POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Planning in Stochastic Environments through Policy Optimization with Mediator Feedback

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Marcello Restelli

Correlatori: Dott. Amarildo Likmeta
Dott. Alberto Maria Metelli

Tesi di Laurea di:
Jacopo Germano, matricola 946847

Anno Accademico 2020-2021

Acknowledgements

I would like to thank my supervisor, Prof. M. Restelli, for his support, encouragement and patience, as well as the assistant supervisors Dott. A. Likmeta and Dott. A.M. Metelli for their precious help in suggesting improvements to this work, for their availability to clarify my doubts and to review my work. Without their precious experience and help, I would not have been able to complete such a challenging thesis.

I would also like to thank my parents who allowed me to study what I like and who have been constantly supporting me during my years of study. I would like to thank my sister Martina for being close to me in the toughest moments.

Last but not least, I would like to thank all my friends who made my university years unforgettable, and all the people who have been close to me: Andrea, Aurora, Carlo, Chiara, Davide, Enrico, Francesca, Gabriele, Giovanni, Leonardo, Lorenzo, Luca, Marco, Martina, Matteo, Nicola, Nicole, Riccardo, Stefano.

Contents

List of Figures

List of Algorithms

Abstract

Within the domain of Reinforcement Learning, the task of *online planning* in *continuous stochastic environments* has proven to be challenging. Over the years, many *sparse-lookahead-tree*-based methods have been designed to solve this problem, followed by the more optimization-oriented *policy-search*-based planners. The goal of this thesis is to develop a policy-search-based online planning algorithm which employs a randomized exploration strategy combined with multiple importance sampling estimation. We develop a continuous bandit algorithm (a sample-based optimizer of stochastic functions defined over compact sets), providing a theoretical analysis of its properties. We employ our algorithm as the core policy optimizer in a policy-search-based planner, testing its performance on two benchmark environments taken from the literature. We compare the performance of our algorithm with that of two the state-of-the-art solutions, showing that it can achieve good, yet not optimal, results for large budget values and showing that it can outperform a deterministic approach when limited budget is available.

Estratto in Italiano

Questa tesi si sviluppa nell'ambito del *Reinforcement Learning*, nel quale si ha un *agente* che ha come obiettivo massimizzare il suo *guadagno* durante l'interazione con un determinato *ambiente*. L'ambiente, inteso come un'entità esterna che può trovarsi in diversi possibili stati, evolve secondo leggi in generale non deterministiche e influenzate in parte dalle azioni effettuate dall'agente stesso. Sta all'agente quindi scegliere accuratamente le azioni da effettuare in modo da guidare l'evoluzione dell'ambiente seguendo il proprio interesse. Il framework utilizzato per rappresentare questo scenario è il Markov Decision Process e l'agente deve ripetutatamente scegliere un'azione fino a quando il processo non termina o un determinato orizzonte temporale non viene raggiunto. Da questi presupposti nasce la branca di ricerca che si occupa di "online planning". Con il termine "planning" si intende la ricerca delle azioni ottime da scegliere in tutti i possibili stati dell'ambiente. Con il termine "online" si intende che invece che pianificare l'interazione tutta in un una volta (che sarebbe come dire risolvere il MDP), l'agente deve pianificare solo la prossima azione da scegliere, per poi applicarla, osservare l'evoluzione dell'ambiente e ottenere di nuovo il tempo necessario a pianificare la prossima azione. Ovviamente si tratta di un compito più semplice rispetto alla soluzione dell'intero MDP e questa semplificazione non viene effettuata senza una buona ragione: si applicano le tecniche di online planning ad ambienti troppo estesi e/o complessi per essere risolti completamente. In questo scenario, l'agente che pianifica ha tipicamente a disposizione un *simulatore* dell'ambiente con cui sta interagendo. L'agente non può osservare il funzionamento interno del simulatore ma è libero di utilizzarlo per effettuare esperimenti su come l'ambiente si potrebbe evolvere in seguito alla scelta di determinate azioni. In particolare, può simulare possibili sequenze di azioni che conducono alla terminazione del processo o al raggiungimento dell'orizzonte temporale prefissato. Queste simulazioni "complete" sono chiamate *rollout* e tipicamente le risorse a disposizione dell'agente per scegliere la prossima azione sono espresse in termini

del numero di rollout che gli è consentito generare tramite il simulatore prima di dover scegliere la prossima azione.
In questo scenario, un approccio è quello basato sulla creazione di un "lookahead tree", un albero che rappresenta le informazioni raccolte sull'evoluzione dell'environment in seguito alla scelta di diverse azioni, che viene costruito tramite i rollout ottenuti con il simulatore e che viene infine usato per scegliere l'azione più promettente. Un secondo e differente approccio è rappresentato dalla "policy search", che parte dalla definizione dello *spazio delle policy*. Una *policy* rappresenta una funzione o un insieme di regole che definiscono l'azione che l'agente deve scegliere come una funzione dello stato dell'ambiente. Definito quindi lo spazio delle policy come un insieme di policy, l'approccio policy search si occupa di trovare la policy migliore in questo spazio utilizzando il simulatore per guidare la ricerca.
In questa tesi, sviluppiamo un algoritmo di online planning che si basa sull'approccio policy-search. Dapprima proponiamo un algoritmo per affrontare problemi bandit continui, vale a dire per ottimizzare una funzione stocastica raccogliendone un numero finito di campioni. Per lo sviluppo di questo algoritmo bandit, assumiamo che la funzione da ottimizzare sia definita su un insieme compatto multidimensionale, ogni elemento del quale rappresenta un vettore di parametri. Dopo aver derivato una discretizzazione di questo spazio continuo, fondamentalmente basata sulla costruzione di una griglia multi-dimensionale, il nostro algoritmo effettua un'esplorazione casuale guidata dalla griglia stessa. Le regioni vicine a punti che hanno dato risultati promettenti sono oggetto di un'esplorazione più intensa caratterizzata dalla raccolta di più campioni. Per sfruttare il carattere probabilistico della nostra esplorazione utilizziamo degli stimatori statistici basati su tecniche chiamate tecniche di "importance sampling". Concludiamo la presentazione del nostro algoritmo bandit con un'analisi teorica di alcune proprietà riguardanti la qualità della soluzione che restituisce.
Successivamente, utilizziamo questo algoritmo bandit per ottenere un algoritmo di online planning basato su policy-search utilizzandolo proprio per effettuare il passo di ottimizzazione della policy, che è definita su uno spazio compatto di parametri. Testiamo questo algoritmo su due ambienti noti nella letteratura, specificamente "Mini-golf" e "MountainCar". Arricchiamo questi test con la rappresentazione grafica e il commento di fenomeni inerenti l'approccio policy-search in sé, come l'esistenza di una "value function" definita sullo spazio delle policy che ivi definice una superficie, e il decadimento di performance che accompagna il nostro approccio di esplorazione casuale se la randomicità non è adeguatamente limitata.
Alla misura di performance del nostro algorimo affianchiamo quella di due al-

goritmi rappresentanti lo stato dell'arte: MCTS-DPW (Couetoux and Doghmen, 2011) per avere un confronto con un algorimo basato sull'approccio "lookahead tree" e UGapEb (Gabillon et al., 2012) per confrontare la nostra esplorazione causale con un approccio deterministico basato sulla stessa grigliatura.

Chapter 1

Introduction

The task of *planning* is one of the most fascinating challenges in the context of *Reinforcement Learning* (Sutton and Barto, 2018). The framework is quite simple: an *agent* must interact with an *environment* by choosing *actions*. The actions will generate a *change* in the environment, that will give the agent a *reward* and notify it of the change so that the agent can choose again, and so on. Despite the simplicity and intuitiveness of the framework, to develop algorithms that can interact well with new environments, obtaining as high cumulative rewards as possible, is a challenging task.

After the exact methods have been developed to find the optimal behaviour of the agent in simple environments, the attention of the researchers has moved to the setting in which computing an exact solution is impossible due to the computational requirements. For this scenario, different frameworks and algorithms have been proposed to formalize which information is available to the agent and to compute an approximate solution. The main instance of this planning setting is that of *Monte Carlo Planning*, initiated by Kearns et al. (2001), in which the evolution dynamics of the environment are assumed to be so complex, meaning that the environment can be in so many different states and can evolve to so many different states depending on the agent's choice, that a complete description of such dynamics is not available: the agent will never have the possibility to "look at the full picture". In this setting, the only source of information that the agent can use is a *black-box simulator* of the environment, which can be used to obtain samples of how the environment could evolve and what reward it could yield if a certain action is chosen in a certain state. Within this framework, two research directions have been mainly explored in the subsequent works, namely the *lookahead-tree*-based approaches and the *policy-search*-based ap-

proaches.
The former use the simulator to build what is called a *lookahead tree*, which is a tree whose leaves represent the future states of the environment after different sequences of actions have been chosen from the current state, and uses this tree to choose the next action. The main contributions to this approach are brought by the popular UCT algorithm (Kocsis and Szepesvári, 2006), which brilliantly employs techniques from the bandit literature to build the lookahead tree in a sample-efficient way, and by the subsequent improvements to UCT to adapt it to the most difficult environments. For example, the algorithm developed by Couetoux and Doghmen (2011) which introduced techniques to artificially control the width of this tree, resulting in great performance improvements.
The online *policy-search*-based approaches, like that of Papini et al. (2019) with the OPTIMIST algorithm, typically search for an optimal *policy* describing how to choose the actions in each state rather than simply looking for the next action to choose in the current state. In such setting, the search space is usually a compact set of parameter vectors and each vector represents a policy (for example, if the action is a real number and the state of the environment is described by two real numbers, the space of linear policies is completely described by the real vectors of dimension three: one parameter for each state variable plus one for the constant term). In order to search this parameter space, the black-box simulator can be used to sample the rewards that a specific parameter vector (therefore, a specific policy) would collect if adopted to choose the actions from the current state onwards. After the optimal policy has been found, it is used to select the next action to choose in the current state in the interaction with the environment.
As the title suggests, the contributions of this thesis belong to the category of *policy-search*-based planning techniques.

1.1 Motivations

Among the many possible environments, the most challenging seem to be the ones whose state is represented by real numbers (therefore called *continuous* environments) and whose transitions are *stochastic*, meaning that the next state is selected according to a probability distribution over the continuous set of states.
When dealing with continuous stochastic environments, the basic mechanism behind the sparse lookahead-tree approaches, which rely on repeated exploration of the promising future paths, fails due to the uncountable number of paths. While this limit has in principle been overcome by pairing UCT with

progressive widening techniques (Couetoux and Doghmen, 2011), the performance of the resulting planning algorithm heavily depends of the tuning of some parameters. The optimal values of such parameters are environment-dependent and, if they are not tuned properly, the resulting algorithm may perform terribly.
From the perspective of policy-search-based approaches, the difference between environments with deterministic transitions and those with stochastic transitions is more evident in the degree of stochasticity in the performance of the policies themselves. For instance, in the deterministic case, assuming that the rewards are deterministic too, it is sufficient to try each policy once. Conversely, if either the rewards or the transitions or both are stochastic, more trials will be needed to have an accurate estimate of the performance of one policy. To design effective ways of exploring the policy space, enjoying strong theoretical guarantees on the quality of the solution, is currently the main goal in this line of research.

1.2 Contributions

In line with the recent works proposing randomized approaches to policy-search-based online planning, like that of Metelli et al. (2020), we try to give an algorithmic contribution to this field: we propose a randomized continuous bandit algorithm to perform optimization of stochastic functions employing the *multiple importance sampling* technique. For our algorithm, we provide an analysis from the theoretical point of view characterizing the quality of the results that it can achieve. We then show how the problem of online planning, when approached from the policy search perspective, under specific assumptions can be seen as a sequence of continuous bandit instances, justifying the development of our algorithm. Lastly, we apply our algorithm to planning as stated above in two popular environments, namely *Mini-golf* and *MountainCar*, comparing its performance with that of the current state-of-the-art algorithms and providing interpretations to the results.

1.3 Structure of the Thesis

The thesis is structured in six Chapters:

1. Introduction: in this Chapter, we present the scope of the thesis, briefly summarizing what the original contributions are and how they link to previous works.

2. Preliminaries: in this Chapter, we report all the notions that the reader will need in order to better understand the contents of the remaining Chapters. In this Chapter, the reader will find all the relevant background from the formal definition of a Markov Decision Process to the notions of Reinforcement Learning, online planning, bandit algorithms and multiple importance sampling.

3. State of the Art: in this Chapter, we report the current state-of-the-art algorithms in the planning area with a detailed explanation of how they work and of what are the innovative ideas that support their performance.

4. A Randomized Approach to Best Arm Identification in Continuous Bandits: in this Chapter, we present our randomized continuous bandit algorithm with detailed descriptions of every step and with all the relevant pseudo-code. At the end of the Chapter, we provide theoretical results involving our algorithm and its performance.

5. Application to Online Planning: in this Chapter, we describe all the experiments that we have performed. Detailed descriptions are given for both the environments and for all the parameter-tuning steps, and lastly a comparative view of the performance of our algorithm and that of the state-of-the-art competitors is given.

6. Conclusions: in this Chapter, we give a summary of the pros and cons of the application of our algorithm to online planning. We give interpretations of what could be the source of better or worse performances when compared to other algorithms, lastly indicating possible directions of future research.

Chapter 2

Preliminaries

2.1 Summary

In this Chapter the fundamental definitions and theoretical background necessary to understand the rest of our work are given. We start with the framework of *Markov Decision Processes*, then we give an introduction about *Reinforcement Learning* and *planning*, explaining the particular setting of *online planning*. We report the *stochastic bandit* frameworks for both the cases of finite and infinite number of arms. Finally, we describe the *multiple importance sampling* technique along with the *balance heuristic* and *weight truncation*, which will be used in the following Chapters.

2.2 Markov Decision Processes

A Markov Decision Process (MDP) (Puterman, 2005) is a framework which models the following scenario: an agent is interacting with a probabilistic system, influencing its behaviour by making a sequence of choices. Based on the resulting behaviour of the system, the agent can be more or less satisfied.

2.2.1 The Formal Model

Formally, a Markov Decision Process is a 5-uple $\langle \mathcal{T}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$ where:

- $\mathcal{T}$ is the set of *decision epochs*, the set of instants in time in which the agent is requested to make a choice. It can be discrete, so that the instants could be enumerated $\mathcal{T} = \{t_1, t_2, t_3, ...\}$, or continuous. In the following, we will focus on MDPs with discrete sets of decision epochs, with a *finite* number of decision epochs $\mathcal{T} = \{t_1, ..., t_N\}$ where N is referred to as the *horizon*.

- $\mathcal{S}$ is the set of possible *states* in which the system can be. The states can be discrete or continuous. In the former case, we speak of *finite* MDP, in the latter we speak of *continuous* MDP.

- $\mathcal{A}$ is the set of possible *actions* that the agent can choose from: more specifically, $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$ with $\mathcal{A}_s$ being the set of actions available to the agent when the system is in state s. Like the states, the actions can be discrete or continuous.

- $\mathcal{R}$ is the *reward function* which, for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}_s$, provides the reward $\mathcal{R}(s, a)$ that the agent will receive choosing such action in that state. It is required that the reward is always finite. The reward could be stochastic, such that when the action is chosen the reward is sampled from a distribution. In this case, the reward function indicates the expected value of the distribution.

- $\mathcal{P}$ is the *transition probability function* which, for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}_s$, provides the probability distribution $\mathcal{P}(s'|s, a)$ over the next state s'.

2.2.2 Policies

Given an MDP $\langle \mathcal{T}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, a *policy* $\pi(s)$ is a function prescribing the action to choose in each state $s \in \mathcal{S}$. The policies can be *stochastic*, thus describing a probability distribution over the actions $\mathcal{A}_s$ rather then one specific action. In this case, the policy $\pi(s, a)$ indicates the probability of playing action a in state s.

2.2.3 Solution Concepts

Given a policy, the rewards that the agent will collect until the end of the interaction entirely depend on how the stochastic quantities of the process realize: the sequence of rewards $(r_1, ..., r_N)$ is indeed the result of a random experiment. Therefore, in order to assess how well a policy performs on a given MDP, optimality criteria are defined as a function of such sequence of rewards. The two most common optimality criteria are the *expected total reward*, which is the *expected value* of the summation $\sum_{i=1}^{N} r_i$, and the *expected total discounted reward*, which is similar to the former but introduces a *discount factor* as we wil see in the next paragraph. The reason to introduce a discount factor is that one might prefer, among a set of policies that have the same (or very similar) expected total reward, the one that collects

the reward sooner. Different discount factors will thus result in different attitudes of the agent towards risk.

The Expected Total Discounted Reward

The *Expected total discounted reward* is the most common optimality criterion, and is the criterion that we adopt. Calling r_i the reward obtained after choosing the action at epoch i and assuming that policy π is used for the whole process and that the system is initially in state s, it is defined as:

$$V_N^\pi(s) = E\left[\sum_{t=1}^{N} \gamma^{t-1} r_t \;\middle|\; Initial\ state = s,\ Policy = \pi\right],$$

where γ, the *discount factor*, is a real number $0 \leqslant \gamma < 1$. In particular, γ can be seen as the value at time n of a reward of 1 received at time $n+1$, and is used so that rewards collected far in the future are less appealing. In the limit case of $\gamma = 1$ the rewards are not discounted.

State Value Function

Given an MDP $\langle \mathcal{T}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, a policy $\pi(s)$ and an optimality criterion such as the *expected total discounted reward* one can compute the value, induced by the policy, of such criterion. This is a function of the state and of the number of remaining decision epochs and goes under the name of *state value function*, indicated as $V_N^\pi(s)$.
If the *expected total discounted reward* criterion is used, its definition allows to derive constraints that are always satisfied by the value functions.
In the case of a *finite* number of remaining decision epochs, the state value functions at each epoch n are subject to the following dynamic constraint, known as the *Bellman Expectation Equation* (Bellman, 1957):

$$V_n^\pi(s) = \sum_{a \in \mathcal{A}_s} \pi(s,a)\Big(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V_{n-1}^\pi(s')\Big).$$

In matrix notation, we can write it as follows:

$$V_n^\pi = R^\pi + \gamma P^\pi V_{n-1}^\pi,$$

where

- $V_n^\pi \in \mathbb{R}^{|\mathcal{S}| \times 1}$ is the column vector of the state values with horizon n,
- $R^\pi \in \mathbb{R}^{|\mathcal{S}| \times 1}$ is the column vector such that

$$R_s^\pi = \sum_{a \in \mathcal{A}_s} \pi(s,a)\mathcal{R}(s,a),$$

- $P^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is the matrix of the transition probabilities induced by the policy, such that

$$P^\pi_{s,s'} = \sum_{a\in\mathcal{A}_s} \pi(s,a)\mathcal{P}(s'|a,s).$$

In the *infinite horizon* case, the dependence on the number of remaining decision epochs is removed and the relation becomes:

$$V^\pi_\infty(s) = \sum_{a\in\mathcal{A}_s} \pi(s,a)\Big(\mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a)V^\pi_\infty(s')\Big).$$

In matrix form, removing the "∞" subscript, it becomes

$$V^\pi = R^\pi + \gamma P^\pi V^\pi,$$

which admits the closed form solution

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi.$$

If the initial state is also stochastic, given the probability $\mu(s)$ of system starting in state s, one can compute the expected total discounted reward of policy π as:

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

State-Action Value Function

Similarly to the state value function, it is possible to define the *state-action value function*, indicated as $Q^\pi_N(s,a)$, which represents the expected total discounted reward for the agent of choosing action a in state s when there are N remaining decision epochs.
In the case of *finite* horizon, the state-action value functions are subject to the following dynamic constraints:

$$\begin{aligned} Q^\pi_n(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a)V^\pi_{n-1}(s') \\ &= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a) \sum_{a'\in\mathcal{A}} \pi(s',a')Q^\pi_{n-1}(s',a'). \end{aligned}$$

In the case of *infinite* horizon, the dependence on the number of remaining decision epochs disappears and the relation becomes:

$$\begin{aligned} Q^\pi(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a)V^\pi(s') \\ &= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a) \sum_{a'\in\mathcal{A}} \pi(s',a')Q^\pi(s',a'). \end{aligned}$$

Optimal Policies

An optimal policy is a policy π^* such that:

$$V_n^{\pi^*}(s) \geqslant V_n^{\pi}(s) \quad \forall s \in \mathcal{S}, \forall n \in \mathcal{T}, \quad \forall \pi.$$

When the dynamics of the MDP ($\mathcal{R}$, $\mathcal{P}$) are known and the cardinalities of ($\mathcal{S}$, $\mathcal{A}$) are small enough, the expected total discounted reward of a given policy can be computed exactly in closed form and the search for the optimal policy can be carried on by dynamic programming approaches.
When the optimal policy π^* and/or the corresponding value function $V^{\pi^*}(s)$ are known, the MDP is considered *solved*.
The state-action value function $Q_N^{\pi^*}(s,a)$ of a finite time horizon induced by an optimal policy π^* satisfies the following equation, known as the *Bellman Optimality Equation* (Bellman, 1957)

$$Q_n^{\pi^*}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}'_s} Q_{n-1}^{\pi^*}(s',a')$$

While the state value function satisfies

$$V_n^{\pi^*}(s') = \max_{a' \in \mathcal{A}_{s'}} Q_n^{\pi^*}(s',a').$$

Due to the presence of the non-linear operator *max*, the Bellman Optimality Equation does not admit a closed-form solution.

2.3 Reinforcement Learning

If finding an optimal policy in an MDPs with known dynamics and reward function and small state and action spaces can be considered a "solved problem" from the research perspective, it is not the case for MDPs with huge state/action spaces (that is, spaces that cannot be enumerated in a practical time on a computer) or for the setting in which the dynamics $\mathcal{R}$, $\mathcal{P}$ are not known. In these cases one has to search for the best policy with Reinforcement Learning techniques, as computing the exact solution with traditional methods is *impossible*.

2.3.1 Planning

The task of *planning* is that of finding an optimal policy for an MDP. The agent may have different levels of access to the transition/reward model of the MDP:

- The definition of $\mathcal{R}, \mathcal{P}$ is *known*, meaning that the agent knows the probability of each reward and transition.
- The definition of $\mathcal{R}, \mathcal{P}$ is *unknown* but there is a *simulator* which can be used by the agent to sample transitions and rewards according to $\mathcal{R}, \mathcal{P}$.
- The definition of $\mathcal{R}, \mathcal{P}$ is *unknown* and the agent does not have a simulator: in this case the agent will either learn the policy by interacting directly with the environment or learn an approximation of the transition/reward model that will be used to find the policy.

In our work, we consider the second case in which the environment dynamics are unknown but a simulator is available. Given a policy π, we can use the simulator of the environment to run *episodes*, i.e. *full sequences of interactions* choosing the actions as prescribed by π at each decision epoch until a terminal state is reached. In this setting, it is common to set a fixed maximum amount of episodes, the *budget*, that the agent can collect before yielding the result.

2.3.2 Online planning

In standard planning the agent is looking for the optimal policy for the whole model. In *online* planning, the agent is interacting with an environment and *at each decision epoch* is required to find, using a given *budget*, the best next action to choose, rather than a policy. The action is used in the environment, the next state is reached and the agent receives a new observation and more budget to find the next action, and so on.
The *budget* is a measure of the maximum amount of experience the agent can collect before performing a choice.

Monte Carlo Planning

The phrase *Monte Carlo Planning* is used to indicate the *online planning* setting in which the dynamics of the environment are unknown but a *simulator* (in the literature also called *generative copy*, *generative model*, *simulation environment*) of the environment is available. At each step of the interaction with the environment (which will be referred to in the following as the *real* environment), the agent can collect a certain number of episodes with the simulator.
In this setting, the *budget* can either be the total number of episodes run or the total number of steps taken in the generative model.

2.4 Stochastic Multi-armed Bandits

In the context of reinforcement learning, a *bandit* problem (Lattimore and Szepesvári, 2020) is a special case of MDP with unknown stochastic rewards. In this MDP there is only one state in which K actions (in the following called *arms*) are available, all leading back to the same state: the goal of the agent is to collect the highest total reward in a limited number of interactions N. Intuitively, the agent must find the best arms and keep pulling (choosing) them, taking into account the stochasticity of the rewards: if a low reward was received by one arm it does not necessarily mean that the expected reward of that arm is low. The name comes from the traditional example of a row of K lever-operated slot machines (which were also called "One-armed bandits" because they steal your money, with the lever being the one arm).

2.4.1 The Cumulative Regret

A n-rounds interaction can be represented as a sequence $(A_1, X_1, ..., A_n, X_n)$ of the arm chosen and the reward that was received. Let $\mu_1, ..., \mu_k$ be the expected rewards of the arms, ordered such that $\mu_k \geqslant \mu_{k+1}$ (this can be assumed without loss of generality). In order to measure how well the agent has performed, the *expected reward* of the pulled arms is compared with that of a "clairvoyant" agent which knows the arm with highest expected reward since the beginning and pulls it at each round.
The cumulative regret is therefore defined as:

$$R_n = n\mu_1 - E\left[\sum_{t=1}^{n} X_t\right],$$

where the expectation is taken with respect to the stochasticity of the rewards and of the choices of the learner (which, even if deterministic, depend on the stochastic observed rewards).

Exploration versus Exploitation trade-off

In order to achieve a small regret, the agent must pull the best arms. In order to find the best arms, a certain amount of *exploration* is needed, that is pulling arms in order to *gain more information* on their expected reward even if at the moment of the choice there is another arm with higher empirical mean. In other words, exploring means *ignoring the current best* in order to search for a better alternative *without the guarantee of finding it.* Obviously, exploring comes with the risk of collecting low rewards without

finding any good arm. Rather then exploring, the agent could be *exploiting*, that is choosing the current best arm even though it could be sub-optimal. Whenever the agent is required to minimize the *cumulative regret*, optimal exploration versus exploitation trade-off must be carefully accomplished: if too much exploration is performed the agent will incur in large regret due to too many pulls of sub-optimal arms, if too much exploitation is performed the agent will incur in large regret due to never finding the optimal arm.

2.4.2 The Upper Confidence Bound algorithm

The Upper Confidence Bound algorithm (UCB) (Auer et al., 2002) applies the "optimism in face of uncertainty" principle, stating that one should behave as if the environment is as nice as reasonably possible, to tune the exploration-exploitation trade-off. In order to do so, it computes for each arm a an upper confidence bound U_a, that is a value that is greater or equal to the true mean μ_a with probability at least $1 - \delta$, and then pulls the arm with the highest upper bound.
Let $(X_t)_{t=1}^n$ be a sequence of independent 1-subgaussian random variables with mean μ and let $\hat{\mu} = \frac{1}{n}\sum_{t=1}^n X_t$. It holds

$$P\left(\mu \geqslant \hat{\mu} + \sqrt{\frac{2\log(1/\delta)}{n}}\right) \leqslant \delta.$$

For each arm a, for which the reward sequence $(X_{a,1}, ..., X_{a,T_a})$ was observed, the upper bound is computed as

$$U_a = \frac{1}{T_a}\sum_{i=1}^{T_a} X_{a,i} + \sqrt{\frac{2\log(1/\delta)}{T_a}}$$

and the next arm x to be pulled is

$$x = \operatorname*{argmax}_a U_a.$$

UCB was proven to suffer a regret that grows logarithmically with the number of rounds.

Schedules for the error probability δ

In order to require that the confidence intervals hold at every round and for every arm, a time-dependent error probability is commonly used so that the bounds computed at round t hold with probability at least $1 - \delta_t$. If we define the event $E_{t,i}$ as "The bound of arm i at round t does not hold",

we can bound the probability of the event $E =$"At least one bound at one round does not hold" as

$$P(E) \leqslant \sum_{t=1}^{n} \sum_{i=1}^{K} P(E_{t,i}) = K \sum_{t=1}^{n} \delta_t = \delta.$$

By make the last equation hold, it is possible to obtain a schedule δ_t which ensures an overall error probability less than or equal to δ. A common choice is to set δ_t as a function of $\frac{1}{t^2}$.

2.4.3 Best Arm Identification

In the stochastic multi-armed bandit framework, optimizing the cumulative regret is not the only interesting task from the research perspective. As pointed out by Bubeck et al. (2010), some real life occurrences of a Multi Armed Bandit problem do not require to minimize the cumulative regret, but rather require to find the best arm. To show this fact, the simple yet effective example proposed by the authors is that of trials for cosmetic products, in which a testing phase is used to find the best among different product. Only after the testing phase is completed will the best product be commercialized, and the only interest of the company in this scenario is to pick the best product, disregarding the cumulative rewards collected during the testing phase.
The best arm identification setting is also called the *full exploration* setting, since the learner is not interested in exploiting.

The Simple Regret

In order to evaluate a learner in the full exploration setting, assuming that after n rounds the learner has recommended the arm I_n, the simple regret r_n is defined as:

$$r_n = \mu_1 - \mu_{I_n}.$$

Similarly to the cumulative regret, it is a measure of how wrong the learner's recommendation was.

2.4.4 Infinitely-Armed Bandits

It may happen in realistic scenarios to be required to find the best among a set of candidates, which is the main applicative scenario of multi-armed bandits, without having the possibility to try them all. This is the constraint

that the arms are *effectively infinite*, and is often present due to scarcity of time or computational resources combined with a high number of arms. This constraint introduces another problem which is not addressed by the finite bandit literature: whether it is better to try a new, unseen candidate or to exploit the already seen ones.

To formalize this scenario, mainly two frameworks have been used in the literature, requiring different properties of the function $\mu : x \to \mathbb{R}$,which represents the mean reward of each arm x.

Reservoir distribution for $\mu(x)$

In this setting, started by Berry et al. (1997), the average reward of each arm is sampled from a distribution called "reservoir distribution": $\mu(x) \sim \mathcal{P}$. As there is no correlation among average rewards of different arms, the learner is mainly concerned with trying a sufficiently high number of arms so that at least one of them will have an ϵ-optimal reward with a certain probability. In order to provide theoretical guarantees in this framework, assumptions are made on $\mathcal{P}$ and in particular on its upper-tail behaviour: one example of such assumption is that done by Carpentier and Valko (2015), requiring that there exist $\beta > 0$ and two constants E, E' such that

$$\forall \epsilon > 0 \quad E\epsilon^{\beta} \leqslant P(\mu(x) \geqslant \mu^* - \epsilon) \leqslant E'\epsilon^{\beta},$$

where μ^* is the best value of the average reward that can be sampled.

Regularity of $\mu(x)$

Kleinberg et al. (2008), Bubeck et al. (2011), Grill et al. (2015) assume that the function μ has some regularity property, for example that μ is locally Lipschitz. Such regularity assumptions allow, for instance, to deduce that the neighbours of a bad arm are bad, with a confidence that varies with the regularity measure that is considered. In this setting the arms are typically the points of a convex set $\mathcal{X} \subseteq \mathbb{R}^d$, over which the function $\mu : \mathcal{X} \to \mathbb{R}$ is defined.

In particular, in this setting, the task of *best-arm identification* coincides with that of *black-box optimization* of a stochastic function: both require finding the optimum point x^* with the smallest number of samples from the environment / function.

This framework is also referred to as the *continuous bandit* framework.

2.5 Importance Sampling Techniques

2.5.1 Importance Sampling

Importance sampling (Cochran, 2007) allows to estimate the expected value of a function under a *target* distribution (P) using samples drawn from a different distribution (Q), which is called *behavioural*.
Given a measurable space $(\mathcal{Z}, \mathcal{F})$ and two probability measures defined on such space P, Q such that $P \ll Q$ (P is *absolutely continuous* w.r.t Q, that is $Q(A) = 0 \implies P(A) = 0 \;\; \forall A \in \mathcal{F}$, events that are *almost impossible* under Q are *almost impossible* under P as well), there exists a $\mathcal{F}$-measurable function $w : \mathcal{Z} \to [0, \infty)$ such that for any measurable set $A \in \mathcal{F}$ it holds

$$P(A) = \int_A w dQ.$$

The function w is called the *Radon-Nikodym derivative* (Nikodym, 1930) of P with respect to Q, denoted as $w_{P/Q}$, and represents the ratio between the density of P and that of Q, if they exist. If p and q are the density functions of P and Q respectively, it holds

$$w_{P/Q} = \frac{p}{q}.$$

Given the sequence of i.i.d. (independent, identically distributed) outcomes $(z_1, ..., z_N)$ sampled from Q, the goal is to estimate $\mu = E_{z \sim P}[f(z)]$.
The *importance sampling estimator* of μ is

$$\widehat{\mu}_{IS} = \frac{1}{N} \sum_{i=1}^{N} w_{P/Q}(z_i) f(z_i).$$

$\widehat{\mu}_{IS}$ is *unbiased* (Owen, 2013), that is $E_{z_i \sim Q}[\widehat{\mu}_{IS}] = \mu$.

2.5.2 Multiple Importance Sampling

Multiple importance sampling is a generalization of importance sampling which allows to take samples from different behavioural distributions $Q_1, ..., Q_K$, under the hypothesis that $P \ll Q_k$ for $k = 1, ..., K$.
Let $\beta_1(z), ..., \beta_K(z)$ be *mixture weights*, i.e. $\forall z \in \mathcal{Z}$

$$\sum_{i=1}^{K} \beta_i(z) = 1,$$
$$\beta_k(z) \geqslant 0 \text{ for } k = 1, ..., K.$$

Assume that from each behavioural distribution Q_k the sequence of N_k samples $(z_{1,k}, ..., z_{N_k,k})$ was drawn. The multiple importance sampling estimator for μ is:

$$\widehat{\mu}_{MIS} = \sum_{k=1}^{K} \frac{1}{N_k} \sum_{i=1}^{N_k} \beta_k(z_{i,k}) w_{P/Q_k}(z_{i,k}) f(z_{i,k}),$$

and is unbiased for any valid choice of the mixture weights.

2.5.3 The Balance Heuristic estimator

A valid choice of the importance weights which has desirable variance properties is the *balance heuristic* (Veach and Guibas, 1995), where each weight is defined as:

$$\beta_k(z) = \frac{N_k}{\sum_{j=1}^{K} N_j q_j(z)} q_k(z).$$

The balance heuristic estimator is then:

$$\widehat{\mu}_{BH} = \sum_{k=1}^{K} \sum_{i=1}^{N_k} \frac{p(z_{i,k})}{\sum_{j=1}^{K} N_j q_j(z_{i,k})} f(z_{i,k}).$$

The Rényi divergence

To quantify the variance of $\widehat{\mu}_{BH}$ the Rényi divergence (Rényi, 1961) is used. Given two probability measures P, Q on the same measurable space $(\mathcal{Z}, \mathcal{F})$ such that $P \ll Q$, the α-Rényi divergence $D_\alpha(P\|Q)$ is defined as:

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \log \int_{\mathcal{Z}} (w_{P/Q})^\alpha dQ.$$

and the *exponentiated α-Rényi divergence* $d_\alpha(P\|Q)$ is defined as

$$d_\alpha(P\|Q) = e^{D_\alpha(P\|Q)}.$$

Bounding the variance of $\widehat{\mu}_{BH}$

Papini et al. (2019) showed that the variance of $\widehat{\mu}_{BH}$ is bounded by a function of $d_2(P\|\mathbf{\Phi})$, where $N = \sum_{k=1}^{K} N_k$, $\mathbf{\Phi} = \frac{1}{N} \sum_{k=1}^{K} N_k Q_k$ is a finite mixture and $\|f\|_\infty$ is the max-norm of f (i.e. the maximum value assumed by $|f|$):

$$\mathrm{Var}\left[\widehat{\mu}_{BH}\right] \leqslant \|f\|_\infty^2 \frac{d_2(P\|\mathbf{\Phi})}{N}.$$

Weights truncation

Metelli et al. (2018) have shown that the importance sampling estimator $\widehat{\mu}_{IS}$ presents problematic tail behaviours due to huge values occasionally assumed by some weights, preventing the use of exponential concentration inequalities. Ionides (2008), in his work, proposed the *truncation heuristic* to mitigate this problem, that is to fix a threshold M and to clip to the threshold the weights that exceed its value.
The resulting estimators are the *truncated importance sampling* estimator

$$\breve{\mu}_{IS} = \frac{1}{N} \sum_{i=1}^{N} \min \left\{ M, w_{P/Q}(z_i) \right\} f(z_i)$$

and the *truncated balance heuristic* estimator

$$\breve{\mu}_{BH} = \frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \min \left\{ M, \frac{p(z_{i,k})}{\sum_{j=1}^{K} \frac{N_j}{N} q_j(z_{i,k})} \right\} f(z_{i,k})$$

Bounding the bias and variance of $\breve{\mu}_{BH}$

Truncating the weights introduces a bias in the estimators. Papini et al. (2019) in their work showed that, assuming that there exists $\epsilon \in (0, 1]$ such that $d_{1+\epsilon}(P \| Q_k) < \infty$ for $k = 1, ..., K$, the bias and the variance of $\breve{\mu}_{BH}$ are bounded by a function of M, $d_{1+\epsilon}(P \| \mathbf{\Phi})$ and $\|f\|_\infty$.
In our work, we will assume $d_2(P \| Q_k) < \infty$ for $k = 1, ..., K$, therefore $\epsilon = 1$. The bound on the bias is then:

$$0 \leqslant \mu - E_{z_{i,k} \sim Q_k} \left[\breve{\mu}_{BH} \right] \leqslant \|f\|_\infty \frac{d_2(P \| \mathbf{\Phi})}{M}$$

and the bound on the variance is:

$$\mathrm{Var} \left[\breve{\mu}_{BH} \right] \leqslant \|f\|_\infty^2 \frac{d_2(P \| \mathbf{\Phi})}{N}.$$

Chapter 3

State of the Art

3.1 Summary

In this Chapter we report some of the state-of-the-art algorithms for both the planning and the continuous bandit scenarios. We start with the anonymous algorithm developed by Kearns et al., generically referred to by the authors as "A sparse sampling algorithm", later referred to as "Monte Carlo planning" by Kocsis and Szepesvári and more generally by the literature as "SparseSampling". Then we report UCT, the most successful instance of a Monte Carlo Tree Search algorithm, which is still a state-of-the-art planning algorithm for deterministic MDPs. We show the limitations of UCT and the proposed solution, and we report MCTS-DPW which is the current state-of-the-art in the case of continuous MDPs with stochastic transitions. We report the OPTIMIST algorithm as a policy-search-based planner. Lastly, we report two bandit algorithms: UGapE for the finite bandit best-arm-identification framework and HOO for the framework of continuous bandits.

3.2 Sparse-lookahead-tree Planning Algorithms

3.2.1 Monte Carlo Planning

The first Monte Carlo planning algorithm to be supported by a theoretical analysis is that of Kearns et al. (2001). The authors propose a planning algorithm targeting MDPs with discrete action and state spaces and deterministic rewards whose complexity does not depend on the number of states $|\mathcal{S}|$.

Their algorithm basically performs a depth-first search on the fixed-height tree of the next states which can be reached from the root in h steps. Their

algorithm can be expressed in pseudo-code (with simplifications with respect to the original) with the recursive procedure of algorithm 3.1.

Algorithm 3.1 Algorithm $\mathcal{A}$

```
procedure ESTIMATE-Q(s, γ, h, C)
    if h=0 then
        return 0, ..., 0
    end if
    for a ∈ A_s do
        S_a ← {C states sampled from P(·|s, a)}
        Q̂_{s,a} ← R(s, a) + γ (1/C) Σ_{s'∈S_a} max {ESTIMATE-Q(s', γ, h − 1, C)}
    end for
    return Q̂_{s,a_1}, ..., Q̂_{s,a_k}
end procedure
```

The constant C is computed as a first step as a function of the maximum error allowed ϵ, of the discount factor γ and of the maximum reward that can be obtained in the MDP.
The running time of their algorithm $\mathcal{A}$ is

$$T^{\mathcal{A}} \in O\left(\left(\frac{k}{\epsilon(1-\gamma)}\right)^{\frac{1}{1-\gamma}\log\left(\frac{1}{\epsilon(1-\gamma)}\right)}\right)$$

while the error is guaranteed to be smaller than ϵ:

$$|V^{\mathcal{A}}(s) - V^*(s)| \leqslant \epsilon.$$

Despite not depending on the number of states, this algorithm is still exponentially slow. We chose to report it here as it is the precursor of current state-of-the-art sparse-lookahead-tree algorithms as MCTS, MCTS-DPW.

3.2.2 Upper Confidence Trees

Kocsis and Szepesvári (2006) developed UCT (Upper Confidence Trees), a sparse-lookahead-tree based algorithm which, rather than exploring uniformly all the actions available at each state, employs a UCB-like technique to choose the most promising actions to explore.
The lookahead tree is composed of nodes which represent states. At the beginning of the search, the only node in the tree is the root, which represents the current environment state.

The Monte Carlo Tree Search loop

In any MCTS algorithm, the tree is built and explored at the same time according to the 4-phases loop depicted in Figure 3.1.
We give a brief explanation of each phase:

1. Selection: Select a node in the tree from which not all the actions have been tried according to a *tree policy*.

2. Expansion: Expand the tree by simulating one of the untried actions and adding the next state as a child node.

3. Evaluation: Evaluate the new node by running a roll-out according to a *default policy*.

4. Backpropagation: propagate the result of the roll-out along the chosen path.

The selection phase is usually performed by recursively descending the lookahead tree, using the *tree policy* to select the actions. If the transitions are stochastic, it is possible that the selected action leads to a different state than the one stored in the tree: in this case, the expansion phase is performed by adding this new state to the tree.

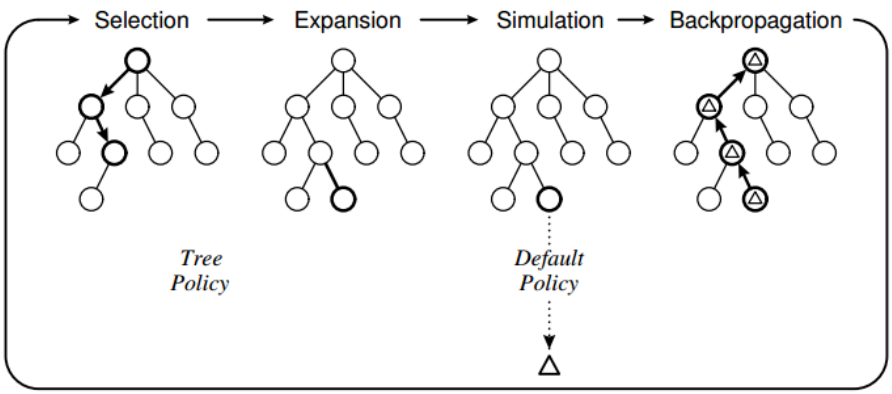

Figure 3.1: The phases of Monte Carlo Tree Search algorithms. Source: Browne et al. (2012)

UCT pseudo-code

The main contribution of UCT is the *tree policy* that the authors use in the selection phase of the MCTS loop: each step in the descent of the tree is

treated as a multi-armed bandit instance, therefore choosing at each state the action with the highest upper confidence bound to the cumulative reward. Despite being just one of the possible implementations of a Monte Carlo Tree Search algorithm, in the following we will refer to UCT as MCTS as many authors have done in the subsequent literature.

The pseudo-code of UCT is reported in algorithm 3.2.

Algorithm 3.2 UCT

```
procedure UCT(s)
    repeat
        SEARCH(s, 0)
    until Timeout
    return BEST-ACTION(s, 0)
end procedure

procedure SEARCH(state, depth)
    if state is terminal then
        return 0
    end if
    if state is a leaf of the lookahead tree then
        return EVALUATE(state)
    end if
    action ← argmax_a UPPER-BOUND(state, a)
    nextState, reward ← SIMULATE-ACTION(state, action)
    q ← reward + γSEARCH(nextState, depth + 1)
    UPDATE-VALUE(state, action, q, depth)
    return q
end procedure
```

It can be noticed that the lookahead tree is recursively descended by the SEARCH procedure and that at each state the action a with the highest value of UPPER-BOUND(s, a) is selected.

The EVALUATE$(state)$ function usually runs a simulation from the given state using the *default policy* to select the actions until either the time horizon a or terminal state is reached, and returns the discounted cumulative reward that was collected. An example of default policy is to choose the next action randomly.

The UPPER-BOUND function is defined as follows:

$$\text{UPPER-BOUND}(s,a) = \widehat{Q}(s,a) + 2C_p\sqrt{\frac{\ln N_{s,d}}{N_{s,a,d}}}$$

Where C_p is an appropriate constant, $N_{s,d}$ is the number of times state s was visited at depth d and $N_{s,a,d}$ the number of times action a was selected from state s at depth d.

UCT performance

As the available budget increases, the action proposed by UCT after the search is theoretically guaranteed to converge to the optimal one.
UCT shows great performance improvements against its competitors in the domain of *discrete* stochastic MDPs, while it still fails to achieve good performance in the case of *continuous* MDPs.

3.2.3 MCTS with Double Progressive Widening

Due to the bandit approach at each decision epoch, traditional MCTS fails to perform well when the state and/or action spaces are continuous and the transitions are stochastic: the main reason for this failure is that the stochastic transitions over a continuous state space will never reach the same state twice, making an upper confindence bound approach impossible.
Couetoux and Doghmen (2011) tackle the problem by adding two *progressive widening* layers to plain MCTS: basically, the number of children of each node of the tree is artificially limited by a *sub-linear* function of its number of visits. This artificial limit is of course not fixed: as a node is visited more times the limit will become higher and the tree will *progressively widen*.
In order to apply these progressive widening layers, the lookahead tree is composed by two kinds of nodes: the *decision nodes*, which represent states in which the agent is required to select an action, and the *chance nodes*, which represent the stochastic transition to a next state after choosing an action. The children of a decision node DN_s representing state s are the actions $a_1, ..., a_{k_s}$ available in state s. Once one action a has been selected, a chance node $CN_{s,a}$ representing the pair (s,a) is created, if not already present, and added to the children of DN_s. The children of the chance node $CN_{s,a}$ are the decision nodes representing the next states sampled from the *generative model* according to $\mathcal{P}(\cdot|s,a)$. A chance node could in principle have an infinite number of children if the transition is to a continuous state: for this reason, the second progressive widening layer is introduced to not allow it to grow more than the agent can reasonably explore.

The choice of the action

The first progressive widening layer is needed to deal with the infinite number of actions available at each state, as it would be impossible to try them all. Progressive widening for action selection had already been proposed by Coulom (2006) and had been implemented by Rolet et al. (2009) in their *BAAL* algorithm.

In order to restrict the exploration only to some of the actions, an initial exploration order among the actions is fixed: $(a_1, a_2, ...)$. Computationally, it is not necessary to store the order into a data structure, but rather it is sufficient be able to generate the sequence of actions one by one so that the next action in the sequence can be obtained if required.
The action selection works as follows: each state s can try at most k_s actions and, every time an action needs to be chosen, only the first k_s ones will be considered. Therefore the algorithm will pick the most promising action among $(a_1, ..., a_{k_s})$.
At each node s, the number k_s of possible actions that the algorithm must consider is computed as

$$k_s = \lceil C N_s^{\alpha} \rceil ,$$

where the constants $C > 0$ and $\alpha \in (0, 1)$ can be used to tune the exploration and N_s is the number of visits to node s.

When this technique is combined with the selection strategy of UCT, whenever an action must be selected in a node s the following steps happen:

1. k_s is computed
2. the number of different actions which have been previously tried from s is compared with k_s

If the already-tried actions are *less* than k_s, a new action is added to the tree and selected. Conversely, if k_s different actions have already been tried from s, a standard multi-armed bandit technique is employed to select the action among the already-present ones.
The expression "Progressive Widening" thus refers to the widening of the set of actions that the algorithm will consider as decision nodes are visited more and more. This uses the intuition that nodes that are visited more often are deemed more promising by the UCT selection, and as a consequence a more thorough exploration of the action space is needed.

The stochastic transition

The second layer of progressive widening is needed to deal with the stochastic transitions: while Monte Carlo Tree Search needs to visit the most promising nodes many times, if the transitions are stochastic the probability of ending in the same state twice could be null (this is exactly the case if the state space is continuous), needing a (potentially) infinite number of rounds to converge.

The restriction of the stochastic transitions works as follows: the number of next states that can be reached after choosing action a at node s is limited by the number $k'_{s,a}$. This number will increase as the action a is selected more times at node s, to ensure that the transitions of seemingly good actions are sufficiently explored. The algorithm will enforce this limit by keeping a list $l_{s,a}$ of the states that have been reached after playing action a in node s.

Whenever action a is selected from state s, the following steps happen:

1. $k'_{s,a}$ is computed

2. the number of states in $l_{s,a}$ is compared with $k'_{s,a}$.

If *less* than $k'_{s,a}$ states are present in $l_{s,a}$, a transition is sampled according to $\mathcal{P}(\cdot|s,a)$ and the resulting state is added to the list if not already present. Conversely, if $l_{s,a}$ already contains $k'_{s,a}$ states, one of them is chosen with probability $p(s') = \frac{N_{s,a,s'}}{N_{s,a}}$, where $N_{s,a}$ is the number of times action a was selected at node s and $N_{s,a,s'}$ is the number of times the node s' was sampled from $\mathcal{P}(\cdot|s,a)$ as the next state after choosing action a at node s.
The number k' is computed as

$$k'_{s,a} = \left\lceil CN^{\beta}_{s,a} \right\rceil,$$

where the constant C is the same of the action selection and $\beta \in (0,1)$ can be used to control the widening speed.

The introduction of the second progressive widening layer leads to great performance improvements with respect to single progressive widening on MDPs with stochastic transitions to many possible next states.

3.3 Policy Search Algorithms

3.3.1 OPTIMIST

Another approach to online planning in continuous state/action MDPs is that of *Policy Search*, where the agent looks for an optimal policy $\pi_{\boldsymbol{\theta}*}$ to play on the real environment among a family of policies, usually parametrized by a vector $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^m$. Papini et al. (2019) propose *OPTIMIST* (Optimistic Policy opTImization via Multiple Importance Sampling with Truncation), an algorithm that performs the policy search with an optimistic exploration approach, exploiting the inherent *structure* of the problem which, in this case, can be intuitively expressed as "Similar policies will have similar returns".

Action-based versus Parameter-based Policy Optimization

Their algorithm is designed to be adapted both to *action-based* policy optimization, where the goal is to maximize the cumulative expected reward of the chosen policies, and *parameter-based* policy optimization, where the agent is working on *hyperpolicies* (policies defined over the policies) and the goal is to maximize the cumulative expected reward of the chosen hyperpolicies.

Performance measures

Given a policy $\pi_{\boldsymbol{\theta}}$, one can run an *episode* choosing actions according to such policy. A trajectory τ is the sequence $(s_0, a_0, s_1, a_1, ..., s_{H-1}, a_{H-1})$ of states encountered and actions chosen (the authors consider only the *finite* trajectories of length H). Since the environment (and, potentially, also the policy) is stochastic, each policy induces a distribution over the trajectories, whose density is denoted as $p_{\boldsymbol{\theta}}$.
The discounted reward over the trajectory τ is defined as:

$$R(\tau) = \sum_{h=0}^{H-1} \gamma^h r_{h+1}.$$

From the reward they define the expected performance $J_{\boldsymbol{\theta}}$ under the policy $\pi_{\boldsymbol{\theta}}$ as:

$$J(\boldsymbol{\theta}) = E_{\tau \sim p_{\boldsymbol{\theta}}} \left[R(\tau) \right].$$

This is the performance measure employed in the *action-based* policy optimization paradigm, where the goal of the learner is to maximize the cumu-

lative expected reward of the chosen policies:

$$\max_{\boldsymbol{\theta}_0,..,\boldsymbol{\theta}_T} \sum_{t=0}^{T} E_{\tau_t \sim p_{\boldsymbol{\theta}_t}} \left[R(\tau_t) \right] = \max_{\boldsymbol{\theta}_0,..,\boldsymbol{\theta}_T} \sum_{t=0}^{T} J(\boldsymbol{\theta}_t).$$

When working with the *parameter-based* policy optimization paradigm, they define probability distributions over policy parameters $\nu_{\boldsymbol{\xi}}$ called *hyperpolicies*, where each hyperpolicy is parametrized by the vector $\boldsymbol{\xi} \in \Xi \subseteq \mathbb{R}^d$. $\boldsymbol{\xi}$ are the hyperpolicy parameters, or *hyperparameters*.
For each episode t, a vector of hyperparameters $\boldsymbol{\xi}_t$ is selected. The policy parameters $\boldsymbol{\theta}_t$ are then drawn from $\nu_{\boldsymbol{\xi}_t}$: $\boldsymbol{\theta}_t \sim \nu_{\boldsymbol{\xi}_t}$ and a new trajectory is obtained using $\pi_{\boldsymbol{\theta}_t}$.
The objective in this case is to maximize the sum over the rounds of the expected rewards of the hyperpolicies chosen by the agent:

$$\max_{\boldsymbol{\xi}_0,..,\boldsymbol{\xi}_T} \sum_{t=0}^{T} E_{\boldsymbol{\theta}_t \sim \nu_{\boldsymbol{\xi}_t}} \left[J(\boldsymbol{\theta}_t) \right] = \max_{\boldsymbol{\xi}_0,..,\boldsymbol{\xi}_T} \sum_{t=0}^{T} J(\boldsymbol{\xi}_t).$$

Multiple Importance Sampling Estimation

To estimate $J(\boldsymbol{\xi})$ (or $J(\boldsymbol{\theta})$ in the action-based paradigm), OPTIMIST employs the *Balance Heuristic estimator*, with a truncation schedule M_t rather than a fixed threshold. Let us focus on the *parameter-based* scenario and let $\boldsymbol{\Phi}_t$ be the *mixture* of the hyperpolicies chosen up until round t. The mixture $\boldsymbol{\Phi}_t$ is itself a distribution over the policy parameters, defined as $\boldsymbol{\Phi}_t(\boldsymbol{\theta}) = \frac{1}{t}\sum_{i=0}^{t} \nu_{\boldsymbol{\xi}_i}(\boldsymbol{\theta})$. Intuitively, it represents the *average hyperpolicy* employed by the algorithm up to round t.

The authors also show that, when performing *truncated multiple importance sampling* estimation, under the assumptions of 2.5.3, if an *adaptive truncation* is used, a *confidence interval* for $\breve{\mu}_{BH}$ (or equivalently, for μ) can be computed as a function of M, $d_{1+\epsilon}(P\|\boldsymbol{\Phi})$ and $\|f\|_\infty$.

In the case with $\epsilon = 1$, if $M_N = \sqrt{\frac{N d_2(P\|\boldsymbol{\Phi})}{\log \frac{1}{\delta}}}$ is used, then with probability at least $1-\delta$ it holds

$$\mu \leqslant \breve{\mu}_{BH} + \|f\|_\infty \left(\sqrt{2} + \frac{4}{3}\right) \sqrt{\frac{d_2(P\|\boldsymbol{\Phi}) \log \frac{1}{\delta}}{N}}$$

Applying this result to the estimation of $J(\boldsymbol{\xi})$, denoting with $\breve{J}_{BH,t}(\boldsymbol{\xi})$ the corresponding truncated multiple importance sampling with balance heuris-

tic estimator, the upper confidence bound

$$B_t(\boldsymbol{\xi}, \delta_t) = \check{J}_{BH,t}(\boldsymbol{\xi}) + \|f\|_\infty \left(\sqrt{2} + \frac{4}{3}\right) \sqrt{\frac{d_2(\nu_{\boldsymbol{\xi}} \| \boldsymbol{\Phi}_t) \log \frac{1}{\delta_t}}{N}}$$

is computed and, according to the OFU principle, the hyperpolicy with the highest upper bound is selected.

3.3.2 The Regret

Let us focus on the case $\epsilon = 1$, in the *action-based* paradigm. Let the regret after T rounds be defined as

$$R(T) = TJ(\boldsymbol{\theta}^*) - \sum_{t=0}^{T} J(\boldsymbol{\theta}_t)$$

Assuming that the policies have finite $d_2(p_{\boldsymbol{\theta}_1} \| p_{\boldsymbol{\theta}_2}) \leqslant v < +\infty$, in the case of *discrete* Θ, OPTIMIST suffers a regret

$$R(T) \in O(\sqrt{T}).$$

In the case $\Theta \subseteq [-D, D]^d$, under the assumption that $J(\cdot)$ is *Lipschitz*, OPTIMIST suffers a regret

$$R(T) \in O(\sqrt{dT}).$$

In the case in which $\Theta \subseteq [-D, D]^d$ is continuous, the authors propose a version of OPTIMIST which implements, at each round, an adaptive discretization of Θ to derive a finite set of policies. In this final case, under the assumption that $J(\cdot)$ is *Lipschitz*, the regret is

$$R(T) \in O(d\sqrt{T}).$$

3.4 Finite Bandit Algorithms

3.4.1 UGapE

A Unified Approach

In the context of best-arm identification in finite bandits, one of the main state-of-the-art algorithms is UGapE (Unified Gap-based Exploration) (Gabillon et al., 2012). In the literature, the problem has been formulated both in terms of *fixed budget*, where the learner must yield the recommendation after a fixed number n of rounds, and *fixed confidence*, where the learner

must yield a the recommendation, with at least a fixed statistical confidence $1 - \delta$ of having found the best arm, within the smallest number of rounds. The authors developed a meta-algorithm, UGapE, to solve the best-arm-identification problem for both these frameworks with minimal changes to the implementation, as can be seen from the simplified pseudo-code reported in algorithms 3.3 and 3.4.

The Task of m ϵ-best-arms identification

UGapE solves the more difficult m ϵ-best-arms-identification problem, where the learner must find a set of m arms such that the simple regret of the worst of the arms is at most ϵ. In this case, the simple regret r_k of arm k is defined as $\mu_{(m)} - \mu_k$, where the notation $\mu_{(m)}$ indicates the mean reward of the m-th-best arm. Of course, the traditional problem of best-arm-identification is a particular instance of this problem where $m = 1$ and $\epsilon = 0$.

Algorithm 3.3 UGapEb

```
procedure UGapEb(ε, m, n, a)
    Pull each arm once                          ▷ Initial Round-Robin
    t ← number of arms
    while t < n do
        SELECT-ARM(t)
        t ← t + 1
    end while
    return Ω(n) = argmin_{J(t)} R_{J(t)}(t)
end procedure
```

The Arm Selection

The procedure SELECT-ARM(t), which constitutes the core of the algorithm, is implemented in the same way for both the settings. First, the quantities $U_k(t), L_k(t)$ are computed for each arm k as $U_k(t) = \widehat{\mu}_k(t) + \beta_k(t)$, $L_k(t) = \widehat{\mu}_k(t) - \beta_k(t)$. Quite intuitively they represent, respectively, an upper and a lower bound to the mean μ_k of arm k. Then, the index $R_k(t)$ is computed as the difference between the m-th highest upper bound among the other arms and $L_k(t)$. In formulas, $R_k(t) = \max_{i \neq k}^{(m)} U_i(t) - L_k(t)$. This index is an upper bound to the simple regret of arm k. Then, the set $J(t)$ is computed as the set of the m arms with the smallest values of $R_k(t)$. The arms u_t, l_t are then computed as $u_t = \text{argmax}_{j \notin J(t)} U_j(t)$ and $l_t = \text{argmin}_{i \in J(t)} L_i(t)$.

Algorithm 3.4 UGapEc

```
procedure UGapEc(ε, m, δ, c)
    Pull each arm once                          ▷ Initial Round-Robin
    t ← number of arms
    while R_{J(t)}(t) ⩾ ε do
        SELECT-ARM(t)
        t ← t + 1
    end while
    return J(t)
end procedure
```

They represent the best possible arm that was left out of $J(t)$ and the worst possible arm that was included.
Once u_t, l_t have been computed, the most uncertain arm among the two (the one with the largest $\beta(t)$) is pulled. The pseudo-code for the procedure SELECT-ARM(t) is reported in algorithm 3.5.

The bound radius $\beta_k(t)$ is defined differently for the two settings. In the fixed-budget setting we have

$$\beta_k(t) = b\sqrt{\frac{a}{T_k(t)}},$$

where the rewards of the arms are assumed to be in $[0, b]$, $T_k(t)$ is the number of times arm k was pulled and a is a tunable parameter.

In the fixed-confidence setting we have

$$\beta_k(t) = b\sqrt{\frac{c\log\left(\frac{4Kt^3}{\delta}\right)}{T_k(t)}},$$

Algorithm 3.5 UGapE, arm selection

```
procedure SELECT-ARM(t)
    Compute R_k(t) for each arm
    Identify the set J(t) of m arms with smallest R(t)
    Identify the arms u_t and l_t
    Pull the arm I(t) = argmax_{i∈{u_t,l_t}} β_i(t)
    Observe the reward, update the history of pulls and rewards
end procedure
```

where c is a tunable parameter which can also be set to the default, theoretically recommended value of 0.5.

Performance

The performance of UGapE is backed up by strong theoretical guarantees in both the fixed budget and fixed confidence settings: in the fixed budget setting, the probability of returning the wrong set of arms decays exponentially in the budget n according to the relation

$$\widetilde{\delta} = P(r_{\Omega(n)} \geqslant \epsilon) \leqslant 2Kne^{-2a}.$$

In the fixed confidence setting, the number of ruonds $\widetilde{n}$ necessary to yield the solution is upper bounded according to the relation

$$P\left(r_{\Omega(\widetilde{n})} \leqslant \epsilon \wedge \widetilde{n} \leqslant K + O\left(H_\epsilon \log \frac{H_\epsilon}{\delta}\right)\right) \geqslant 1 - \delta,$$

where H_ϵ is the *complexity measure* of the m ϵ-best arms identification problem, defined by the authors as

$$H_\epsilon = \sum_{i=1}^{K} \frac{b^2}{\max\left\{\frac{\Delta_i+\epsilon}{2}, \epsilon\right\}^2},$$

$$\Delta_k = \begin{cases} \mu_k - \mu_{(m+1)} & \text{If } k \text{ is in the top } m \text{ arms} \\ \mu_{(m)} - \mu_k & \text{If } k \text{ is in not in the top } m \text{ arms} \end{cases}.$$

3.5 Continuouos bandit Algorithms

3.5.1 HOO

The Hierarchical Optimistic Optimization strategy, developed by Bubeck et al. (2011), tackles continuous bandits by incrementally building an estimate of the mean reward function f over $\mathcal{X}$. The estimate will be more precise around the maxima while it will be less precise where the observed reward was low, as it can be observed in Figure 3.3.
To achieve this incremental accuracy, HOO maintains a binary tree whose nodes represent regions of $\mathcal{X}$, such that the union of all the nodes covers all $\mathcal{X}$ and that deeper nodes in the tree are associated with smaller regions. To each node is associated an index B representing an optimistic estimate to the maximum mean reward that can be obtained in that region. The B index is used to select the leaf from which a point $x \in \mathcal{X}$ (an arm) will be

chosen and pulled.

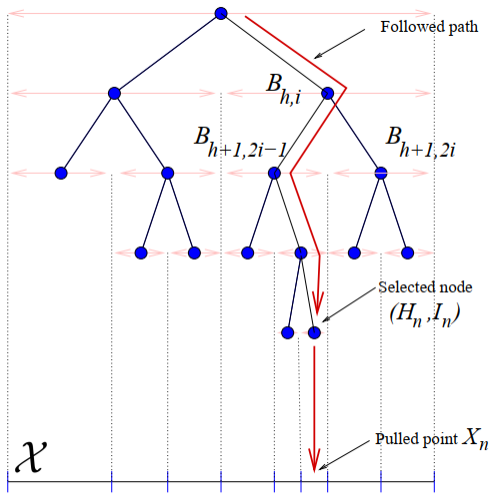

Figure 3.2: The covering of $\mathcal{X}$ and the selection of the node to expand performed by HOO. Source: Bubeck et al. (2011).

The pseudo-code of HOO is reported in algorithm 3.6.
The B values are propagated from the leaves at each round. When a node is a leaf, the B values of the children are assumed to have the default value $+\infty$. The backpropagation rule is as follows:

$$a.B = \min\{a.U, max\{a.leftchild.B, a.rightchild.B\}\}$$

The intuition is that if the B values of the children of node a are valid upper bounds to the mean reward in the regions that they cover, since the region covered by a is the union of those regions then max $\{a.leftchild.B, a.rightchild.B\}$ is a valid upper bound to the mean reward of the region covered by a. Furthermore, since also $a.U$ is a valid upper bound in the region covered by a, the tightest bound is given by the minimum of the two quantities.

Algorithm 3.6 HOO

```
procedure HOO
    for t = 1, ..., n do
        node ← root
        while node is not a leaf do                          ▷ tree descent loop
            if node.leftchild.B > node.rightchild.B then
                node ← node.leftchild
            else if node.leftchild.B < node.rightchild.B then
                node ← node.rightchild
            else
                node ← random(node.leftchild, node.rightchild)
            end if
        end while
        node ← random(node.leftchild, node.rightchild)
        𝒯 ← 𝒯 ∪ {node}                                       ▷ add the new node
        choose x ∈ node.region                               ▷ the selection is arbitrary
        obtain reward y from the environment
        for a in ancestors(node) ∪ {node} do
            a.T ← a.T + 1
            a.μ̂ ← (1 − 1/a.T) a.μ̂ + y/a.T                    ▷ update the mean
        end for
        for a in 𝒯 do
            a.U ← a.μ̂ + sqrt(2 ln t / a.T) + ν1 ρ^h          ▷ Update the U values
        end for
        backpropagate the B values from the leaves to the root
    end for
end procedure
```

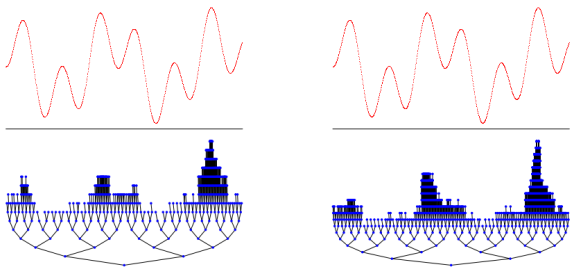

Figure 3.3: The HOO binary tree after 1,000 (left) and 10,000 (right) rounds. In the top part, the mean reward function is shown. Source: Bubeck et al. (2011).

Chapter 4

A Randomized Approach to Best Arm Identification in Continuous Bandits

4.1 Summary

We propose a randomized algorithm for the *continuous bandit* framework. Our algorithm derives a discrete set of meta-arms (probability distributions over the continuous arms), which share an artificially-induced form of *mediator feedback* (Metelli et al., 2020), entirely by construction. Then, we tackle the problem of finding the best meta-arm as a multi-armed bandit problem. We use the Truncated Balance Heuristic Estimator to extract information from the mediator feedback, in order to effectively use the samples from all the meta-arms. In the last Section, we provide some theoretical results concerning our algorithm and its performance.

4.2 The Framework

A continuous bandit problem is a pair $\langle \mathcal{X}, M \rangle$ (Bubeck et al., 2011) where $\mathcal{X}$ is a measurable space (in our case, $\mathcal{X} = [a_1, b_1] \times ... \times [a_d, b_d] \subseteq \mathbb{R}^d$) and M is a mapping assigning to every point $x \in \mathcal{X}$ a probability distribution over the reals, with mean $f(x)$. At every decision epoch t, the learner will choose an arm X_t and will receive a reward Y_t sampled from $M(X_t)$. We assume without loss of generality that $Y_t \in [0, 1]$.
Let:

$$f^* = \sup_{x \in \mathcal{X}} f(x),$$

be the expected reward of the best arm. At round n, the *cumulative pseudo-regret* of the learner who selected the sequence of arms $(X_1, ..., X_n)$ is defined as:

$$R(n) = nf^* - \sum_{t=1}^{n} f(X_t).$$

For the best-arm-identification task, assuming that Z_n is the recommendation of the learner after n rounds, the *simple regret* is defined as:

$$r(n) = f^* - f(Z_n).$$

4.3 The Meta-arms

Our algorithm first computes a special form of *discretization* of the set of arms $\mathcal{X}$: we call this *finite* set $\widehat{\mathcal{X}}$, the set of the *meta-arms*. To each meta-arm $\widehat{x} \in \widehat{\mathcal{X}}$ is associated a *probability distribution* $p_{\widehat{x}}$ on the original set of arms $\mathcal{X}$. Intuitively, in order to pull the meta-arm $\widehat{x}$, an arm x is sampled from $p_{\widehat{x}}$ and is subsequently pulled from the environment. The reward obtained is used to estimate the expected reward of the meta-arm $\widehat{x}$ and of all the meta-arms that are *similar* to it thanks to the *mediator feedback*.

4.3.1 The Mediator Feedback

Whenever a meta-arm $\widehat{x}$ is pulled, an arm x is sampled according to its probability density function $p_{\widehat{x}}$ and pulled, obtaining the reward y. The reward y can be used to estimate the expected reward not only of $\widehat{x}$, but also of every other meta-arm that puts positive probability mass on the arm x. This allows to use the collected samples to estimate the expected reward of an arbitrary large set of meta-arms, by keeping track both of the return y and of the arm x that was sampled each time a meta-arm $\widehat{x}$ is pulled. Metelli et al. (2020) define the return pair $(x, y(x))$ as "Mediator feedback", the rationale being that the arm x *mediates* between the pulled meta-arm $\widehat{x}$ and the obtained reward y. In the following, let us denote the expected reward of meta-arm $\widehat{x}$ with $\mu(\widehat{x})$.
Our algorithm, therefore, performs best-arm identification on the set of meta-arms $\widehat{X}$, exploiting the information that can be obtained with the mediator feedback.

4.3.2 The Probability Density Function $p_{\widehat{x}}$

We employ multi-dimensional Gaussian meta-arms, located so that the space $\mathcal{X}$ is covered in a grid fashion and each meta-arm has its mean on a different

point of the grid. The reader may find it clarifying to check Figure 5.6 in Chapter 5 for a graphical example of such covering. The probability density functions $\{p_{\hat{x}}\}$ associated to our meta-arms are multivariate normal distributions over $\mathcal{X}$. First, we choose the standard deviation σ_l for each dimension $l = 1, ..., d$ of the arms domain, then we set the number of meta-arms to place along each dimension. This is set indirectly through the parameter s_l, called "standard deviations to neighbour": the meta-arms are placed at the extreme points of $\mathcal{X}$ and evenly distributed inside so that the means of two neighbour meta-arms along dimension l differ at most $2s_l\sigma_l$. The result is a multi-dimensional grid covering $\mathcal{X}$ so that to each point of the grid $x = (x_1, ..., x_d)$ is associated a multivariate normal distribution with mean $(x_1, ..., x_d)$ and covariance matrix $\begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_d^2 \end{bmatrix}$.

4.4 The Algorithm

Our algorithm is similar the majority of *finite* bandit algorithms that can be found in the literature: until there is available budget some indices are computed and used to select the next arm to pull and the corresponding reward is collected. At the end, the algorithm outputs a recommendation of what the best arm is. The main difference is that our algorithm does not work with the arms, which in this case are infinite, but rather works on the meta-arms. It would be correct to say that our algorithm performs best-meta-arm identification.

4.4.1 The Construction of the Meta-arms

The very first step performed by our algorithm is building the set of meta-arms according to the given ranges $[a_1, b_1], ..., [a_d, b_d]$ of the arms set and to the parameters $(\sigma_1, s_1), ..., (\sigma_d, s_d)$. In particular, for each dimension $l = 1, ..., d$ the number of grid points is computed as $g_l = \left\lceil \frac{b_l - a_l}{2\sigma_l s_l} \right\rceil$. Then, the domain D_l is computed as the set of g_l uniformly spread points in $[a_l, b_l]$ including the extremes. The set of the means of our meta-arms is therefore $\mathcal{M} = D_1 \times ... \times D_d$. Clearly, $\mathcal{M} \subseteq \mathcal{X}$. For each point $m \in \mathcal{M}$, one meta-arm $\hat{x}_m$ is built such that $p_{\hat{x}_m} = \mathcal{N}\left(m, \begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_d^2 \end{bmatrix}\right)$. Once the meta-arms are built, the best-meta-arm identification process can begin.

4.4.2 The Choice of the First Meta-arm

The first meta-arm to be pulled is arbitrarily set to be the most central one: it is the meta-arm such that along each dimension $l = 1, ..., d$ the mean is the median of D_l.

4.4.3 The Pull of a Meta-arm

Whenever our algorithm *pulls* a meta-arm $\widehat{x}$, the environment is eventually sampled in a point which is likely to be around the mean of $p_{\widehat{x}}$. More precisely, a point in the domain $x \in \mathcal{X}$ is sampled according to $p_{\widehat{x}}$. Then, the environment is queried in the point x, obtaining the reward y. The pulled meta-arm $\widehat{x}$ is added to the history of pulled meta-arms, as well as the pair (x, y) which constitutes the *mediator feedback*.

4.4.4 The Selection of the Next Meta-arm

After at least one meta-arm has been pulled, the algorithm must select after each round which meta-arm to pull based on the available data. We do this in two ways: our algorithm can in fact be set to work in two modes. The first mode is the *optimistic* mode: at each round, the meta-arm with the highest upper-bound is selected. The second mode is the *UGapE* mode, which employs the selection logic of the UGapE algorithm (Gabillon et al., 2012), pulling the meta-arm that will contribute most to the identification of a best meta-arm among the empirically best. This second procedure is more complex and is described in detail in Section 4.4.7.

4.4.5 The Final Recommendation

When all the budget has been consumed, our algorithm will search in the *continuous* space of all the possible meta-arms, therefore also the meta-arms corresponding to points which were not present in the grid, for a local maximum.

4.4.6 The Confidence Bounds

Like in most bandit algorithms, the choices are driven by statistical confidence bounds computed from the available data. Given the history $\mathcal{H}_t = (\widehat{x}_1, x_1, y_1, ..., \widehat{x}_t, x_t, y_t)$ available before round $t + 1$, we compute the confidence bounds for the mean reward $\mu(\widehat{x})$ of each meta-arm $\widehat{x}$. Let $\mathbf{\Phi}_t$ be the *mixture* of the probability density functions of the meta-arms pulled up to round t. $\mathbf{\Phi}_t$ is itself a probability density function and is defined

by $\mathbf{\Phi}_t(x) = \frac{1}{t}\sum_{i=1}^{t} p_{\widehat{x}_t}(x)$. $\mathbf{\Phi}_t$ represents the probability density function of the "average meta-arm" that was pulled up to round t. The statistical properties of our estimators are expressed as a function of $d_2(p_{\widehat{x}} \| \mathbf{\Phi}_t)$ (see section 2.5.3). However, computing such divergence is prohibitive in terms of computational cost, as there is no known closed form and the only option is to compute the integral numerically. To circumvent this issue, we use the following harmonic mean upper-bound of $d_2(p_{\widehat{x}} \| \mathbf{\Phi}_t)$ (Papini et al., 2019):

$$d_2(p_{\widehat{x}} \| \mathbf{\Phi}_t) \leqslant \frac{t}{\sum_{i=1}^{t} \frac{1}{d_2(p_{\widehat{x}} \| p_{\widehat{x}_i})}} = d_{2,hm,t}(\widehat{x}).$$

For each meta-arm $\widehat{x}$ we compute the *adaptive truncation threshold*

$$M_t(\widehat{x}) = \sqrt{\frac{t d_{2,hm,t}(\widehat{x})}{\alpha \log(t+1)}},$$

the *truncated importance weight* of each sample i, for $i = 1, ..., t$

$$\breve{w}(\widehat{x})_i = \min\left\{M_t(\widehat{x}), \frac{p_{\widehat{x}}(x_i)}{\mathbf{\Phi}_t(x_i)}\right\},$$

the *truncated balance heuristic estimator*

$$\breve{\mu}_{BH,t}(\widehat{x}) = \frac{1}{t}\sum_{i=1}^{t} \breve{w}(\widehat{x})_i y_i,$$

the *confidence bound radius*

$$B_t(\widehat{x}) = (1+\sqrt{2})\sqrt{\frac{\alpha \log(t+1) d_{2,hm,t}(\widehat{x})}{t}},$$

and finally the confidence interval

$$\breve{\mu}_{BH,t}(\widehat{x}) - B_t(\widehat{x}) \leqslant \mu(\widehat{x}) \leqslant \breve{\mu}_{BH,t}(\widehat{x}) + B_t(\widehat{x}).$$

The expressions of the truncation threshold and of the confidence bound radius are the ones employed by Metelli et al. (2020). The term α in the radius expression is a tunable parameter such that $\alpha > 1$ which can be used to control the amount of exploration: the greater α, the larger the bounds, the more the algorithm will try potentially sub-optimal meta-arms.

4.4.7 The Pseudo-Code

In Algorithm 4.1 is reported the pseudo-code of our algorithm.

Algorithm 4.1 The main loop of our algorithm

```
procedure TMIS-BEST-ARM-IDENTIFICATION(𝒳, (σ_1, ..., σ_d), (s_1, ..., s_d))
    𝒳̂ ← BUILD-META-ARMS(𝒳, (σ_1, ..., σ_d), (s_1, ..., s_d))
    x̂_1 ← SELECT-FIRST-META-ARM(𝒳̂)
    ℋ_1 ← PULL-META-ARM(x̂_1)
    t ← 2
    while t ⩽ n do
        x̂_t ← SELECT-NEXT-META-ARM(𝒳̂, ℋ_{t−1})
        ℋ_t ← CONCATENATE(ℋ_{t−1}, PULL-META-ARM(x̂_t))
    end while
    return FIND-BEST-ARM(ℋ_n)
end procedure

procedure PULL-META-ARM(x̂)
    sample x ∼ p_x̂
    sample y from the bandit environment (y ∼ M(x))
    return (x̂, x, y)
end procedure
```

The Selection of the Next Meta-arm

For the choice of the next meta-arm to pull we explored two possibilities, both using the confidence bound for the *truncated balance heuristic* estimator.

Optimistic Meta-arm Selection This selection technique is the UCB-like optimistic strategy employed by Papini et al. (2019) in OPTIMIST: at each round, the meta-arm with the highest upper bound is selected.

$$\widehat{x}_i = \operatorname*{argmax}_{\widehat{x} \in \widehat{\mathcal{X}}} \breve{\mu}_{BH,t}(\widehat{x}) + B_t(\widehat{x})$$

This procedure is reported in pseudo-code in algorithm 4.2.

Algorithm 4.2 Optimistic mode selection

```
procedure SELECT-NEXT-META-ARM-OPTIMISTIC(𝒳̂, ℋ_{t−1})
    compute U_t(x̂) = μ̆_{BH,t}(x̂) + B_t(x̂) for each x̂ ∈ 𝒳̂
    return argmax_{x̂∈𝒳̂} U_t(x̂)
end procedure
```

Full-exploration Meta-arm Selection The second selection technique is the one developed by Gabillon et al. (2012) for the UGapE algorithm. Differently from the optimistic approach, the goal here is to have the confidence bound of the best meta-arm disjoint from the confidence bounds of the other meta-arms. This strategy therefore will sometimes pull the second-best arm. First, the index $R_t(\widehat{x})$ is computed for each meta-arm. This index is an upper bound on the simple regret of $\widehat{x}$. Then, identify the meta-arm with smallest R_t. Let $l_t = \operatorname{argmin}_{\widehat{x}\in\widehat{\mathcal{X}}} R_t(\widehat{x})$. Then, let u_t be the best meta-arm among the remaining: $u_t = \operatorname{argmax}_{\widehat{x}\in\widehat{\mathcal{X}},\widehat{x}\neq l_t} U_t(\widehat{x})$. Pull the most uncertain meta-arm (that is, the one with larger $B_t(\widehat{x})$) between l_t and u_t.

$$\widehat{x}_{t+1} = \operatorname*{argmax}_{\widehat{x}\in\{l_t,u_t\}} B_t(\widehat{x})$$

This procedure is reported in pseudo-code in algorithm 4.3.

Algorithm 4.3 UGapE mode selection

procedure SELECT-NEXT-META-ARM-UGAPE($\widehat{\mathcal{X}}, \mathcal{H}_{t-1}$)
  compute $U_t(\widehat{x}) = \breve{\mu}_{BH,t}(\widehat{x}) + B_t(\widehat{x})$ for each $\widehat{x} \in \widehat{\mathcal{X}}$
  compute $L_t(\widehat{x}) = \breve{\mu}_{BH,t}(\widehat{x}) - B_t(\widehat{x})$ for each $\widehat{x} \in \widehat{\mathcal{X}}$
  compute $R_t(\widehat{x}) = \max_{\substack{\widehat{x}'\in\widehat{\mathcal{X}}\\ \widehat{x}'\neq\widehat{x}}} U_t(\widehat{x}') - L_t(\widehat{x})$
  compute $l_t = \operatorname{argmin}_{\widehat{x}\in\widehat{\mathcal{X}}} R_t(\widehat{x})$
  compute $u_t = \operatorname{argmax}_{\substack{\widehat{x}\in\widehat{\mathcal{X}}\\ \widehat{x}\neq l_t}} U_t(\widehat{x})$
  return $\operatorname{argmax}_{\widehat{x}\in\{l_t,u_t\}} B_t(\widehat{x})$
end procedure

The Choice of the Recommendation

After the budget has been consumed, the algorithm must recommend an *arm* based on the collected data. This is done in the procedure FIND-BEST-ARM($\mathcal{H}_n$). In order to find the best point, our algorithm:

1. Sorts the meta-arms in decreasing order by the estimator $\mu_{BH,t}(\widehat{x})$.
2. Computes the set $\widehat{\mathcal{X}}^*$ of the $\overline{k}$ best meta-arms by taking the first $\overline{k}$. In our experiments, we employed $\overline{k} = 5$.
3. Numerically performs gradient ascent using each $\widehat{x} \in \widehat{\mathcal{X}}^*$ as a different starting point, thus considering a continuous of meta-arms including points that were not in the grid.

4. Returns the point $x \in \mathcal{X}$ such that the meta-arm centered on x was returned by the gradient ascent as local optimum and its estimator is the highest among the other local optima.

4.5 Theoretical Analysis

We provide a few theoretical results to characterize our algorithm. We consider the more general case of optimization of a *stochastic function* $\rho(x)$ defined over a set $\mathcal{X}$, with mean reward $f(x)$. The optimization is carried on by *Policy Search*, where the policies $p_{\boldsymbol{\theta}}$ are parametrized by a compact set Θ. The policies are used to decide which is the next point to sample, so that first a point $x \in \mathcal{X}$ is sampled according to the policy and then the stochastic function $\rho(x)$ is sampled in that point. The results of this analysis can be restricted to the case of our algorithm keeping in mind that the set of parameter vectors Θ coincides with the set of arms $\mathcal{X}$, the policies $p_{\boldsymbol{\theta}}$ coincide with the probability distributions of the meta-arms $p_{\hat{x}}$ and the mean reward function is $f(x)$ in both settings.

4.5.1 Setting

Let $\Theta \subseteq \mathbb{R}^m$ be a compact set, $\mathcal{X}$ be a set. For every parameter $\boldsymbol{\theta} \in \Theta$, a policy $p_{\boldsymbol{\theta}}(\cdot)$ is a probability distribution over $\mathcal{X}$. For every decision $x \in \mathcal{X}$, the reward distribution is $\rho(\cdot|x)$, whose decision expected reward is:

$$f(x) = \int_{\mathbb{R}} y\rho(\mathrm{d}y|x).$$

The policy expected reward is:

$$\mu(\boldsymbol{\theta}) = \int_{\mathcal{X}} f(x)p_{\boldsymbol{\theta}}(\mathrm{d}x). \tag{4.1}$$

The goal of the decision maker is to find the optimal parameter vector $\boldsymbol{\theta}$ (or equivalently, the optimal policy):

$$\mu^* = \sup_{\boldsymbol{\theta}\in\Theta} \mu(\boldsymbol{\theta}) \qquad \Longleftrightarrow \qquad \boldsymbol{\theta}^* \in \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \mu(\boldsymbol{\theta}). \tag{4.2}$$

Let us denote with $\boldsymbol{\theta}_n$ the parameter vector recommended at round n. Our performance index is the policy simple regret:

$$r_\mu(n) = \mu^* - \mu(\boldsymbol{\theta}_n). \tag{4.3}$$

We could also consider the decision simple regret:

$$r_f(n) = f^* - \mu(\boldsymbol{\theta}_n). \tag{4.4}$$

4.5.2 Assumptions

We list all the assumptions that we will use in the following. Not all will be employed together.

The following assumption requires that, for each pair of policies defined by the vectors $\boldsymbol{\theta}, \boldsymbol{\theta}'$ the 1-norm (the sum of the absolute values of the elements) of the vector containing the difference of the *probability density functions* $p_{\boldsymbol{\theta}'}(x) - p_{\boldsymbol{\theta}}(x)$ evaluated in each point $x \in \mathcal{X}$ is upper bounded by a *semi-metric* (a non-negative, symmetric function ℓ such that $\ell(\boldsymbol{\theta}, \boldsymbol{\theta}') = 0 \iff \boldsymbol{\theta} = \boldsymbol{\theta}'$) over $\boldsymbol{\theta}, \boldsymbol{\theta}'$. This assumption is not constraining when one is free to design the policy space. It is satisfied in the case of multivariate normal policies, as in our case.

Assumption 1 (Regularity of $p_{\boldsymbol{\theta}}(\cdot)$) *There exists a semimetric $\ell_p : \Theta^2 \to \mathbb{R}_{\geqslant 0}$ such that:*

$$\|p_{\boldsymbol{\theta}'}(\cdot) - p_{\boldsymbol{\theta}}(\cdot)\|_1 \leqslant \ell_p(\boldsymbol{\theta}, \boldsymbol{\theta}'). \tag{4.5}$$

The following assumption requires that no point $x \in \mathcal{X}$ has infinite reward.

Assumption 2 (Boundedness of $f(x)$) *There exists constant $F \in \mathbb{R}_{\geqslant 0}$ such that*

$$\sup_{x \in \mathcal{X}} |f(x)| \leqslant F. \tag{4.6}$$

The following assumption requires that the mean reward function f has some smoothness property. While in principle one could design a continuous bandit problem with non-smooth mean reward functions (for example by taking a smooth one and then setting the mean reward of a specific point to a very high value without changing that of the near points), this regularity assumption is always done when analysing a problem of online optimization of continuous functions, as bounding the simple regret without it would be impossible.

Assumption 3 (Regularity of $f(x)$) *There exists a semimetric $\ell_f : \Theta^2 \to \mathbb{R}_{\geqslant 0}$ such that:*

$$\left|f(x) - f(x')\right| \leqslant \ell_f(x, x'). \tag{4.7}$$

The following assumption is done to lighten the notation and to re-use the upper bounds existing in the literature without loss of generality. In our case, it holds by definition.

Assumption 4 *The reward $r \sim \rho(x)$ is bounded in $[0, 1]$.*

The following assumption requires that *absolute continuity* holds for each pair of policies. It is equivalent to requiring that all the policies assign zero probability to the same set of points, and positive probability to the others. In our case, since all the multivariate normal policies assign positive probability to each point, it is satisfied.

Assumption 5 *All the policies are absolutely continuous:*

$$p_{\boldsymbol{\theta}}(\cdot) \ll p_{\boldsymbol{\theta}'}(\cdot) \qquad \forall \boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta.$$

4.5.3 Theoretical Results

Regularity of μ

We show that moving to the expected value in policy space the reward function μ is regular even if f is not.

Lemma 1 *Under Assumptions 1 and 2, it holds that:*

$$|\mu(\boldsymbol{\theta}) - \mu(\boldsymbol{\theta}')| \leqslant F \ell_p(\boldsymbol{\theta}, \boldsymbol{\theta}'); \tag{4.8}$$

$$\sup_{\boldsymbol{\theta} \in \Theta} |\mu(\boldsymbol{\theta})| \leqslant F. \tag{4.9}$$

Proof 1 *For the first inequality:*

$$|\mu(\boldsymbol{\theta}) - \mu(\boldsymbol{\theta}')| = \left| \int_{\mathcal{X}} \left(p_{\boldsymbol{\theta}}(\mathrm{d}x) - p_{\boldsymbol{\theta}'}(\mathrm{d}x) \right) f(x) \right| \leqslant F \left\| p_{\boldsymbol{\theta}}(\cdot) - p_{\boldsymbol{\theta}'}(\cdot) \right\|_1 .$$

The second inequality is trivial.

Upper Bound to $f^* - \mu^*$

Under the regularity of f, the "loss" of moving to a randomized policy space is bounded:

Lemma 2 *Under Assumptions 3, it holds that:*

$$f^* - \mu^* \leqslant \inf_{x^* \in \mathcal{X} : f^* = f(x^*)} \inf_{\boldsymbol{\theta} \in \Theta} \int_{\mathcal{X}} p_{\boldsymbol{\theta}}(\mathrm{d}x) \ell_f(x^*, x).$$

Proof 2 *Let*

$$\begin{aligned} f^* &= \sup_{x \in \mathcal{X}} f(x) \\ \mu^* &= \sup_{\boldsymbol{\theta} \in \Theta} \int_{\mathcal{X}} p_{\boldsymbol{\theta}}(\mathrm{d}x) f(x) \\ \mathcal{X}^* &= \{x \in \mathcal{X} : f(x) = f^*\}, \end{aligned}$$

Then

$$f^* - \mu^* = \inf_{\boldsymbol{\theta} \in \Theta} \int_{\mathcal{X}} p_{\boldsymbol{\theta}}(\mathrm{d}x)(f^* - f(x)),$$

and in particular $\forall x^* \in \mathcal{X}^*$ *it holds:*

$$\begin{aligned} f^* - \mu^* &= \inf_{\boldsymbol{\theta} \in \Theta} \int_{\mathcal{X}} p_{\boldsymbol{\theta}}(\mathrm{d}x) \left(f(x^*) - f(x)\right) \\ &\leqslant \inf_{\boldsymbol{\theta} \in \Theta} \int_{\mathcal{X}} p_{\boldsymbol{\theta}}(\mathrm{d}x) \ell_f(x^*, x). \qquad \text{(By Assumption 3)} \end{aligned}$$

It is therefore possible to select x^* *so that upper bound is minimized, obtaining the statement of the Lemma*

Upper Bound to $r_\mu(n)$ in Expectation

Since the truncated importance sampling estimator with the balance heuristic is used, when the optimistic selection strategy is used our algorithm is similar to an instance of OPTIMIST (Papini et al., 2019). We can use the proof of the regret bound in case $\mathcal{X}$ is a compact space $\mathcal{X} \subseteq [-D, D]^d$ (Theorem 3), instantiated with $\epsilon = 1$, which states that if the confidence schedule $\delta_t = \frac{6\delta}{\pi^2 t^2 (1 + d^d t^{2d})}$ is used with the adaptive truncation threshold $M_t = \sqrt{\frac{t d_2(p_{\boldsymbol{\theta}} \| \Phi_t)}{\log(\frac{1}{\delta_t})}}$, the Cumulative Regret $R_\mu(n)$ is with probability at least $1 - \delta$:

$$R_\mu(n) \leqslant \Delta_0 + \frac{\pi^2 L D}{6} + C \sqrt{nv \left(2(d+1)\log n + d \log d + \log \frac{\pi^2}{3\delta}\right)},$$

where C is defined as

$$C = 2\left(\sqrt{2} + \frac{5}{3}\right) \|f\|_\infty,$$

Δ_0 is the simple regret of the initial arm p_0 and v is the maximum divergence between any policy and the mixture:

$$v = \sup_{\boldsymbol{\theta} \in \Theta} d_2(p_{\boldsymbol{\theta}} \| \boldsymbol{\Phi}_t).$$

We can select a time-independent value for α which ensures that the upper bounds employed by our algorithm are larger or equal to those of OPTIMIST by ensuring

$$\alpha \log(t+1) \geqslant \log \frac{1}{\delta_t},$$

which is satisfied by any $\alpha \geqslant \frac{\log \frac{\pi^2 d^d}{3\delta}}{\log 2} + 2d + 2$.

Since the upper bounds still hold, all the high-probability properties of the bounds must hold too. Adapting the proof of Theorem 3, specifically by inserting $\alpha \log(t+1)$ instead of $\log \frac{1}{\delta_t}$ in Eq. (67) of the supplementary paper, and to all the subsequent steps, we obtain that with probability at least $1-\delta$ it holds:

$$R_\mu(n) \in O(\sqrt{\alpha v n \log n}).$$

We now employ the fact that $E[r_\mu(n)] \leqslant \frac{E[R_\mu(n)]}{n}$. Supposing that after having pulled n meta-arms the algorithm's recommendation is drawn with uniform probability from the sequence of pulled meta-arms, therefore giving higher probability of being recommended to the most pulled ones, it holds that:

$$E_{Uniform}[E[r_\mu(n)]] = \frac{1}{n}\sum_{t=1}^{n} E[\Delta_t] = \frac{1}{n} E\left[\sum_{t=1}^{n} \Delta_t\right] = \frac{E[R_\mu(n)]}{n}$$

where Δ_t is the simple regret of the meta-arm pulled at round t.
Since our algorithm recommends the meta-arm with the highest estimator given the observed samples, without assigning any probability of being recomended to the empirically sub-optimal ones, it cannot do worse that the above case. Thus, we obtain the final upper bound:

$$E[r_\mu(n)] \in O(\sqrt{\alpha v \log n})$$

4.5.4 The Variance and the Lipschitz Constant

Assume that the function f is Lipschitz, that is there exists a constant L such that:

$$\|f(x_1) - f(x_2)\| \leqslant L\|x_1 - x_2\| \quad \forall x_1, x_2 \in \mathcal{X}.$$

The intuition suggests that the optimal value of σ should be related to that of L: indeed, the function g (the mean reward in the meta-arm space) is a regularized version of f. This regularization could be better understood by thinking of how g is built: in each point x, $g(x)$ is equal to a mixture of the values of f. Considering a point x' infinitely close to x, the value of $g(x')$ is obtained as the mixture of the same points (all the domain $\mathcal{X}$) but with all the weights changed by an infinitely small amount, thus resulting

in an infinitely small variation. In our algorithm, higher values of σ give higher importance to the samples from one meta-arm in the estimates of the reward of its neighbours: this means that with higher σ the expected reward of the meta-arms is more similar to that of their neighbours. The intuition suggests that introducing such artificial regularity could be beneficial if some regularity is already present in the original function f, in a way that the more f is regular, the more this artificial regularity can be exploited.

Chapter 5

Application to Online Planning

5.1 Summary

In this Chapter, we test the performance of our bandit algorithm when employed as the policy optimizer in a policy-search online planning framework. We present the Mini-golf environment as the first benchmark and the continuous, stochastic version of OpenAI Gym's MountainCar environment as second one. On both environments we test the performance of our algorithm, of our implementation of MCTS-DPW and that of a planner which employs UGapE to perform best-arm-identification on the discretized policy space corresponding to the meta-arms grid employed by our algorithm. For each environment we show the value function represented as a surface on the policy space, the parameter tuning steps and the performance with the selected parameters, providing explanations to the observed results.

5.2 The Online Planning Loop

We employ our bandit algorithm to perform *online Monte Carlo planning*. Our planner will therefore be sequentially interacting with the environment, each time being required to find the next action using a limited *budget*. Rather than looking directly for the best action, our planner will search for the best policy among a set of policies parametrized by a compact set $\Theta \subseteq \mathbb{R}^d$, which in the experimental evaluation we defined to be the family of *linear policies*. At each interaction, this policy search is approached as a continuous bandit problem, employing our randomized continuous best-arm-identification algorithm. Once a policy is found, it is used to select

the next action to choose in the environment. Our online planning loop is graphically represented in Figure 5.1.

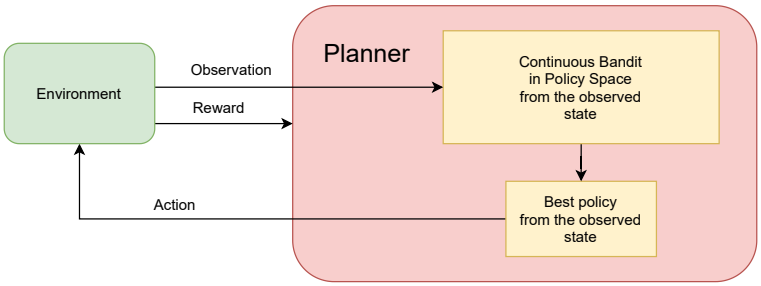

Figure 5.1: Schematics of the online planning loop of our algorithm.

5.3 Policy Search as Continuous Bandit

Thanks to the *generative model* and to the fact that the current state is known, the MDP becomes a equivalent to a *stochastic function* defined over the set of the possible policies, effectively becoming an instance of a continuous bandit problem. To make this consideration more clear, we will now introduce the "Mini-golf" environment, an MDP with continuous states and actions and with stochastic transitions which we used as an initial benchmark for our algorithm.

5.4 The Mini-golf Environment

5.4.1 Environment Definition

As the name suggests, this environment mimics the game of Mini-golf, where the player's goal is to put the ball into the hole with the minimum number of shots. The implementation of the transition function is inspired to the paper of Penner (2002), where the author studies the physics involved in the act of putting a ball. The state of this simple environment, represented in Figure 5.2, is the distance separating the ball from the hole. Therefore, $\mathcal{S} = \mathbb{R}^+$ and the action space is $\mathcal{A}_s = \mathbb{R}^+ \quad \forall s \in \mathcal{S}$, that is in each state the agent can choose a non-negative real number. At each state, the value of the chosen action represents the strength with which the ball is hit, which will make the ball progress towards the hole in a straight line. Whenever the player performs an action without putting the ball in the hole, they get a negative reward of -1. If the ball ends *inside* the hole, they get a reward

of 0 and the process stops. If the ball ends *past* the hole, they get a reward of -100 and the process stops. If the *time horizon* expires before the ball has reached the hole, the process stops. Since we have set a horizon of 20, the maximum reward achievable is $R_{max} = 0$ if the ball is put at the first shot and $R_{min} = -119$ if 19 shots do not reach the hole and the final shot throws the ball past it.

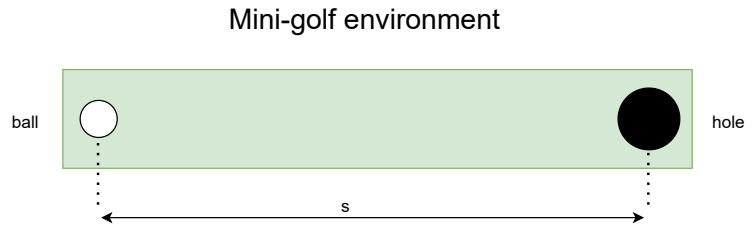

Figure 5.2: Representation of the Mini-golf environment.

Solving this environment could seem easy at a first glance, but indeed it is not straightforward. What makes it difficult is the heavy stochasticity introduced in the transition function. First, the action a chosen by the learner is clipped to the interval $[0.00001, 5]$. Then, a random noise $\xi \in (-1, 1)$ is sampled. The initial velocity of the ball is computed as $v_0 = l * a * (1 + \xi)$, where l is the *putter length*, an environment parameter which defaults to 1. This means that the action could potentially be multiplied by a factor of 2 or 0, leading to very different results. The *deceleration* α of the ball is also computed with the *friction* environment parameter, so that, if the ball is not put, the time τ before the ball stops is computed as $\tau = \frac{v_0}{|\alpha|}$ and the next state is $s_{t+1} = s_t - v_0\tau + \frac{1}{2}|\alpha|\tau^2$. The formulas reported by Penner are used to determine the minimum initial speed v_{min} and the maximum initial speed v_{max} of the ball that will make it fall into the hole. If $v_0 > v_{max}$, intuitively, the ball will go past the hole.

5.4.2 The Policies

We consider the family of linear policies: since the state is 1-dimensional, we have a 2-dimensional parameter space $\Theta \subseteq \mathbb{R}^2$. We chose to restrict the domain of both parameteres to $[0, 2]$ so that $\Theta = [0, 2]^2$. To each parameter vector $\boldsymbol{\theta} = (\theta_1 \;\; \theta_2) \in \Theta$ is associated the deterministic policy $\pi_{\boldsymbol{\theta}}(s) = \theta_1 + \theta_2 s$.

The Value Function $V_n^{\pi_{\boldsymbol{\theta}}}(s_0)$ in Policy Space

Given the initial state s_0 and the remaining horizon n, to each point $\boldsymbol{\theta} \in \Theta$ corresponds a different policy $\pi_{\boldsymbol{\theta}}$ and therefore a different value of $V_n^{\pi_{\boldsymbol{\theta}}}(s_0)$. Let us remind that the value $V_n^{\pi_{\boldsymbol{\theta}}}(s_0)$ is the expected cumulative reward if policy $\pi_{\boldsymbol{\theta}}$ is used from the state s_0 until the end of the process (or until the horizon n is reached). When we are searching the policy space, therefore, we are looking for maxima on the surface defined over Θ by $\phi(\boldsymbol{\theta}) = V_n^{\pi_{\boldsymbol{\theta}}}(s_0)$. In Figure 5.3 we show this surface for a specific initial state.

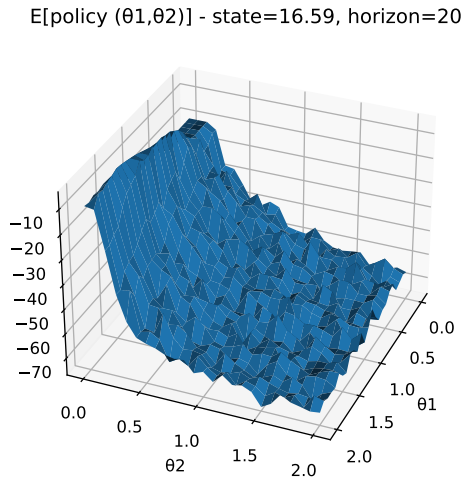

Figure 5.3: The surface $\phi(\boldsymbol{\theta})$. The approximation was obtained by covering Θ with a 25×25 grid and averaging the cumulative reward collected by each point over 400 runs

It can be noticed in Figure 5.3 that an optimal linear policy from the initial state $s_0 = 16.59$ with a remaining horizon $n = 20$ has an expected cumulative reward of about -3, indicating that no good linear policy can conveniently put the ball in less than 3 rounds (on average). This is due to the high imbalance between the negative reward of using one more round, which is -1, and that of sending the ball beyond the hole, which is -100. This disproportion, together with the high impact of the stochastic noise in the transition function, will make any learner adopt a very conservative policy.

At this point, the careful reader will have noticed the equivalence between this policy optimization and an instance of a continuous bandit problem: indeed, to turn the former into the latter it is enough to set $\mathcal{X} = \Theta$, define

the pull of one arm (or one policy) as the run of an episode using such policy and to rescale the reward of the bandit environment. Since the reward of the bandit environment is the cumulative reward of the Mini-golf environment, given the cumulative reward Y obtained from the MDP the bandit will return $y = \frac{Y - R_{min}}{R_{max} - R_{min}}$: this ensures $y \in [0, 1]$.
Once we have cast the problem as a continuous bandit instance, we can run our algorithm to find the best parameters (and, therefore, the best policy).

5.4.3 Building the Meta-arms

Selection of the Variance

In order to build the meta-arms of our algorithm we must choose appropriate values for $(\sigma_1,\ \sigma_2)$ and $(s_1,\ s_2)$. Let us remind that in the context of our algorithm applied to continuous bandits the quantity s_l is half the number of σ_l that divide the mean of two neighbour meta-arms along dimension l, and is not to be confused with the state of a MDP.
To find an appropriate value for the variance we first set $\sigma_1 = \sigma_2 = \sigma$. Then, given the optimal linear policy $\boldsymbol{\theta}^* = (1.47\ \ 0.14)$, we evaluate the degradation of the performance of $\widehat{x}^*$, the meta-arm such that $p_{\widehat{x}^*} = \mathcal{N}\left(\boldsymbol{\theta}^*, \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}\right)$, for different values of σ.
Figure 5.4 shows the degradation of the expected reward of $\widehat{x}^*$ as σ grows.

Based on such empirical observation, we chose to employ $\sigma = 0.1$. The idea behind such choice is that when the meta-arms will be built using such value of σ we can expect to have some meta-arms with an expected reward that is similar to that of $\widehat{x}^*$, even if the meta-arm grid built by our algorithm will not contain exactly $\widehat{x}^*$.

Selection of s

The tuning of the parameter s seems not to be as intuitive as that of σ: different values of s will lead to a different number of meta-arms as well as to different values of the divergence $d_2(\cdot\|\cdot)$ between the meta-arms. When constrained by a limited budget, whether it is convenient or not to increase the density of the grid is itself a difficult question: while a denser grid will lead to a finer search of the space, if the rewards are stochastic the accuracy of the estimate for each point will be generally lower. To this consideration it must be added that a denser grid will lead to smaller values of the divergence $d_2(p_{\widehat{x}_1}\|p_{\widehat{x}_2})$ between two neighbour meta-arms $\widehat{x}_1, \widehat{x}_2$, which will reduce the

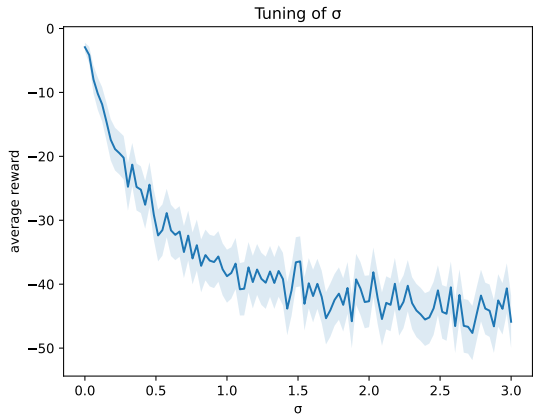

Figure 5.4: The decay of the performance due to the randomization. The 2-std ($\approx 95\%$) confidence intervals for the expected value are shown. The plot was obtained by covering the domain $[0, 3]$ of σ with a 100 points one-dimensional grid and averaging the cumulative reward for each value of σ over 400 runs.

confidence bound radius more and lead to a stronger re-usage of the samples and faster convergence of the algorithm.

Accounting for all of this complexity, we resorted to trying many values for the parameter s and taking the best one. The results are reported in Figure 5.5.

Based on such experimental evidence, we chose to set $s = 1.54$. This value will result in a smaller number of meta-arms while still maintaining roughly the same performance of 0.8, which is the point of the domain with the highest performance (by a very small amount).

Example of Meta-arms for the Mini-golf Environment

In Figure 5.6 we can see how the *probability density functions* associated to the meta-arms cover the parameter space Θ for different values of s.

5.4.4 Tuning of α

As last tuning step, we focused on the parameter α in the expression of the confidence radius. We tested the performance of our algorithm for different values of α. The results are shown in Figure 5.7.

The empirical results of this last tuning step suggest to use $\alpha = 2.5$.

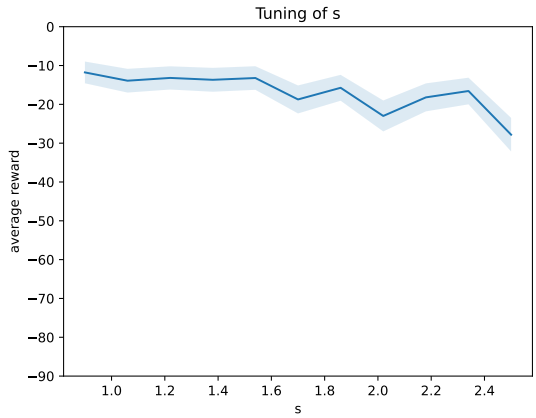

Figure 5.5: The average performance of our algorithm on the Mini-golf environment with $\sigma = 0.1$ in 500 rounds for different values of s. The plot was obtained covering the domain $[0.8, 2.6]$ of s with a 11-points one-dimensional grid, taking 400 samples for each value of s. The 2-std confidence intervals for the expected value are shown.

5.4.5 Experimental Results

The *budget*

Our algorithm needs to compute the *probability density function* of the current mixture $\mathbf{\Phi}_t$ of pulled meta-arms at every sampled point at every round. Thus, a quadratic dependence on the number of rounds is unavoidable, making our algorithm unsuitable for scenarios where a large budget is available. In our tests, we have set the maximum budget to be 500 rounds, as we have experimentally seen that the larger budget values take a long computational time. In order to compare our algorithm with other state-of-the-art solutions, we tune them for the best performance with the same budget.

Performance of Our Algorithm

With the selected parameters $\sigma = 0.1$, $s = 1.54$, $\alpha = 2.5$ we tested the average performance of our algorithm in the online planning scenario for different values of the budget for both the optimistic and UGapE-like meta-arm selection policies. The results are reported in Figures 5.8 and 5.9.

MCTS-DPW

We tested our implementation of MCTS-DPW on the Mini-golf environment. We implemented the first version of the algorithm, as described by Couëtoux

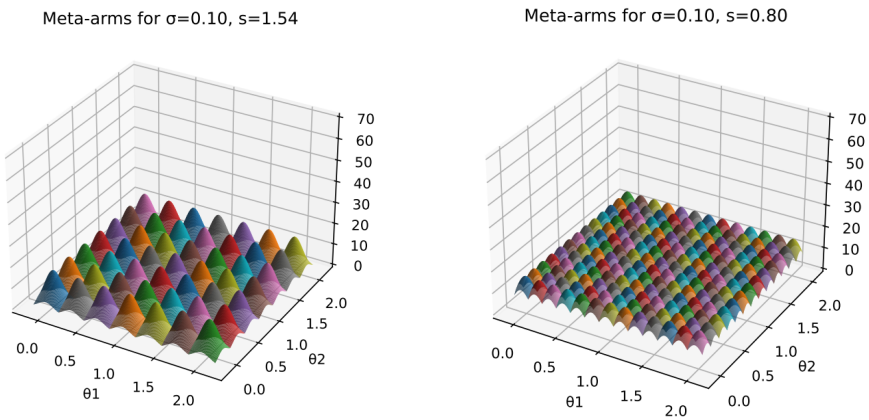

Figure 5.6: Covering of Θ for $\sigma = 0.1, s = 1.54$ (left), $\sigma = 0.1, s = 0.8$ (right).

et al. (2011), where the same parameter α is used for both the progressive widening layers.

Tuning the parameter α We have run tests on the Mini-golf environment for different values of α with the fixed budget of 500. The results that we achieved with our implementation of MCTS-DPW are shown in Figure 5.10.

MCTS-DPW Performance With the empirically optimal value of $\alpha = 0.725$, we have measured the average performance of MCTS-DPW for different values of the budget. The results are shown in Figure 5.11.

UGapEb

In order to assess if our randomized exploration strategy brings an advantage over the standard grid search, we test the performance of the UGapEb algorithm, the fixed-budget version of the best-arm-identification algorithm UGapE. We make it perform best-arm identification on the subset of arms $\mathcal{M} \subseteq \Theta$ containing only the points which correspond to the means of the meta-arms used by our algorithm.

Tuning the parameter a We have run tests on the Mini-golf environment for different values of the parameter a with the fixed budget of 500. The results that we achieved with our implementation of UGapEb are shown in Figure 5.12.

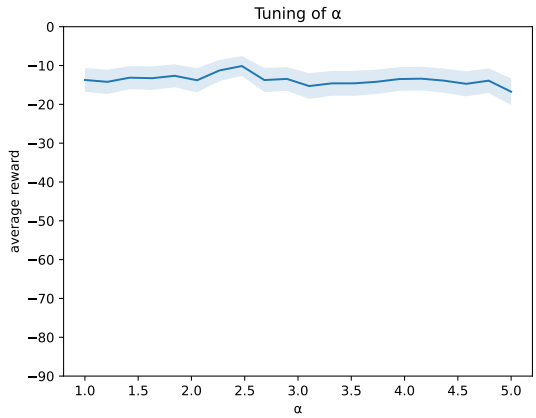

Figure 5.7: The average performance of our algorithm on the Mini-golf environment with $\sigma = 0.1$, $s = 1.54$ in 500 rounds for different values of α. The plot was obtained covering the domain $[1, 5]$ of α with a 20-points one-dimensional grid, taking 400 samples for each value of α. The 2-std confidence intervals for the expected value are shown.

UGapEb Performance With the empirically optimal value of $a = 1$, we tested the performace of UGapEb for different values of the budget. The results are shown in Figure 5.13.

Comparative Performance

In Figure 5.14 we report the results in the same plot for an easier comparison. The empiric results show that on the Mini-golf environment both UGapE and our algorithm were able to outperform MCTS-DPW. This could probably be explained by the different search spaces, since MCTS is building a lookahead tree while we are optimizing the expected reward over a 2-dimensional policy space. Also, according to the results on this environment, there is no statistical evidence to indicate that either of the two selection modes of our algorithm is better. Lastly, it can be seen that on the Mini-golf environment UGapE is performing better than our algorithm while employing the same grid. This might be due to very near-optimal parameters already being present in the grid, making the randomized exploration superfluous: if this is the case, the randomized exploration combined with multiple importance sampling will require more samples than the UGapE one because, in a non-trivial problem, moving to the meta-arms space the sub-optimality gaps between the meta-arms corresponding to the grid points become smaller, making the best-meta-arm-identification task harder.

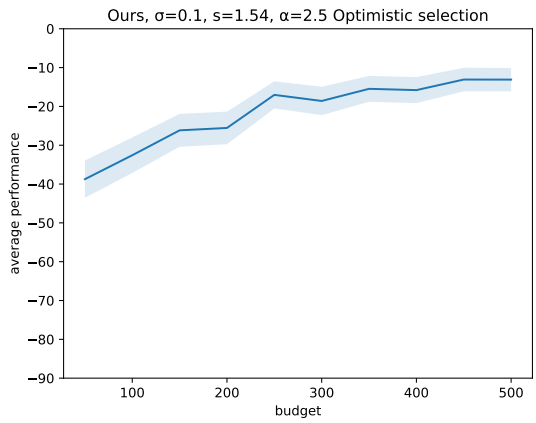

Figure 5.8: Average performance of our algorithm on Mini-golf if the optimistic meta-arm selection is used. The 2-std confidence interval for the expected value are shown. The data were collected by running 400 simulations for each of the considered budget values.

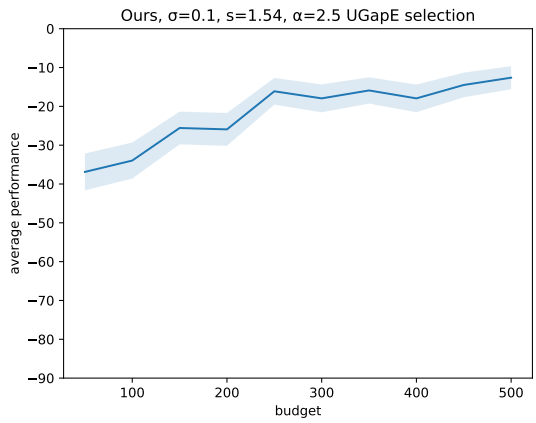

Figure 5.9: Average performance of our algorithm on Mini-golf if the UGapE meta-arm selection is used. The 2-std confidence interval for the expected value are shown. The data were collected by running 400 simulations for each of the considered budget values.

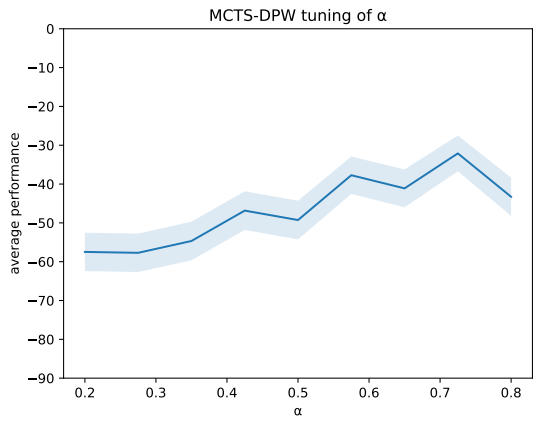

Figure 5.10: Average performance of MCTS-DPW on the Mini-golf environment for different values of α with budget 500. The plot was obtained covering the domain $[0.2, 0.8]$ of α with a 9-points one-dimensional grid, taking 400 samples for each value of α. The 2-std confidence intervals for the expected value are shown.

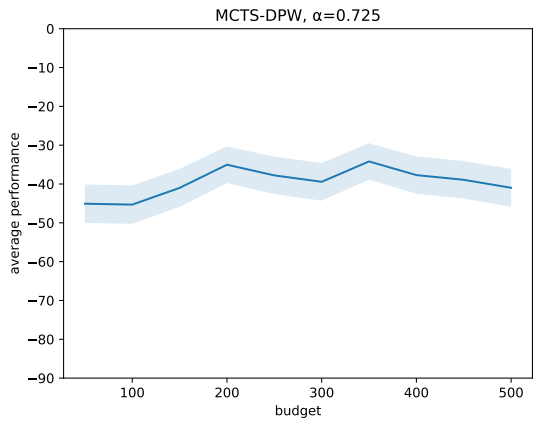

Figure 5.11: Average performance of MCTS-DPW on Mini-golf with $\alpha = 0.725$. The plot was obtained by running 400 simulations for each value of the budget. The 2-std confidence intervals for the expected value are shown.

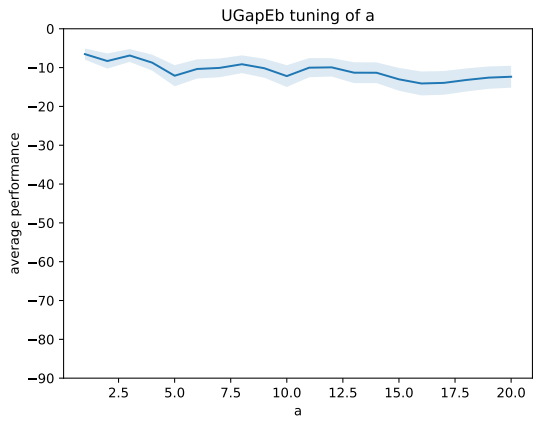

Figure 5.12: Average performance of UGapEb on Mini-golf for different values of a. The plot was obtained by covering the domain $[1, 20]$ of a with a 1-dimensional grid with 20 points and running 400 simulations for each value of the a with budget 500. The 2-std confidence intervals for the expected value are shown.

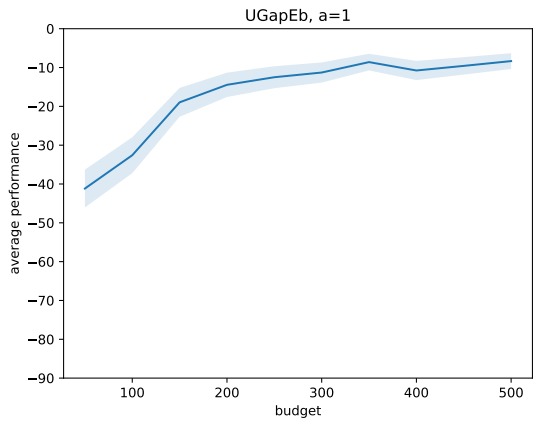

Figure 5.13: Average performance of UGapEb on Mini-golf, obtained by running 400 simulations for each value of the budget with $a = 1$. The 2-std confidence intervals for the expected value are shown.

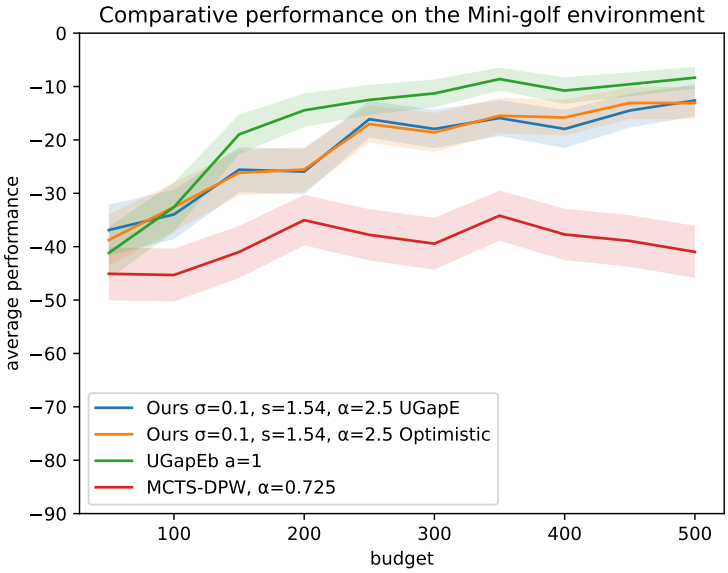

Figure 5.14: Comparative performance on the Mini-golf environment

5.5 The Stochastic, Continuous MountainCar Environment

As second benchmark environment, we employ the stochastic, continuous-actions version of the popular MountainCar environment available on the OpenAI Gym website. To perform our experiments, we used the python implementation by O. Sigaud, which is freely available online, introducing a stochastic noise to achieve the non-deterministic transitions.

5.5.1 Environment Definition

In this environment, the agent is controlling a car which starts in a valley between two hills, as can be seen in Figure 5.15. The goal of the agent is to reach the top of the right hill. The difficulty in this environment lies in the fact that the car is not powerful enough to directly climb the right hill: in order to reach the goal, it has to swing up and down the hills to increase its momentum and eventually be able to reach the flag, requiring a non-trivial planning ability.

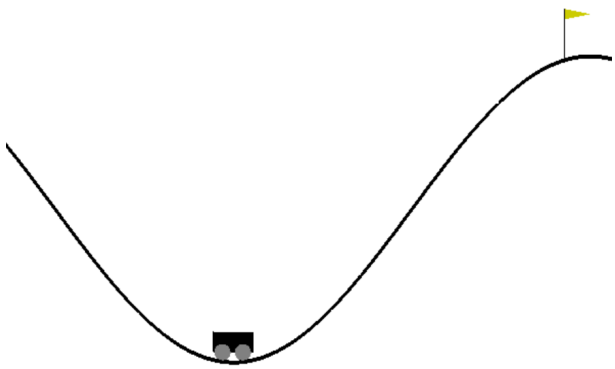

Figure 5.15: Graphic representation of the Continuous MountainCar environment as generated by the script

The continuous state of this environment is 2-dimensional and is represented by the horizontal distance of the car from the center of the valley and by its horizontal velocity, $s = (x, v)$. The action a of the agent is always a real number in the interval $[-1, 1]$. A random noise is sampled as $\xi_t \sim \mathcal{U}(0, 2)$ and the transition is as follows: $v_{t+1} = v_t + ap\xi_t - 0.0025\cos(3x_t)$ and $x_{t+1} = x_t + v_{t+1}$, where the parameter p, the "power" of the car's engine, can be modified to make the environment easier or more difficult. The

rewards are defined as $r(s, a) = r(a) = -0.1a^2$, that is the car must reach the goal spending as little energy as possible. If the car reaches the top of the right hill, the agent receives a positive reward of 100. Calling H the horizon, we set the minimum possible reward as $r_{min} = -0.1H$, and the maximum as $r_{max} = 100$. Of course the car is not powerful enough to reach the top in one step: r_{max} is just an upper bound to the best possible reward.

5.5.2 The Policies

We employ linear policies *without* the constant term, in order to keep the dimensionality small. Therefore, $\pi_{\boldsymbol{\theta}}(s) = \pi_{\boldsymbol{\theta}}(x, v) = \theta_1 x + \theta_2 v$. As parameter space, we employ $\Theta = [-4, 4] \times [-4, 4]$.

The Persistence

In order to reach the goal, the car needs to swing up and down the hills to increase its velocity and finally climb the right hill.
When tested with horizon $H = 600$, the number of rounds needed to reach the goal with the optimal *linear* policy without constant term is approximately 540, swinging from one hill to the other about 8 times. This means that, in order to have a meaningful policy space to search, the minimum horizon to employ is 540 plus some margin. In the context of online planning, testing the performance of an algorithm scales with H^2: indeed, one must perform one planning step (which has a complexity of at least bH, b being the budget, since each budget unit corresponds to one rollout until the end of the process) for each step in the episode, which again depends on H, up until the desired number of samples is collected (one sample is the reward obtained during one full episode). For this reason, a horizon of 540 is prohibitively large for our computational resources. To overcome this issue, we employed the *persistence* technique. The persistence is the number of environment steps that each choice of an action will last: with persistence 4, for example, the action resulting from the search step will be used for 4 steps in the environment. This means that, to the learner, the horizon appears to be reduced by a factor of 4. Of course the number of transitions performed in the environment remains the same: what changes is the number of search steps required to the planner, which is the slowest procedure in the algorithm. It must be noticed that introducing the persistence changes the granularity of the control that the agent can exert on the environment, potentially leading to different optimal policies and outcomes. Therefore, one should not expect our results to be valid also in the case of

different horizon / persistence combinations.

For the MountainCar environment we set a horizon of 150 and a persistence of 4. With this configuration, the optimal policy in our parameter space reaches the hill in about 115 steps (which correspond to 460 transitions in the environment), obtaining a reward of roughly 92.

The Value Function in Policy Space

In Figure 5.16 is shown the value surface in our policy space for the initial state $(-0.6, 0)$. Looking at the surface, one can notice a region of the policy parameters which allows the agent to reach the goal (specifically, the region with $\theta 1 \leqslant 0$ and $\theta_2 \geqslant 0$). One may also notice that in the region which does not reach the goal, the value function is seemingly independent from θ_2 and is shaped like a negative parabolic function of θ_1. This is due to the fact that the velocity is initially very small ($\leqslant 0.005$ due to the initial slope), cancelling the effect of θ_2, and if $\theta_1 \geqslant 0$ the car is indeed "braking" when descending from the hills, preventing the velocity from increasing. The parabolic shape is due to the reward function, which is a negative quadratic function of the action and therefore of the predominant parameter θ_1.

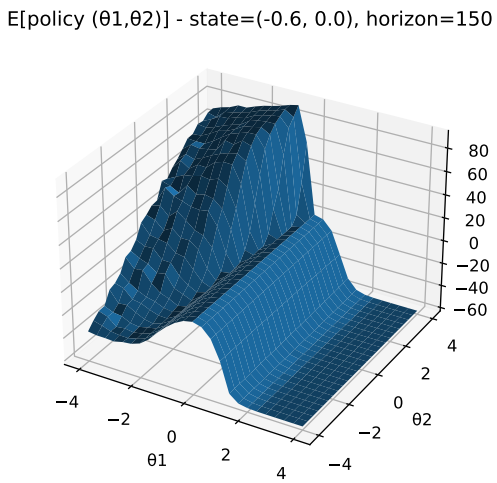

Figure 5.16: The surface defined by the expected total reward of the policies on the MountainCar environment. The approximation was obtained by covering Θ with a 25×25 grid and by sampling the model 400 times in each point of the grid.

5.5.3 Building the Meta-arms

Selection of the Variance

Also for this environment, we tuned the variance starting from the performance decay that different values of σ would cause when starting from the initial state $(-0.6, 0)$. As one can notice from Figure 5.16, the maximum located at $\boldsymbol{\theta}^* = (-0.667, 4)$ is near a precipitous fall to the value of -5. This suggests intuitively that a meta-arm with center on $\boldsymbol{\theta}^*$ will suffer a high decay in performance due to the randomization. In Figure 5.17 we show how different values of the standard deviation impact the performance of the meta-arms $\boldsymbol{\theta}^*$ and $\boldsymbol{\theta}_1 = (-2.33, 4)$, which is located in a more regular region. From this experimental data we selected the value $\sigma = 0.2$, the rationale being that we must not set σ to be too large in order not to suffer from the aforementioned pitfall and we must not set σ to be too small in order to not have either too many meta-arms (for small values of s) or few meta-arms too far from each other (for $s > 3$).

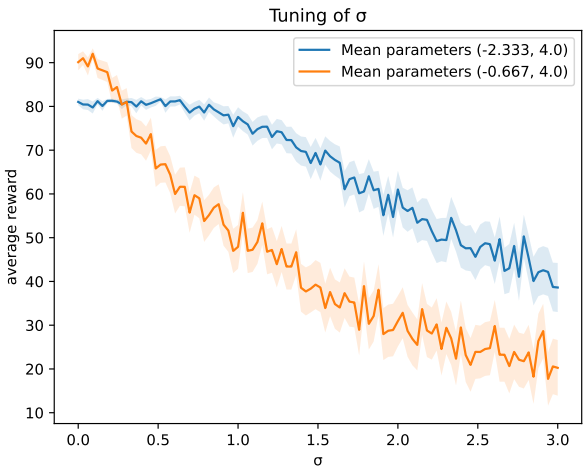

Figure 5.17: Performance decay due to the randomization for the meta-arm with mean $(-2.33, 4)$ (blue) and $(-0.667, 4)$ (orange). The plots were obtained by covering the domain $[0, 3]$ of σ with a 1-dimensional grid with 100 points and taking 400 samples for each point. The 2-std confidence intervals for the expected value are reported.

Selection of s

After setting $\sigma = 0.2$, we ran tests to select an appropriate value for s. We tried the candidate values $1.0, 1.1, 1.15, 1.2, 1.3, 1.4, 1.5, 1.65$ which led

respectively to a total of $361, 324, 324, 256, 225, 196, 169, 144$ meta-arms. The results are reported in Figure 5.18. From the experiments it turned out that many of the candidates led to very similar performance. We chose to employ $s = 1.65$ in order to have a smaller number of meta-arms.

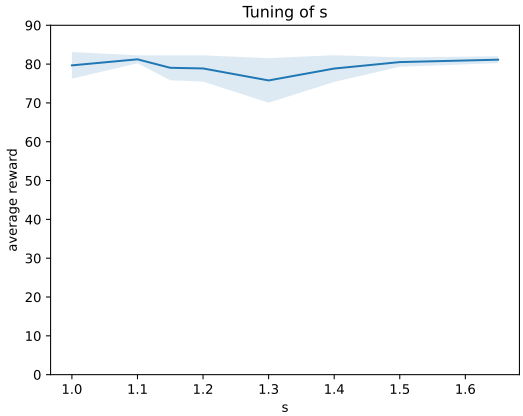

Figure 5.18: The performance of our algorithm with $\sigma = 0.2$ for different values of s. The plot was obtained by taking 64 samples for each point in the domain $\{1.0, 1.1, 1.15, 1.2, 1.3, 1.4, 1.5, 1.65\}$ of s. The 2-std confidence intervals for the expected value are reported.

5.5.4 Tuning of α

With the values $\sigma = 0.2$, $s = 1.65$ we tuned the parameter α. The results of this tuning step, reported in Figure 5.19, suggest to employ $\alpha = 3$, as its performance confidence bounds suggest that it can hardly be worse than the other values.

5.5.5 Experimental results

Performance of Our Algorithm

With the selected values $\sigma = 0.2$, $s = 1.65$, $\alpha = 3$ we tested our algorithm in both selection modes. The results are shown in Figures 5.20 (Optimistic mode) and 5.21 (UGapE mode). It can be noticed in the plots that even with a budget of 50 our algorithm is able to reach the goal (due to the positive average reward), and with a budget of 150 or greater, corresponding roughly to one pull for each meta-arm, it is able to reach the goal consistently enough to have its average performance greater or equal to 60.

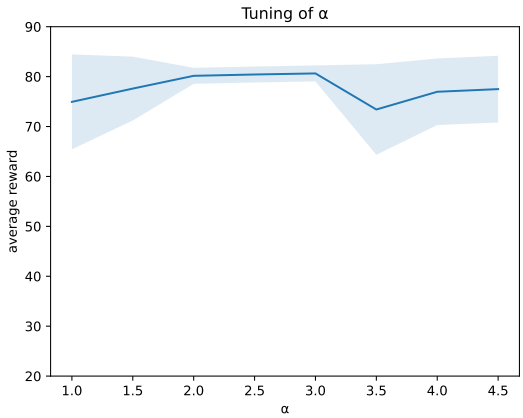

Figure 5.19: The performance of our algorithm with $\sigma = 0.2$ and $s = 1.65$ for different values of α. The plot was obtained by covering the domain $[1, 4.5]$ of α with a 1-dimensional grid with 8 points and taking 32 samples for each point. The 2-std confidence intervals for the expected value are reported.

MCTS-DPW

Tuning the parameter α We tuned α with the fixed budget of 500. The results are shown in Figure 5.22. Given the high increase in performance associated to the value $\alpha = 0.5$, we set this as the value to employ in the tests.

MCTS-DPW Performance After setting $\alpha = 0.5$, we measured the average performance of MCTS-DPW for different values of the budget. The results are shown in Figure 5.23. As it can be seen, MCTS-DPW with a budget of at least 150 will consistently achieve a good performance on this environment, while being able to reach the goal even with the very limited budget of 50. The performance of MCTS-DPW is astonishingly good, as the expectation was that with a horizon of 150, with the stochastic transitions and with the continuous states it would take much more than 50 rollouts to identify the best action for any lookahead-tree-based approach. The reason behind this expectation was that, since during the rollouts the actions are sampled uniformly in $[-1, 1]$, each rollout would contain enough "wrong" actions that the car would accelerate in the wrong direction too much and never be able accumulate enough velocity to reach the goal. This intuition turned out to be wrong. We observed two interesting behaviours in the simulations:

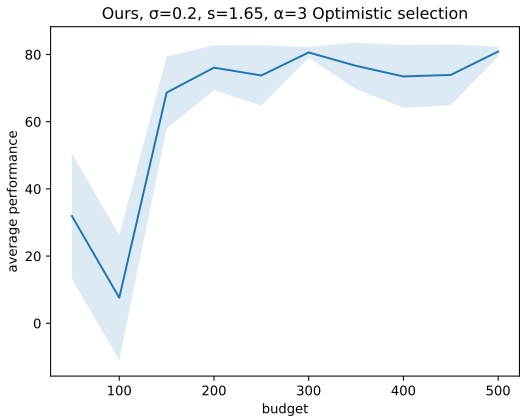

Figure 5.20: Average performance of our algorithm if optimistic meta-arm selection is used. The 2-std confidence interval for the expected value are shown. The data were collected by running 32 simulations for each of the considered budget values.

1. The uniform policy leads to the goal in the long term. We started from this observation: if at some point the action $a = 0$ was fixed an chosen at all the subsequent interactions, the car would oscillate without ever decreasing its maximum velocity. This is easily understandable by looking at the dynamics of the environment in which the horizontal position x becomes a sinusoidal function. From this, it would seem intuitive to think that selecting actions uniformly with mean 0 should not lead, in the long run, to a reduction of the car's velocity. We tested this specific situation without the stochastic noise, obtaining the surprising result that the car is indeed *gaining* velocity even if the actions are sampled uniformly, meaning that the uniform policy is indeed able to make the car oscillate higher and higher.

2. The noise is helping the uniform policy. We tested the uniform policy *with* the stochastic noise, to discover that the presence of the noise will increase the speed of the oscillation, making it reach higher altitudes in a smaller number of rounds.

From this observation, it seems that in our environment the uniform policy, which is the *default policy* employed by MCTS-DPW in the rollouts, is already a good policy. Therefore, when at the root the planner is trying a good action there are good chances that the rollout reaches the goal, if the action was near-optimal. This drives MCTS-DPW's search in such an accurate way that even with ridiculously small budgets it is able to consistently

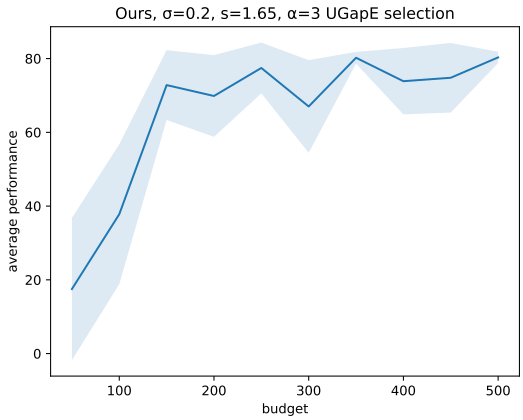

Figure 5.21: Average performance of our algorithm if UGapE meta-arm selection is used. The 2-std confidence interval for the expected value are shown. The data were collected by running 32 simulations for each of the considered budget values.

solve this environment. In order to prevent this behaviour, the stochastic noise should be reduced to a much smaller range (for example, $[0.95, 1.05]$) and a *friction* term should be added in the transition function, so that if the action 0 is repeatedly selected the car will eventually stop.

UGapEb

Tuning the parameter a We have run tests on the MountainCar environment for different values of the parameter a with the fixed budget of 500. The results are shown in Figure 5.24. The experiments suggest to employ the value $a = 1$.

UGapEb Performance With the empirically optimal value of $a = 1$, we ran tests for different budget values. The result is shown in Figure 5.25. It is evident from the plot that the performance of the deterministic approach heavily depends on the budget. In particular, for budget values smaller than 200 the UGapE planner consistently fails to reach the goal: this is probably due to the fact that in order to try each grid point once the planner needs 144 rounds and the remaining budget is not enough to discriminate between the seemingly good arms.

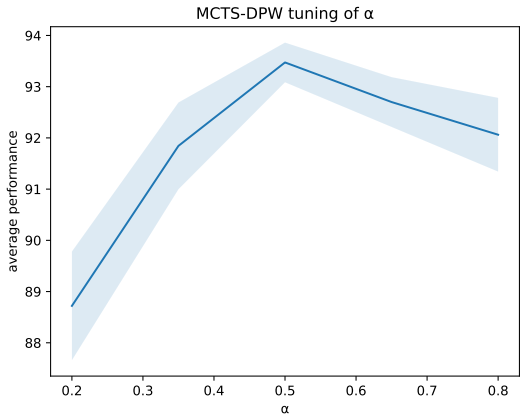

Figure 5.22: Average performance of MCTS-DPW on the MountainCar environment for different values of α with budget 500. The plot was obtained covering the domain $[0.2, 0.8]$ of α with a 5-points one-dimensional grid, taking 32 samples for each value of α. The 2-std confidence intervals for the expected value are shown.

Comparative Performance

The performances of the different algorithms are reported in figure 5.26. From the data one can notice that while MCTS-DPW and UGapE are achieving a better result for budgets greater or equal to 200, our algorithm is indeed outperforming the UGapE-based deterministic planner by a fair amount for the smaller budget values, when working in either of the selection modes.

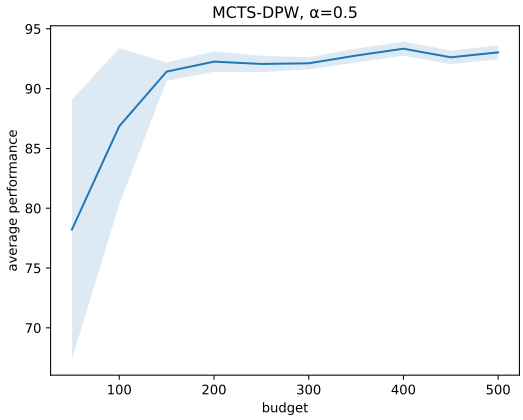

Figure 5.23: Average performance of MCTS-DPW on MountainCar with $\alpha = 0.5$. The plot was obtained by running 32 simulations for each value of the budget. The 2-std confidence intervals for the expected value are shown.

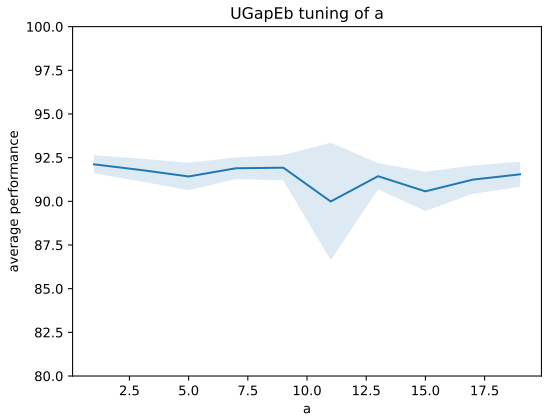

Figure 5.24: Average performance of UGapEb on MountainCar for different values of a. The plot was obtained by covering the domain $[1, 19]$ of a with a 1-dimensional grid with 10 points and running 64 simulations for each value of a with budget 500. The 2-std confidence intervals for the expected value are shown.

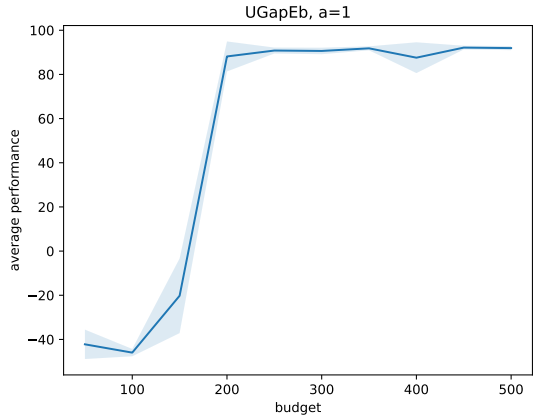

Figure 5.25: Average performance of UGapEb on MountainCar with $a = 1$ for different values of the budget. The plot was obtained by running 32 simulations for each value of the budget. The 2-std confidence intervals for the expected value are shown.

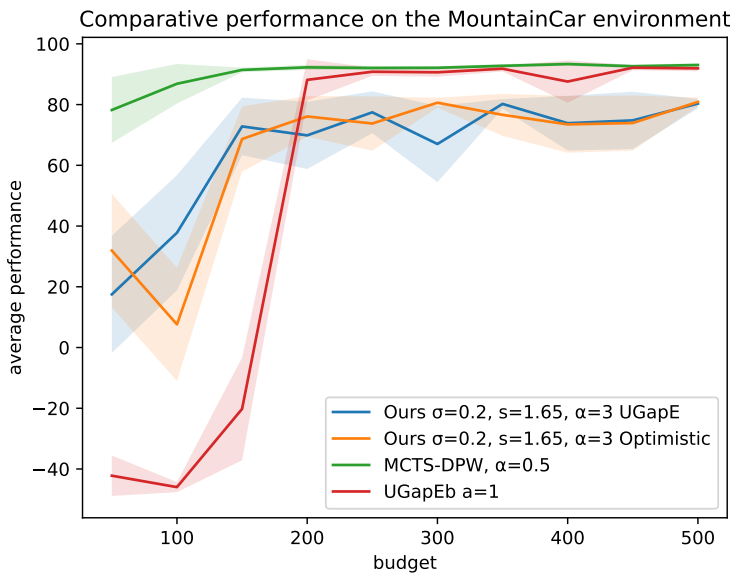

Figure 5.26: Comparative performance on the MountainCar environment

Chapter 6

Conclusions

In order to perform online planning, we developed a randomized continuous bandit algorithm which relies on multiple importance sampling estimation and we employed it as the core policy optimizer in the policy-search online planning loop. We tested the resulting planner on two benchmark environments: *Mini-golf* and *MountainCar*. On the same environments, we tested also two state-of-the-art solutions: the MCTS-DPW planner and a policy-search-based planner which starts from the same grid of points covering the policy space as our algorithm and then employs the UGapEb best-arm-identification algorithm to find the best policy in the grid. The choice of the two algorithms was not coincidental: the comparison with MCTS-DPW is done in order to assess the performance of the most general-purpose, lookahead-tree based planner for continuous, stochastic MDPs on the specific environments that we chose. The comparison with the UGapEb planner is done to assess whether the randomized exploration, and the subsequent importance estimation, could bring an advantage over a standard grid search.

The first conclusion that we draw, both from the theoretical standpoint (in Section 5.4.5) and from the empirical observation during our tests, is that employing the multiple importance sampling estimators with the balance heuristic requires a rather long computational time, which is due to the computation of the value of the current mixture in the past points: at each round, the newly pulled meta-arm must be added to the value of the mixture for all the points sampled in the previous rounds. In practice, our implementation would take a relatively long time for budgets greater or equal to 600 even on environments with a short horizon such as Mini-golf. This indicates that an algorithm like ours could be more suitable for settings where little budget is available.

The second conclusion that we draw is that the methods based on lookahead trees are not easily comparable with those based on policy-search: this is suggested by the difference in the performance of MCTS and our algorithm in the two environments. In the Mini-golf environment, the policy-search-based approaches have achieved a better performance than that of MCTS-DPW, while in the MountainCar environment the performance is similar. Despite both approaches were tested with the same budget per planning step (in terms of rollouts on the simulation environment), it is far from trivial to tell whether the difference in performance is due to a better usage of the samples or rather to a lucky choice of the parameter space. To this considerations one must add that different degrees of stochasticity in the environment have in general a different impact on the difficulty of the task in the two approaches, making it even harder to address what could be the source of better performance on one particular instance of environment.
The third conclusion is that, on the two environments that we tested, the randomized exploration did not lead to a better performance for sufficiently large budget values, as it can be seen by the higher rewards obtained by the UGapEb planner. There were two reasons to expect that our randomized exploration could obtain an overall better performance on a continuous environment: the first reason is that our algorithm can sample in any point of the parameter space, using the grid only as an indicator of the regions to explore more thoroughly, potentially finding maxima which are not present on the grid itself; the second reason is that the mediator feedback, which allows to use all the samples to estimate the performance of all the policies, implicitly assumes some regularity on the reward function in the parameter space, and in the continuous environments that we used this regularity is indeed present, as it can be seen by the plots, meaning that we are exploiting one additional hypothesis. A reason for the observed performance decrease with respect to UGapEb could be that the increase in difficulty of finding the best meta-arm due to the randomization was not properly compensated by the extra information yielded by the mediator feedback.
A notable result achieved by our algorithm is outperforming the deterministic UGapE-based approach when constrained by a very limited budget, as can be seen from the tests on the MountainCar environment, showing that the randomized exploration combined with mediator feedback, if properly tuned, could give an advantage in optimizing continuous, regular, stochastic functions in such scenario.
A direction of future research could be to investigate what are the conditions, in terms of regularity and stochasticity of the rewards in the parameter space, that enable an effective usage of a randomized approach like ours and

what are the optimal values of the algorithm parameters.

Bibliography

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2–3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352. URL https://doi.org/10.1023/A:1013689704352.

Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957. ISSN 00959057, 19435274. URL http://www.jstor.org/stable/24900506.

Donald A. Berry, Robert W. Chen, Alan Zame, David C. Heath, and Larry A. Shepp. Bandit problems with infinitely many arms. *The Annals of Statistics*, 25(5):2103 – 2116, 1997. doi: 10.1214/aos/1069362389. URL https://doi.org/10.1214/aos/1069362389.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration for multi-armed bandit problems, 2010.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvari. X-armed bandits, 2011.

Alexandra Carpentier and Michal Valko. Simple regret for infinitely many armed bandits, 2015.

W.G. Cochran. *Sampling Techniques, 3Rd Edition.* A Wiley publication in applied statistics. Wiley India Pvt. Limited, 2007. ISBN 9788126515240. URL https://books.google.it/books?id=xbNn41DUrNwC.

Adrien Couetoux and Hassen Doghmen. Adding Double Progressive Widening to Upper Confidence Trees to Cope with Uncertainty in Planning Problems. In *The 9th European Workshop on Reinforcement Learning (EWRL-9)*, Athens, Greece, September 2011. URL `https://hal.inria.fr/hal-00745207`.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 433–445, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25566-3.

Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In Paolo Ciancarini and H. Jaap van den Herik, editors, *5th International Conference on Computer and Games*, Turin, Italy, May 2006. URL `https://hal.inria.fr/inria-00116992`.

Victor Gabillon, Mohammad Ghavamzadeh, and Alessandro Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Jean-Bastien Grill, Michal Valko, and Remi Munos. Black-box optimization of noisy functions with unknown smoothness. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Edward L Ionides. Truncated importance sampling. *Journal of Computational and Graphical Statistics*, 17(2):295–311, 2008.

Michael Kearns, Yishay Mansour, and Andrew Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49, 06 2001. doi: 10.1023/A:1017932429737.

Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces, 2008.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. doi: 10.1017/9781108571401.

Alberto Maria Metelli, Matteo Papini, Francesco Faccio, and Marcello Restelli. Policy optimization via importance sampling. *arXiv preprint arXiv:1809.06098*, 2018.

Alberto Maria Metelli, Matteo Papini, Pierluca D'Oro, and Marcello Restelli. Policy optimization as online learning with mediator feedback, 2020.

O. Nikodym. Sur une généralisation des intégrales de M. J. Radon. *Fundam. Math.*, 15:131–179, 1930. ISSN 0016-2736. doi: 10.4064/fm-15-1-131-179.

Art B. Owen. *Monte Carlo theory, methods and examples*. 2013.

Matteo Papini, Alberto Maria Metelli, Lorenzo Lupo, and Marcello Restelli. Optimistic policy optimization via multiple importance sampling. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4989–4999. PMLR, 09–15 Jun 2019. URL `http://proceedings.mlr.press/v97/papini19a.html`.

Albert Penner. The physics of putting. *Canadian Journal of Physics*, 80: 83–96, 02 2002. doi: 10.1139/p01-137.

M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 2005. ISBN 9780471727828. URL `https://books.google.it/books?id=Y-gmAQAAIAAJ`.

Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–561. University of California Press, 1961.

Philippe Rolet, Michèle Sebag, and Olivier Teytaud. Upper Confidence Trees and Billiards for Optimal Active Learning. In *CAP09*, Hammamet, Tunisie, Tunisia, 2009. URL `https://hal.inria.fr/inria-00369787`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 419–428, New York, NY, USA, 1995. Association for Computing Machinery. ISBN 0897917014. doi: 10.1145/218380.218498. URL `https://doi.org/10.1145/218380.218498`.