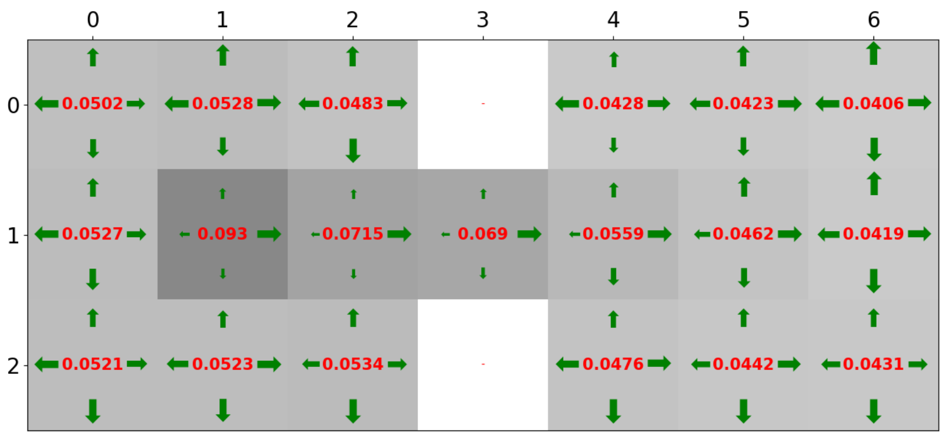


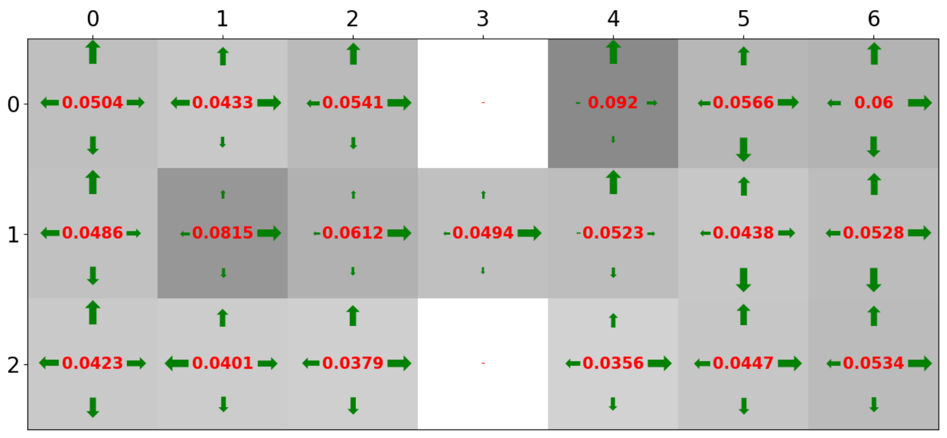
Figure 8.12: The limits of the non-consistent upper bound.

8.2.3 The Resulting Covering Policies

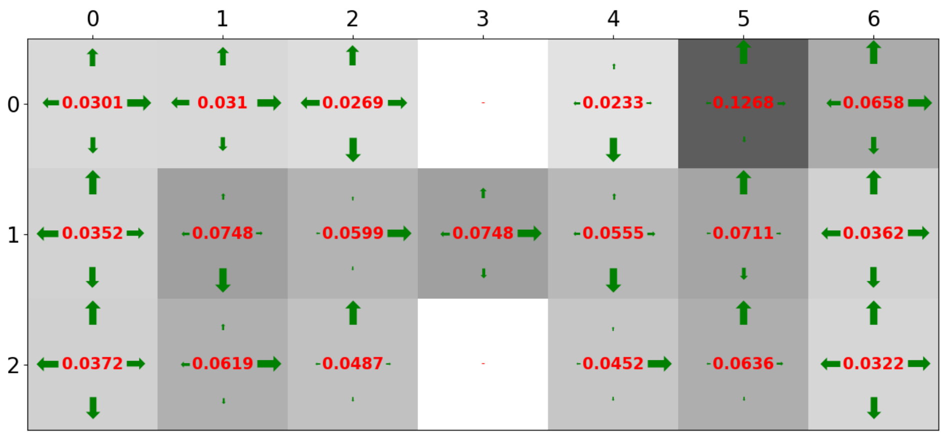
We report in Figure 8.13 the obtained configuration of the leader's coalition. As the reader can see, the covering policies are effectively partitioning the environment at hand. Interestingly, we can spot different behaviors in the individual components. The leader in Figure 8.13a is uniformly visiting the environment, slightly focusing on the left room. The second and third leaders (Figure 8.13b and 8.13c), are exhibiting a specialized behavior, visiting more often the upper part of the right room. Finally, the fourth leader in Figure 8.13d focuses on the leftmost side of the environment and to the opposite upper-right corner. We argue that these policies are satisfyingly partitioning the underlying MDP by visiting different regions of the environment, as shown in the images. Moreover, since the number of covering policies is small, the components are forced to be fairly stochastic, as each one of them has many distributions to cover. We argue that, by incrementing the number of leaders, each element of the coalition specializes to cover a smaller region of the MDP, while still collectively covering the full distribution space. To empirically support this claim, we report the component in Figure 8.14, which was taken from a coalition of ten elements. As the reader can notice, its policy is much more specialized, in particular, to explore the lower part of the left room.



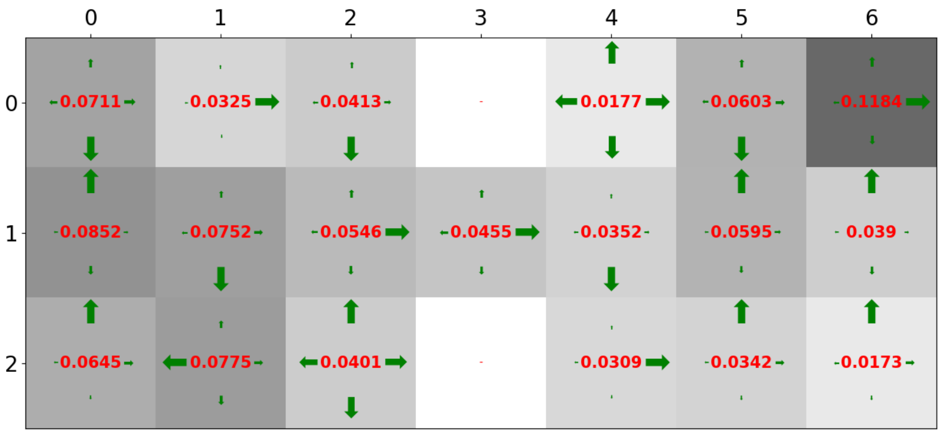
(a) First leader component.



(b) Second leader component.



(c) Third leader component.



(d) Fourth leader component.

Figure 8.13: Resulting leader coalition.

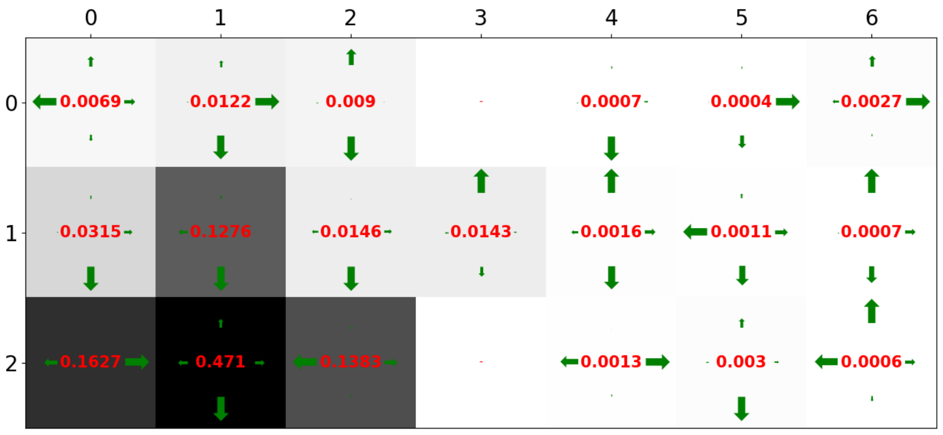


Figure 8.14: Specialized leader component.

Chapter 9

Discussion and Conclusions

In the following chapter, we will propose some interesting directions for future works (Section 9.1) and we will summarize the main results presented in this document (Section 9.2).

9.1 Discussion

Thanks to the unconventional nature of the presented topic, it is easy to see many potential directions for future works. In the following, we will present the three tracks we believe to be the most compelling ones.

9.1.1 Extension to Model-Free Setting

Many of the components of the solution procedure we presented in this thesis make explicit use of the transition model of the environment. Unfortunately, in real-case scenarios we cannot always rely on the complete knowledge of the environment and, therefore, it would be interesting to extend the proposed techniques to the case where the environment is unknown. The most straightforward solution is to estimate the transition model from samples and to keep everything else as it is in the proposed algorithm. An interesting alternative would be to directly estimate the gradients of the two players without relying on the transition model. Nonetheless, integrating these techniques in the complete algorithm could prove an interesting, and worthwhile, challenge.

9.1.2 Exploring Better Upper Bounds

The main limitation of the solution proposed in this thesis is the non-consistency of the upper bound in Equation (7.4). Unfortunately, with such an upper bound, the coverage guarantee we can provide is capped by a problem-dependent threshold. Even though we proposed a consistent alternative in Section 7.3, it requires finding the global optimum of a surrogate problem, which does not directly translate to a formalization amenable to efficient global optimization. It would be certainly compelling to investigate how to perform such optimization and to devise tighter upper bounds, in order to improve the coverage guarantee we can provide with the algorithm.

9.1.3 Development of Custom Algorithms

Reducing the policy search problem to a finite MAB problem is only one of the possible applications that make use of a compressed policy space. We indicated OPTIMIST [16] as one of the algorithms that would straightforwardly exploit such compression, but we argue that many other algorithms in the reinforcement learning scenario could benefit from having at their disposal a reduced representation of the policy space. In our opinion, this thesis could pave the way for novel policy search algorithms that are specifically designed to exploit a compressed policy space.

9.2 Conclusions

The main contribution of this thesis is a novel approach to the reward-free optimization setting, addressing the *compression* of the *policy space* available to the agent. This unconventional objective aims to reduce the complexity of policy search algorithms, by providing the agent a reduced set of representative elements, that can be used to estimate the value of any element in the policy space. In this thesis, we formalized the policy space compression problem as a non-convex two-player game, and we devised a sound procedure to find locally optimal solutions to such a game. Moreover, we presented theoretical guarantees on the algorithm's result, which we empirically verified with some practical examples.

We positively believe the *policy space compression* to be an interesting problem for the reinforcement learning community, and we hope to see more research avenues building on the ideas presented in this writing.

Bibliography

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[2] Martin J. Osborne. *An introduction to game theory*. Oxford Univ. Press, 2004.

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

[4] Lihong Li. Sample complexity bounds of exploration. *Adaptation, Learning, and Optimization*, 2012.

[5] Chi Jin, Akshay Krishnamurthy, Max Simchowitz, and Tiancheng Yu. Reward-free exploration for reinforcement learning. In *International Conference on Machine Learning*, 2020.

[6] Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, 2019.

[7] Jean Tarbouriech and Alessandro Lazaric. Active exploration in markov decision processes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

[8] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

[9] Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. A policy gradient method for task-agnostic exploration. *arXiv preprint arXiv:2007.04640*, 2020.

[10] Mirco Mutti and Marcello Restelli. An intrinsically-motivated approach for learning highly exploring and fast mixing policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[11] Chuheng Zhang, Yuanying Cai, Longbo Huang, and Jian Li. Exploration by maximizing r\'enyi entropy for zero-shot meta rl. *arXiv preprint arXiv:2006.06193*, 2020.

[12] Liu Hao and Abbeel Pieter. Behavior from the void: Unsupervised active pre-training. *arXiv preprint arXiv:2103.04551*, 2021.

[13] Emilie Kaufmann, Pierre Ménard, Omar Darwiche Domingues, Anders Jonsson, Edouard Leurent, and Michal Valko. Adaptive reward-free exploration. In *Algorithmic Learning Theory*, 2021.

[14] Zihan Zhang, Simon S Du, and Xiangyang Ji. Nearly minimax optimal reward-free reinforcement learning. *arXiv preprint arXiv:2010.05901*, 2020.

[15] Ruosong Wang, Simon S Du, Lin Yang, and Russ R Salakhutdinov. On reward-free reinforcement learning with linear function approximation. In *Advances in Neural Information Processing Systems*, 2020.

[16] Matteo Papini, Alberto Maria Metelli, Lorenzo Lupo, and Marcello Restelli. Optimistic policy optimization via multiple importance sampling. In *ICML*, 2019.

[17] Alberto Maria Metelli, Matteo Papini, Pierluca D'Oro, and Marcello Restelli. Policy optimization as online learning with mediator feedback. *arXiv preprint arXiv:2012.08225*, 2020.

[18] Alberto Maria Metelli, Matteo Papini, Francesco Faccio, and Marcello Restelli. Policy optimization via importance sampling. *Advances in Neural Information Processing Systems*, 2018.

[19] Tanner Fiez and Lillian Ratliff. Gradient descent-ascent provably converges to strict local minmax equilibria with a finite timescale separation. *arXiv preprint arXiv:2009.14820*, 2020.

[20] Chi Jin, Praneeth Netrapalli, and Michael Jordan. What is local optimality in nonconvex-nonconcave minimax optimization? In *International Conference on Machine Learning*, 2020.

[21] Alan Malek, Yasin Abbasi-Yadkori, and Peter Bartlett. Linear programming for large-scale markov decision problems. In *International Conference on Machine Learning*, 2014.

[22] Art B. Owen. *Monte Carlo theory, methods and examples.* Avaliable at `https://statweb.stanford.edu/~owen/mc/`, 2013.

[23] Corinna Cortes, Yishay Mansour, and Mehryar Mohri. Learning bounds for importance weighting. In *Nips*, 2010.

[24] Dimitri Bertsekas. *Dynamic Programming and Optimal Control, Vol II: Approximate Dynamic Programming.* MIT, 2012.

[25] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 1992.

[26] Marvin Zhang, Zoe McCarthy, Chelsea Finn, Sergey Levine, and Pieter Abbeel. Learning deep neural network policies with continuous memory states. In *2016 IEEE international conference on robotics and automation (ICRA)*, 2016.

[27] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Found. Trends Robot*, 2013.

[28] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, 2014.

[29] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

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

[31] J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 2001.

[32] Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. In *Conference on Learning Theory*, 2020.

[33] Kaiqing Zhang, Alec Koppel, Hao Zhu, and Tamer Basar. Global convergence of policy gradient methods to (almost) locally optimal policies. *SIAM Journal on Control and Optimization*, 2020.

[34] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Adaptive step-size for policy gradient methods. *Advances in Neural Information Processing Systems*, 2013.

[35] John Nash. Non-cooperative games. *Annals of Mathematics*, 1951.

[36] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 2009.

[37] Lillian J Ratliff, Samuel A Burden, and S Shankar Sastry. On the characterization of local nash equilibria in continuous games. *IEEE Transactions on Automatic Control*, 2016.

[38] Tanner Fiez, Benjamin Chasnov, and Lillian Ratliff. Implicit learning dynamics in stackelberg games: Equilibria characterization, convergence analysis, and empirical study. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[39] J. von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 1928.

[40] Constantinos Daskalakis, Stratis Skoulakis, and Manolis Zampetakis. The complexity of constrained min-max optimization. *arXiv preprint arXiv:2009.09623*, 2020.

[41] Eric Mazumdar, Lillian J. Ratliff, and S. Shankar Sastry. On gradient-based learning in continuous games. *SIAM Journal on Mathematics of Data Science*, 2020.

[42] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms.* Cambridge University Press, 2020.

[43] Andrea Tirinzoni, Alessandro Lazaric, and Marcello Restelli. A novel confidence-based algorithm for structured bandits. In *International Conference on Artificial Intelligence and Statistics*, 2020.

[44] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations.* Springer, 1972.

[45] Maher Nouiehed, Maziar Sanjabi, Tianjian Huang, Jason D Lee, and Meisam Razaviyayn. Solving a class of non-convex min-max games using iterative first order methods. *Advances in Neural Information Processing Systems*, 2019.

[46] Tetsuro Morimura, Eiji Uchibe, Junichiro Yoshimoto, Jan Peters, and Kenji Doya. Derivatives of logarithmic stationary distributions for policy gradient reinforcement learning. *Neural computation*, 2010.

[47] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, 2016.

[48] Yannick Schroecker and Charles L Isbell. State aware imitation learning. *Advances in Neural Information Processing Systems*, 2017.

[49] Giorgia Ramponi and Marcello Restelli. Newton-based policy optimization for games. *arXiv preprint arXiv:2007.07804*, 2020.

[50] Santiago Paternain, Aryan Mokhtari, and Alejandro Ribeiro. A newton-based method for nonconvex optimization with fast evasion of saddle points. *SIAM Journal on Optimization*, 2019.

[51] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 1948.

[52] Jingbin Liu, Xinyang Gu, and Shuai Liu. Policy optimization reinforcement learning with entropy regularization. *arXiv preprint arXiv:1912.01557*, 2019.

[53] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

[54] KB Petersen and MS Pedersen. The matrix cookbook. technical university of denmark. *Technical Manual*, 2008.

[55] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust markov decision processes. *Mathematics of Operations Research*, 2013.

[56] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003.

[57] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 2021.

[58] Kurt M Anstreicher. Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. *Journal of Global Optimization*, 2009.

[59] Zhi-Quan Luo, Wing-Kin Ma, Anthony Man-Cho So, Yinyu Ye, and Shuzhong Zhang. Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, 2010.

[60] Jaehyun Park and Stephen Boyd. General heuristics for nonconvex quadratically constrained quadratic programming. *arXiv preprint arXiv:1703.07870*, 2017.

[61] Tim Van Erven and Peter Harremos. Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*, 2014.

[62] Imre Csiszár and János Körner. *Information theory: coding theorems for discrete memoryless systems.* Cambridge University Press, 2011.

[63] Igal Sason and Sergio Verdú. Upper bounds on the relative entropy and rényi divergence as a function of total variation distance for finite alphabets. In *2015 IEEE Information Theory Workshop-Fall (ITW)*, 2015.