From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning

Rémi Munos

Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–115
© xxxx xxxxxxxxx
DOI: xxxxxx

From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning

Rémi Munos[1]

[1] *INRIA Lille – Nord Europe, remi.munos@inria.fr*

Abstract

This work covers several aspects of the *optimism in the face of uncertainty* principle applied to large scale optimization problems under finite numerical budget. The initial motivation for the research reported here originated from the empirical success of the so-called *Monte-Carlo Tree Search* method popularized in Computer Go and further extended to many other games as well as optimization and planning problems. Our objective is to contribute to the development of theoretical foundations of the field by characterizing the complexity of the underlying optimization problems and designing efficient algorithms with performance guarantees.

The main idea presented here is that it is possible to decompose a complex decision making problem (such as an optimization problem in a large search space) into a sequence of elementary decisions, where each decision of the sequence is solved using a *(stochastic) multi-armed bandit* (simple mathematical model for decision making in stochastic environments). This so-called *hierarchical bandit* approach (where the reward observed by a bandit in the hierarchy is itself the return of an-

other bandit at a deeper level) possesses the nice feature of starting the exploration by a quasi-uniform sampling of the space and then focusing progressively on the most promising area, at different scales, according to the evaluations observed so far, and eventually performing a local search around the global optima of the function. The performance of the method is assessed in terms of the optimality of the returned solution as a function of the number of function evaluations.

Our main contribution to the field of function optimization is a class of hierarchical optimistic algorithms designed for general search spaces (such as metric spaces, trees, graphs, Euclidean spaces, ...) with different algorithmic instantiations depending on whether the evaluations are noisy or noiseless and whether some measure of the "smoothness" of the function is known or unknown. The performance of the algorithms depends on the local behavior of the function around its global optima expressed in terms of the quantity of near-optimal states measured with some metric. If this local smoothness of the function is known then one can design very efficient optimization algorithms (with convergence rate independent of the space dimension), and when it is not known, we can build adaptive techniques that can, in some cases, perform almost as well as when it is known.

In order to be self-contained, we start with a brief introduction to the stochastic multi-armed bandit problem in Chapter 1 and describe the UCB (Upper Confidence Bound) strategy and several extensions. In Chapter 2 we present the Monte-Carlo Tree Search method applied to Computer Go and show the limitations of previous algorithms such as UCT (UCB applied to Trees). This provides motivation for designing theoretically well-founded optimistic optimization algorithms. The main contributions on hierarchical optimistic optimization are described in Chapters 3 and 4 where the general setting of a semi-metric space is introduced and algorithms designed for optimizing a function assumed to be locally smooth (around its maxima) with respect to a semi-metric are presented and analyzed. Chapter 3 considers the case when the semi-metric is known and can be used by the algorithm, whereas Chapter 4 considers the case when it is not known and describes an adaptive technique that does almost as well as when it is known. Finally in Chapter 5 we describe optimistic strategies for a specific structured problem, namely the planning problem in Markov decision processes with infinite horizon discounted rewards.

Contents

Optimism

Optimists and pessimists inhabit different worlds, reacting to the same circumstances in completely different ways.

Learning to Hope, Daisaku Ikeda.

Habits of thinking need not be forever. One of the most significant findings in psychology in the last twenty years is that individuals can choose the way they think.

Learned Optimism, Martin Seligman.

1

The stochastic multi-armed bandit problem

We start with a brief introduction to the stochastic multi-armed bandit problem. This is a simple mathematical model for sequential decision making in unknown random environments that illustrates the so-called *exploration-exploitation trade-off*. Initial motivation in the context of clinical trials dates back to the works of Thompson [104, 103] and Robbins [92]. In this chapter we mainly describe a strategy that illustrates the *optimism in the face of uncertainty* principle, namely the UCB algorithm (where UCB stands for upper confidence bound) introduced by Auer, Cesa-Bianchi, and Fischer in [13]. This principle recommends following the optimal policy in the most favorable environment compatible with the observations. In a multi-armed bandit the set of “compatible environments” is the set of possible distributions of the arms that are likely to have generated the observed rewards. The UCB strategy uses a particularly simple representation of this set of compatible environments as a set of high-probability confidence intervals (one for each arm) for the expected value of the arms. Then the strategy consists in selecting the arm with highest upper-confidence-bound (the optimal strategy for the most favorable environment). We introduce the setting of the multi-armed bandit problem in Section 1.1.1, then present the UCB algorithm in Section 1.1.2 and existing lower bounds in Section 1.1.3. In Section 1.2 we describe extensions of the optimistic approach to the case of an infinite set of arms, either when the set is denumerable (in which case a stochastic assumption is made) or where it is continuous but the reward function has a known structure (e.g. linear, Lipschitz).

1.1 The multi-armed stochastic bandit

1.1.1 Setting

Consider K arms (actions, choices) defined by some distributions $(\nu_k)_{1\leq k\leq K}$ with bounded support (here we will assume that it is $[0,1]$) that are initially unknown to the player. At each round $t=1,\dots,n$, the player selects an arm $I_t\in\{1,\dots,K\}$ and obtains a reward $X_t\sim\nu_{I_t}$, which is a random sample drawn from the distribution of the corresponding arm I_t, and is assumed to be independent of previous rewards. The goal of the player is to maximize the sum of obtained rewards in expectation.

Define $\mu_k=\mathbb{E}_{X\sim\nu_k}[X]$ as the mean values of each arm, and $\mu^*=\max_k\mu_k=\mu_{k^*}$ as the mean value of one best arm k^* (there may exist several).

If the arm distributions were known, the agent would select the arm with the highest mean at each round and obtain an expected cumulative reward of $n\mu^*$. However, since the distributions of the arms are initially unknown, he needs to pull each arm several times in order to acquire information about the arms (this is called the *exploration*) and while his knowledge about the arms improves, he should pull increasingly often the apparently best ones (this is called the *exploitation*). This illustrates the so-called *exploration-exploitation trade-off*.

In order to assess the performance of any strategy, we compare its performance to an oracle strategy that would know the distributions in advance (and would thus play the optimal arm). For that purpose we define the notion of *cumulative regret*: at round n,

$$R_n\overset{\text{def}}{=}n\mu^*-\sum_{t=1}^{n}X_t. \tag{1.1}$$

This defines the loss, in terms of cumulative rewards, resulting from not knowing from the beginning the reward distributions. We are thus interested in designing strategies that have a low cumulative regret.

Notice that using the tower rule, the expected regret can be rewritten:

$$\mathbb{E}R_n = n\mu^* - \mathbb{E}\Big[\sum_{t=1}^{n} \mu_{I_t}\Big] = \mathbb{E}\Big[\sum_{k=1}^{K} T_k(n)(\mu^* - \mu_k)\Big] = \sum_{k=1}^{K} \mathbb{E}[T_k(n)]\Delta_k, \tag{1.2}$$

where $\Delta_k \overset{\text{def}}{=} \mu^* - \mu_k$ is the *gap* in terms of expected rewards, between the optimal arm and arm k, and $T_k(n) \overset{\text{def}}{=} \sum_{t=1}^{n} \mathbf{1}\{I_t = k\}$ is the number of pulls of arm k up to time n.

Thus a good algorithm should not pull sub-optimal arms too many times. Of course, in order to acquire information about the arms, one needs to explore all the arms and thus pull sub-optimal arms. The regret measures how fast one can *learn* relevant quantities about some unknown environment for the purpose of *optimizing* some criterion. This combined learning-optimizing objective is central to the exploration-exploitation trade-off.

Proposed solutions Initially formulated by [92], this exploration-exploitation problem is not entirely solved yet. However there have been many approaches developed in the past, including:

- *Bayesian exploration*: A prior is assigned to the arm distributions and an arm is selected as a function of the their posterior distribution (such as the Thompson strategy [104, 103] which has been analyzed recently [7, 71], the Gittins indexes, see [58, 59], and optimistic Bayesian algorithms such as [99, 70]).
- *ϵ-greedy exploration*: The empirical best arm is played with probability $1-\epsilon$ and a random arm is chosen with probability ϵ (see e.g. [13] for an analysis),
- *Soft-max exploration*: An arm is selected with a probability that depends on the (estimated) performance of this arm given previous reward samples (such as the EXP3 algorithm

introduced in [14], see also the *learning-from-expert* setting [41]).

- *Follow the perturbed leader*: The empirical mean reward of each arm is perturbed by a random quantity and the best perturbed arm is selected (see e.g. [69, 79]).
- *Optimistic exploration*: Select the arm with the largest high-probability upper-confidence-bound (initiated by [81, 36]), an example of which is the UCB algorithm [13] described in the next section.

1.1.2 Upper Confidence Bounds (UCB) algorithms

The Upper Confidence Bounds (UCB) strategy [13] consists in selecting at each time step t an arm with largest B-values:

$$I_t \in \arg\max_{k\in\{1,\dots,K\}} B_{t,T_k(t-1)}(k),$$

where the B-value of an arm k is defined as:

$$B_{t,s}(k) \overset{\text{def}}{=} \hat{\mu}_{k,s} + \sqrt{\frac{3\log t}{2s}}, \tag{1.3}$$

where $\hat{\mu}_{k,s} \overset{\text{def}}{=} \frac{1}{s}\sum_{i=1}^{s} X_{k,i}$ is the empirical mean of the s first rewards received from arm k, where we write $X_{k,i}$ for the reward received when pulling arms k for the i-th time (i.e., by defining the random time $\tau_{k,i}$ to be the instant when we pull arm k for the i-th time, we have $X_{k,i} = X_{\tau_{k,i}}$). We described here a slightly modified version of UCB1 where the constant defining the confidence interval is $3/2$ instead of 2 in the original version.

This strategy follows the so-called *optimism in the face of uncertainty* principle since it selects the optimal arm in the most favorable environments that are (in high probability) compatible with the observations. Indeed the B-values $B_{t,s}(k)$ are high-probability upper-confidence-bounds on the mean-value of the arms μ_k. More precisely for any $1 \leq s \leq t$, we have $\mathbb{P}(B_{t,s}(k) \geq \mu_k) \leq 1-t^{-3}$. This bound comes from the Chernoff-Hoeffding inequality which is described below. Let

$Y_i \in [0,1]$ be independent copies of a random variable of mean μ. Then

$$\mathbb{P}\big(\frac{1}{s}\sum_{i=1}^{s} Y_i - \mu \geq \epsilon\big) \leq e^{-2s\epsilon^2} \quad \text{and} \quad \mathbb{P}\big(\frac{1}{s}\sum_{i=1}^{s} Y_i - \mu \leq -\epsilon\big) \leq e^{-2s\epsilon^2}. \tag{1.4}$$

Thus for any fixed $1 \leq s \leq t$,

$$\mathbb{P}\Big(\hat{\mu}_{k,s} + \sqrt{\frac{3\log t}{2s}} \leq \mu_k\Big) \leq e^{-3\log(t)} = t^{-3}, \tag{1.5}$$

and

$$\mathbb{P}\Big(\hat{\mu}_{k,s} - \sqrt{\frac{3\log t}{2s}} \geq \mu_k\Big) \leq e^{-3\log(t)} = t^{-3}. \tag{1.6}$$

We now deduce a bound on the expected number of plays of sub-optimal arms by noticing that with high probability, the sub-optimal arms are not played whenever their UCB is below μ^*.

Proposition 1.1. Each sub-optimal arm k is played in expectation at most

$$\mathbb{E}T_k(n) \leq 6\frac{\log n}{\Delta_k^2} + \frac{\pi^2}{3} + 1$$

time. Thus the cumulative regret of UCB is bounded as

$$\mathbb{E}R_n = \sum_k \Delta_k \mathbb{E}T_k(n) \leq 6 \sum_{k:\Delta_k>0} \frac{\log n}{\Delta_k} + K\big(\frac{\pi^2}{3} + 1\big).$$

First notice that the dependence in n is logarithmic. This says that out of n pulls, the sub-optimal arms are played only $O(\log n)$ times, and thus the optimal arm (assuming there is only one) is played $n - O(\log n)$ times. Now, the constant factor in the logarithmic term is $6\sum_{k:\Delta_k>0}\frac{1}{\Delta_k}$ which deteriorates when some sub-optimal arms are very close to the optimal one (i.e., when Δ_k is small). This may seem counter-intuitive, in the sense that for any fixed value of n, if all the arms have a very small Δ_k, then the regret should be small as well (and this is indeed true since the regret is trivially bounded by $n \max_k \Delta_k$ whatever the algorithm). So this result should be understood (and is meaningful) for a fixed problem (i.e., fixed Δ_k) and for n sufficiently large (i.e., $n > \min_k 1/\Delta_k^2$).

Proof. The proof is simple. Assume that a sub-optimal arm k is pulled at time t. This means that its B-value is larger than the B-values of the other arms, in particular that of the optimal arm k^*:

$$\hat{\mu}_{k,T_k(t-1)} + \sqrt{\frac{3\log t}{2T_k(t-1)}} \geq \hat{\mu}_{k^*,T_{k^*}(t-1)} + \sqrt{\frac{3\log t}{2T_{k^*}(t-1)}}. \tag{1.7}$$

This implies that either the empirical mean of the optimal arm is not within its confidence interval:

$$\hat{\mu}_{k^*,T_{k^*}(t-1)} + \sqrt{\frac{3\log t}{2T_{k^*}(t-1)}} < \mu^*, \tag{1.8}$$

or the empirical mean of the arm k is not within its confidence interval:

$$\mu_{k,T_k(t-1)} > \mu_k + \sqrt{\frac{3\log t}{2T_k(t-1)}}, \tag{1.9}$$

otherwise, we deduce that

$$\mu_k + 2\sqrt{\frac{3\log t}{2T_k(t-1)}} \geq \mu^*,$$

which is equivalent to $T_k(t-1) \leq \frac{6\log t}{\Delta_k^2}$.

This says that whenever $T_k(t-1) \geq \frac{6\log t}{\Delta_k^2} + 1$, either arm k is not pulled at time t, or one of the two small probability events (1.8) or (1.9) does not hold. Thus writing $u \overset{\text{def}}{=} \frac{6\log t}{\Delta_k^2} + 1$, we have:

$$\begin{aligned} T_k(n) &\leq u + \sum_{t=u+1}^{n} \mathbf{1}\{I_t = k; T_k(t) > u\} \\ &\leq u + \sum_{t=u+1}^{n} \mathbf{1}\{(1.8) \text{ or } (1.9) \text{ fails}\}. \end{aligned} \tag{1.10}$$

Now, the probability that (1.8) fails is bounded by

$$\mathbb{P}\Big(\exists 1 \leq s \leq t, \hat{\mu}_{k^*,s} + \sqrt{\frac{3\log t}{2s}} < \mu^*\Big) \leq \sum_{s=1}^{t} \frac{1}{t^3} = \frac{1}{t^2},$$

using Chernoff-Hoeffding inequality (1.5). Similarly the probability that (1.9) fails is bounded by $1/t^2$, thus by taking the expectation in (1.10) we deduce that

$$\begin{aligned} \mathbb{E}[T_k(n)] &\leq \frac{6\log(n)}{\Delta_k^2} + 1 + 2\sum_{t=u+1}^{n} \frac{1}{t^2} \\ &\leq \frac{6\log(n)}{\Delta_k^2} + \frac{\pi^2}{3} + 1 \qquad (1.11) \end{aligned}$$

□

The previous bound depends on some properties of the distributions: the gaps Δ_k. The next result states a problem-independent bound.

Corollary 1.1. The expected regret of UCB is bounded as:

$$\mathbb{E}R_n \leq \sqrt{Kn\big(6\log n + \frac{\pi^2}{3} + 1\big)} \qquad (1.12)$$

Proof. Using Cauchy-Schwarz inequality and the bound on the expected number of pulls of the arms (1.11),

$$\begin{aligned} R_n &= \sum_k \Delta_k \sqrt{\mathbb{E}T_k(n)}\sqrt{\mathbb{E}T_k(n)} \\ &\leq \sqrt{\sum_k \Delta_k^2 \mathbb{E}T_k(n) \sum_k \mathbb{E}T_k(n)} \\ &\leq \sqrt{Kn\big(6\log n + \frac{\pi^2}{3} + 1\big)}. \end{aligned}$$

□

1.1.3 Lower bounds

There are two types of lower bounds: (1) The problem-dependent bounds [81, 37] which say that for a given problem, any "admissible" algorithm will suffer -asymptotically- a logarithmic regret with a constant factor that depends on the arm distributions. (2) The problem-independent bounds [41, 30] which states that for any algorithm and any time-horizon n, there exists an environment on which this algorithm will have a regret at least of order $\sqrt{Kn}$.

Problem-dependent lower bounds: Lai and Robbins [81] considered a class of one-dimensional parametric distributions and showed that any admissible strategy (i.e. such that the algorithm pulls any sub-optimal arm k at most a sub-polynomial number of times: $\forall \alpha > 0$, $\mathbb{E}T_k(n) = o(n^\alpha)$) will asymptotically pull in expectation any sub-optimal arm k at least:

$$\liminf_{n\to\infty} \frac{\mathbb{E}T_k(n)}{\log n} \geq \frac{1}{\mathcal{K}(\nu_k, \nu_{k^*})} \tag{1.13}$$

times (which, from (1.2), enables the deduction of a lower bound on the regret), where $\mathcal{K}(\nu_k, \nu_{k^*})$ is the Kullback-Leibler (KL) divergence between ν_k and ν_{k^*} (i.e., $\mathcal{K}(\nu, \kappa) \overset{\text{def}}{=} \int_0^1 \frac{d\nu}{d\kappa} \log \frac{d\nu}{d\kappa} d\kappa$ if ν is dominated by κ, and $+\infty$ otherwise).

Burnetas and Katehakis [37] extended this result to several classes $\mathcal{P}$ of multi-dimensional parametric distributions. By writing

$$\mathcal{K}_{\inf}(\nu, \mu) \overset{\text{def}}{=} \inf_{\kappa \in \mathcal{P}: E(\kappa) > \mu} \mathcal{K}(\nu, \kappa),$$

(where μ is a real number such that $E(\nu) < \mu$), they showed the improved lower bound on the number of pulls of sub-optimal arms:

$$\liminf_{n\to\infty} \frac{\mathbb{E}T_k(n)}{\log n} \geq \frac{1}{\mathcal{K}_{\inf}(\nu_k, \mu^*)}. \tag{1.14}$$

Those bounds consider a fixed problem and show that any algorithm that is "good enough" on all problems (i.e. what we called an admissible algorithm) cannot be extremely good on any specific instance, and thus needs to suffer some incompressible regret. Note also that these problem-independent lower-bounds are of an asymptotic nature and do not say anything about the regret at any finite time n.

A problem independent lower-bound: In contrast to the previous bounds, we can also derive finite-time bounds that do not depend on the arm distributions: For any algorithm and any time horizon n, there exists an environment (arm distributions) such that this algorithm will suffer some incompressible regret on this environment. We deduce the minimax lower-bounds (see e.g. [41, 30]):

$$\inf \sup \mathbb{E}R_n \geq \frac{1}{20}\sqrt{nK},$$

where the inf is taken over all possible algorithms and the sup over all possible reward distributions of the arms.

1.1.4 Recent improvements

Notice that in the problem-dependent lower-bounds (1.13) and (1.14), the rate is logarithmic, like for the upper bound of UCB, however the constant factor is not the same. In the lower bound it uses KL divergences whereas in the upper bounds the constant is expressed in terms of the difference between the means. From Pinsker's inequality (see e.g. [41]) we have: $\mathcal{K}(\nu,\kappa) \geq (E[\nu]-E[\kappa])^2$ and the discrepancy between $\mathcal{K}(\nu,\kappa)$ and $(E[\nu]-E[\kappa])^2$ can be very large (e.g. for Bernoulli distributions with parameters close to 0 or 1). It follows that there is a potentially large gap between the lower and upper bounds, which motivated several recent attempts to reduce the gap between the upper and lower bounds. The main line of research consists in tightening the concentration inequalities defining the upper confidence bounds.

A first improvement was made in [10] who introduced UCB-V (UCB with variance estimate) that uses a variant of Bernstein's inequality to take into account the empirical variance of the rewards (in addition to their empirical mean) to define tighter UCB on the mean reward of the arms:

$$B_{t,s}(k) \overset{\text{def}}{=} \hat{\mu}_{k,s} + \sqrt{2\frac{V_{k,s}\log(1.2t)}{s}} + \frac{3\log(1.2t)}{s}. \tag{1.15}$$

They proved that the regret is bounded as follows:

$$\mathbb{E}R_n \leq 10\Big(\sum_{k:\Delta_k>0}\frac{\sigma_k^2}{\Delta_k}+2\Big)\log(n),$$

which scales with the actual variance of the arms.

Then [64, 63] proposed the DMED algorithm and proved an asymptotic bound that achieves the asymptotic lower-bound of [37]. Notice that [81] and [37] also provided an algorithm with asymptotic guarantees (under more restrictive conditions). It is only in [54, 85, 39] that a finite-time analysis was derived for KL-based UCB algorithms, KL-UCB and $\mathcal{K}_{\text{inf}}$-UCB, that achieve the asymptotic lower bounds of [81] and [37] respectively. Those algorithms make use of KL divergences in the definition of the UCBs and use the full empirical reward distribution (and not only the two first moments). In addition to their improved analysis in comparison to regular UCB algorithms, several experimental studies showed their improved numerical performance.

Finally let us also mention that the logarithmic gap between the upper and lower problem-independent bounds (see (1.12) and (1.14)) has also been closed (up to a constant factor) by the MOSS algorithm of [11], which achieves a minimax regret bound of order $\sqrt{Kn}$.

1.2 Extensions

The principle of optimism in the face of uncertainty has been successfully extended to several variants of the multi-armed stochastic bandit problem, notably when the number of arms is large (possibly infinite) compared to the number of rounds. In those situations one cannot even pull each arm once and thus in order to achieve meaningful results we need to make some assumptions about the unobserved arms. There are two possible situations:

- When the previously observed arms do not give us any information about unobserved arms. This is the case when there is no structure in the rewards. In those situations, we may rely on a probabilistic assumption on the mean value of any unobserved arm.
- When the previously observed arms can give us some information about unobserved arms: this is the case of structured rewards, for example when the mean reward function is a linear, convex, or Lipschitz function of the arm position, or also when the rewards depend on some tree or graph structure.

We now briefly describe those two situations.

1.2.1 Unstructured rewards

The so-called *many-armed* bandit problem considers a countably infinite number of arms where there is no structure among arms. Thus at any round t the rewards obtained by pulling previously observed arms do not give us information about unobserved arms.

To illustrate, think of the problem of selecting a restaurant for dinner in a big city like Paris. Each day you go to a restaurant and receive a reward reflective of how much you enjoyed the food you were served. You may decide to go back to one of the restaurants you have already visited either because the food you received was good (exploitation) or because you have not been there many times and want to try another dish (exploration). However you may also want to try a new restaurant (discovery) chosen randomly (if you don't have prior information). Of course there are many other applications of this exploration-exploitation-discovery trade-off, such as in marketing (e.g. you want to send catalogs to good customers, uncertain customers, or random people), in mining for valuable resources (such as gold or oil) where you want to exploit good wells, explore unknown wells, or start digging at a new location.

A strong probabilistic assumption that has been made in [17, 19] to model such situations is that the mean-value of any unobserved arm is a random variable that follows some known distribution. More recently this assumption has been weakened in [108] with an assumption focussing on this distribution upper tail only. More precisely, we assume that there exists $\beta > 0$ such that the probability that the mean-reward μ of a new randomly chosen arm is ϵ-optimal, is of order ϵ^β:

$$\mathbb{P}(\mu(\text{new arm}) > \mu^* - \epsilon) = \Theta(\epsilon^\beta), [1] \tag{1.16}$$

where $\mu^* = \sup_{k \geq 1} \mu_k$ is the supremum of the mean-reward of the arms.

Thus the parameter β characterizes the probability of selecting a near-optimal arm. A large value of β indicates that there is a small chance that a new random arm will be good, thus we would need to pull many arms in order to achieve a low regret (defined as in (1.1) with respect to μ^* and not to the best pulled arm).

[1] We write $f(\epsilon) = \Theta(g(\epsilon))$ if $\exists c_1, c_2, \epsilon_0, \forall \epsilon \leq \epsilon_0, c_1 g(\epsilon) \leq f(\epsilon) \leq c_2 g(\epsilon)$.

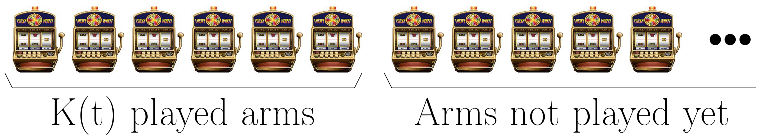

Fig. 1.1 The UCB-AIR strategy: UCB-V algorithm is played on an increasing number $K(t)$ or arms

The UCB-AIR (UCB with Arm Increasing Rule) strategy introduced in [108] consists in playing a UCB-V strategy [10] (see (1.15)) on a set of arms that is increasing in time. Thus at each round, either an arm already played (set of active arms) is chosen using the UCB-V strategy, or a new random arm is selected. Theorem 4 of [108] states that by selecting at each round t a number of active arms defined by

$$K(t) = \begin{cases} \lfloor t^{\frac{\beta}{2}} \rfloor & \text{if } \beta < 1 \text{ and } \mu^* < 1 \\ \lfloor t^{\frac{\beta}{\beta+1}} \rfloor & \text{if } \beta \geq 1 \text{ or } \mu^* = 1 \end{cases}$$

then the regret of UCB-AIR is upper-bounded as:

$$R_n \leq \begin{cases} C\big(\log n\big)^2 \sqrt{n} & \text{if } \beta < 1 \text{ and } \mu^* < 1 \\ C\big(\log n\big)^2 n^{\frac{\beta}{1+\beta}} & \text{if } \mu^* = 1 \text{ or } \beta \geq 1 \end{cases},$$

where C is a (numerical) constant.

This setting illustrates the *exploration-exploitation-discovery tradeoff* where exploitation means pulling an apparently good arm (based on previous observations), exploration means pulling an uncertain arm (already pulled), and discovery means trying a new arm.

An important aspect of this model is that the coefficient β characterizes the probability of choosing randomly a near-optimal arm (thus the proportion of near-optimal arms), and the UCB-AIR algorithm requires the knowledge of this coefficient (since β is used for the choice of $K(t)$). An open question is whether it is possible to design an *adaptive strategy* which would show similar performance even when β is unknown.

Here we see an important characteristic of the performance of the optimistic strategy in a stochastic bandit setting, that will appear several times in different settings in the next chapters:

- The performance depends on a **measure of the quantity of near-optimal solutions,**
- and on **the knowledge we have about this measure.**

1.2.2 Structured bandit problems

In structured bandit problems we assume that the mean-reward of an arm is a function of some arm parameters, where the function belongs to some known class. This includes situations where "arms" denote paths in a tree or a graph (and the reward of a path being the sum of rewards obtained along the edges), or points in some metric space where the reward function has specific structure.

A well-studied case is the *linear bandit* problem where the set of arms $\mathcal{X}$ lies in a Euclidean space $I\!R^d$ and the mean-reward function is linear with respect to (w.r.t.) the arm position $x \in \mathcal{X}$: at time t, one selects an arm $x_t \in \mathcal{X}$ and receives a reward $r_t \overset{\text{def}}{=} \mu(x_t) + \epsilon_t$, with the mean-reward linear function $\mu(x) \overset{\text{def}}{=} x \cdot \theta$ where $\theta \in I\!R^d$ is some (unknown) parameter, and ϵ_t is a (centered, independent) observation noise. The regret is defined w.r.t. the best possible arm $x^* \overset{\text{def}}{=} \arg\max_{x \in \mathcal{X}} \mu(x)$:

$$R_n \overset{\text{def}}{=} n\mu(x^*) - \mathbb{E}\Big[\sum_{t=1}^{n} r_t\Big].$$

Several optimistic algorithms have been introduced and analyzed, such as the *confidence ball* algorithms in [46], as well as refined variants in [95, 2]. The main bounds on the regret are either problem-dependent, of the order $O\Big(\frac{\log n}{\Delta}\Big)$ (where Δ is the mean-reward difference between the best and second best extremal points), or problem-independent of the order[2] $\widetilde{O}(d\sqrt{n})$. Several extensions to the linear setting have been considered, such as *Generalized Linear models* [49] and *sparse linear bandits* [40, 3].

Another popular setting is when the mean-reward function $x \mapsto \mu(x)$ is convex [51, 5] in which case regret bounds of order

[2] where $\widetilde{O}$ stands for a O notation up to a polylogarithmic factor

$O(\text{poly}(d)\sqrt{n})$ can be achieved[3].

Now, other weaker assumptions on the mean-reward function have been considered, such as a Lipschitz assumption in [76, 6, 12, 77] or an even weaker local assumption in [29]. This setting of bandits in metric spaces as well as more general spaces will be investigated in depth in Chapters 3 and 4.

To conclude this brief overview on multi-armed bandits, it is worth mentioning that there have been huge developments in the field of Bandit Theory over the last few years which have produced emerging fields such as *contextual bandits* (where the rewards depend on some observed contextual information), *adversarial bandits* (where the rewards are chosen by an adversary instead of being stochastic), and has drawn strong links with other fields such as *online-learning* (where a statistical learning task is performed online given limited feedback) and *learning from experts* (where one has to perform almost as well as the best expert). The interested reader may consider the following books and PhD theses [41, 30, 84, 31].

1.3 Conclusion

This chapter presented a brief overview of the multi-armed bandit problem which can be seen as a tool for rapidly selecting the best action among a set of possible ones, assuming that each reward sample provides information about the value (mean-reward) of the selected action. In the following we would like to use this tool as a building block to solve more complicated problems where the action space is larger (for example when it is a sequence of actions, or a path in a tree), which would consist in *combining bandits in a hierarchy.* The next chapter introduces the historical motivation for our interest in this problem while the other chapters provide some theoretical and algorithmic material.

[3] where $\text{poly}(d)$ refers to a polynomial in d

2

Historical motivation: Monte-Carlo Tree Search

This chapter presents the historical motivation for our involvement in the topic of hierarchical bandits. It starts with an **experimental success**: UCB-based bandits (see previous Chapter) used in a hierarchy demonstrated impressive performance for tree search in the field of Computer Go, such as in the Go programs Crazy-Stone [45] and MoGo [109, 55]. This impacted the field of *Monte-Carlo-Tree-Search* (MCTS) [43, 24] which provided a simulation-based approach to game programming and has also been used in other sequential decision making problems. However, the analysis of the popular UCT (Upper Confidence Bounds applied to Trees) algorithm [78] have been a **theoretical failure**: the algorithm may perform very poorly (much worse than a uniform search) on some problems and it does not enjoy any finite-time performance guarantee [44].

In this chapter we briefly review the initial idea of performing efficient tree search by assigning a bandit algorithm to each node of the tree and following an optimistic search strategy that explores in priority the most promising branches (according to previous reward samples). We then mention the theoretical difficulties and illustrate the possible failure of such approaches. This was the starting point for

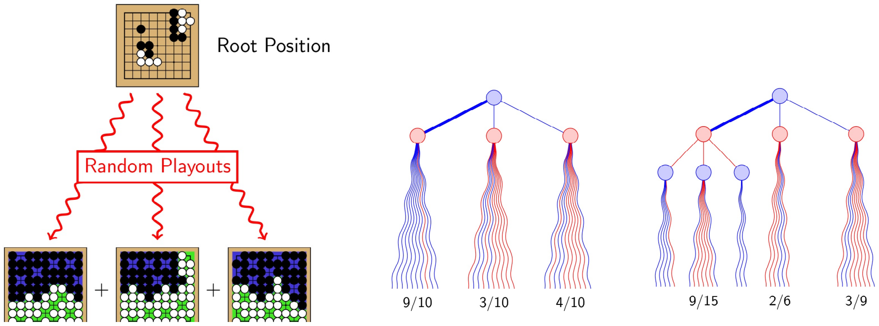

Fig. 2.1 Illustration of the Monte-Carlo Tree Search approach (Courtesy of Rémi Coulom from his talk *The Monte-Carlo revolution in Go*). Left: Monte-Carlo evaluation of a position in Computer Go. Middle: each initial move is sampled several times. Right: The apparently best moves are sampled more often and the tree structure grows.

designing alternative algorithms (described in later chapters) with theoretical performance guarantees which will be analyzed in terms of a new measure of complexity.

2.1 Historical motivation in Computer Go

The use of Monte-Carlo simulations in Computer Go started with the pioneering work of Brügmann [25] followed by Bouzy, Cazenave and Helmstetter [23, 22]. Note that a similar idea was introduced by Abramson in [4] for other games such as Othello. A position is evaluated by running many “playouts” (simulations of a sequence of random moves generated alternatively from the player and the adversary) starting from this position until a terminal configuration is reached. This enables to score each playout (where the winner is decided from a single count of the respective territories), and then averaging the resulting scores. See the illustration in Figure 2.1. This method approximates the value of a Go position (which is actually the solution of a max-min problem) by an average, and thus even if the number of runs goes to infinity, there is not necessary convergence of this average to the max-min value.

An important step was achieved by Coulom [45] in his Crazy-Stone program. In this program, instead of selecting the moves according to a uniform distribution, the probability distribution over possible moves is updated after each simulation so that more weight is assigned to moves that achieved better scores in previous runs (see Figure 2.1). In addition, an incremental tree representation adding a leaf to the current tree representation at each playout enables the construction of an asymmetric tree where the most promising branches (according to the previously observed rewards) are explored to a greater depth.

This was the starting point of the so-called *Monte-Carlo tree search* (MCTS) (see e.g. [43, 24]) that aims to approximate the solution of a max-min problems with a weighted average.

This idea of starting with a uniform sampling over a set of available moves (or actions) and progressively focusing on the best actions according to previously observed rewards is reminiscent of the bandit problem discussed in the previous Chapter. The MoGo program initiated by Yizao Wang, Sylvain Gelly, Olivier Teytaud, Pierre-Arnaud Coquelin and myself [55] started from this simple observation and the idea of performing a tree search by assigning a bandit algorithm to each node of the tree. We started with the UCB algorithm and this lead to the so-called UCT (Upper Confidence Bounds applied to Trees) algorithm, which was independently developed and analyzed by Csaba Szepesvári and Levente Kocsis [78]. Several major improvements (such as the use of features in the random playouts, the Rapid Action Value Estimation (RAVE), the parallelization of the algorithm, and the introduction of opening books) [56, 91, 21, 97, 43, 57] enabled the MoGo program to rank among the best Computer Go programs (see e.g. [82, 1]) until 2012.

2.2 Upper Confidence Bounds in Trees (UCT)

In order to illustrate the UCT algorithm [78], consider a tree search optimization problem on a uniform tree of depth D where each node has K children. A reward distribution ν_i is assigned to each leaf i (there are K^D such leaves) and the goal is to find the path (sequence of nodes from the root) to a leaf with highest mean-value $\mu_i \overset{\text{def}}{=} E[\nu_i]$. Define the value of any node k as $\mu_k \overset{\text{def}}{=} \max_{i\in\mathcal{L}(k)} \mu_i$, where $\mathcal{L}(k)$ denotes the set of leaves in the branch starting from k.

At any round t, the UCT algorithm selects a leaf I_t of the tree and receives a reward $r_t \sim \nu_{I_t}$ which enables it to update the B-values of all nodes in the tree. The way the leaf is selected is by following a path starting from the root and from each node j along the path, the next node is chosen to be the one with highest B-value among the children nodes, where the B-value of any child k of node j is defined as:

$$B_t(k) \overset{\text{def}}{=} \hat{\mu}_{k,t} + c\sqrt{\frac{\log T_j(t)}{T_k(t)}}, \tag{2.1}$$

where c is a numerical constant, $T_k(t) \overset{\text{def}}{=} \sum_{s=1}^{t} \mathbf{1}\{I_s \in \mathcal{L}(k)\}$ is the number of paths that went through node k up to time t (and similarly for $T_j(t)$), and $\hat{\mu}_{k,t}$ is the empirical average of rewards obtained from leaves originating from node k, i.e.,

$$\hat{\mu}_{k,t} \overset{\text{def}}{=} \frac{1}{T_k(t)} \sum_{s=1}^{t} r_s \mathbf{1}\{I_s \in \mathcal{L}(k)\}.$$

The intuition for the UCT algorithm is that at the level of a given node j, there are K possible choices, i.e. arms, corresponding to the children nodes, and the use of a UCB-type of bandit algorithm should enable the selection of the best arm given noisy rewards samples.

Now, when the number of simulations goes to infinity, since UCB selects all arms infinitely often (indeed, thanks to the log term in the definition of the B-values (2.1), when a children node k is not chosen, its B-value increases and thus it will eventually be selected, as long as its parent j is), we deduce that UCT selects all leaves infinitely often. Thus from an immediate backward induction from the leaves to the root of the tree we deduce that UCT is consistent, i.e. for any node k, $\lim_{t\to\infty} \hat{\mu}_t(k) = \mu(k)$, almost surely.

The main reason this algorithm demonstrated interesting numerical performance in several large tree search problems is that it explores in priority the most promising branches according to previously observed sample rewards. This mainly happens in situations where the reward function possesses some smoothness property (so that initial random reward samples provide information about where the search should focus) or when no other technique can be applied (e.g. in Computer Go where the branching factor is so large that regular minimax or alpha-beta methods fail). See [42, 97, 43, 24] and the references therein for different variants of MCTS and applications to games and other search, optimization, and control problems. These types of algorithms appear as possible alternatives to usual depth-first or breadth-first search techniques and apparently implement an optimistic exploration of the search space. Unfortunately in the next Section we show that this algorithm does not enjoy any finite-time performance guarantee and performs very poorly on some problems.

2.3 No finite-time performance for UCT

The main problem comes from the fact that the reward samples r_t obtained from any node k are not independent and identically distributed (i.i.d.). Indeed, such a reward $r_t \sim \nu_{I_t}$ depends on the selected leaf $I_t \in \mathcal{L}(k)$, which itself depends on the arm selection process along the path from node k to the leaf I_t, thus potentially on all previously observed rewards. Thus the B-values $B_t(k)$ defined by (2.1) do not define high-probability upper-confidence-bounds on the value μ_k of the arm (i.e. we cannot apply Chernoff-Hoeffding inequality). Thus the analysis of UCB seen in Section 1.1.2 does not apply.

The potential risk of UCT is to stop exploring the optimal branch too early because the current B-value of that branch is under-estimated. It is true that the algorithm is consistent (as discussed previously) and the optimal path will eventually be discovered but the time it takes for the algorithm to do so can be desperately long.

This point is described in [44] with an illustrative example reproduced in Figure 2.2. This is a binary tree of depth D. The rewards are deterministic and defined as follows: For any node of depth $d < D$ in the optimal branch (rightmost one), if Left action is chosen, then a reward of $\frac{D-d}{D}$ is received (all leaves in this branch have the same reward). If Right action is chosen, then this moves to the next node in the optimal branch. At depth $D-1$, Left action yields reward 0 and Right action reward 1.

For this problem, as long as the optimal reward has not been observed, from any node along the optimal path, the left branches seem better than the right ones and are thus explored exponentially more

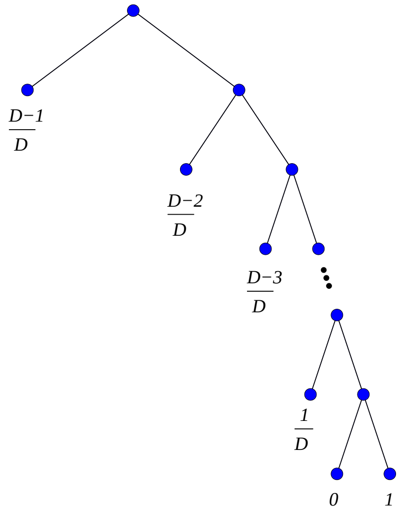

Fig. 2.2 An example of tree for which UCT performs very poorly.

often. Therefore, the time required before the optimal leaf is eventually reached is huge and we can deduce the following lower-bound on the regret of UCT:

$$R_n = c\,\underbrace{\exp(\exp(\dots\exp(}_{D\text{ times}}1)\dots)) + \Omega(\log(n)),$$

for some constant c. The first term of this bound is a constant independent of n (thus the regret is asymptotically of order $\log n$ as proven in (2.1)) but this constant is "D-uply" exponential. In particular this is much worse than a uniform sampling of all the leaves which will be "only" exponential in D.

The reason why this is a particularly hard problem for UCT is that the initial reward samples collected by the algorithm are strongly misleading at each level along the optimal path. Actually, since the B-values do not represent high-probability UCB on the true value of the nodes, the UCT strategy does not implement the *optimism in the face of uncertainty* principle.

This observation is the historical motivation for the research described in the next chapters. UCT is very efficient in some well-structured problems and could be very inefficient in tricky problems (the majority of them...). Our objectives are now to recover the *optimism in the face of uncertainty* principle by defining algorithms making use of *true* high-probability UCBs. So we need to define the classes of problems for which performance guarantees can be obtained, or better, define new measures of the problem complexity and derive finite-time performance bounds in terms of this measure of complexity in situations where this quantity is known, and when it is not.

3

Optimistic optimization with known smoothness

In this chapter we consider the *optimism in the face of uncertainty* principle applied to the problem of black-box optimization of a function f given (deterministic or stochastic) evaluations of the function.

We search for a good approximation of the maximum of a function $f : \mathcal{X} \to I\!R$ using a finite number n (i.e. the numerical budget) of function evaluations. More precisely, we want to design a sequential exploration strategy $\mathcal{A}$ of the search space $\mathcal{X}$, i.e. a sequence $x_1, x_2, \ldots, x_n$ of states of $\mathcal{X}$, where each x_t may depend on previously observed values $f(x_1), \ldots, f(x_{t-1})$, such that at round n (which may or may not be known in advance), the algorithm $\mathcal{A}$ recommends a state $x(n)$ with the highest possible value. The performance of the algorithm is assessed by the loss (or simple regret):

$$r_n = \sup_{x \in \mathcal{X}} f(x) - f(x(n)). \tag{3.1}$$

Here the performance criterion is the closeness to optimality of the recommendation made after n evaluations to the function. This criterion is different from the cumulative regret previously defined in the multi-armed bandit setting (see Chapter 1):

$$R_n \overset{\text{def}}{=} \sup_{x \in \mathcal{X}} f(x) - \sum_{t=1}^{n} f(x_t), \tag{3.2}$$

which measures how well the algorithm succeeds in selecting states with good values while exploring the search space (notice that we write $x_1, \ldots, x_n$ as the states selected for evaluation, whereas $x(n)$ refers to the recommendation made by the algorithm after n observations, and may differ from x_n). The two settings provide different exploration-exploitation tradeoffs in the multi-armed bandit setting (see [27, 9] for thorough comparison between the settings). In this chapter we consider the loss criterion (3.1), which induces the so-called **numerical exploration-exploitation trade-off**, since it more naturally relates to the problem of function optimization given a finite simulation budget (whereas the cumulative regret (3.2) mainly applies to the problem of optimizing while learning an unknown environment).

Since the literature on global optimization is very important, we only mention the works that are closely related to the optimistic strategy described here. A large body of algorithmic work has been developed using branch-and-bound techniques [86, 61, 72, 65, 90, 52, 100] such as Lipschitz optimization where the function is assumed to be globally Lipschitz. For illustration purpose, Section 3.1 provides an intuitive introduction to the optimistic optimization strategy in the case when the function is assumed to be Lipschitz. The next sample is chosen to be the maximum of an upper-bounding function which is built from previously observed values and knowledge of the function smoothness. This enables the algorithm to achieve a good numerical exploration-exploitation trade off that makes an efficient use of the available numerical resources in order to rapidly estimate the maximum of f.

However the main contribution of this chapter (starting from Section 3.2 where the general setting is introduced) is to considerably weaken the assumptions made in most of the previous literature since we do not require the space $\mathcal{X}$ to be a metric space but only to be equipped with a semi-metric ℓ, and we relax the assumption that f is globally Lipschitz into a much weaker assumption that f is locally smooth w.r.t. ℓ (this definition is made precise in Section 3.2.2). In this chapter we assume that **the semi-metric ℓ (under which f is smooth) is known**.

The case of deterministic evaluations is presented in Section 3.3 where a first algorithm, Deterministic Optimistic Optimization (DOO) is introduced and analyzed. In Section 3.4, the same ideas are extended to the case of stochastic evaluations of the function, which corresponds to the so-called *$\mathcal{X}$-armed bandit*, and two algorithms Stochastic Optimistic Optimization (StoOO) and Hierarchical Optimistic Optimization (HOO) are described and analyzed.

The main result is that we can characterize the performance of those algorithms using a measure that depends both on the function f and the semi-metric ℓ, which represents the quantity of near-optimal states and is called the **near-optimality dimension of f under ℓ**. We show that if the behavior of the function around its (global) maxima is known, then one can select the semi-metric ℓ such that the corresponding near-optimality dimension is low, which implies efficient optimization algorithms (whose loss rate does not depend on the space dimension). However the performance deteriorates when this smoothness is not correctly estimated.

3.1 Illustrative example

In order to illustrate the approach, we consider the simple case where the space $\mathcal{X}$ is metric (let ℓ denote the metric) and the function $f : \mathcal{X} \to I\!R$ is Lipschitz continuous, i.e., for all $x, y \in \mathcal{X}$,

$$|f(x) - f(y)| \leq \ell(x, y). \tag{3.3}$$

Define the numerical budget n as the total number of calls to the function. At each round for $t = 1$ to n, the algorithm selects a state $x_t \in X$, then either (in the **deterministic case**) observes the exact value of the function $f(x_t)$, or (in the **stochastic case**) observes a noisy estimate r_t of $f(x_t)$, such that $\mathbb{E}[r_t|x_t] = f(x_t)$.

This chapter is informal and all theoretical results are deferred to the next chapters (which describe a much broader setting where the function does not need to be Lipschitz and the space does not need

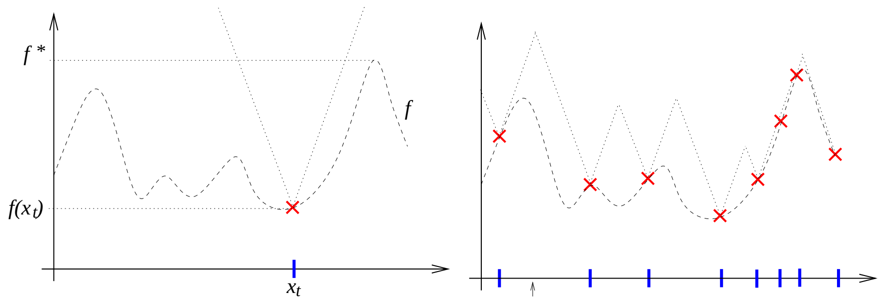

Fig. 3.1 Left: The function f (dotted line) is evaluated at a point x_t, which provides a first upper bound on f (given the Lipschitz assumption). Right: several evaluations of f enable the refinement of its upper-bound. The optimistic strategy samples the function at the point with highest upper-bound.

to be metric). The purpose of this chapter is simply to provide some intuition of the optimistic approach for the optimization problem.

3.1.1 Deterministic setting

In this setting, the evaluations are deterministic, thus exploration does not refer to improving our knowledge about some stochastic environment but consists in evaluating the function at unknown but possibly important areas of the search space, in order to estimate the global maximum of the function.

Given that the function is Lipschitz continuous and that we know ℓ, an evaluation of the function at any point x_t enables the definition of an upper envelope of f: for all $x \in \mathcal{X}$, $f(x) \leq f(x_t) + l(x, x_t)$. This upper envelope is refined after each evaluation of f by taking the minimum of the previous upper-bounds (see illustration on Figure 3.1): for all $x \in \mathcal{X}$,

$$f(x) \leq B_t(x) \overset{\text{def}}{=} \min_{1 \leq s \leq t} \left[f(x_s) + l(x, x_s) \right]. \tag{3.4}$$

Now, the optimistic approach consists in selecting the next state x_{t+1} as the point with highest upper bound:

$$x_{t+1} = \arg\max_{x \in \mathcal{X}} B_t(x). \tag{3.5}$$

We can say that this strategy follows an **"optimism in the face**

of computational uncertainty" principle. The uncertainty does not come from the stochasticity of some unknown environment (as it was the case in the stochastic bandit setting), but from the uncertainty about the function given that the search space may be infinite and we possess a finite computational budget only.

Remark 3.1. Notice that we only need the property that $B_t(x)$ is an upper-bound on $f(x)$ at the (global) maxima x^* of f. Indeed, the algorithm selecting at each round a state $\arg\max_{x\in\mathcal{X}} B_t(x)$ will not be affected by having a $B_t(x)$ function under-evaluating $f(x)$ at sub-optimal points $x \neq x^*$. Thus in order to apply this optimistic sampling strategy, one really needs (3.4) to hold for x^* only (instead of requiring it for all $x \in \mathcal{X}$). Thus we see that the global Lipschitz assumption (3.3) may be replaced by the much weaker assumption that for all $x \in \mathcal{X}$, $f(x^*)-f(x) \leq \ell(x, x^*)$. This case will be further detailed in Section 3.2.

Several issues remain to be addressed: (1) How do we generalize this approach to the case of stochastic rewards? (2) How do we deal with the computational problem of computing the maximum of the upper-bounding function in (3.5)? Question 1 is the object of the next subsection, and Question 2 will be addressed by considering a hierarchical partitioning of the space that will be discussed in Section 3.2.

3.1.2 Stochastic setting

Now consider the stochastic case, where the evaluations to the function are perturbed by noise (see Figure 3.2). More precisely, an evaluation of f at x_t returns a noisy estimate r_t of $f(x_t)$ where we assume that $\mathbb{E}[r_t|x_t] = f(x_t)$.

In order to follow the *optimism in the face of uncertainty* principle, one would like to define a high probability upper bound $B_t(x)$ on $f(x)$ at any state $x \in \mathcal{X}$ and select the point with highest bound $\arg\max_{x\in\mathcal{X}} B_t(x)$. So the question is how to define this UCB function.

A possible answer to this question is to consider a given subset $X_i \subset \mathcal{X}$ containing x and define a UCB on $\sup_{x\in X_i} f(x)$. This can be done by averaging the rewards observed by points sampled in X_i and using the Lipschitz assumption on f.

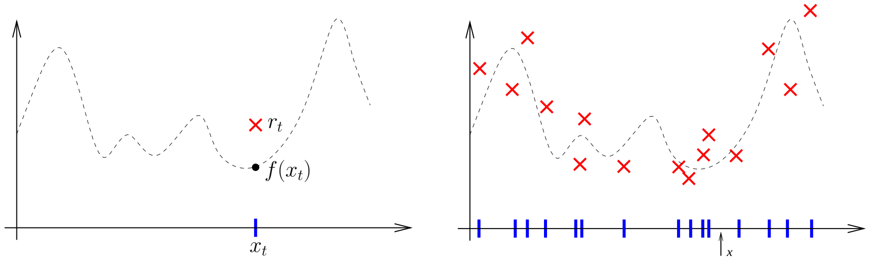

Fig. 3.2 The evaluation of the function is perturbed by a centered noise: $\mathbb{E}[r_t|x_t] = f(x_t)$. How should we define a high-probability upper-confidence-bound on f at any state x in order to implement the *optimism in the face of uncertainty* principle?

More precisely, let $T_i(t) \overset{\text{def}}{=} \sum_{u=1}^{t} \mathbf{1}\{x_u \in X_i\}$ be the number of points sampled in X_i and write τ_s the absolute time instant when X_i was sampled for the s-th time, i.e. $\tau_s = \min\{u : T_i(u) = s\}$. Notice that $\sum_{u=1}^{t}(r_u - f(x_u))\mathbf{1}\{x_u \in X_i\} = \sum_{s=1}^{T_i(t)}(r_{\tau_s} - f(x_{\tau_s}))$ is a Martingale (w.r.t. the filtration generated by the sequence $\{(r_{\tau_s}, x_{\tau_s})\}_s$). Thus, we have

$$
\begin{aligned}
&\mathbb{P}\Big(\frac{1}{T_i(t)}\sum_{s=1}^{T_i(t)}\big[r_{\tau_s} - f(s_{\tau_s})\big] \le -\epsilon_{t,T_i(t)}\Big)\\
&\le \mathbb{P}\Big(\exists 1 \le u \le t, \frac{1}{u}\sum_{s=1}^{u}\big[r_{\tau_s} - f(s_{\tau_s})\big] \le -\epsilon_{t,u}\Big)\\
&\le \sum_{u=1}^{t}\mathbb{P}\Big(\frac{1}{u}\sum_{s=1}^{u}\big[r_{\tau_s} - f(s_{\tau_s})\big] \le -\epsilon_{t,u}\Big)\\
&\le \sum_{u=1}^{t} e^{-2u\epsilon_{t,u}^2},
\end{aligned}
$$

where we used a union bound in the third line and Hoeffding-Azuma inequality [16] in the last derivation. For any $\delta > 0$, setting $\epsilon_{t,u} = \sqrt{\frac{\log t/\delta}{2u}}$ we deduce that with probability $1 - \delta$, we have

$$
\frac{1}{T_i(t)}\sum_{s=1}^{T_i(t)} r_{\tau_s} + \sqrt{\frac{\log t/\delta}{2T_i(t)}} \ge \frac{1}{T_i(t)}\sum_{s=1}^{T_i(t)} f(s_{\tau_s}). \tag{3.6}
$$

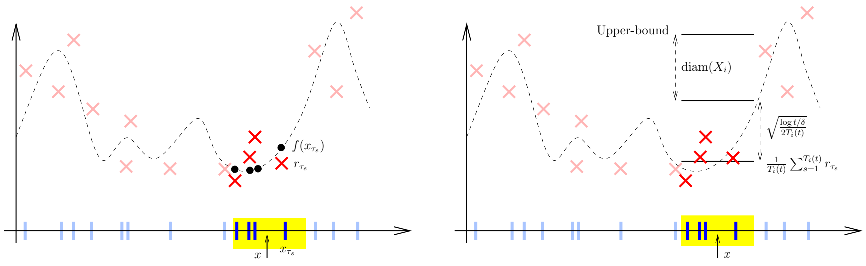
$f(x_{\tau_s})$
r_{τ_s}
x
x_{τ_s}
Upper-bound
$\mathrm{diam}(X_i)$
$\sqrt{\frac{\log t/\delta}{2T_i(t)}}$
$\frac{1}{T_i(t)}\sum_{s=1}^{T_i(t)} r_{\tau_s}$
x

Fig. 3.3 A possible way to define a high-probability bound on f at any $x \in \mathcal{X}$ is to consider a subset $X_i \ni x$ and average the $T_i(t)$ rewards obtained in this subset $\sum_{s=1}^{T_i(t)} r_{\tau_s}$, then add a confidence interval term $\sqrt{\frac{\log t/\delta}{2T_i(t)}}$, and add the diameter $\mathrm{diam}(X_i)$. This defines an UCB (with probability $1-\delta$) on f at any $x \in X_i$.

Now we can use the Lipschitz property of f to define a high probability UCB on $\sup_{x\in X_i} f(x)$. Indeed each term in the r.h.s. of (3.6) is bounded as $f(x_{\tau_s}) \geq \max_{x\in X_i} f(x) - \mathrm{diam}(X_i)$, where the diameter of X_i is defined as $\mathrm{diam}(X_i) \overset{\text{def}}{=} \max_{x,y\in X_i} \ell(x,y)$. We deduce that with probability $1-\delta$, we have

$$B_{t,T_i(t)}(X_i) \overset{\text{def}}{=} \frac{1}{T_i(t)} \sum_{s=1}^{T_i(t)} r_{\tau_s} + \sqrt{\frac{\log t/\delta}{2T_i(t)}} + \mathrm{diam}(X_i) \geq \max_{x\in X_i} f(x). \quad (3.7)$$

This UCB is illustrated in Figure 3.3.

Remark 3.2. We see a trade off in the choice of the size of X_i: The bound (3.7) is poor either (1) when $\mathrm{diam}(X_i)$ is large, or (2) when X_i contains so few samples (i.e. $T_i(t)$ is small) that the confidence interval term is large.

Ideally we would like to consider several possible subsets X_i (of different size) containing a given $x \in \mathcal{X}$ and define several UCBs on $f(x)$ and select the tightest one: $B_t(x) \overset{\text{def}}{=} \min_{i;x\in X_i} B_{t,T_i(t)}(X_i)$. Now, an optimistic strategy would simply compute the tightest UCB at each state $x \in \mathcal{X}$ according to the rewards already observed, and choose the next state to sample as the one with highest UCB, as in (3.5).

However this poses several problems: (1) One cannot consider concentration inequalities on an arbitrarily large number of subsets (since

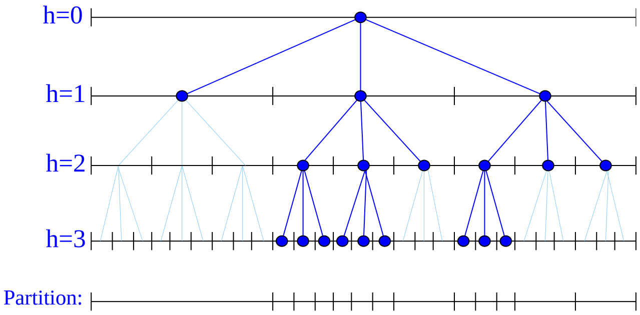

Fig. 3.4 Hierarchical partitioning of the space $\mathcal{X}$ equivalently represented by a K-ary tree (here $K = 3$). The set of leaves of any subtree corresponds to a partition of $\mathcal{X}$.

we would need a union bound over too large a number of events), (2) From a computational point of view, it may not be easy to compute the maximum point of the bounds if the shapes of the subsets are arbitrary. In order to provide a simple answer to both issues we consider a **hierarchical partitioning of the space**. This is the approach followed in the next section, which introduces the general setting.

3.2 General setting

3.2.1 Hierarchical partitioning

In order to address the computational problem of computing the optimum of the upper-bound (3.5) described above, our algorithms will use a hierarchical partitioning of the space $\mathcal{X}$.

More precisely, we consider a set of partitions of $\mathcal{X}$ at all scales $h \geq 0$: For any integer h, $\mathcal{X}$ is partitioned into a set of K^h subsets $X_{h,i}$ (called cells), where $0 \leq i \leq K^h - 1$. This partitioning may be represented by a K-ary tree where the root corresponds to the whole domain $\mathcal{X}$ (cell $X_{0,0}$) and each cell $X_{h,i}$ corresponds to a node (h,i) of the tree (indexed by its depth h and index i), and each node (h,i) possesses K children nodes $\{(h+1,i_k)\}_{1\leq k\leq K}$ such that the associated cells $\{X_{h+1,i_k}, 1 \leq k \leq K\}$ form a partition of the parent's cell $X_{h,i}$.

In addition, to each cell $X_{h,i}$ is assigned a specific state $x_{h,i} \in X_{h,i}$, that we call the *center* of $X_{h,i}$ where f may be evaluated.

3.2.2 Assumptions

We now state 4 assumptions: Assumption 1 is about the semi-metric ℓ, Assumption 2 is about the smoothness of the function w.r.t. ℓ, and Assumptions 3 and 4 are about the shape of the hierarchical partitioning w.r.t. ℓ.

Assumption 1 (Semi-metric). We assume that $\mathcal{X}$ is equipped with a semi-metric $\ell : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$. We recall that this means that for all $x, y \in \mathcal{X}$, we have $\ell(x,y) = \ell(y,x)$ and $\ell(x,y) = 0$ if and only if $x = y$.

Note that we do not require that ℓ satisfies the triangle inequality (in which case, ℓ would be a metric). An example of a metric space is the Euclidean space $\mathbb{R}^d$ with the metric $\ell(x,y) = \|x - y\|$ (Euclidean norm). Now consider $\mathbb{R}^d$ with $\ell(x,y) = \|x - y\|^\alpha$, for some $\alpha > 0$. When $\alpha \leq 1$, then ℓ is also a metric, but whenever $\alpha > 1$ then ℓ does not satisfy the triangle inequality anymore, and is thus a semi-metric only.

Now we state our assumption about the function f.

Assumption 2 (Local smoothness of f). There exists at least one global optimizer $x^* \in \mathcal{X}$ of f (i.e., $f(x^*) = \sup_{x \in \mathcal{X}} f(x)$) and for all $x \in \mathcal{X}$,

$$f(x^*) - f(x) \leq \ell(x, x^*). \tag{3.8}$$

This condition guarantees that f does not decrease too fast around (at least) one global optimum x^* (this is a sort of a locally one-sided Lipschitz assumption). Note that although it is required that (3.8) be satisfied for all $x \in \mathcal{X}$, this assumption essentially sets constraints to the function f locally around x^* (since when x is such that $\ell(x, x^*) > \text{range}(f) \overset{\text{def}}{=} \sup f - \inf f$, then the assumption is automatically satisfied). Thus when this property holds, we say that f **is locally smooth w.r.t. ℓ around its maximum**. See an illustration in Figure 3.5.

Now we state the assumptions about the hierarchical partitioning.

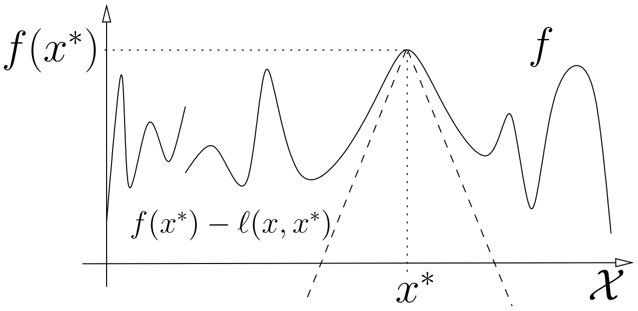

Fig. 3.5 Illustration of the local smoothness property of f around x^* w.r.t. the semi-metric ℓ: the function $f(x)$ is lower-bounded by $f(x^*) - \ell(x, x^*)$. This essentially constrains f around x^* since for x away from x^* the function can be arbitrarilly non-smooth (e.g., discontinuous).

Assumption 3 (Bounded diameters). There exists a decreasing sequence $\delta(h) > 0$, such that for any depth $h \geq 0$ and for any cell $X_{h,i}$ of depth h, we have $\sup_{x \in X_{h,i}} \ell(x_{h,i}, x) \leq \delta(h)$.

Assumption 4 (Well-shaped cells). There exists $\nu > 0$ such that for any depth $h \geq 0$, any cell $X_{h,i}$ contains a ℓ-ball of radius $\nu\delta(h)$ centered in $x_{h,i}$.

In this chapter, we consider the setting where Assumptions 1-4 hold for a specific semi-metric ℓ, and that **the semi-metric ℓ is known from the algorithm**.

3.3 The DOO Algorithm

The Deterministic Optimistic Optimization (DOO) algorithm described in Figure 3.6 uses explicitly the knowledge of ℓ (through the use of $\delta(h)$).

DOO builds incrementally a tree $\mathcal{T}_t$ for $t = 1 \ldots n$, starting with the root node $\mathcal{T}_1 = \{(0, 0)\}$, and by selecting at each round t a leaf of the current tree $\mathcal{T}_t$ to expand. Expanding a leaf means adding its

```
**Initialization:** $\mathcal{T}_1 = \{(0,0)\}$ (root node)
**for** $t = 1$ to $n$ **do**
   Select the leaf $(h,j) \in \mathcal{L}_t$ with maximum $b_{h,j} \overset{\text{def}}{=} f(x_{h,j}) + \delta(h)$ value.
   Expand this node: add to $\mathcal{T}_t$ the $K$ children of $(h,j)$ and evaluate the
   function at the points $\{x_{h+1,j_1}, \ldots, x_{h+1,j_K}\}$
**end for**
Return $x(n) = \arg\max_{(h,i)\in\mathcal{T}_n} f(x_{h,i})$
```

Fig. 3.6 Deterministic Optimistic Optimization (DOO) algorithm.

K children to the current tree (this corresponds to splitting the cell $X_{h,j}$ into K children-cells $\{X_{h+1,j_1}, \ldots, X_{h+1,j_K}\}$) and evaluating the function at the centers $\{x_{h+1,j_1}, \ldots, x_{h+1,j_K}\}$ of the children cells. We write $\mathcal{L}_t$ the leaves of $\mathcal{T}_t$ (set of nodes whose children are not in $\mathcal{T}_t$), which are the set of nodes that can be expanded at round t.

The algorithm computes a b-value $b_{h,j} \overset{\text{def}}{=} f(x_{h,j}) + \delta(h)$ for each leaf $(h,j) \in \mathcal{L}_t$ of the current tree $\mathcal{T}_t$ and selects the leaf with highest b-value to expand next. Once the numerical budget is over (here, n node expansions, thus nK function evaluations), DOO returns the evaluated state $x(n) \in \{x_{h,i}, (h,i) \in \mathcal{T}_n\}$ with highest value.

This algorithm follows an optimistic principle because it expands at each round a cell that may contain the optimum of f, based on the information about (i) the previously observed evaluations of f, and (ii) the knowledge of the local smoothness property (3.8) of f (since ℓ is known).

Thus the use of the hierarchical partitioning provides a computationally efficient implementation of the optimistic sampling strategy described in Section 3.1 and illustrated in Figure 3.1. The (numerically heavy) problem of selecting the state with highest upper-bound (3.5) is replaced by the (easy) problem of selecting the cell with highest upper bound to expand next.

3.3.1 Analysis of DOO

Notice that Assumption 2 implies that the b-value of any cell containing x^* upper bounds f^*, i.e., for any cell $X_{h,i}$ such that $x^* \in X_{h,i}$,

$$b_{h,i} = f(x_{h,i}) + \delta(h) \geq f(x_{h,i}) + \ell(x_{h,i}, x^*) \geq f^*.$$

As a consequence, a leaf (h, i) such that $f(x_{h,i}) + \delta(h) < f^*$ will never be expanded (since at any time t, the b-value of such a leaf will be dominated by the b-value of the leaf containing x^*). We deduce that DOO only expands nodes of the set $I \overset{\text{def}}{=} \cup_{h \geq 0} I_h$, where

$$I_h \overset{\text{def}}{=} \{\text{nodes } (h, i) \text{ such that } f(x_{h,i}) + \delta(h) \geq f^*\}.$$

In order to derive a loss bound we now define a measure of the quantity of near-optimal states, called *near-optimality dimension*. This measure is closely related to similar measures introduced in [77, 28]. For any $\epsilon > 0$, let us write

$$\mathcal{X}_\epsilon \overset{\text{def}}{=} \{x \in \mathcal{X}, f(x) \geq f^* - \epsilon\}$$

the set of ϵ-optimal states.

Definition 3.1. The η-near-optimality dimension is the smallest $d \geq 0$ such that there exists $C > 0$ such that for any $\epsilon > 0$, the maximal number of disjoint ℓ-balls of radius $\eta\epsilon$ and center in $\mathcal{X}_\epsilon$ is less than $C\epsilon^{-d}$.

Note that d is not an intrinsic property of f: it characterizes both f and ℓ (since we use ℓ-balls in the packing of near-optimal states), and also depends on the constant η.

Remark 3.3. Notice that in the definition of the near-optimality dimension, we require the packing property to hold for any $\epsilon > 0$. We can also define a *local* near-optimality dimension by requiring this packing property to only hold for all $\epsilon \leq \epsilon_0$ for some $\epsilon_0 \geq 0$. However if the space $\mathcal{X}$ has finite packing dimension, the near-optimality and local near-optimality dimensions coincide. Only the constant C in their definition may change. Thus we see that the near-optimality dimension

d captures a local property of f near x^* whereas the corresponding constant C depends on the global shape of f.

We now bound the number of nodes in I_h.

Lemma 3.1. Let d be the ν-near-optimality dimension (where ν is defined in Assumption 4), and C the corresponding constant. Then

$$|I_h| \le C\delta(h)^{-d}.$$

Proof. From Assumption 4, each cell (h,i) contains a ball of radius $\nu\delta(h)$ centered in $x_{h,i}$, thus if $|I_h| = |\{x_{h,i} \in \mathcal{X}_{\delta(h)}\}|$ exceeded $C\delta(h)^{-d}$, this would mean that there exists more than $C\delta(h)^{-d}$ disjoint ℓ-balls of radius $\nu\delta(h)$ with center in $\mathcal{X}_{\delta(h)}$, which contradicts the definition of d. □

We now provide our loss bound for DOO.

Theorem 3.1. Let us write $h(n)$ the smallest integer h such that $C\sum_{l=0}^{h}\delta(l)^{-d} \ge n$. Then the loss of DOO is bounded as $r_n \le \delta(h(n))$.

Proof. Let $(h_{\max}, j_{\max})$ be the deepest node that has been expanded by the algorithm up to round n. We known that DOO only expands nodes in the set I. Thus the total number of expanded nodes n is such that

$$\begin{aligned} n &= \sum_{l=0}^{h_{\max}} \sum_{j=0}^{K^l-1} \mathbf{1}\{(h,j) \text{ has been expanded}\} \\ &\le \sum_{l=0}^{h_{\max}} |I_l| \le C \sum_{l=0}^{h_{\max}} \delta(l)^{-d}, \end{aligned}$$

from Lemma 3.1. Now from the definition of $h(n)$ we have $h_{\max} \ge h(n)$. Now since node $(h_{\max}, j_{\max})$ has been expanded, we have that $(h_{\max}, j_{\max}) \in I$, thus

$$f(x(n)) \ge f(x_{h_{\max},j_{\max}}) \ge f^* - \delta(h_{\max}) \ge f^* - \delta(h(n)).$$

□

Now, let us make the bound more explicit when the diameter $\delta(h)$ of the cells decreases exponentially fast with their depth (this case is rather general as illustrated in the examples described next, as well as in the discussion in [29]).

Corollary 3.1. Assume that $\delta(h) = c\gamma^h$ for some constants $c > 0$ and $\gamma < 1$.

- If $d > 0$, then the loss decreases polynomially fast:

$$r_n \le \left(\frac{C}{1-\gamma^d}\right)^{1/d} n^{-1/d}.$$

- If $d = 0$, then the loss decreases exponentially fast:

$$r_n \le c\gamma^{(n/C)-1}.$$

Proof. From Theorem 3.1, whenever $d > 0$ we have $n \le C\sum_{l=0}^{h(n)} \delta(l)^{-d} = Cc^{-d}\frac{\gamma^{-d(h(n)+1)}-1}{\gamma^{-d}-1}$, thus $\gamma^{-dh(n)} \ge \frac{n}{Cc^{-d}}\big(1-\gamma^d\big)$, from which we deduce that $r_n \le \delta(h(n)) \le c\gamma^{h(n)} \le \big(\frac{C}{1-\gamma^d}\big)^{1/d} n^{-1/d}$.

Now, if $d = 0$ then $n \le C\sum_{l=0}^{h(n)} \delta(l)^{-d} = C(h(n)+1)$, and we deduce that the loss is bounded as $r_n \le \delta(h(n)) = c\gamma^{(n/C)-1}$. □

Remark 3.4. Notice that in Theorem 3.1 and Corollary 3.1 the loss bound is expressed in terms of the number of node expansions n. The corresponding number of function evaluations is Kn (since each node expansion generates K children where the function is evaluated).

3.3.2 Examples

Example 1: Let $\mathcal{X} = [-1,1]^D$ and f be the function $f(x) = 1 - \|x\|_\infty^\alpha$, for some $\alpha \ge 1$. Consider a $K = 2^D$-ary tree of partitions with (hyper)-squares. Expanding a node means splitting the corresponding square in 2^D squares of half length. Let $x_{h,i}$ be the center of $X_{h,i}$.

Consider the following choice of the semi metric: $\ell(x,y) = \|x-y\|_\infty^\beta$, with $\beta \le \alpha$. We have $\delta(h) = 2^{-h\beta}$ (recall that $\delta(h)$ is defined in terms of ℓ), and $\nu = 1$. The optimum of f is $x^* = 0$ and f satisfies the local smoothness property (3.8). Now let us compute its near-optimality dimension. For any $\epsilon > 0$, $\mathcal{X}_\epsilon$ is the L_∞-ball of radius $\epsilon^{1/\alpha}$ centered in 0, which can be packed by $\big(\frac{\epsilon^{1/\alpha}}{\epsilon^{1/\beta}}\big)^D$ L_∞-balls of diameter ϵ (since a L_∞-balls of diameter ϵ is a ℓ-ball of diameter $\epsilon^{1/\beta}$). Thus the near-optimality dimension is $d = D(1/\beta - 1/\alpha)$ (and the constant $C = 1$). From Corollary 3.1 we deduce that (i) when $\alpha > \beta$, then $d > 0$ and in this case, $r_n = O\big(n^{-\frac{1}{D}\frac{\alpha\beta}{\alpha-\beta}}\big)$, and (ii) when $\alpha = \beta$, then $d = 0$ and the loss decreases exponentially fast: $r_n \leq 2^{1-n}$.

It is interesting to compare this result to a uniform sampling strategy (i.e., the function is evaluated at the set of points on a uniform grid), which would provide a loss of order $n^{-\alpha/D}$. We observe that DOO is better than uniform whenever $\alpha < 2\beta$ and worse when $\alpha > 2\beta$.

This result provides some indication on how to choose the semi-metric ℓ (thus β), which is a key ingredient of the DOO algorithm (since $\delta(h) = 2^{-h\beta}$ appears in the b-values): β should be as close as possible to the true α (which can be seen as a local smoothness order of f around its maximum), but never larger than α (otherwise f does not satisfy the local smoothness property (3.8) any more).

Example 2: The previous analysis generalizes to any function that is locally equivalent to $\|x - x^*\|^\alpha$, for some $\alpha > 0$ (where $\|\cdot\|$ is any norm, e.g., Euclidean, L_∞, or L_1), around a global maximum x^* (among a set of global optima assumed to be finite). More precisely, we assume that there exists constants $c_1 > 0$, $c_2 > 0$, $\eta > 0$, such that

$$\begin{aligned} f(x^*) - f(x) &\leq c_1\|x - x^*\|^\alpha, \quad \text{for all } x \in \mathcal{X}, \\ f(x^*) - f(x) &\geq c_2 \min(\eta, \|x - x^*\|)^\alpha, \quad \text{for all } x \in \mathcal{X}. \end{aligned}$$

Let $\mathcal{X} = [0,1]^D$. Again, consider a $K = 2^D$-ary tree of partitions with (hyper)-squares. Let $\ell(x,y) = c\|x - y\|^\beta$ with $c_1 \leq c$ and $\beta \leq \alpha$ (so that f satisfies (3.8)). For simplicity we do not make explicit all the constants using the O notation for convenience (the actual constants depend on the choice of the norm $\|\cdot\|$). We have $\delta(h) = O(2^{-h\beta})$. Now, let us compute the local near-optimality dimension. For any small enough $\epsilon > 0$, $\mathcal{X}_\epsilon$ is included in a ball of radius $(\epsilon/c_2)^{1/\alpha}$ centered in

x^*, which can be packed by $O\big(\frac{\epsilon^{1/\alpha}}{\epsilon^{1/\beta}}\big)^D$ ℓ-balls of diameter ϵ. Thus the local near-optimality dimension (thus the near-optimality dimension in light of Remark 3.3) is $d = D(1/\beta - 1/\alpha)$, and the results of the previous example apply (up to constants), i.e. for $\alpha > \beta$, then $d > 0$ and $r_n = O\big(n^{-\frac{1}{D}\frac{\alpha\beta}{\alpha-\beta}}\big)$. And when $\alpha = \beta$, then $d = 0$ and one obtains the exponential rate $r_n = O(2^{-\alpha(n/C-1)})$.

We deduce that the behavior of the algorithm depends on our knowledge of the local smoothness (i.e. α and c_1) of the function around its maximum. Indeed, if this smoothness information is available, then one should define the semi-metric ℓ (which impacts the algorithm through the definition of $\delta(h)$) to match this smoothness (i.e. set $\beta = \alpha$) and derive an exponential loss rate. Now if this information is unknown, then one should underestimate the true smoothness (i.e. by choosing $\beta \leq \alpha$) and suffer a loss $r_n = O\big(n^{-\frac{1}{D}\frac{\alpha\beta}{\alpha-\beta}}\big)$, rather than overestimating it ($\beta > \alpha$) since in this case, (3.8) may not hold anymore and there is a risk that the algorithm converges to a local optimum (thus suffering a constant loss).

3.3.3 Illustration

We consider the optimization of the function $f(x) = \big[\sin(13x)\sin(27x) + 1\big]/2$ in the interval $\mathcal{X} = [0,1]$ (plotted in Figure 3.7). The global optimum is $x^* \approx 0.86442$ and $f^* \approx 0.975599$. The top part of Figure 3.7 shows two simulations of DOO, both using a numerical budget of $n = 150$ evaluations to the function, but with two different metrics ℓ.

In the first case (left figure), we used the property that f is globally Lipschitz and its maximum derivative is $\max_{x\in[0,1]} |f'(x)| \approx 13.407$. Thus with the metric $\ell_1(x,y) \overset{\text{def}}{=} 14|x-y|$, f is Lipschitz w.r.t. ℓ_1 and (3.8) holds. We remind that DOO algorithm requires the knowledge of the metric since the diameters $\delta(h)$ are defined in terms of this metric. Thus since we considered a dyadic partitioning of the space (i.e. $K = 2$), we used $\delta(h) = 14 \times 2^{-h}$ in the algorithm.

In the second case (right figure), we used the property that $f'(x^*) = 0$, thus f is locally quadratic around x^*. Since $f''(x^*) \approx 443.7$, us-

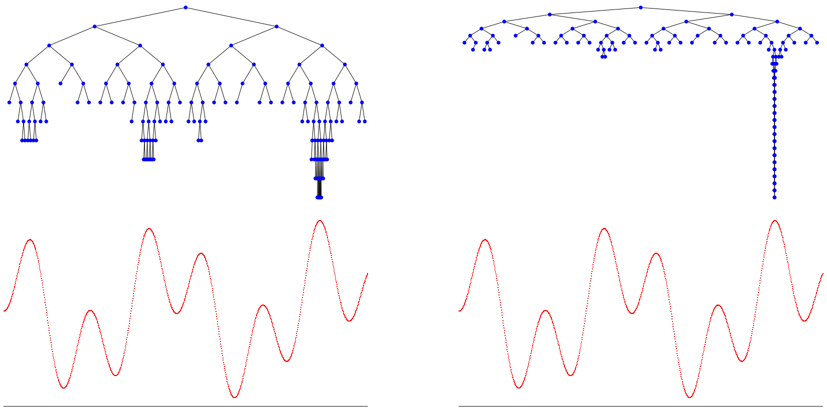

Fig. 3.7 The trees $\mathcal{T}_n$ built by DOO after $n = 150$ rounds with the choice of $\ell(x, y) = 14|x - y|$ (left) and $\ell(x, y) = 444|x - y|^2$ (right) rounds. The function (shown in the bottom part of the figure) is $x \in [0, 1] \longmapsto f(x) = 1/2\big(\sin(13x)\sin(27x) + 1\big)$. Note that the tree is extensively refined where the function is near-optimal, while it is much less developed in other regions. Using a metric that reflects the quadratic regularity of f around its maximum enables a much more precise refinement to the discretization around x^*.

ing a Taylor expansion of order 2 we deduce that f is locally smooth (i.e. satisfies (3.8)) w.r.t. $\ell_2(x, y) \overset{\text{def}}{=} 222|x - y|^2$. Thus here we defined $\delta(h) = 222 \times 2^{-2h}$.

Table 3.8 reports the numerical loss of DOO with these two metrics. As mentioned in previous subsection, the behavior of the algorithm heavily depends on the choice of metric. Although f is locally smooth (i.e. satisfies (3.8)) w.r.t. both metrics, the near-optimality of f w.r.t. ℓ_1 is $d = 1/2$ (as discussed in Example 2 above) whereas it is $d = 0$ w.r.t. ℓ_2. Thus ℓ_2 is better suited for optimizing this function since in that case, the loss decreases exponentially fast with the number of evaluations (instead of polynomially when using ℓ_1). The choice of the constants in the definition of the metric is also important. If we were to use a larger constant in the definition of the metric, the effect would be a more uniform exploration of the space at the beginning. This will impact the constant factor in the loss bound but not the rate (since the rate only depends on the near-optimality dimension d which characterizes a local behavior of f around x^* whereas the constant

n	uniform grid	DOO with ℓ_1	DOO with ℓ_2
50	1.25×10^{-2}	2.53×10^{-5}	1.20×10^{-2}
100	8.31×10^{-3}	2.53×10^{-5}	1.67×10^{-7}
150	9.72×10^{-3}	4.93×10^{-6}	4.44×10^{-16}

Fig. 3.8 Loss r_n for different values of n for a uniform grid and DOO with the two semi-metric ℓ_1 and ℓ_2.

factor also depends on the corresponding constant C depending on the global shape of f).

Now, we should be careful of not selecting a metric (such as $\ell_3(x, y) \overset{\text{def}}{=} |x - y|^3$) which is overestimating the true smoothness of f around its optimum since in this case (3.8) would not hold anymore and the algorithm might not converge to the global optimum at all (it can be stuck in a local maximum).

Thus we see that the main difficulty for applying this technique boils down to the lack of knowledge that we may have about the smoothness of the function around its maximum (or equivalently the metric under which the function is locally smooth). In Chapter 4 we will consider adaptive techniques that apply even when this smoothness is unknown. But before this, let us discuss the stochastic case in the next section.

3.4 $\mathcal{X}$-armed bandits

We now consider the case of noisy evaluations of the function, as in Subsection 3.1.2: At round t, the observed value (reward) is $r_t = f(x_t) + \epsilon_t$, where ϵ_t is an independent sample of a random variable (whose law may depend on x_t) such that $\mathbb{E}[\epsilon_t | x_t] = 0$. We also assume that the rewards r_t are bounded in $[0, 1]$. Thus the setting is a stochastic multi-armed bandit with the set of arms being $\mathcal{X}$. There are several ways to extend the deterministic case described in the previous section to this stochastic setting:

The simplest way consists in sampling several times each point in order to build an accurate estimate of the value at that point, before deciding to expand the corresponding node. This leads to a direct extension of DOO where an additional term in the definition of the

```
Parameters: error probability $\delta > 0$, time horizon $n$
Initialization: $\mathcal{T}_1 = \{(0,0)\}$ (root node)
for $t = 1$ to $n$ do
    For each leaf $(h,j) \in \mathcal{L}_t$, compute the b-values $b_{h,j}(t)$ according to (3.9).
    Select $(h_t, j_t) = \arg\max_{(h,j)\in\mathcal{L}_t} b_{h,j}(t)$
    Sample state $x_t \overset{\text{def}}{=} x_{h_t,j_t}$ and collect reward $r_t = f(x_t) + \epsilon_t$.
    If $T_{h,j}(t) \geq \frac{\log(n^2/\delta)}{2\delta(h)^2}$, expand this node: add to $\mathcal{T}_t$ the $K$ children of $(h,j)$
end for
Return the deepest node among those that have been expanded:

    $$x(n) = \arg \max_{x_{h,j}:(h,j)\in\mathcal{T}_n\setminus\mathcal{L}_n} h.$$
```

Fig. 3.9 Stochastic Optimistic Optimization (StoOO) algorithm

b-values accounts for a high-probability estimation interval. The corresponding algorithm is called Stochastic DOO (StoOO) and is close in spirit to the Zooming algorithm of [77]. The analysis is simple but the time horizon n needs to be known in advance (thus this is not an anytime algorithm). This algorithm is described in Subsection 3.4.1.

Now, another way consists in expanding the selected node each time we collect a sample. Thus the sampled points may always be different. In that case we can use the approach illustrated in Subsection 3.1.2 to generate high-probability upper bounds on the function in each cell of the tree in order and define a procedure to select in an optimistic way a leaf to expand at each round. The corresponding algorithm, Hierarchical Optimistic Optimization (HOO) is described in Subsection 3.4.2. The benefit is that HOO does not require the knowledge of the time horizon n (thus is anytime) and is more efficient in practice than StoOO (although this improvement is not reflected in the loss bounds). However it requires a slightly stronger assumption on the smoothness of the function.

3.4.1 Stochastic Optimistic Optimization (StoOO)

In the stochastic version of DOO the algorithm computes the b-values of all the leaves $(h,j) \in \mathcal{L}_t$ of the current tree as

$$b_{h,j}(t) \overset{\text{def}}{=} \hat{\mu}_{h,j}(t) + \sqrt{\frac{\log(n^2/\delta)}{2T_{h,j}(t)}} + \delta(h), \tag{3.9}$$

where $\hat{\mu}_{h,j}(t) \overset{\text{def}}{=} \frac{1}{T_{h,j}(t)} \sum_{s=1}^{t} r_s \mathbf{1}\{x_s \in X_{h,j}\}$ is the empirical average of the rewards received in $X_{h,j}$, and $T_{h,j}(t) \overset{\text{def}}{=} \sum_{s=1}^{t} \mathbf{1}\{x_s \in X_{h,j}\}$ is the number of times (h,j) has been selected up to time t. We use the convention that if a node (h,j) has not been sampled at time t then $T_{h,j}(t) = 0$ and the b-value is $+\infty$.

The algorithm is similar to DOO, see Figure 3.9, except that a node (h,j) is expanded only if $x_{h,j}$ has been sampled at least a certain number of times. Another noticeable difference is that the algorithm returns a state $x(n)$ which is the deepest among all nodes that have been expanded up to round n.

Analysis of StoOO: For any $\delta > 0$, define the following event

$$\begin{aligned} \xi \overset{\text{def}}{=} \Big\{ & \forall h \geq 0, \forall 0 \leq i < K^h, \forall 1 \leq t \leq n, \\ & \big|\hat{\mu}_{h,j}(t) - f(x_{h,j})\big| \leq \sqrt{\frac{\log(n^2/\delta)}{T_{h,j}(t)}} \Big\}. \end{aligned} \tag{3.10}$$

We now prove that this event holds in high probability:

Lemma 3.2. We have $\mathbb{P}(\xi) \geq 1 - \delta$.

Proof. Let $m \leq n$ be the (random) number of nodes expanded throughout the algorithm. For $1 \leq i \leq m$, write t_i as the time when the i-th node is expanded, and $(\tilde{h}_i, \tilde{j}_i) = (h_{t_i}, j_{t_i})$ the corresponding node. Using a "local clock", denote τ_i^s the time when the node $(\tilde{h}_i, \tilde{j}_i)$ has been selected for the s-th time and write $\tilde{r}_i^s = r_{\tau_i^s}$ the reward obtained. Note that $(h_{\tau_i^s}, j_{\tau_i^s}) = (\tilde{h}_i, \tilde{j}_i)$. Using these notations, the event ξ rewrites

$$\begin{aligned} \xi \quad = \quad \Big\{ & \forall 1 \leq i \leq m, \forall 1 \leq u \leq T_{\tilde{h}_i,\tilde{j}_i}(n), \\ & \big|\frac{1}{u}\sum_{s=1}^{u} \tilde{r}_i^s - f(x_{\tilde{h}_i,\tilde{j}_i})\big| \leq \sqrt{\frac{\log(n^2/\delta)}{u}} \Big\}. \end{aligned}$$

Since we have $\mathbb{E}[r_i^s | x_{\tilde{h}_i,\tilde{j}_i}] = f(x_{\tilde{h}_i,\tilde{j}_i})$, then $\sum_{s=1}^t \tilde{r}_i^s - f(x_{\tilde{h}_i,\tilde{j}_i})$ is a Martingale (w.r.t. the filtration generated by the samples collected at $x_{\tilde{h}_i,\tilde{j}_i}$), and Azuma's inequality [16] applies. Taking a union bound over the number of samples $u \le n$ and the number $m \le n$ of expanded nodes, we deduce the result. □

We now show that in this event of high probability StoOO only expands nodes that are near-optimal. Indeed, similarly to the analysis of DOO, define the sets

$$I_h \overset{\text{def}}{=} \{\text{nodes } (h,i) \text{ such that } f(x_{h,i}) + 3\delta(h) \ge f^*\}.$$

Lemma 3.3. In the event ξ, StoOO only expands nodes of the set $I \overset{\text{def}}{=} \cup_{h\ge 0} I_h$.

Proof. Let (h_t, j_t) be the node expanded at time t. From the definition of the algorithm, since this node is selected we have that its b-value is larger than the b-value of the cell (h_t^*, j_t^*) containing x^*. And since this node is expanded, we have $\sqrt{\frac{\log(n^2/\delta)}{2T_{h_t,j_t}(t)}} \le \delta(h_t)$. Thus,

$$\begin{aligned} f(x_{h_t,j_t}) &\ge \hat{\mu}_{h_t,j_t}(t) - \delta(h_t) && \text{under } \xi \\ &\ge b_{h_t,j_t}(t) - 3\delta(h_t) && \text{since the node is expanded} \\ &\ge b_{h_t^*,j_t^*}(t) - 3\delta(h_t) && \text{since the node is selected} \\ &\ge f(x_{h_t^*,j_t^*}) + \delta(h_t^*) - 3\delta(h_t) && \text{under } \xi \\ &\ge f^* - 3\delta(h_t) && \text{from Assumption (2)} \end{aligned}$$

which ends the proof. □

We now relate the number of nodes in I_h to the near-optimality dimension.

Lemma 3.4. Let d be the $\frac{\nu}{3}$-near-optimality dimension, and C the corresponding constant. Then

$$|I_h| \le C[3\delta(h)]^{-d}.$$

Proof. From Assumption 4, each cell (h, i) contains a ball of radius $\nu\delta(h)$ centered in $x_{h,i}$, thus if $|I_h| = |\{x_{h,i} \in \mathcal{X}_{3\delta(h)}\}|$ exceeded $C[3\delta(h)]^{-d}$, this would mean that there exists more than $C[3\delta(h)]^{-d}$ disjoint ℓ-balls of radius $\nu\delta(h)$ with center in $\mathcal{X}_{3\delta(h)}$, which contradicts the definition of d (take $\epsilon = 3\delta(h)$). □

We now provide a loss bound for StoOO.

Theorem 3.2. Let $\delta > 0$. Let us write $h(n)$ the smallest integer h such that

$$2CK3^{-d}\sum_{l=0}^{h}\delta(l)^{-(d+2)} \geq \frac{n}{\log(n^2/\delta)}.$$

Then with probability $1-\delta$, the loss of StoOO is bounded as

$$r_n \leq \delta(h(n)).$$

Proof. Let $(h_{\max}, j_{\max})$ be the deepest node that has been expanded by the algorithm up to round n. At round n there are two types of nodes: the leaves $\mathcal{L}_n$ (nodes that have not been expanded) and the nodes that have been expanded $\mathcal{T}_n \backslash \mathcal{L}_n$, which from Lemma 3.3, belong to I on the event ξ. Each leaf $j \in \mathcal{L}_n$ of depth h has been pulled at most $\frac{\log(n^2/\delta)}{2\delta(h)}$ times (since it has not been expanded) and its parent (written $(h-1, j')$ below) belongs to I_{h-1}. Thus the total number of expanded nodes n is such that

$$\begin{aligned}
n &= \sum_{l=0}^{h_{\max}}\sum_{j=0}^{K^l-1} T_{l,j}(n)\mathbf{1}\{(h,j) \in I_h\} \\
&\quad + \sum_{l=1}^{h_{\max}+1}\sum_{j=0}^{K^l-1} T_{l,j}(n)\mathbf{1}\{(h-1,j') \in I_{h-1}\} \\
&\leq \sum_{l=0}^{h_{\max}} |I_l|\frac{\log(n^2/\delta)}{2\delta(l)} + (K-1)\sum_{l=1}^{h_{\max}+1}|I_{l-1}|\frac{\log(n^2/\delta)}{2\delta(l-1)} \\
&= K\sum_{l=0}^{h_{\max}} C[3\delta(l)]^{-d}\frac{\log(n^2/\delta)}{2\delta(l)}
\end{aligned}$$

where we used Lemma 3.4 to bound the number of nodes in I_l. Now from the definition of $h(n)$ we have $h_{\max} \geq h(n)$. Now since node $(h_{\max}, j_{\max})$ has been expanded, we have that $(h_{\max}, j_{\max}) \in I$ on ξ and

$$f(x(n)) = f(x_{h_{\max}, j_{\max}}) \geq f^* - 3\delta(h_{\max}) \geq f^* - 3\delta(h(n))$$

happens with probability $1 - \delta$ from Lemma 3.2. □

Now, in the case of exponential diameters we have the following corollary.

Corollary 3.2. Assume that $\delta(h) = c\gamma^h$ for some constants $c > 0$ and $\gamma < 1$. For any $\delta > 0$ the loss of StoOO run with parameter δ is bounded with probability $1 - \delta$ as

$$r_n \leq c_1 \Big[\frac{\log(n^2/\delta)}{n}\Big]^{\frac{1}{d+2}}.$$

with $c_1 \overset{\text{def}}{=} \Big[\frac{2CK3^{-d}}{1-\gamma^{d+2}}\Big]^{\frac{1}{d+2}}$. Now, setting the parameter δ as a function of the time horizon n enables the derivation of the expected loss bound. For example with the choice $\delta = 1/n$ we have $\mathbb{E}r_n = O\Big(\big[\frac{\log n}{n}\big]^{\frac{1}{d+2}}\Big)$.

Proof. From the definition of $h(n)$ in Theorem 3.2, we have

$$\begin{aligned}
\frac{n}{\log(n^2/\delta)} &\leq 2CK3^{-d} \sum_{l=0}^{h(n)} [c\gamma^l]^{-(d+2)} \\
&\leq 2CK3^{-d} c^{-(d+2)} \frac{\gamma^{-(h(n)+1)(d+2)} - 1}{\gamma^{-(d+2)} - 1} \\
&\leq c_1^{d+2} \delta(h(n))^{-(d+2)}.
\end{aligned}$$

Now from Theorem 3.2, $r_n \leq \delta(h(n))$ with probability $1-\delta$ from which we deduce the result in high probability. The result in expectation immediately follows from

$$\mathbb{E}r_n \leq (1-\delta)\delta(h(n)) + \delta = O\Big(\big[\frac{\log n}{n}\big]^{\frac{1}{d+2}}\Big),$$

for the choice $\delta = 1/n$ as the loss is trivially bounded by 1 (since the rewards are in $[0, 1]$). □

Notice that this algorithm requires the knowledge of the time horizon n in advance. Thus this is not an *anytime* algorithm, in contrary to the DOO algorithm. This algorithm is close in spirit to the Zooming algorithm introduced in [77]. In both cases, the algorithm can be made anytime in a somehow artificial way by resorting to the so-called *doubling-trick* technique, which consists in running the algorithm for a given time horizon n_0, and once finished (if $n > n_0$), starting it again with a double time horizon $n_1 = 2n_0$ and repeating this process until the (unknown) horizon n is reached. One can show that the performance of the resulting algorithm is bounded by a similar quantity (to the performance of the algorithm that would know n) up to a constant factor. The main difference between StoOO and Zooming algorithm is that StoOO is given a hierarchical partitioning which constrains the computation of the upper-confidence bounds but as a consequence simplifies the complexity of the sampling strategy, whereas Zooming requires a sampling oracle that can identify states that do not belong to the current covering centered at the set of active states.

In the next subsection we present a modification of the StoOO algorithm, called HOO, which is anytime but which requires a slightly stronger assumption on f, called *weak Lipschitz assumption.*

3.4.2 Hierarchical Optimistic Optimization (HOO)

We make the following assumption on the function f:

Assumption 5 (weak Lipschitz). The function f satisfies that for all $x, y \in \mathcal{X}$,

$$f^* - f(y) \leq f^* - f(x) + \max\big\{f^* - f(x),\, \ell(x,y)\big\}\,. \tag{3.11}$$

Intuitively, this says that around an optimum x^* the values $f(y)$ should be above $f^* - \ell(x^*, y)$, like the local smoothness property (3.8). But in addition, in the vicinity of other arms x, the constraint is milder as the arm x gets worse: around any ϵ–optimal point x the values $f(y)$ should be larger than $f^* - 2\epsilon$ for $\ell(x,y) \leq \epsilon$ and larger than $f(x) - \ell(x,y)$ elsewhere. In other words, there is no sudden and large drop in the mean-payoff function around states with values close to the optimum (note that this property can be satisfied even for discontinuous functions).

The HOO algorithm is described in Figure 3.10. The notation $\mathcal{C}(h,i)$ refers to the set of children of (h,i).

At each round t, the algorithm assigns b-values to all nodes of the current tree T_t, defined as $b_{h,j} = +\infty$ for any leaf $(h,j) \in \mathcal{L}_t$ (from which no sample has been observed yet), and for any node $(h,i) \in \mathcal{T}_t \setminus \mathcal{L}_t$,

$$b_{h,i}(t) \overset{\text{def}}{=} \min\Big\{\hat{\mu}_{h,i}(t)+\sqrt{\frac{2\log t}{T_{h,i}(t)}}+\delta(h), \max_{(h+1,j)\in\mathcal{C}(h,i)} b_{h+1,j}(t)\Big\}. \quad (3.12)$$

Their computation can be done by backward induction, starting from the leaves, up to the root node.

The algorithm works as follows: At each round t a leaf $(h_t, j_t) \in \mathcal{L}_t$ of the current tree is selected. The way this leaf is chosen is by following an "optimistic path" from the root to a leaf where at each node along this path, the child node is the one with the highest b-value (Figure 3.11 illustrates the leaf selection procedure). Then a point x_t is selected arbitrarily in the corresponding domain X_{h_t,j_t} (for example x_{h_t,j_t} but it can be any other point, possibly chosen randomly) and the random reward $r_t = f(x_t) + \epsilon_t$ is observed. Then the b-values of all nodes are updated and the process repeats.

Finally, at round n, the algorithm returns one of the previously sampled states chosen (uniformly) randomly.

An optimistic sampling strategy: By defining the $b^{\min}$-value of any leaf $(h,j) \in \mathcal{L}_t$ as the minimum of the b-values of all its ancestor nodes, i.e.,

$$b^{\min}_{h,j}(t) \overset{\text{def}}{=} \min_{(l,i) \text{ ancestor of } (h,j)} \hat{\mu}_{l,i}(t) + \sqrt{\frac{2\log t}{T_{l,i}(t)}} + \delta(l),$$

we have that $b^{\min}_{h,j}(t)$ is a refined high-probability upper-confidence bound on $\sup_{x\in X_{h,j}} f(x)$ (since each term of the min is). This is a way to implement the idea of improving the UCB using a hierarchy of domains mentioned in Remark 3.2.

```
Parameter: $\delta > 0$
Initialization: $\mathcal{T}_1 = \{(0,0)\}$ (root node)
for $t = 1$ to $n$ do
    Compute the b-values of all nodes in $\mathcal{T}_t$ according to (3.12),
    Select a leaf $(h_t, j_t) \in \mathcal{L}_t$ by following an "optimistic path":
    Let $(h,i) \leftarrow (0,0)$ (start from the root)
    While $(h,i) \in \mathcal{T}_t \setminus \mathcal{L}_t$ do
        $(h,i) \leftarrow \arg \max_{(h+1,j) \in \mathcal{C}(h,i)} b_{h+1,j}(t)$ (Ties broken arbitrarily)
    The selected leaf is $(h_t, j_t) = (h,i)$
    Sample a state $x_t$ arbitrarily in $X_{h_t,j_t}$ (for example $x_t = x_{h_t,j_t}$) and
    collect the reward $r_t = f(x_t) + \epsilon_t$.
    Expand node $(h_t, j_t)$: $\mathcal{T}_{t+1} \leftarrow \mathcal{T}_t \cup \mathcal{C}(h_t, j_t)$ (add the $K$ children of $(h_t, j_t)$)
end for
Return $x(n) \overset{\text{def}}{=} x_T$, where $T \sim \mathcal{U}(\{1, 2, \ldots, n\})$.
```

Fig. 3.10 Hierarchical Optimistic Optimization (HOO) applied to the problem of minimizing the loss r_n.

Actually from the definition of the optimistic path chosen by the HOO algorithm, we have the property that the selected leaf (h_t, j_t) is a leaf with highest $b^{\min}$ value among all leaves in $\mathcal{L}_t$:

$$(h_t, j_t) \in \arg \max_{(h,j) \in \mathcal{L}_t} b^{\min}_{h,j}(t).$$

This is exactly the optimistic methodology introduced in Section 3.1.2, especially described in remark 3.2.

Analysis of HOO The bound reported in [29] is in terms of the cumulative regret $R_n \overset{\text{def}}{=} nf^* - \sum_{t=1}^{n} r_t$, i.e. the difference between the sum of rewards collected by the algorithm up to time n compared to n times the best possible expected reward f^*.

However, from an algorithm achieving a cumulative regret R_n one can design an algorithm that achieves a loss r_n in expectation of $\mathbb{E}r_n = \mathbb{E}R_n/n$. This loss bound is not optimal for finitely many armed bandits (since there exists strategies that achieve exponential loss bounds as discussed in [27, 9]), but in the case of $\mathcal{X}$-armed bandits (where the set of arms is larger than the number of rounds n), this may be unimprovable. The version presented in Figure 3.10 is an adaptation

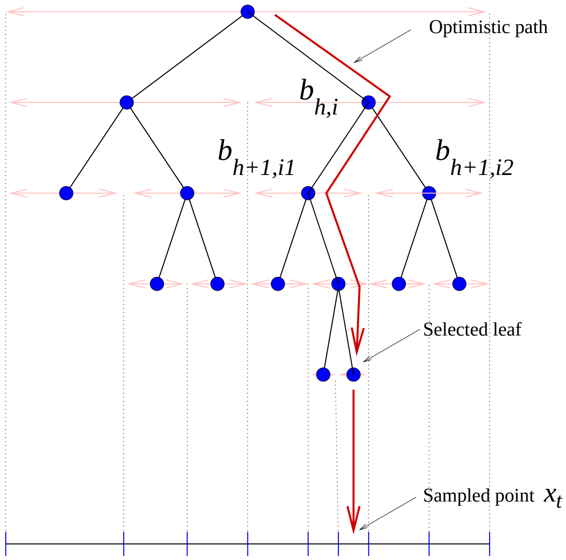

Fig. 3.11 Illustration of the leaf selection procedure in round t. The tree represents $\mathcal{T}_t$. In the illustration, $B_{h+1,i_1}(t) > B_{h+1,i_2}(t)$, therefore, the selected path traverses the node $(h+1, i_1)$. The point x_t is chosen in the selected leaf (h_t, j_t).

of the HOO algorithm where the state $x(n)$ returned at the end of the algorithm is chosen uniformly randomly among the states $\{x_t\}_{1\leq t\leq n}$ sampled by the algorithm up to round n:

$$x(n) \overset{\text{def}}{=} x_T, \text{ where } T \sim \mathcal{U}(\{1, 2, \ldots, n\}). \tag{3.13}$$

Thus we immediately deduce that

$$\mathbb{E}r_n = \mathbb{E}_T[f^* - f(x_T)] = \frac{1}{n}\sum_{t=1}^{n}[f^* - f(x_t)] = \frac{1}{n}\mathbb{E}R_n. \tag{3.14}$$

Theorem 3.3 (Regret bound for HOO [29]). Under Assumption 5 on f. Let d be the $\frac{\nu}{3}$-near-optimality dimension of f w.r.t. ℓ. Then the loss of HOO is upper-bounded as

$$\mathbb{E}r_n = O\Big(\big[\frac{n}{\log n}\big]^{-\frac{1}{d+2}}\Big).$$

Remark 3.5. HOO requires that f satisfies (3.11) which is slightly stronger than (3.8). The reason is that since HOO expands a leaf at each round, it builds a high-probability UCB on $\sup_{x \in X_{h,i}} f(x)$ at a given node (h,i) based on different points in the cell $X_{h,i}$ (in contrary to StoOO that samples several times the same point in order to build an accurate estimate of the value before expanding the node). As a consequence, the rewards collected in sub-optimal cells may significantly impact the cumulative regret. Indeed, consider a sub-optimal cell $X_{h,i}$ (thus $x^* \notin X_{h,i}$) such that $f(x_{h,i}) \geq f^* - \delta(h)$. Assuming that f satisfies (3.8) only, then sampling arbitrarily at $x \in X_{h,i}$ may cause a large cumulative regret (since the function may be arbitrarily low at points $x \neq x_{h,i}$). In contrast, assuming that f satisfies (3.11), one deduce that any sample x in the cell $X_{h,i}$ contributes to the cumulative regret by $f^* - f(x) \leq f^* - f(x_{h,i}) + \max\{f^* - f(x_{h,i}), \ell(x_{h,i}, x)\} \leq 2\delta(h)$ only.

Since the state $x(n)$ returned by the algorithm follows (3.13), the loss r_n of HOO is directly related to the cumulative regret R_n via (3.14). However for the problem of minimizing the loss r_n (that we consider in this paper), it may be possible to define other choices for the recommended state $x(n)$ such that the loss r_n may not be related to the cumulative regret R_n. Such a possible choice would be to return any point in the deepest leaf $\arg\max_{(h,j) \in \mathcal{L}_n} h$ of the final tree $\mathcal{T}_n$ built from HOO. Actually, numerical experiments indicate that this strategy provides better performance than the one defined by (3.13). However, there is currently no theoretical guarantee for it.

The loss bounds of HOO and StoOO are of the same order. The benefit of HOO over StoOO is that it is anytime (i.e. n does not need to be known in advance) and it is usually numerically more efficient since it does not wait until a cell has been sampled enough times to start refining the corresponding node. Thus inside a given cell $X_{h,i}$ the sampling is adaptive even when the number of samples is small, which enables HOO to localize more rapidly the maximum of f within the cell (contrary to StoOO which samples the same state $O(\log(n)/\delta(h)^2)$ times before refining it). Those improvements come at the cost of constraining slightly more the assumption about the function f as explained in the previous remark.

Finally, we provide some numerical experiments on the same one-dimensional problem as described in Subsection 3.3.3. The mean-reward function is $f(x) \overset{\text{def}}{=} \big(\sin(13x)\sin(27x)+1\big)/2$ and the reward collected at a state x_t follows a Bernoulli distribution with parameter $f(x_t)$ (i.e. $r_t = 1$ with probability $f(x_t)$ and $r_t = 0$ with probability $1-f(x_t)$). Figure 3.12 shows the trees built by HOO after $n = 10^2, 10^3, 10^4$, and $n = 10^5$ calls to the function using the ℓ_2-metric. Here the hierarchical partitioning is formed by dyadic intervals, $\delta(h) = 2^{-h}$, and the points x_t are uniformly randomly chosen in the selected cells X_{h_t,j_t}.

A first observation is that tree is more uniformly balanced here than in the deterministic case. The reason is that the loss obtained in this stochastic case (both for StoOO and HOO) is of order $n^{-\frac{1}{d+2}}$, where d is the near-optimality dimension, whereas in the deterministic setting, DOO achieves the improved rate $n^{-1/d}$ when $d > 0$, and even an exponential rate when $d = 0$ (see Corollary 3.1).

A second remark is that, similarly to the deterministic case, the tree is more deeply refined where the mean-payoff function is near-optimal, and the heterogeneous aspect of the tree increases with n: The algorithm starts with a quasi-uniform initial exploration, then rapidly focus on the main peaks, and eventually performs a local search around the global optimum. We can intuitively grasp the advantages of such hierarchical optimistic optimization methods in that they do the best possible exploration given the numerical budget n (and the knowledge of the smoothness of f).

Comparison with UCB-AIR algorithm One can think of applying the UCB-AIR algorithm [108] introduced in Subsection 1.2.1 in this $\mathcal{X}$-armed bandit setting, where new arms would be chosen uniformly at random over the space $\mathcal{X}$.

For illustration, let us compare UCB-AIR with StoOO/HOO on Example 2 described in Section 3.3.2 where $\mathcal{X} = [0,1]^D$ and the mean-reward function f is locally equivalent to $\|x - x^*\|^\alpha$, for some $\alpha > 0$, around a global maximum x^*.

UCB-AIR would pull randomly a new arm X according to the

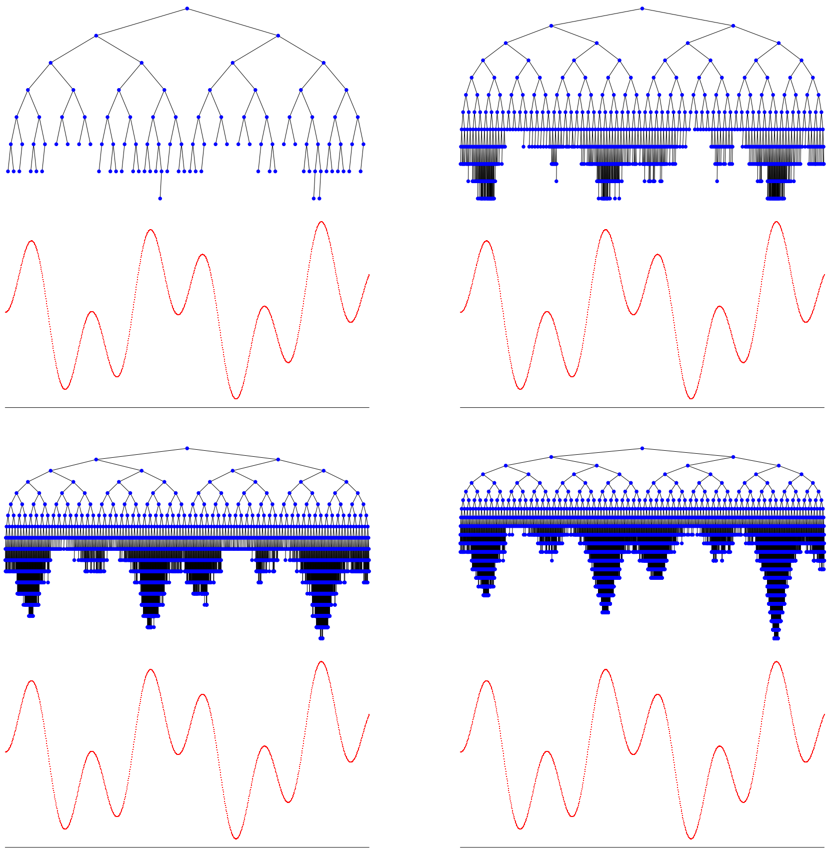

Fig. 3.12 The trees $\mathcal{T}_n$ built by HOO after $n = 100$, 10^3, n^5 and $n = 10^5$ noisy function evaluations. The mean-payoff function (shown in the top part of the figures) is $x \in [0,1] \longmapsto f(x) = \big(\sin(13x)\sin(27x)+1\big)/2$ and the corresponding rewards are Bernoulli-distributed.

Lebesgue measure on $[0,1]^D$, we have: $\mathbb{P}(\mu(X) > \mu^* - \varepsilon) = \Theta(\mathbb{P}(||X - x^*||^\alpha < \varepsilon)) = \Theta(\varepsilon^{D/\alpha})$, for $\varepsilon \to 0$.

Thus Assumption (1.16) holds with $\beta = D/\alpha$, and UCB-AIR provides an expected cumulative regret bounded as (in the case $f^* < 1$) $\mathbb{E}R_n = \tilde{O}(\sqrt{n})$ when $D < \alpha$, and $\mathbb{E}R_n = \tilde{O}(n^{D/(\alpha+D)})$ when $D \geq \alpha$. Using the recommendation strategy of $x(n)$ defined as in (3.13), the expected loss of UCB-AIR is thus:

$$\mathbb{E}r_n = \begin{cases} \tilde{O}\big(n^{-1/2}\big) & \text{for } D < \alpha \\ \tilde{O}\big(n^{-\frac{\alpha}{\alpha+D}}\big) & \text{for } D \geq \alpha \end{cases}$$

Thus the loss is small when the smoothness order α is large, since there is a reasonable chance to find a near-optimal point among a small number of samples chosen uniformly a random. Notice that in order to apply UCB-AIR, the coefficient α should be known.

Now using StoOO or HOO with the semi-metric $\ell(x,y) = \|x-y\|^\beta$ with $\beta \leq \alpha$ implies that the near-optimality dimension is $d = D(1/\beta - 1/\alpha)$ (see Subsection 1.2.1), thus the expected loss of StoOO or HOO is

$$\mathbb{E}r_n = \begin{cases} \tilde{O}\big(n^{-1/2}\big) & \text{for } \alpha = \beta \\ \tilde{O}\big(n^{-\frac{1}{D(1/\beta-1/\alpha)+2}}\big) & \text{for } \alpha > \beta \end{cases}, \tag{3.15}$$

So the important measure of the quality of this strategy is the discrepancy between the actual smoothness order α of f and the "believed" smoothness order β which is used in the algorithm. The closer β is from α, the better (since the near-optimality dimension depends on this discrepancy).

Thus if the local smoothness order α is known, then it is always better to apply StoSOO or HOO with $\beta = \alpha$ than UCB-AIR since the loss is then $\tilde{O}(n^{-1/2})$. If α is not known, then UCB-AIR cannot be applied immediately, and we would have to guess a value of $\beta \leq \alpha$ as close to α as possible. However, β should not be chosen strictly larger than α, otherwise the smoothness property (3.8) or (3.11) does not hold, and the algorithms StoOO and HOO may not converge to the global optimum (i.e. the loss may not converge to 0).

Comparison with UCT Actually, one can see the UCT algorithm [78] exposed in Section 2.2 as a version of HOO where $\delta(h)$ is set to 0 in the definition of the upper-confidence-bounds (3.12) (since when $\delta(h) = 0$ the minimum of the two terms defining the bound is always the first one), which reduces to the UCT bound (2.1). Thus UCT can be seen as a version of HOO where the smoothness of the function is assumed to be infinite (i.e. β is set to ∞). Thus in light of the previous comment, this algorithm does not enjoy any finite-time bound.

Monte-Carlo Tree Search HOO can be seen as a Monte-Carlo Tree Search (MCTS) algorithm as illustrated in Figure 2.1. If we consider choosing the point x_t uniformly at random over the selected cell X_{h_t,j_t} then this is equivalent to performing an (infinite) rollout where uniformly random moves are chosen from node X_{h_t,j_t}. Thus the results presented in this chapter can be seen as preliminary foundations for MCTS in the sense that finite-time performance guarantees are obtained for the problem of function optimization in general spaces (i.e. semi-metric) under the assumption that the mean-reward function satisfies a local smoothness property w.r.t. a known semi-metric.

3.5 Conclusions

The performance of the algorithms DOO, StoOO, HOO described in this chapter depends on the near-optimality dimension d, which characterizes the quantity of near-optimal states of f measured with the semi-metric ℓ. Actually d can be seen as a discrepancy between the actual smoothness order of the function around its maximum and the believed smoothness order that is used in the algorithm (through the choice of ℓ), as illustrated in the previous example where $d = D(1/\beta - 1/\alpha)$. Thus when the local smoothness of f around x^* is known, it can be used for defining ℓ such that the near-optimality dimension is $d = 0$, which leads to a loss bound $r_n = O(n^{-1/2})$ in the stochastic case. Thus we obtain the nice property that **the rate $n^{-1/2}$ is independent of the space dimension**, thus those techniques do not suffer from the so-called "the curse of dimensionality“.

However it is important to notice that the constant factor hidden in the O notation may be exponential in the dimension of the space. This is of course unavoidable when we consider a global optimization problem under such a weak and local assumption on the considered functions. Thus the performance is somehow similar to a Monte-Carlo integration method where the standard deviation of the Monte-Carlo estimate using n random samples is $\sigma(f)n^{-1/2}$. The rate $n^{-1/2}$ is independent of the space dimension, but the constant factor (the standard deviation of f) is usually exponential in the dimension. Thus, in terms of convergence rate, optimizing a function with known smoothness is no more difficult than integrating it!

Now, when the local smoothness of f is not known, or when there is no semi-metric such that $d = 0$ then the loss bound deteriorates and the dimension of the space appears in the rate.

Thus, like in Chapter 1, we see that the performance of the optimistic strategy depends on **the smoothness of f around the global optimum** (expressed in terms of a measure of the quantity of near-optimal states) and on **our knowledge about this smoothness.**

The next chapter presents adaptive techniques that may apply when the smoothness of the function is unknown.

4

Optimistic Optimization with unknown smoothness

We now consider the setting where Assumptions 1, 2, 3, 4 hold for some semi-metric ℓ, but now, **the semi-metric ℓ is unknown from the algorithm**.

The hierarchical partitioning of the space is still given to the algorithm, but since ℓ is unknown, one cannot use the diameter $\delta(h)$ of the cells to design upper-bounds, like in DOO, StoOO, or HOO.

Alternatively, we can think of this setting as a lack of knowledge about the local smoothness of f around its maximum. For example, in the Examples 1 and 2 described in Section 3.3.2 the choice of β (defining the semi-metric ℓ) is difficult if the smoothness order α of f is unknown, but this choice is critical since β should always be less than α (in order to guarantee the convergence of the algorithm) while as close to α as possible in order to optimize the performance.

The question we wish to address here is: If ℓ is unknown, is it possible to implement an optimistic optimization strategy with performance guarantees?

We provide a positive answer to this question and in addition we show that we can do **almost as well as if ℓ were known, for the best possible valid** ℓ (i.e., satisfying Assumptions 1, 2, 3, 4).

Section 4.1 considers the deterministic case while Section 4.2 deals with the stochastic case.

4.1 Simultaneous Optimistic Optimization (SOO) algorithm

In this section we consider the deterministic setting and use the same notations as in Section 3.3.

The idea is to expand at each round simultaneously all the leaves (h, j) of the current tree for which there exists a semi-metric ℓ such that the corresponding upper-bound $f(x_{h,j}) + \sup_{x \in X_{h,j}} \ell(x_{h,j}, x)$ of the leaf (h, j) could be the highest. In other words, we select all cells that are potentially optimal for any valid metric. This is implemented by expanding at each round at most a leaf per depth, and a leaf is expanded only if it has the highest value among all leaves of same or lower depths. The Simultaneous Optimistic Optimization (SOO) algorithm is described in Figure 4.1.

The SOO algorithm takes as a parameter a function $t \to h_{\max}(t)$ which limits the tree to a maximal depth of $h_{\max}(t)$ after t node expansions. Again, $\mathcal{L}_t$ refers to the set of leaves of $\mathcal{T}_t$.

4.1.1 Analysis of SOO

All previously relevant quantities such as the diameters $\delta(h)$, the sets I_h, and the nu-near-optimality dimension d depend on the unknown semi-metric ℓ (which is such that Assumptions 1, 2, 3, 4 are satisfied).

At time t, let us write h_t^* the depth of the deepest expanded node in the branch containing x^* (an optimal branch). Let $(h_t^* + 1, i^*)$ be an optimal node of depth $h_t^* + 1$ (i.e., such that $x^* \in X_{h_t^*+1,i^*}$). Since this node has not been expanded yet, any node (h_t^*+1, i) of depth h_t^*+1 that is later expanded, before $(h_t^* + 1, i^*)$ is expanded, is $\delta(h_t^* + 1)$-optimal. Indeed, $f(x_{h_t^*+1,i}) \geq f(x_{h_t^*+1,i^*}) \geq f^* - \delta(h_t^* + 1)$. We deduce that once an optimal node of depth h is expanded, it takes at most $|I_{h+1}|$ node expansions at depth $h+1$ before the optimal node of depth $h+1$ is expanded. From that simple observation, we deduce the following lemma.

```
The maximum depth function $t \mapsto h_{\max}(t)$ is a parameter of the algorithm.
**Initialization:** $\mathcal{T}_1 = \{(0,0)\}$ (root node). Set $t = 1$.
while True do
  Set $v_{\max} = -\infty$.
  for $h = 0$ to $\min(\mathrm{depth}(\mathcal{T}_t), h_{\max}(t))$ do
    Among all leaves $(h,j) \in \mathcal{L}_t$ of depth $h$, select
        $(h,i) \in \arg\max_{(h,j)\in\mathcal{L}_t} f(x_{h,j})$
    if $f(x_{h,i}) \geq v_{\max}$ then
      Expand this node: add to $\mathcal{T}_t$ the $K$ children $\{(h+1,i_1),\ldots,(h+1,i_K)\}$ and evaluate the function at the corresponding centers $\{x_{h+1,i_1},\ldots,x_{h+1,i_K}\}$
      Set $v_{\max} = f(x_{h,i})$, Set $t = t+1$
      if $t = n$ then Return
          $x(n) \overset{\mathrm{def}}{=} \arg\max_{(h,i)\in\mathcal{T}_n} f(x_{h,i})$
    end if
  end for
end while.
```

Fig. 4.1 Simultaneous Optimistic Optimization (SOO) algorithm.

Lemma 4.1. For any depth $0 \leq h \leq h_{\max}(t)$, whenever $t \geq (|I_0| + |I_1| + \cdots + |I_h|)h_{\max}(t)$, we have $h_t^* \geq h$.

Proof. We prove it by induction. For $h = 0$, we have $h_t^* \geq 0$ trivially. Assume that the proposition is true for all $0 \leq h \leq h_0$ with $h_0 < h_{\max}(t)$. Let us prove that it is also true for h_0+1. Let $t \geq (|I_0|+|I_1|+\cdots+|I_{h_0+1}|)h_{\max}(t)$. Since $t \geq (|I_0|+|I_1|+\cdots+|I_{h_0}|)h_{\max}(t)$ we know that $h_t^* \geq h_0$. So, either $h_t^* \geq h_0+1$ in which case the proof is finished, or $h_t^* = h_0$. In this latter case, consider the nodes of depth h_0+1 that are expanded. We have seen that as long as the optimal node of depth h_0+1 is not expanded, any node of depth h_0+1 that is expanded must be $\delta(h_0+1)$-optimal, i.e., belongs to I_{h_0+1}. Since there are $|I_{h_0+1}|$ of them, after $|I_{h_0+1}|h_{\max}(t)$ node expansions, the optimal one must be expanded, thus $h_t^* \geq h_0+1$. □

Theorem 4.1. Let us write $h(n)$ the smallest integer h such that

$$Ch_{\max}(n)\sum_{l=0}^{h}\delta(l)^{-d} \geq n. \tag{4.1}$$

Then the loss is bounded as

$$r_n \leq \delta\big(\min(h(n)-1, h_{\max}(n))\big). \tag{4.2}$$

Proof. From Lemma 3.1 and the definition of $h(n)$ we have

$$h_{\max}(n)\sum_{l=0}^{h(n)-1}|I_l| \leq Ch_{\max}(n)\sum_{l=0}^{h(n)-1}\delta(l)^{-d} < n,$$

thus from Lemma 4.1, when $h(n)-1 \leq h_{\max}(n)$ we have $h_n^* \geq h(n)-1$. Now in the case $h(n)-1 > h_{\max}(n)$, since the SOO algorithm does not expand nodes beyond depth $h_{\max}(n)$, we have $h_n^* = h_{\max}(n)$. Thus in all cases, $h_n^* \geq \min(h(n)-1, h_{\max}(n))$.

Define (h_n^*, i^*) to be the deepest expanded node containing x^*. Using the local smoothness assumption (3.8), we deduce that:

$$f(x(n)) \geq f(x_{h_n^*,i^*}) \geq f^* - \delta(h_n^*) \geq f^* - \delta(\min(h(n)-1, h_{\max}(n))).$$

□

This result may seem very surprising: although the semi-metric ℓ is not known, the performance is almost as good as for DOO (see Theorem 3.1) which uses the knowledge of ℓ. The main difference is that the maximal depth $h_{\max}(n)$ appears both as a multiplicative factor in the definition of $h(n)$ in (4.1) and as a threshold in the loss bound (4.2). Those two appearances of $h_{\max}(n)$ define a trade-off between deep (large $h_{\max}$) versus broad (small $h_{\max}$) types of exploration. We now illustrate the case of exponentially decreasing diameters.

Corollary 4.1. Assume that $\delta(h) = c\gamma^h$ for some $c > 0$ and $\gamma < 1$. Consider the two cases:

- The near-optimality $d > 0$. Let the depth function $h_{\max}(t) = t^{\epsilon}$, for some $\epsilon > 0$ arbitrarily small. Then, for n large enough (i.e. $n^{\epsilon} \geq \xi \log n$ for some constant ξ that depends on c, C, γ, d) the loss of SOO is bounded as:

$$r_n \leq \Big(\frac{C}{1-\gamma^d}\Big)^{1/d} \gamma^{-1} n^{-\frac{1-\epsilon}{d}}.$$

- The near-optimality $d = 0$. Let the depth function $h_{\max}(t) = \sqrt{t}$. Then the loss of SOO is bounded as:

$$r_n \leq c\gamma^{\sqrt{n}\min(1/C,1)-2}.$$

Proof. From Theorem 3.1, when $d > 0$ we have

$$n \leq C h_{\max}(n) \sum_{l=0}^{h(n)} \delta(l)^{-d} = C c^{-d} h_{\max}(n) \frac{\gamma^{-d(h(n)+1)} - 1}{\gamma^{-d} - 1}$$

thus for the choice $h_{\max}(n) = n^{\epsilon}$, we deduce $\gamma^{-dh(n)} \geq \frac{n^{1-\epsilon}}{Cc^{-d}}\big(1-\gamma^d\big)$. Thus $h(n)$ is logarithmic in n and for n large enough (i.e. when $d \log(1/\gamma) n^{\epsilon} \geq (1-\epsilon)\log n + \log \frac{1-\gamma^d}{Cc^{-d}}$) then $h(n) \leq h_{\max}(n) + 1$, thus

$$\begin{aligned} r_n &\leq \delta\big(\min(h(n)-1, h_{\max}(n))\big) = \delta(h(n)-1) = c\gamma^{h(n)-1} \\ &\leq \Big(\frac{C}{1-\gamma^d}\Big)^{1/d} \gamma^{-1} n^{-\frac{1-\epsilon}{d}}. \end{aligned}$$

Now, if $d = 0$ then $n \leq C h_{\max}(n) \sum_{l=0}^{h(n)} \delta(l)^{-d} = C h_{\max}(n)(h(n)+1)$, thus for the choice $h_{\max}(n) = \sqrt{n}$ we deduce that the loss decreases as:

$$r_n \leq \delta\big(\min(h(n)-1, h_{\max}(n))\big) \leq c\gamma^{\sqrt{n}\min(1/C,1)-2}.$$

□

Remark 4.1. The maximal depth function $h_{\max}(t)$ is still a parameter of the algorithm, which somehow influences the behavior of the algorithm (deep versus broad exploration of the tree). However, for a large class of problems (e.g. when $d > 0$) the choice of the order ϵ does not impact the asymptotic performance of the algorithm.

Since our algorithm does not depend on ℓ, the analysis is actually true **for *any* semi-metric ℓ that satisfies Assumptions 1, 2, 3, 4** thus Theorem 4.1 and Corollary 4.1 hold for the best possible choice of such a ℓ (which may depend on f itself!). In particular, we can think of problems for which there exists a semi-metric ℓ such that the corresponding near-optimality dimension d is 0. See the discussion in Section 4.2.2 below. Now, instead of describing a general class of problems satisfying this property, we illustrate in the next subsection non-trivial optimization problems in $\mathcal{X} = \mathbb{R}^D$ where there exists ℓ such that $d = 0$.

4.1.2 Examples

Example 1: Consider the Example 1 described in Section 3.3.2 where $\mathcal{X} = [-1, 1]^D$ and $f(x) = 1 - \|x\|_\infty^\alpha$, where $\alpha \geq 1$ is unknown. We have seen that DOO with the metric $\ell(x, y) = \|x - y\|_\infty^\beta$ provides a polynomial loss $r_n = O\big(n^{-\frac{1}{D}\frac{\alpha\beta}{\alpha-\beta}}\big)$ whenever $\beta < \alpha$, and an exponential loss $r_n \leq 2^{1-n}$ when $\beta = \alpha$. However, here α is unknown.

Now consider the SOO algorithm with the maximum depth function $h_{\max}(t) = \sqrt{t}$. As mentioned before, SOO does not require ℓ, thus we can apply the analysis for any ℓ that satisfies Assumptions 1, 2, 3, 4. So let us consider $\ell(x, y) = \|x - y\|_\infty^\alpha$. Then $\delta(h) = 2^{-h\alpha}$, $\nu = 1$, and the near-optimality dimension of f under ℓ is $d = 0$ (and $C = 1$). We deduce that the loss of SOO is $r_n \leq 2^{(2-\sqrt{n})\alpha}$. Thus SOO provides a stretched-exponential loss without requiring the knowledge of α.

Note that a uniform grid provides the loss $n^{-\alpha/D}$, which is polynomially decreasing only (and subject to the curse of dimensionality since the rate depends on D). Thus, in this example SOO is always better than both Uniform and DOO except if one knows perfectly α and applies DOO with $\beta = \alpha$ (in which case we obtain an exponential loss). The fact that SOO is not as good as DOO optimally fitted comes from the truncation of SOO at a maximal depth $h_{\max}(n) = \sqrt{n}$ (whereas DOO optimally fitted would explore the tree up to a depth linear in n).

n	loss of SOO
50	$r_n = 3.56 \times 10^{-4}$
100	$r_n = 5.90 \times 10^{-7}$
150	$r_n = 1.92 \times 10^{-10}$

Fig. 4.2 Numerical performance of SOO for the function $f(x) = 1/2\big(\sin(13x)\sin(27x) + 1\big)$

Example 2: The same conclusion holds for Example 2, where we consider a function f defined on $[0,1]^D$ that is locally equivalent to $\|x - x^*\|^\alpha$, for some unknown $\alpha > 0$ (see the precise assumptions in Section 3.3.2). We have seen that DOO using $\ell(x,y) = c\|x - y\|^\beta$ with $\beta < \alpha$ has a loss $r_n = O\big(n^{-\frac{1}{D}\frac{\alpha\beta}{\alpha-\beta}}\big)$, and when $\alpha = \beta$, then $d = 0$ and the loss is $r_n = O(2^{-\alpha(n/C-1)})$.

Now by using SOO (which does not require the knowledge of α) with $h_{\max}(t) = \sqrt{t}$ we deduce the stretched-exponential loss $r_n = O(2^{-\sqrt{n}\alpha/C})$ (by using $\ell(x,y) = \|x - y\|^\alpha$ in the analysis, which gives $\delta(h) = 2^{-h\alpha}$ and $d = 0$).

4.1.3 Illustrations

Figure 4.3 shows the first iterations of the SOO algorithm for the function $f(x) = 1/2\big(\sin(13x)\sin(27x) + 1\big)$ already considered in Section 3.3.3. At each round several cells (indicated by the circled dots and the bold segments) are simultaneously spit. Here we used a branching factor $K = 3$ and the maximal depth function $h_{\max}(t) = \sqrt{t}$.

Table 4.2 reports the loss of SOO for different numerical budgets. In comparison to Table 3.8 the loss of SOO is better than DOO using the sub-optimal semi-metric ℓ_1 and is almost as good DOO with the optimal semi-metric ℓ_2. This corroborates the theoretical guarantees stated in Subsection 4.1.1. Indeed, in this example the near-optimality dimension of f w.r.t. the semi-metric ℓ_2 is $d = 0$, as illustrated in Example 2 in Subsection 4.1.2, thus the loss of SOO is a stretched-exponential.

Figure 4.4 also shows the first iterations of the SOO algorithm for the function $f(x) = x(1-x)\big(4 - \sqrt{|\sin(60x)|}\big)$. We also used $K = 3$

and $h_{\max}(t) = \sqrt{t}$. This function f has a local behavior (around its maximum) $f(x) \equiv f(x^*) - c|x - x^*|^\alpha$, for some constant $c > 0$ and $\alpha = 1/2$. One can easily check that the near-optimality dimension of f w.r.t. the metric $\ell(x, y) \overset{\text{def}}{=} c|x - y|^{1/2}$ is $d = 0$, thus the loss of SOO is also stretched-exponentially decreasing to 0. Notice that SOO neither requires the knowledge of c nor α (in contrast to DOO).

Figure 4.5 illustrates the SOO algorithm for the optimization of a Brownian motion (i.e. f is a function sample of a Gaussian process). We can prove that with high-probability (w.r.t. the random choice of f), f is lower-bounded as $f(x) \geq f(x^*) - c|x - x^*|^\alpha$, for some constant $c > 0$ (which depends on the failure probability) and $\alpha = 1/2$. An open question is whether the near-optimality dimension of f w.r.t. the metric $\ell(x, y) \overset{\text{def}}{=} c|x - y|^{1/2}$ is (in high probability) $d = 0$, in which case SOO would have a stretched-exponential loss, or $d > 0$ for which SOO would have a polynomial loss.

Finally, Figure 4.6 shows a 2-dimensional problem with the function $f(x_1, x_2) = f(x_1)f(x_2)$ where $f(x) = \big(\sin(13x)\sin(27x) + 1\big)/2$. Again we used $h_{\max}(t) = \sqrt{t}$ and $K = 3$ (where a cell is spit in 3 along the longest direction). In this situation again, the near-optimality dimension of f w.r.t. the semi-metric $l(x, y) = c|x_1 - y_1|^2|x_2 - y_2|^2$ (for some constant $c > 0$) is $d = 0$.

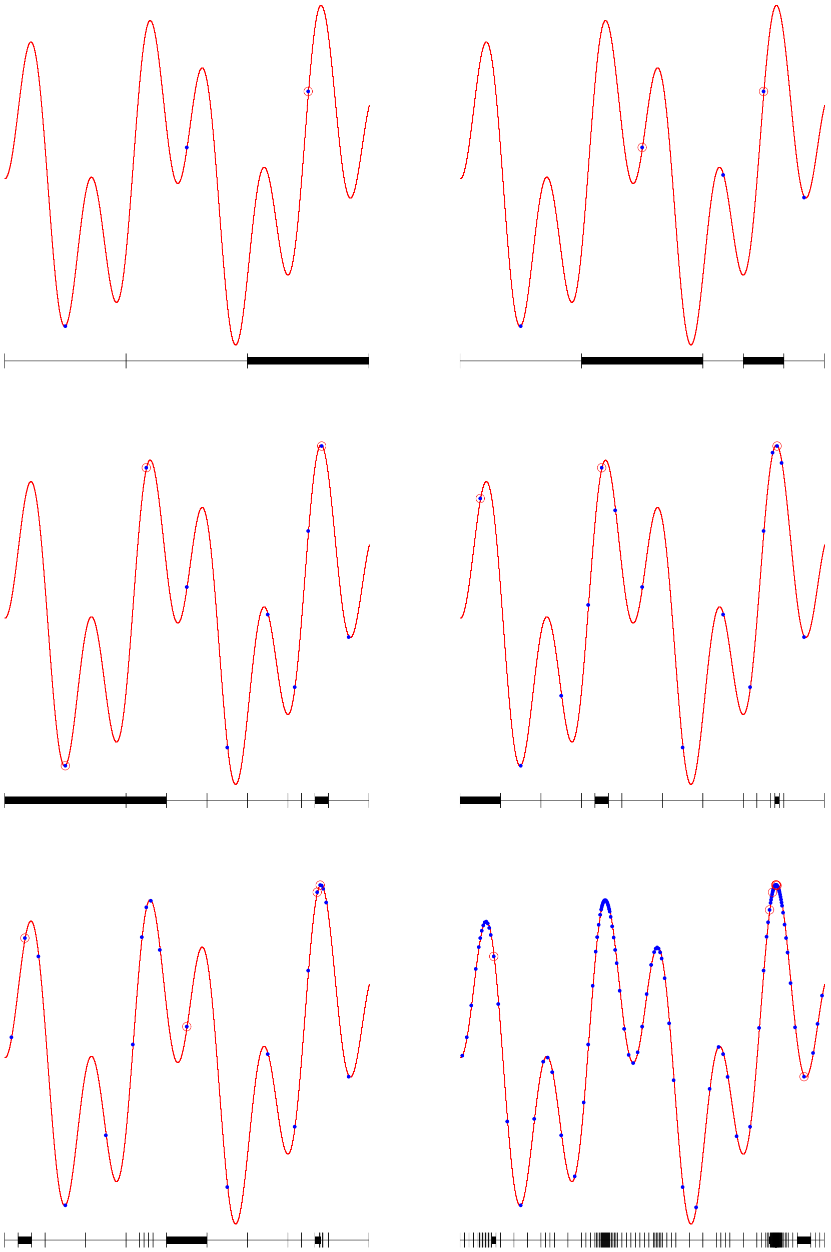

Fig. 4.3 The 5 first iterations of the SOO algorithm and the resulting tree $\mathcal{T}_n$ after $n = 150$ function evaluations. Here $f(x) = \big(\sin(13x)\sin(27x) + 1\big)/2$ and $K = 3$. The blue dots represent the values of the function at the center of the cells. The circle around the dots and the bold segments shows the nodes that are expanded at each iteration.

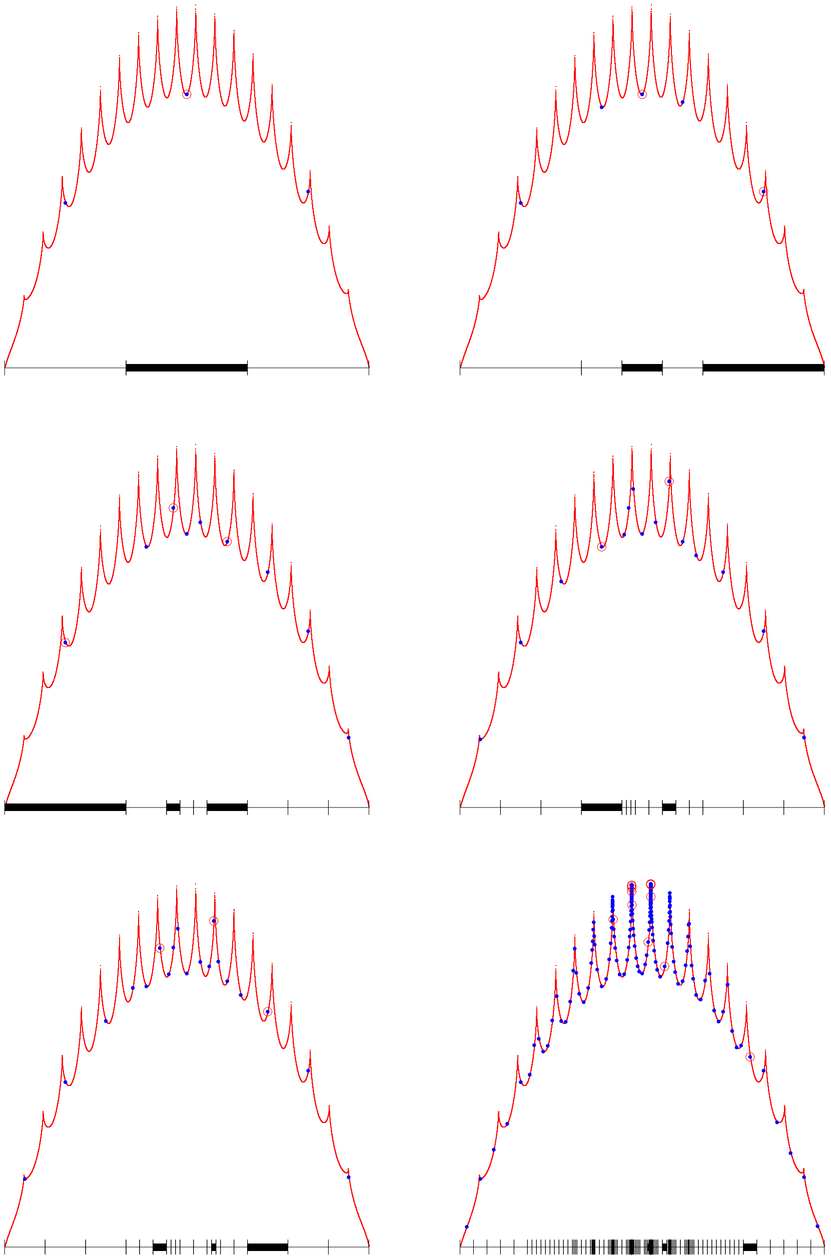

Fig. 4.4 The 5 first iterations of the SOO algorithm and the resulting tree $\mathcal{T}_n$ after $n = 150$ function evaluations. Here $f(x) = x(1-x)\big(4 - \sqrt{|\sin(60x)|}\big)$ and $K = 3$.

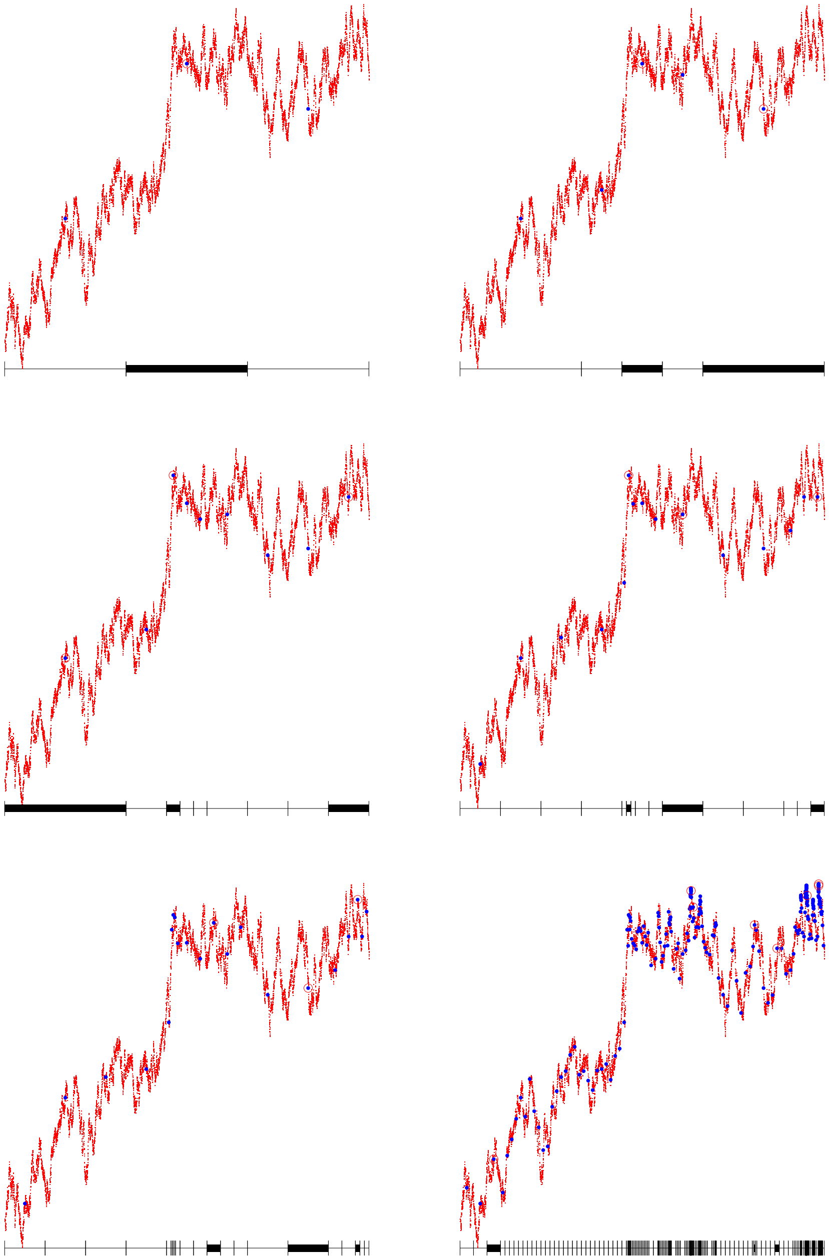

Fig. 4.5 The 5 first iterations of the SOO algorithm and the resulting tree $\mathcal{T}_n$ after $n = 150$ function evaluations. Here $f(x)$ is a Brownian motion sample and $K = 3$.

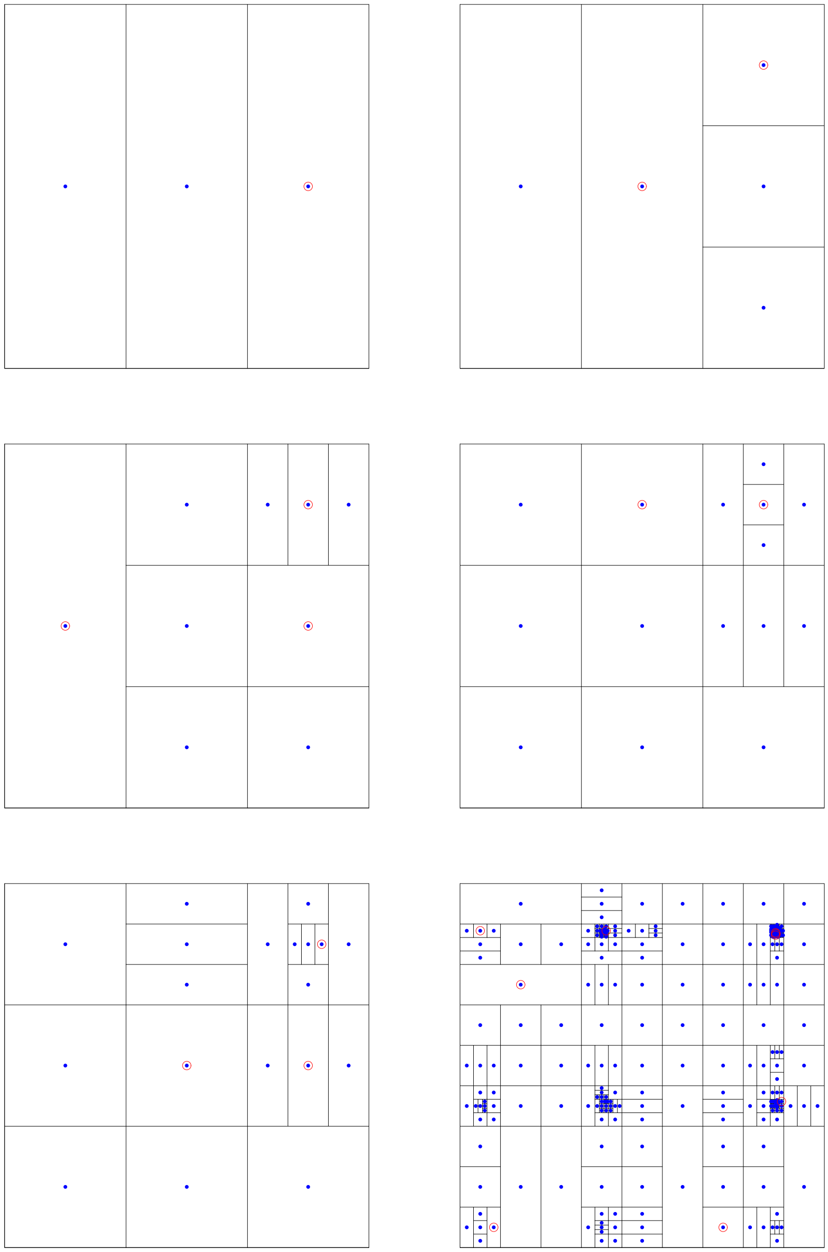

Fig. 4.6 The 5 first iterations of the SOO algorithm and the resulting tree $\mathcal{T}_n$ after $n = 150$ function evaluations. Here we considered the 2-dimensional function $f(x_1, x_2) = f(x_1)f(x_2)$ where $f(x) = \big(\sin(13x)\sin(27x) + 1\big)/2$ and $K = 3$. When a node is expanded, its corresponding cell is split in the widest direction in 3 subsets of same size.

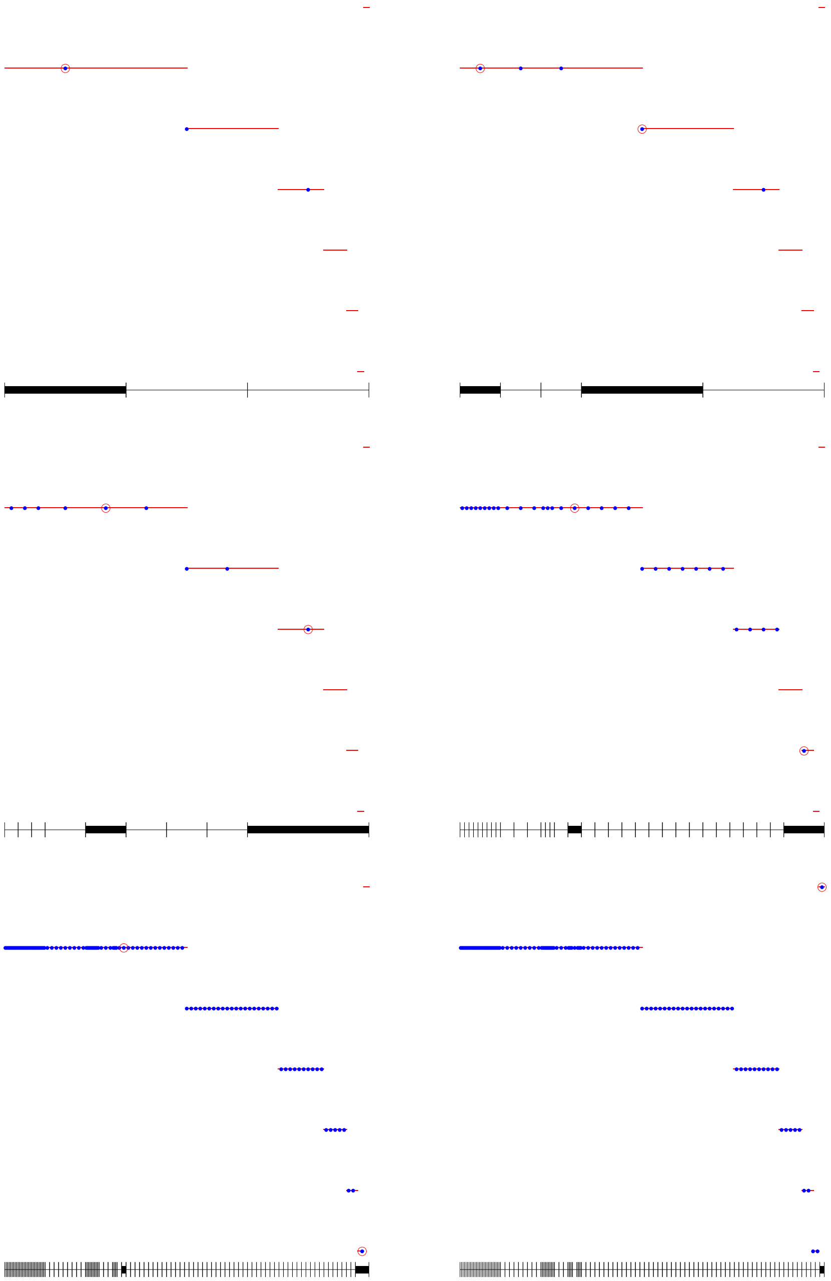

Fig. 4.7 The 3 first iterations of the SOO algorithm and after $n = 17, 53$ and 54 function evaluations on the function built from the example illustrated in Figure 2.2.

Comparison with the DIRECT algorithm: The DIRECT (DIviding RECTangles) algorithm [67, 50, 53] is a Lipschitz optimization algorithm when the Lipschitz constant L of f is unknown. It uses an optimistic splitting technique similar to ours where at each round, it expands the set of nodes that have the highest upper-bound (as defined in DOO) for at least some value of L. To the best of our knowledge, there is no finite-time analysis of this algorithm (only the consistency property $\lim_{n\to\infty} r_n = 0$ is proven in [50]). Our approach generalizes DIRECT and we are able to derive finite-time loss bounds in a much broader setting where the function is assumed to be locally smooth (instead of globally Lipschitz) only and the space is assumed semi-metric only.

We are not aware of other finite-time analyses of similar global optimization algorithms that do not require the knowledge of the smoothness of the function.

SOO is a rank-based algorithm: The algorithm only requires the knowledge of the rank of the function evaluations and not their specific values. Indeed the decision to expand a node only depends on whether the value at this node is larger than the values of all nodes of the same or lower depth. The specific values are not important as long as their rank is preserved. This is also a property shared by the CMA-ES optimization algorithm (see e.g. Figure 10.4 in [15]). Thus if $g : I\!R \mapsto I\!R$ is strictly increasing, SOO will perform identically on f and $g \circ f$. For example SOO will perform identically on $x \mapsto \|x - x^*\|$ and $x \mapsto g(\|x - x^*\|)$. And our analysis of the loss of SOO actually reflect this property since we can choose to define the semi-metric as $\ell(x, y) = g(\|x - y\|)$, as illustrated in subsection 4.1.2 for the case $g(z) = z^\alpha$.

SOO for the unsmooth function illustrated in Figure 2.2 Finally we report in Figure 4.7 the result of SOO applied to the function built from the example illustrated in Figure 2.2 (where we used $D = 6$). This function served as an illustration of the perticularly bad behavior of UCT. If one looks at this function at a high level scale, this function does not possess any smoothness around its maximum. Actually, for this type of functions, the best algorithm would be a uniform search, since no information from higher level can be used to guide the search.

In addition, any optimistic algorithm will be fooled here since, in any cell (not containing the optimum), the function has higher values on the left than on the right. Thus, as long as the optimum is not reached, the search will be focussing more on left branches than on right ones, at all levels, leading to a perticularly misleading behavior. This was all the more true for UCT since the B-values computed by UCT are not true high probability upper-confidence-bounds, and we saw that the number of samples required by UCT to find the optimum can be as bad as $\Omega(\exp(\exp(\ldots\exp(1)\ldots)))$, where the number of intricated exponentials is D.

In contrast, SOO (fitted with $h_{\max}(t) = \sqrt{t}$) requires "only" $\Omega(K^{2D})$ samples to find the optimum. This is because at each iteration SOO expands a node with largest diameter (out of at most $h_{\max}(n)$ nodes). Thus after n node expansions, the $\sqrt{n}$ nodes of lowest depth have all been expanded.

Now if the actual "smoothness" of the function were known (we can show that here $d = 0$ and $C = 2^D$) one could use it to define true upper-confidence-bounds and use it in the DOO algorithm. Such a DOO optimally fitted would expand first the nodes with lowest depth (since the diameter term $\delta(h)$ would dominate the evaluations $f(x_{h,j})$ in the computation of the b-value $b_{h,j}$, see Algorithm 3.6), thus reducing to a uniform search. The resulting number of samples required to find the optimum would be K^D.

Thus in this hard instance of function optimization, the uniform search (or DOO optimally fitted) is exponential in D, SOO is exponential in a multiple of D, whereas UCT is "D-uply" exponential. This example illustrates the fact that any optimistic algorithm that does not know the smoothness of the function may be poorer than a uniform search on perticularly unsmooth functions. But this is the price to pay in order to be able to do much better than uniform as soon as the function possesses some smoothness (even if it is unknown).

4.2 Extensions to the stochastic case

We now consider an extension of SOO to the stochastic case where an evaluation of f at a point x_t returns a noisy estimate r_t of $f(x_t)$ such that $\mathbb{E}[r_t|x_t] = f(x_t)$. We actually follow the same approach as StoOO (see Section 3.4.1) where each state $x_{h,j}$ is sampled several times in order to build an accurate estimate of $f(x_{h,j})$ before the corresponding node $X_{h,j}$ is expanded.

The corresponding algorithm, called StoSOO (for Stochastic and Simultaneous Optimistic Optimization), is described in Figure 4.8 and introduced in [105].

```
Parameters: $\delta > 0$, the max number of samples per node $k > 0$, and the
maximum depth function $t \mapsto h_{\max}(t)$.
Initialization: $\mathcal{T}_1 = \{(0,0)\}$ (root node). Set $t = 1$ (round number)
while $t \leq n$ do
  Set $v_{\max} = 0$.
  For each leaf $(h,j) \in \mathcal{L}_t$, compute the b-values $b_{h,j}(t)$ according to (4.3).
  for $h = 0$ to $\min(\mathrm{depth}(\mathcal{T}_t), h_{\max}(t))$ do
    if $t \leq n$ then
      Among all leaves of depth $h$, select $(h,i) \in \arg\max_{(h,j)\in\mathcal{L}_t} b_{h,j}(t)$
      if $b_{h,i}(t) \geq v_{\max}$ then
        if $T_{h,i}(t) < k$ then
          Sample state $x_t = x_{h,i}$ and collect reward $r_t$
          $t \leftarrow t + 1$
        else
          Add the $K$ children of $(h,i)$ to $\mathcal{T}_t$ (we expand this node)
          Set $v_{\max} = b_{h,i}(t)$.
        end if
      end if
    end if
  end for
end while.
Return the state with highest empirical value corresponding to the set of
expanded nodes:
    $x(n) = \arg\max_{x_{h,j}\in\mathcal{T}_n\setminus\mathcal{L}_n} \hat{\mu}_{h,j}(n).$
```

Fig. 4.8 The Stochastic Simultaneous Optimistic Optimization (StoSOO) algorithm

StoSOO defines the b-values $b_{h,j}(t)$ of any node at round t, by

$$b_{h,j}(t) \overset{\text{def}}{=} \hat{\mu}_{h,j}(t) + \sqrt{\frac{\log(n^2/\delta)}{2T_{h,j}(t)}}, \tag{4.3}$$

where $T_{h,j}(t) \overset{\text{def}}{=} \sum_{s=1}^{t} \mathbf{1}\{x_s \in X_{h,j}\}$ is the number of times (h,j) has been selected up to time t, and $\hat{\mu}_{h,j}(t) \overset{\text{def}}{=} \frac{1}{T_{h,j}(t)} \sum_{s=1}^{t} r_s \mathbf{1}\{x_s \in X_{h,j}\}$ is the empirical average of the rewards received in $X_{h,j}$. In the case of $T_{h,j}(t) = 0$ we let $b_{h,j}(t) = \infty$.

Now, instead of selecting the promising nodes according of their value $f(x_{h,j}$ we select them according to their $b-$value $b_{h,j}$. The parameter k used in the algorithm is the number of samples collected from a state before the corresponding node is expanded. Finally, StoSOO returns the state $x(n)$ with highest empirical value among the set of nodes that have been expanded (thus which have been sampled k times).

4.2.1 Analysis of StoSOO

We have the property that for any $\delta > 0$, defining the event ξ as in (3.10), Lemma 3.2 implies that $\mathbb{P}(\xi) \geq 1 - \delta$. Notice that the b-values $b_{h,j}(t)$ define high-probability upper-confidence-bounds on the values $f(x_{h,j})$ (and not on $\sup_{x \in X_{h,j}} f(x)$ as it was the case for the b-values defined by StoOO in (3.9)).

Thus the intuition of the algorithm is that in the event ξ, the estimation $\hat{\mu}_{h,j}(t)$ of a node $X_{h,j}$ that has been expanded (thus sampled at least k times) is ϵ-close to its value $f(x_{h,j})$, where $\epsilon = \sqrt{\frac{\log(n^2/\delta)}{2k}}$.

Thus, in the event ξ, StoSOO is very close to the algorithm SOO where:

- The sampling budget is only $m = n/k$, which corresponds to the number of nodes that are expanded,
- Each of the m evaluations is only ϵ-correct.

Indeed notice that when a node (h,i) is expanded by StoSOO it means that k samples have been collected from the state $x_{h,i}$, and the ϵ-estimation $\hat{f}(x_{h,i})$ of $f(x_{h,i})$ is at least as good as the estimation $\hat{f}(x_{h',j})$ of nodes that have been expanded at previous depths $h' < h$.

Thus the analysis of StoSOO (in the event ξ) reduces to the analysis of the so-called **"ϵ-perturbed SOO algorithm**, which is defined exactly as the SOO algorithm except that each evaluation to the function is perturbed by at most ϵ (i.e., when sampling a state $x_{h,j}$ one observes $\hat{f}(x_{h,j})$, which is such that $|\hat{f}(x_{h,j}) - f(x_{h,j})| \leq \epsilon$).

Let us now analyze this ϵ-perturbed SOO using a similar proof to that of SOO. Define the sets

$$I_h^\epsilon \overset{\text{def}}{=} \{\text{nodes } (h,i) \text{ such that } f(x_{h,i}) + \delta(h) + 2\epsilon \geq f^*\}.$$

After t (perturbed) function evaluations, let us write h_t^* the depth of the deepest expanded node in the branch containing x^* (an optimal branch). Let (h_t^*+1, i^*) be an optimal node of depth h_t^*+1 (i.e., such that $x^* \in X_{h_t^*+1,i^*}$). Since this node has not been expanded yet, any node (h_t^*+1, i) of depth h_t^*+1 that is later expanded, before (h_t^*+1, i^*) is expanded, is $[\delta(h_t^*+1)+2\epsilon]$-optimal. Indeed,

$$\begin{aligned} f(x_{h_t^*+1,i}) &\geq \hat{f}(x_{h_t^*+1,i}) - \epsilon \geq \hat{f}(x_{h_t^*+1,i^*}) - \epsilon \\ &\geq f(x_{h_t^*+1,i^*}) - 2\epsilon \geq f^* - \big[\delta(h_t^*+1) + 2\epsilon\big]. \end{aligned}$$

We deduce that once an optimal node of depth h is expanded, it takes at most $|I_{h+1}^\epsilon|$ node expansions at depth $h+1$ before the optimal node of depth $h+1$ is expanded, which corresponds to at most $h_{\max}(t)|I_{h+1}^\epsilon|$ function evaluations. We deduce the following lemma as a straightforward extensions of Lemma 4.1.

Lemma 4.2. For any depth $0 \leq h \leq h_{\max}(t)$, whenever $t \geq h_{\max}(t)(|I_0^\epsilon| + |I_1^\epsilon| + \cdots + |I_h^\epsilon|)$, we have $h_t^* \geq h$.

The next lemma bounds the number of nodes in the sets $|I_h^\epsilon|$ for any depth $h \leq h_\epsilon \overset{\text{def}}{=} \min\{h \geq 0, \text{s.t. } \delta(h+1) < \epsilon\}$:

Lemma 4.3. Let d be the $\nu/3$-near-optimality dimension (where ν is defined in Assumption 4), and C be the corresponding constant. Then for any $h \leq h_\epsilon$, we have

$$|I_h^\epsilon| \leq C\big[\delta(h) + 2\epsilon\big]^{-d}.$$

Proof. The proof is similar to that of Lemma 3.1. By contradiction: for $h \leq h_\epsilon$, if $|I_h^\epsilon| > C[\delta(h) + 2\epsilon]^{-d}$ we would have $|I_h^\epsilon| > C[3\delta(h)]^{-d}$, which would mean that there exists more than $C[3\delta(h)]^{-d}$ disjoint ℓ-balls of radius $\nu\delta(h)$ with center in $\mathcal{X}_{\delta(h)}$. This contradicts the fact that d is the $\nu/3$-near-optimality dimension. □

Now we can state our main result for ϵ-perturbed SOO using a budget of m perturbed evaluations of f.

Theorem 4.2. Let d be the $\nu/3$-near-optimality dimension and $h(m)$ be the smallest integer h such that

$$Ch_{\max}(m) \sum_{l=0}^{h} \big[\delta(l) + 2\epsilon\big]^{-d} \geq m. \tag{4.4}$$

Then the loss of ϵ-perturbed SOO is bounded as

$$r_m \leq 2\epsilon + \delta\big(\min(h(m) - 1, h_{\max}(m), h_\epsilon)\big). \tag{4.5}$$

Proof. Consider first the case when $h(m) - 1 \leq h_\epsilon$. Then using a similar argument as in the proof of Theorem 4.1 we deduce that after m node expansions, the depth h_m^* of the deepest expanded node in the branch containing x^* satisfies $h_m^* \geq \min(h(m) - 1, h_{\max}(m))$. Now if $h(m) - 1 > h_\epsilon$, we can use Lemma 4.3 up to depth h_ϵ to deduce similarly that $h_m^* \geq \min(h_\epsilon, h_{\max}(m))$. Thus altogether $h_m^* \geq \min(h(m) - 1, h_\epsilon, h_{\max}(m))$.

Now define (h_m^*, i^*) the optimal node of depth h_m^* (i.e., containing x^*). Let $x_{h,j}$ be the state returned by the algorithm. Thus $\hat{f}(x_{h,j}) \geq \hat{f}(x_{h_m^*,i^*})$ and we deduce that

$$\begin{aligned} f(x_{h,j}) &\geq \hat{f}(x_{h,j}) - \epsilon \geq \hat{f}(x_{h_n^*,i^*}) - \epsilon \\ &\geq f(x_{h_n^*,i^*}) - 2\epsilon \geq f^* - \delta(h_m^*) - 2\epsilon \\ &\geq f^* - \delta(\min(h(m) - 1, h_{\max}(m), h_\epsilon)) - 2\epsilon. \end{aligned}$$

□

We now state our main result for StoSOO in the case when the near-optimality dimension for the best valid semi-metric ℓ is $d = 0$.

Theorem 4.3. Assume there exists a semi-metric ℓ such that f is locally smooth around one global optimum x^* (i.e. such that (3.8) holds) and that Assumptions 1, 3, 4 hold. Assume that the diameters (measured with ℓ) of the cells decrease exponentially fast, i.e. $\delta(h) = c\gamma^h$ for some $c > 0$ and $\gamma < 1$. Assume that the $\nu/3$-near-optimality dimension is $d = 0$ (and write C the corresponding constant). Then the expected loss of StoSOO run with parameters k, $h_{\max}(t) = \sqrt{t/k}$, and $\delta > 0$, is bounded as:

$$\mathbb{E}[r_n] \leq (2 + 1/\gamma)\sqrt{\frac{\log(n^2/\delta)}{2k}} + c\gamma^{\sqrt{n/k}\min(1/C,1)-1} + \delta. \qquad (4.6)$$

In particular, for the choice $k = \frac{n}{(\log n)^3}$ and $\delta = 1/\sqrt{n}$, we have

$$\mathbb{E}[r_n] = O\Big(\frac{(\log n)^2}{\sqrt{n}}\Big).$$

Proof. In the event ξ, the StoSOO algorithm behaves like the ϵ-perturbed SOO with $\epsilon = \sqrt{\frac{\log(n^2/\delta)}{2k}}$ run for $m = n/k$ rounds (node expansions).

When $d = 0$, from Theorem 4.2, we have that $m \leq Ch_{\max}(m)\sum_{l=0}^{h(m)}\big[\delta(l) + 2\epsilon\big]^{-d} = Ch_{\max}(m)(h(m) + 1)$, thus for $h_{\max}(m) = \sqrt{m}$ we deduce that the loss of ϵ-perturbed SOO (thus the loss of StoSOO in the event ξ) is at most:

$$\begin{aligned} r_n &\leq 2\epsilon + \delta\big(\min(h(m) - 1, h_{\max}(m), h_\epsilon)\big) \\ &\leq 2\epsilon + \delta(h_\epsilon) + \delta\big(\min(h(m) - 1, h_{\max}(m))\big) \\ &\leq (2 + 1/\gamma)\epsilon + c\gamma^{\sqrt{m}\min(1/C,1)-2}. \end{aligned}$$

The bound on the expected loss of StoSOO follows from the fact that ξ holds with probability $1 - \delta$.

Finally, for the specific choice $k = \frac{n}{(\log n)^3}$ we notice that the second term in the bound (4.6) is a $o(1/\sqrt{n})$. □

Thus in the case the near-optimality dimension for the best valid semi-metric is $d = 0$ and the diameters are exponentially decreasing, StoSOO achieves the same rate $1/\sqrt{n}$ (neglecting logarithmic factors) as StoOO and HOO (which required the knowledge of the semi-metric ℓ). In the next subsection we discuss this important case $d = 0$.

4.2.2 The case $d = 0$

Notice that SOO and StoSOO algorithms do not require the knowledge of the semi-metric ℓ; the semi-metric is only used in the analysis of the algorithm. Thus one can choose the best possible semi-metric ℓ, **possibly according to the function f itself**, as long as it satisfies the following properties:

- f should be locally smooth w.r.t. ℓ around a global optimum x^* (i.e. such that (**??**) holds)
- The diameters of the cells (measured with ℓ) should decrease exponentially fast
- There exists $C > 0$ such that for any $\epsilon > 0$, the maximal number of disjoint ℓ-balls of radius $\nu\epsilon$ centered in $\mathcal{X}_\epsilon$ is less than C (i.e. the near-optimality dimension d is 0).

In Examples 1 and 2 we illustrated the case of functions f defined on $[0,1]^D$ that are locally equivalent to a polynomial of degree α around their maximum, i.e., $f(x) - f(x^*) = \Theta(\|x - x^*\|^\alpha)$ for some $\alpha > 0$, where $\|\cdot\|$ is any norm. The precise definition is given in Example 2 of Subsection 3.3.2. In light of the discussion in Subsection 4.1.2, the choice of semi-metric $\ell(x,y) \overset{\text{def}}{=} \|x - x^*\|^\alpha$ implies that the near-optimality dimension $d = 0$.

More generally, this results extends to any function whose upper- and lower envelopes around x^* are of same order. More precisely, we assume that there exists constants $c > 0$, and $\eta > 0$, such that

$$\min(\eta, c\ell(x,x^*)) \leq f(x^*) - f(x) \leq \ell(x,x^*), \quad \text{for all } x \in \mathcal{X}. \tag{4.7}$$

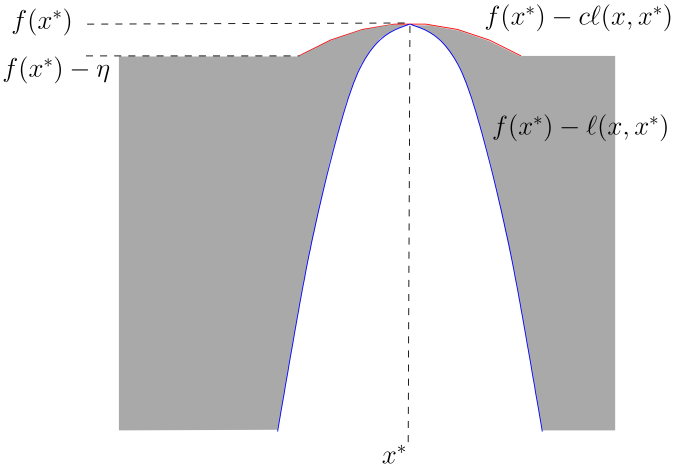

Fig. 4.9 Any function satisfying (4.7) (i.e., lying in the gray area) has a near-optimality dimension $d = 0$ since it possesses a lower- and upper-envelopes that are of same order around x^*.

Now, one can even define the semi-metric ℓ according to the behavior of f around x^* in order that (**??**) holds. For example if the space $\mathcal{X}$ is a normed space (with norm $\|\cdot\|$), one can define the metric $\ell(x, y) \overset{\mathrm{def}}{=} \tilde{\ell}(\|x - y\|)$ with

$$\tilde{\ell}(r) \overset{\mathrm{def}}{=} \sup_{x;\|x^*-x\| \leq r} f(x^*) - f(x).$$

Thus $f(x^*) - \ell(x, x^*)$ naturally forms a lower-envelope of f. Thus assuming that the first inequality of (4.7) (upper-envelope) holds, then the near-optimality dimension is $d = 0$ again. This is the case in particular when the function is strongly concave, or only locally strongly concave (i.e. only in a η-neighborhood of x^*).

However, although the case $d = 0$ is quite general, it does not hold in situations where there is a discrepancy between the upper- and lower-envelopes of f around x^* as illustrated in Figure 4.10.

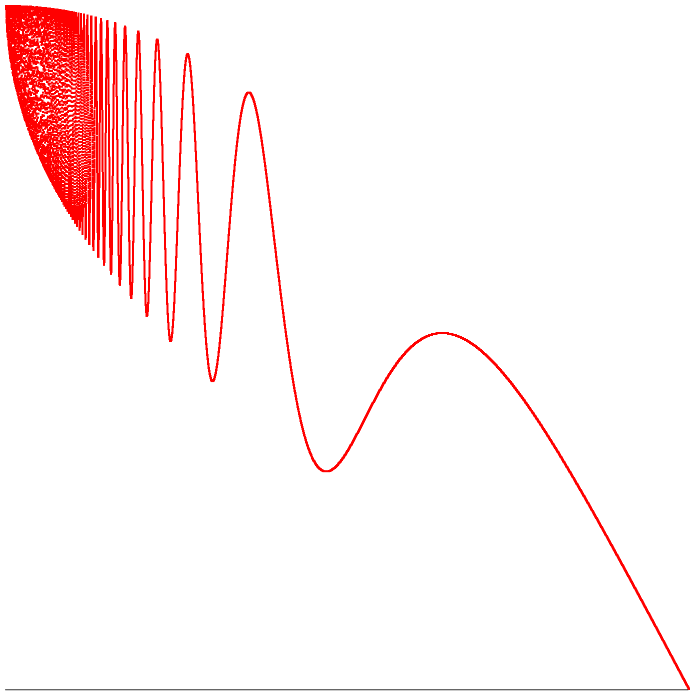

Fig. 4.10 We illustrate the case of a function with different order in the upper and lower envelopes. Here $f(x) = 1 - \sqrt{x} + (-x^2 + \sqrt{x}) * (\sin(1/x^2) + 1)/2$. The lower-envelope is of order $1/2$ whereas the upper one is of order 2. We deduce that $d = 3/2$.

Finally, as discussed in Remark 3.3, the near-optimality dimension d is a *local* property of f near x^* since it coincides with the local near-optimality dimension (defined in the same remark). However the corresponding constant C in the definition 3.1 depends on the global shape of f. For instance, in a Euclidean space $\mathcal{X}$, assume that a function f has a near-optimality dimension d around x^* with a corresponding constant C. Now consider the function $\tilde{f}$ defined as $\tilde{f}(x) \overset{\text{def}}{=} \max_{1 \le i \le k} f(x^* - x + x_i)$, where $\{x_1, \ldots, x_k\}$ are k points in $\mathcal{X}$ (i.e. $\tilde{f}$ is the maximum of k translated copies of f). Thus $\tilde{f}$ possesses

k global optima $\{x_1, \ldots, x_k\}$ and the near-optimality dimension of $\tilde{f}$ is still d but the corresponding constant can be as large as kC (this is simply because it requires at most k times more balls to cover the set of ϵ-optimal states of $\tilde{f}$, than it takes for ϵ-optimal states of f).

4.3 Conclusions

Assuming that the function f is locally smooth w.r.t. some semi-metric ℓ enables the design of optimistic exploration strategies, even when ℓ is unknown. Since the algorithm does not depend on ℓ, the loss analysis can be undertaken using the best possible valid (i.e. such that Assumptions 1, 2, 3, 4 hold) semi-metric. In the deterministic case, the SOO algorithm performs almost as well as DOO optimally-fitted.

In the stochastic case, the StoSOO algorithms performs almost as well as StoOO or HOO only in the case when there exists a valid semi-metric such that the corresponding near-optimality dimension d is 0. We showed that this already covers a large class of functions. Now, when this is not the case (as illustrated in Figure 4.10) the problem of designing an algorithm that would do almost as well as StoOO or HOO for the smallest $d > 0$ corresponding to a valid semi-metric, is open.

5

Optimistic planning

In this chapter we consider the optimistic approach for solving planning problems. In comparison to the previous chapters about optimization, the planning problem introduces some structure in the search space and the function to be optimized. Here, the search space is the set of possible policies (where a policy is an action to follow in each possible situation), and the function to optimized (the so-called value function) depends on the sum of rewards along the trajectories resulting from the policy that is evaluated.

We consider that a full model of the dynamics and the reward function is available but each call to the model has some numerical cost. Thus our goal is to return the best possible plan given a finite number of calls to the model (our numerical budget). A possible setting is the following.

Online planning: We consider a class of online model-based algorithms that, at each step, looks at the current state of the system and uses the model to predict the system's response to various sequences of actions. Exploiting these predictions, an action that is as good as possible is applied in the real world, which results in a new state. The entire cycle then repeats. In computer science such algorithms belong to the planning class [80] and are known as online planning [73, 88] or lazy planning [47]. While we use the name ‘online planning’ and mainly refer to the computer science literature, it must be emphasized that such algorithms are also widely studied in systems and control, where they are known as model-predictive or receding-horizon control [83, 38]. In the AI community, related works are the classical A* heuristic search [87] and the AO* variant from [62].

More precisely, at time k, let the current state of the system be x_k. Our goal is to select an action a_k to follow. In order to do so, we perform a simulated search (planning) in the set of all possible policies starting from the current state x_k using a finite numerical budget (here a finite number n of calls to the generative model), and this search returns a recommended action a_k to follow. Then this action is executed in the real environment, which generates a transition to a next state x_{k+1}. Then another search is performed from this new state, and the same procedure is repeated again and again. This is called *online planning* because the planning is performed online at each time step. Since time is limited for selecting each action, the planning part should be as efficient as possible given the time (or numerical budget) allowed.

The goal of this “online planning” is thus to perform in each current state a search in a policy space starting from that state and using a finite budget n and return a recommended action whose quality is almost as good as the best action to follow from that state.

The online planning approach is different from the value-function and policy search methods usually considered in dynamic programming and reinforcement learning [101, 20, 102, 96, 33]; the latter methods usually seek a global solution, whereas online planning finds actions on demand, locally for each state where they are needed. Online planning is therefore much less dependent on the state space size.

In this chapter we present three settings where the optimistic principle can guide us in performing this search [32]. In all settings we consider an infinite-time horizon with discounted rewards. Section 5.1 considers the case of deterministic dynamics and reward functions, Section 5.2 the case of general stochastic rewards with deterministic dynamics, and Section 5.3 the general case of Markov Decision Processes.

In all three situations we provide performance bounds on the loss (how close the quality of the recommended action is from that of the optimal action) as a function of the number of calls to the model. For clarity, in this chapter we will make use standard notations in control that differ from the notations used in previous chapters.

5.1 Deterministic dynamics and rewards

5.1.1 Setting and notations

Here the dynamics and reward functions are deterministic. Let us write X the state space, A the action space, $f : X \times A \to X$ the transition dynamics, and $r : X \times A \to I\!R$ the reward function. Thus if at time t, the current state is $x_t \in X$ and the chosen action a_t, then the system jumps to the next state $x_{t+1} = f(x_t, a_t)$ and a reward $r(x_t, a_t)$ is received. Again we will assume that all rewards are in the interval $[0, 1]$.

We assume that the state space is large (possibly infinite), and the action space is finite, with K possible actions. We consider an infinite-time horizon problem with discounted rewards ($0 \le \gamma < 1$ is the discount factor). For any policy $\pi : X \to A$ we define the value function $V^\pi : X \to I\!R$ associated to that policy:

$$V^\pi(x) \overset{\text{def}}{=} \sum_{t \ge 0} \gamma^t r(x_t, \pi(x_t)),$$

where x_t is the state of the system at time t when starting from x (i.e. $x_0 = x$) and following policy π.

We also define the Q-value function $Q^\pi : X \times A \to I\!R$ associated to a policy π, for each state-action pair (x, a), as the value of playing a in x and π after:

$$Q^\pi(x, a) \overset{\text{def}}{=} r(x, a) + \gamma V^\pi(f(x, a)).$$

We have the property that $V^\pi(x) = Q^\pi(x, \pi(x))$. Now the optimal value function (respectively Q-value function) is defined as: $V^*(x) \overset{\text{def}}{=} \sup_\pi V^\pi(x)$ (respectively $Q^*(x, a) \overset{\text{def}}{=} \sup_\pi Q^\pi(x, a)$, which corresponds to playing a now and optimally after). And from the dynamic programming principle, we have the Bellman equations:

$$V^*(x) = \max_{a \in A} \big[r(x, a) + \gamma V^*(f(x, a)) \big]$$

$$Q^*(x,a) = r(x,a) + \gamma \max_{b \in A} Q^*(f(x,a),b).$$

5.1.2 Planning under finite numerical budget

We assume that we possess a generative model of f and r that can be used to generate simulated transitions and rewards. We want to make the best possible use of this model in order to return a single action (or a sequence of actions), given an initial state. The action-selection procedure takes as input the current state of the system and outputs an action $a(n)$ using at most n calls to the generative model. The amount n of available numerical resources may not be known before they are all used (e.g. because of time constraints), so we wish to design anytime algorithms that can return an action $\mathcal{A}(n)$ for any time n. Our goal is that the proposed action $a(n)$ be as close as possible to the optimal action in that state, and we define the performance loss r_n resulting from choosing this action and then following an optimal path instead of following an optimal path from the beginning:

$$r_n \overset{\text{def}}{=} \max_{a \in A} Q^*(x,a) - Q^*(x,a(n)). \tag{5.1}$$

Thus the goal is to find the best way to explore the environment (first phase) so that, once the available resources have been used, the agent is able to make the best possible recommendation on the action to play in the environment.

From an action-selection algorithm one may define a policy π which would select in each state encountered along a trajectory the action recommended by the algorithm using n calls to the model. The previous definition of the loss is motivated by the fact that an algorithm with small loss at each state (say $r_n \leq \epsilon$) will generate a policy π which is $\frac{\epsilon}{1-\gamma}$-optimal, i.e. for all x, $V^*(x) - V^\pi(x) \leq \frac{\epsilon}{1-\gamma}$ (see [66]).

5.1.3 The planning tree

For a given initial state x, consider the (infinite) planning tree defined by all possible sequences of actions (thus all possible reachable states starting from x). Write A^∞ the set of infinite sequences $(a_0, a_1, a_2, \dots)$ where $a_t \in A$. The branching factor of this tree is the number of actions

$|A| = K$. Since the dynamics are deterministic, to each finite sequence $a \in A^d$ of length d corresponds a state that is reachable starting from x by following a sequence of d actions.

Using standard notations over alphabets, we write $A^0 = \{\emptyset\}$, A^* the set of finite sequences, for $a \in A^*$ we write $h(a)$ the length of a, and $aA^h = \{aa', a' \in A^h\}$, where aa' denotes the sequence a followed by a'. We identify the set of finite sequences $a \in A^*$ to the set of nodes of the tree.

The value $v(a)$ of an infinite sequence $a \in \mathcal{A}^\infty$ is the discounted sum of rewards along the trajectory starting from the initial state x and defined by the choice of this sequence of actions:

$$v(a) \overset{\text{def}}{=} \sum_{t \geq 0} \gamma^t r(x_t, a_t), \text{ where } x_0 = x, \text{ and } x_{t+1} = f(x_t, a_t).$$

Now, for any finite sequence $a \in A^*$ (or node) we define the value $v(a) = \sup_{a' \in \mathcal{A}^\infty} v(aa')$. We write $v^* = v(\emptyset) = \sup_{a \in \mathcal{A}^\infty} v(a)$ the optimal value. We also define the u- and b-values (respectively lower- and upper- bounds on v) as

$$l(a) \overset{\text{def}}{=} \sum_{t=0}^{h(a)} \gamma^t r(x_t, a_t) \tag{5.2}$$

$$b(a) \overset{\text{def}}{=} l(a) + \frac{\gamma^{h(a)+1}}{1-\gamma}, \tag{5.3}$$

Indeed, since all rewards are in $[0, 1]$ we have that $l(a) \leq v(a) \leq b(a)$.

At any finite time t an algorithm has expanded a set of t nodes, which define the expanded tree $\mathcal{T}_t$. Expanding a node $a \in A^h$ means using the generative model f and r to generate transitions and rewards for the K children nodes aA. The set of leaves of $\mathcal{T}_t$ represents the set of nodes that can be expanded at time $t+1$ and is denoted by $\mathcal{L}_t$.

Thus, once a node, $a \in A^*$ is expanded, the values $l(a)$ and $b(a)$ can be computed (since they only depend on rewards obtained along the finite sequence a).

5.1.4 Minimax bounds

First, consider a uniform exploration policy, defined by expanding at each round t a node in $\mathcal{L}_t$ with the smallest depth. Now, at time n

(once n nodes have been expanded), the algorithm returns the immediate action with largest u-value: $a(n) \overset{\text{def}}{=} \arg\max_{a\in A} l(a)$ (ties broken arbitrarily).

This strategy expands the set of sequences in a uniform fashion; hence, at round $n = 1 + K + K^2 + \cdots + K^d = \frac{K^{d+1}-1}{K-1}$, all nodes of depth up to d have been expanded. Thus the value $l(a)$ of each action $a \in A$ is known up to an error $v(a) - l(a) \leq \frac{\gamma^{d+1}}{1-\gamma}$, since the rewards of all paths up to depth d have been seen, and the remaining rewards from depths $d+1$ on sum to at most $\frac{\gamma^{d+1}}{1-\gamma}$. We deduce an upper-bound on the loss of uniform planning:

$$r_n \leq \frac{1}{\gamma(1-\gamma)}\big[n(K-1)+1\big]^{-\frac{\log 1/\gamma}{\log K}}. \tag{5.4}$$

In addition we have a matching lower-bound: For any algorithm and any n, there exists a reward function, such that the loss is at least

$$r_n \geq \frac{\gamma}{1-\gamma}\big[n(K-1)+1\big]^{-\frac{\log 1/\gamma}{\log K}}. \tag{5.5}$$

The proof of those results can be found in [66]. We thus observe that the uniform planning strategy achieves a loss of $O(n^{-\frac{\log 1/\gamma}{\log K}})$ in a minimax sense (i.e. for any possible environment). And the lower-bound tells us that (up to a constant factor) there is no algorithm that can do better uniformly over all problems.

However, this does not tell us that there is not better algorithms for some problems. In the next section we show that strictly better algorithms can be designed for specific class of problems.

5.1.5 Optimistic planning

The infinite set of sequences A^∞ is our search space (denoted by $\mathcal{X}$ in previous sections) and each $a \in \mathcal{A}^\infty$ is a point in that space. The value $v(a)$ of each sequence $a \in A^\infty$ is the sum of discounted rewards along the sequence. Now, by defining the metric $\ell(a, a') = \frac{\gamma^{h(a,a')}}{1-\gamma}$, where $h(a, a') \overset{\text{def}}{=} \max\{t \geq 0, \forall 0 \leq s \leq t, a_s = a'_s\}$, we have the property that for all $a, a' \in A^\infty$,

$$|v(a) - v(a')| \leq \ell(a, a'),$$

thus the value function v is Lipschitz w.r.t. the metric ℓ.

Any subtree $\mathcal{T}_t$ corresponds to a partitioning of A^∞ into t subsets. Expanding a leaf $a \in \mathcal{L}_t$ of this tree means splitting the corresponding subset into K smaller subsets aa', for $a \in A$. To each subset $a \in \mathcal{L}_t$ the value $b(a)$ is an upper-bound on $v(a)$.

Thus one may apply the DOO algorithm from Section 3.3: at each round t, expand the leaf of the expanded tree with highest b-value. And after n node expansions, return the action with highest u-value (where the values are defined as in 5.2).

This defines an algorithm, called Optimistic Planning algorithm (OPD) (see Algorithm 1), that builds an asymmetric planning tree aiming at exploring first the most promising parts of the tree. Branches with low rewards close to the root will not be further explored and only near-optimal paths will be continually expanded.

Algorithm 1 Optimistic Planning algorithm (OPD)

```
Expand the root.
for $t = 1$ to $n$ do
  Expand a node $a_t \in \arg\max_{a \in \mathcal{L}_t} b(a)$,
end for
return  Action $\arg\max_{a \in A} l(a)$
```

Although OPD is directly inspired from DOO, there are two important differences with DOO: (1) here we have a structured problem where the value $v(a)$ of any point $a \in A^\infty$ is the sum of (discounted) rewards along an (infinite) sequence of actions, and (2) the budget n represents the number of calls to the generative model (i.e. transitions and rewards) and not directly the number of evaluations of the function v.

Analysis The loss of OPD is bounded as

$$r_n(\mathcal{A}_O) \leq \frac{\gamma^{d_n}}{1-\gamma}, \tag{5.6}$$

where d_n is the maximal depth of nodes in $\mathcal{T}_n$ ([66]). As a consequence, for any reward function, the upper bound on the loss for the optimistic planning is never larger than that of the uniform planning (indeed since the uniform exploration is the exploration strategy with minimal depth d_n for a given n, thus the depth obtained when using OPD is at least as high as that of the uniform one).

However the lower bound tells us that no improvement (over uniform planning) may be expected in a worst-case setting. In order to quantify possible improvement over uniform planning, one thus needs to define specific classes of problems.

We now define a measure of the quantity of near-optimal sequences. More precisely, by denoting $\mathcal{T}^+ \subset \mathcal{T}^\infty$ the set of sequences in A^h, for any h, that are $\frac{\gamma^{h+1}}{1-\gamma}$-optimal, we define $\kappa \in [1, K]$ as the branching factor of $\mathcal{T}^+$:

$$\kappa = \limsup_{h\to\infty} \left| \left\{ a \in A^h : v(a) \geq v^* - \frac{\gamma^{h+1}}{1-\gamma} \right\} \right|^{1/h}. \tag{5.7}$$

This measure is closely related to the notion of near-optimality dimension d (and corresponding constant C) introduced in Chapter 3.3. Indeed, if there are $C'\kappa^h$ (for some constant C') sequences of length h in $\mathcal{T}^+$, then the corresponding nodes represents a set of ℓ-balls of diameter $\frac{\gamma^{h+1}}{1-\gamma}$ that form a packing of the set of (infinite) sequences that are $\frac{\gamma^{h+1}}{1-\gamma}$-optimal. Writing $\epsilon = \frac{\gamma^{h+1}}{1-\gamma}$ we have that the set of ϵ-optimal points of A^∞ can be packed by $C'\kappa^h = C\epsilon^{-d}$ such ℓ-balls with the near-optimality dimension d and corresponding constant C being:

$$d = \frac{\log \kappa}{\log 1/\gamma} \text{ and } C = C'\kappa(1-\gamma)^{-d}. \tag{5.8}$$

We have the following result:

Theorem 5.1. If $\kappa > 1$ then the loss of OPD is $r_n = O\big(n^{-\frac{\log 1/\gamma}{\log \kappa}}\big)$.

If $\kappa = 1$ and there are at most C' sequences of length h in $\mathcal{T}^+$ (for any $h \geq 0$), the loss decreases exponentially fast as $r_n = O\big(e^{-\frac{\log 1/\gamma}{C'} n}\big)$.

The proof of this result can be found in [66], but in light of the previous discussion, it is a direct consequence of the analysis of DOO.

Some intuition about $\mathcal{T}^+$: By definition, $\mathcal{T}^+$ is the set of finite sequences that are $\frac{\gamma^{h+1}}{1-\gamma}$-optimal, thus from any sequence $a \in \mathcal{T}^+$, given the sequence of rewards obtained along this sequence, one cannot decide whether this sequence belongs to an optimal sequence or not. Now once a sequence does not belong to $\mathcal{T}^+$, it is not useful to further expand it since it is clear that whatever the later rewards, it will not be part of an optimal sequence. Thus $\mathcal{T}^+$ is exactly the set of sequences that deserve to be further expanded in order to find the optimal path.

The nice property of OPD is that it only expands nodes in $\mathcal{T}^+$ (which explains why the performance of OPD is expressed in terms of the branching factor κ of $\mathcal{T}^+$). This implies that OPD cannot be improvable in the class of problems defined by a given κ.

Indeed, if we characterize the class of problems $\mathcal{P}(\kappa)$ by all environments having a set $\mathcal{T}^+$ with branching factor κ, we have that the loss of OPD on any problem $P \in \mathcal{P}(\kappa)$ satisfies: $r_n(P) = O\big(n^{-\frac{\log 1/\gamma}{\log \kappa}}\big)$. And we deduce a κ-minimax lower bound: for any algorithm, for any $\kappa \in [1, K]$, there exists a problem $P \in \mathcal{P}(\kappa)$ such that the loss of this algorithm applied to P is at least $r_n = \Omega\big(n^{-\frac{\log 1/\gamma}{\log \kappa}}\big)$.

Thus OPD is κ-minimax optimal.

Remark 5.1. OPD greatly improves over the uniform planning whenever there is a small proportion of near-optimal paths (i.e. κ is small), and the bound is always at least as good as that for uniform planning. The case $\kappa = 1$ provides exponential rates. In particular, this is the case where there exists a depth h_0 such that any sequence of depth $h \geq h_0$ along an optimal path, the gap in the Q-values at the corresponding state x_h is lower bounded by a quantity independent of h: $\exists \Delta > 0$, for all $h \geq h_0$,

$$V^*(x_h) - \max_{a \in A \text{ s.t. } Q^*(x_h,a) < V^*(x_h)} Q^*(x_h, a) \geq \Delta. \tag{5.9}$$

Indeed in such a situation, the number of nodes in a sub-optimal branch departing from any state x_h (along the optimal path) is at most K^H where $\gamma^H/(1-\gamma) \geq \Delta$. Thus $\left|\left\{a \in A^h : v(a) \geq v^* - \frac{\gamma^{h+1}}{1-\gamma}\right\}\right|$ is bounded by a constant independent of h, thus $\kappa = 1$.

SOO for planning? In previous sections (see e.g. Section 5.3.2) we built a metric ℓ defined over the space of policies, such that the value function v is Lipschitz w.r.t. ℓ (see e.g. (5.14)). Now it could be the case that the value function possesses some additional local smoothness around the optimal policy π^*, in the sense that there exists another semi-metric ℓ' of "higher order" such that (3.8) holds, i.e. for all π, $v(\pi^*) - v(\pi) \leq \ell'(\pi^*, \pi)$ (in a way similar to the example illustrated in Section 3.3.3 where the function f was globally Lipschitz w.r.t. ℓ_1 and locally smooth w.r.t. the higher-order semi-metric ℓ_2). In such cases, it would be interesting to use a version of SOO for planning. In the deterministic case described in Section 5.1, an extension of OPD to the simultaneous node expansion strategy implemented in SOO is straightforward and is expected to improve the numerical performances in some planning problems that possess such higher order smoothness.

5.2 Deterministic dynamics, stochastic rewards

Now we consider the problem of planning in environments where transitions are deterministic but rewards are stochastic. Thus for any state x and action $a \in A$, the call to the generative model returns a transition to a unique next-state $f(x, a)$ and a reward sample drawn (independently from previous samples) from a probability distribution $\nu(x, a)$ (with mean $r(x, a)$) on $[0, 1]$. Thus several calls to the generative model for each state action (x, a) are required in order to estimate precisely the average reward $r(x, a)$. Again we consider an infinite-time horizon problem with discounted rewards and the value function is defined identically as in Section 5.1.1.

Now consider the planning problem given an initial state x and define the set of infinite sequences of actions A^∞ like in Subsection 5.1.2. For any finite sequence $a \in A^*$, we write $\nu(a)$ the corresponding reward distribution, and $r(a)$ its expectation. During the exploration of the environment, the agent iteratively selects sequences of actions, under the global constraint that he can not take more than n actions in total, and receives a reward after each action. For $a \in A^h$, write $Y_h^m \sim \nu(a)$ the reward sample collected when selecting the sequence a for the m^{th} time.

5.2.1 OLOP algorithm

We now describe the Open Loop Optimistic Planning (OLOP) introduced in [26]. In that paper, the term "open-loop" referred to policies that were the function of a sequence of actions only and not of the underlying resulting states. However in the setting described here (where the transitions are deterministic), the underlying state is uniquely defined by the sequence of actions, thus the planning is actually closed-loop.

The OLOP algorithm is described in Algorithm 2. Given a budget n (which needs to be known before the algorithm starts), the algorithms generates M sequences of actions of length L (where $L \times M \leq n$). The algorithm defines b-values assigned to any sequence of actions in A^L. At time $m = 0$, the b-values are initialized to $+\infty$. Then, after episode $m \geq 1$, the b-values are defined as follows: For any $1 \leq h \leq L$, for any $a \in A^h$, let

$$T_a(m) = \sum_{s=1}^{m} \mathbf{1}\{a^s_{1:h} = a\}$$

be the number of times we played a sequence of actions beginning with a. Now we define the empirical average of the rewards for the sequence a as:

$$\widehat{\mu}_a(m) = \frac{1}{T_a(m)} \sum_{s=1}^{m} Y^s_h \mathbf{1}\{a^s_{1:h} = a\},$$

if $T_a(m) > 0$, and 0 otherwise. The corresponding upper confidence bound on the value of the sequence of actions a is by definition:

$$u_a(m) = \sum_{t=1}^{h} \left(\gamma^t \widehat{\mu}_{a_{1:t}}(m) + \gamma^t \sqrt{\frac{2 \log M}{T_{a_{1:t}}(m)}} \right) + \frac{\gamma^{h+1}}{1-\gamma},$$

if $T_a(m) > 0$ and $+\infty$ otherwise. Now that we have upper confidence bounds on the value of many sequences of actions we can sharpen these bounds for the sequences $a \in A^L$ by defining the b-values as:

$$b_a(m) = \inf_{1 \leq h \leq L} u_{a_{1:h}}(m). \qquad (5.10)$$

At each episode $m = 1, 2, \ldots, M$, OLOP selects a sequence $a^m \in A^L$ with highest b-value, observes the rewards $Y^m_t \sim \nu(a^m_{1:t})$, $t = 1, \ldots, L$

provided by the environment, and updates the b-values. At the end of the exploration phase, OLOP returns an action that has been the most often played, *i.e.* $a(n) = \arg\max_{a \in A} T_a(M)$.

Algorithm 2 Open Loop Optimistic Planning

```
Let $M$ be the largest integer such that $M\lceil \log M/(2\log 1/\gamma)\rceil \leq n$.
Let $L = \lceil \log M/(2\log 1/\gamma)\rceil$.
for $m = 1$ to $M$ do
  Computes the $b$-values at time $m-1$ for sequences of actions in $A^L$ using (5.10) and chooses a sequence that maximizes the corresponding $b$-value:
      $a^m \in \arg\max_{a \in A^L} b_a(m-1).$
end for
return Action $a(n) = \arg\max_{a \in A} T_a(M)$.
```

5.2.2 Analysis of OLOP

Let $\kappa \in [1, K]$ be defined as

$$\kappa = \limsup_{h \to \infty} \left| \left\{ a \in A^h : v(a) \geq v^* - 2\frac{\gamma^{h+1}}{1-\gamma} \right\} \right|^{1/h}. \tag{5.11}$$

Notice that this definition is very close to (5.7), where the additional 2 factor accounts for the additional uncertainty due to the empirical estimation of the rewards.

Theorem 5.2. For any $\kappa' > \kappa$, the expected loss is bounded as[1]:

$$\mathbb{E}r_n = \begin{cases} \tilde{O}\left(n^{-\frac{\log 1/\gamma}{\log \kappa'}}\right) & \text{if } \gamma\sqrt{\kappa'} > 1, \\ \tilde{O}\left(n^{-\frac{1}{2}}\right) & \text{if } \gamma\sqrt{\kappa'} \leq 1. \end{cases}$$

[1] We say that $u_n = \tilde{O}(v_n)$ if there exists $\alpha, \beta > 0$ such that $u_n \leq \alpha(\log(v_n))^\beta v_n$

5.2.3 Discussion

In this section we compare the performance of OLOP with previous algorithms that can be adapted to our framework. This discussion is summarized in Figure 5.1. We also point out several open questions raised by these comparisons.

Comparison with HOO/StoOO/Zooming algorithms In Section 5.1.5 we showed that the mapping $a \in A^\infty \mapsto v(a)$ is Lipschitz w.r.t. some metric ℓ. Thus we could use the HOO algorithm described in Section 3.4.2 (or the zooming algorithm of [75]) and derive performance bounds in terms of the near-optimality dimension $d = \frac{\log \kappa}{\log 1/\gamma}$ (see (5.8)). The expected loss of HOO is thus of order

$$\mathbb{E} r_n = \tilde{O}(n^{-1/(d+2)}) = \tilde{O}(n^{-\frac{\log 1/\gamma}{\log \kappa + 2 \log 1/\gamma}}). \tag{5.12}$$

Clearly, this rate is always worse than the ones in Theorem 5.2. This is expected since these algorithms do not use the specific structure of the global reward function (which is the sum of rewards obtained along a sequence) to generalize efficiently the estimation of rewards across arms. More precisely, they do not consider the fact that a reward sample observed for an arm (or sequence) ab provides information for the estimation of any other arm in aA^∞. Thus we see that is it crucial to take into account the specific tree structure of the rewards.

Comparison with UCB-AIR: When one knows that there are many near-optimal sequences of actions (i.e. when κ is close to K), then one may be convinced that among a certain number of paths chosen uniformly at random, there exists at least one which is very good with high probability. This idea is exploited by the UCB-AIR algorithm of [108], introduced in Section 1.2.1, designed for infinitely many-armed bandits, where at each round one chooses either to sample a new arm (or sequence in our case) uniformly at random, or to re-sample an arm that has already been explored (using a UCB-like algorithm to choose which one). The regret bound of UCB-AIR is expressed in terms of the probability of selecting an ϵ-optimal sequence when one chooses the actions uniformly at random. More precisely, the important quantity β

is such that this probability is of order of ϵ^{β}. Again, one can see that κ is closely related to β. Indeed, our assumption says that the proportion of *epsilon*-optimal sequences of actions (with $\epsilon = 2\frac{\gamma^{h+1}}{1-\gamma}$) is $O(\kappa^h)$, resulting in $\kappa = K\gamma^{\beta}$. Thanks to this result, we can see that applying UCB-AIR in our setting yields the loss bounded by:

$$\mathbb{E}r_n = \begin{cases} \tilde{O}(n^{-\frac{1}{2}}) & \text{if } \kappa > K\gamma \\ \tilde{O}(n^{-\frac{1}{1+\beta}}) = \tilde{O}(n^{-\frac{\log 1/\gamma}{\log K/\kappa + \log 1/\gamma}}) & \text{if } \kappa \le K\gamma \end{cases}$$

As expected, UCB-AIR is very efficient when there is a large proportion of near-optimal paths. Note also that UCB-AIR requires the knowledge of β (or equivalently κ), whereas OLOP (or HOO/Zooming) does not.

Figure 5.1 shows a comparison of the exponents in the loss bounds for OLOP, uniform planning, UCB-AIR, and HOO (in the case $K\gamma^2 > 1$). We note that the rate for OLOP is better than UCB-AIR when there is a small proportion of near-optimal paths (small κ). Uniform planning is always dominated by OLOP and corresponds to a minimax lower bound for any algorithm. HOO/Zooming are always strictly dominated by OLOP and they do not attain minimax performances.

Open questions are whether or not (1) one can do as well as UCB-AIR (for large κ) when κ is unknown, (2) one can do better than both OLOP and UCB-AIR in intermediate cases (i.e. when $1/\gamma^2 < \kappa < \gamma K$).

Comparison with OPD Remarkably, in the case $\kappa\gamma^2 > 1$, we obtain the same rate for the loss as planning with deterministic rewards. Thus, in this case, we can say that planning with stochastic rewards (under deterministic transitions) is not harder than planning with deterministic rewards.

5.3 Markov decision processes

Now we consider the setting of Markov decision processes where transitions are stochastic. More precisely we denote by $p(y|x,a)$ the probability of a transition from x to y given action a. Here we assume that the number of possible next-states N is finite,

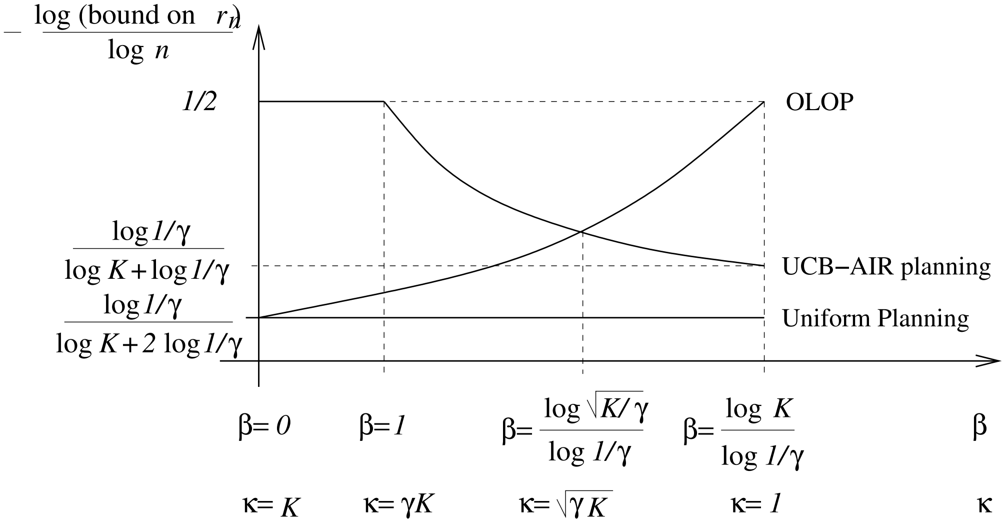

Fig. 5.1 Comparison of the exponent rate of the bounds on the simple regret for OLOP, uniform planning, UCB-AIR, and HOO/Zooming, as a function of $\kappa \in [1, K]$, in the case $K\gamma^2 > 1$.

i.e. $\sup_{x \in X, a \in A} |\{y; p(y|x,a) > 0\}| \overset{\text{def}}{=} N < \infty$. We also assume that the rewards $r(x,a)$ are deterministic and lie in $[0,1]$.

Again we consider a infinite-time horizon problem with discounted rewards. For any policy $\pi : X \to A$ the value function is defined as the expected sum of rewards:

$$V^\pi(x) \overset{\text{def}}{=} \mathbb{E}\big[\sum_{t \geq 0} \gamma^t r(x_t, \pi(x_t))\big],$$

where x_t is the state of the system at time t when starting from x (i.e. $x_0 = x$) and following policy π. We also define the Q-value function $Q^\pi : X \times A \to I\!R$ associated to a policy π, in state-action (x,a), as:

$$Q^\pi(x,a) \overset{\text{def}}{=} r(x,a) + \gamma \sum_y p(y|x,a) V^\pi(y).$$

The optimal value function (respectively Q-value function) is defined as $V^*(x) \overset{\text{def}}{=} \sup_\pi V^\pi(x)$ (respectively $Q^*(x,a) \overset{\text{def}}{=} \sup_\pi Q^\pi(x,a)$), and the Bellman equations can be written

$$V^*(x) = \max_{a \in A} \left[r(x,a) + \gamma \sum_y p(y|x,a) V^*(y) \right]$$

$$Q^*(x,a) = r(x,a) + \gamma \sum_y p(y|x,a) \max_{b \in A} Q^*(y,b).$$

We assume that we possess a full model of the transition probabilities p and the reward function r, which can be used by the planning algorithm. The model takes as input a state x and returns for each action a the reward $r(x,a)$ as well as the N next states y and the corresponding transition probabilities $p(y|x,a)$. An algorithm takes as input an initial state x, and outputs an action $a(n)$ using at most n calls to the generative model. Again the performance is assessed with the loss $r_n(\mathcal{A})$ of choosing $a(n)$ and then following an optimal path instead of following an optimal path from the beginning, as defined in (5.1).

This setting is different from the two previous sections in the fact that the space of policies cannot be identified with the set of infinite sequences of actions anymore, since a policy depends on the actual resulting states and not only on the sequence of actions.

5.3.1 Optimistic Planning in MDP

The Optimistic Planning in MDP (OP-MDP) algorithm [35, 34] works by building incrementally a tree corresponding to the set of states that can be reached from the initial state. Notice that several nodes may correspond to the same state (because there may be different transitions from the root state to a given state). Such duplicates could be merged which would close the tree into a graph; however here we restrict ourselves to a simple version of OP-MDP that ignores duplicates (thus each node corresponds to a unique path to any state).

We use the following notations: $\mathcal{T}$ denotes the infinite planning tree and $\mathcal{T}_n \subset \mathcal{T}$ is the subtree resulting from n node expansions, as illustrated in Figure 5.2 for $n = 4$. $\mathcal{L}_t$ is the set of leaves of $\mathcal{T}_t$. We write x_i the state associated to any node $i \in \mathcal{T}$. For any policy $\pi : \mathcal{T} \mapsto A$ defined over the tree $\mathcal{T}$, we denote by $\mathcal{T}^\pi$ the (infinite) subtree

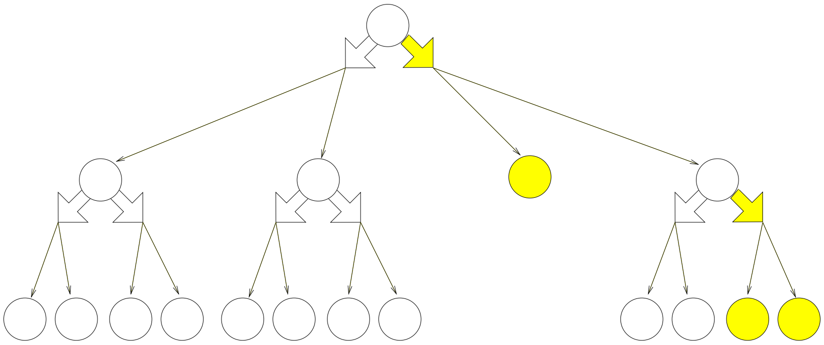

Fig. 5.2 The subtree corresponding to the set of states that can be reached from the initial state. The big arrows represent the actions ($K = 2$) and the thin arrows the transitions to the next states ($N = 2$). Here 4 nodes have been expanded. The optimistic policy and the leaves of the resulting optimistic subtree are represented in yellow.

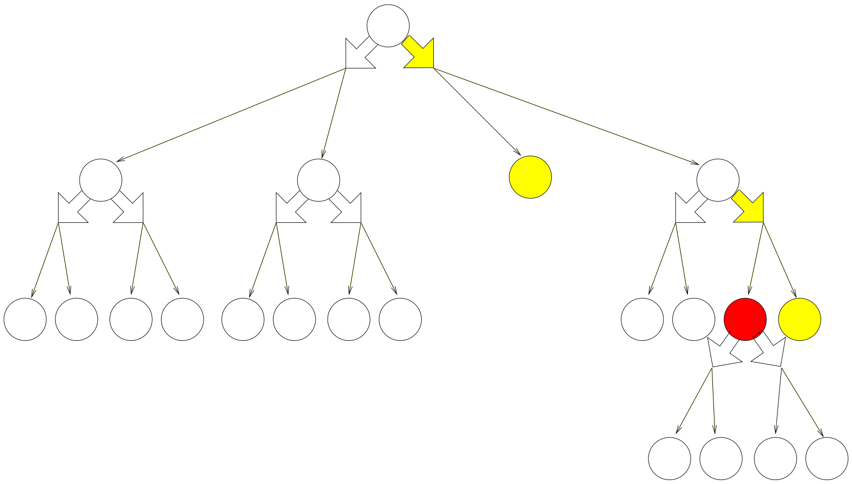

Fig. 5.3 Among the leaves of the current optimistic subtree, the one with largest contribution $p(i)\frac{\gamma^{h(i)}}{1-\gamma}$ is expanded (represented in red): a call to the model returns the rewards and transition probabilities to the next states for each action.

corresponding to the set of nodes that are reachable when following π. For any finite subtree $\mathcal{T}' \subset \mathcal{T}$, we write a policy-class $\Pi : \mathcal{T}' \mapsto A$ as a set of policies $\pi : \mathcal{T} \mapsto A$ that share the same actions on $\mathcal{T}'$. We write $\mathcal{T}^{\Pi}$ the corresponding (finite) subtree.

Algorithm 3 describes OP-MDP. T_0 is initialized to be the root node, and for each $t = 1$ to $n-1$, a leaf J_t of $\mathcal{L}_t$ is selected and expanded, which results in adding KN children nodes (number of actions K times number of next states N) to the current tree. After n node expansions, OP-MDP returns the first action of the current optimal policy.

The way the leaf J_t is selected is by first computing the optimistic policy-class Π_t^+ and then selecting a leaf of the corresponding subtree with largest "contribution". More precisely, at each round t, we define the b-values and u-values of any node of the current tree $\mathcal{T}_t$ as follows: for any leaf $j \in \mathcal{L}_t$, $b_t(j) \overset{\text{def}}{=} \frac{1}{1-\gamma}$ and $u_t(j) \overset{\text{def}}{=} 0$, and for any other node $i \in \mathcal{T}_t \setminus \mathcal{L}_t$ define

$$\begin{aligned} b_t(i) &\overset{\text{def}}{=} \max_{a \in A} \big[r(x_i, a) + \gamma \sum_{j \in \mathcal{C}(i,a)} p(x_j | x_i, a) b_t(j) \big], \\ u_t(i) &\overset{\text{def}}{=} \max_{a \in A} \big[r(x_i, a) + \gamma \sum_{j \in \mathcal{C}(i,a)} p(x_j | x_i, a) u_t(j) \big], \end{aligned}$$

where $\mathcal{C}(i,a)$ denotes the set of children nodes of node i when choosing action a.

By a backward induction starting from the leaves up to the root, we immediately deduce that the b-value (respectively the u-value) of any node $i \in \mathcal{T}_t$ provides an upper-bound (resp. a lower bound) on the optimal value function at the corresponding state: $u_t(i) \leq V^*(x_i) \leq b_t(i)$, for any t.

We define the optimistic policy-class $\Pi_t^+ : \mathcal{T}_t \mapsto A$ as the optimal policy for the b-values for any $i \in \mathcal{T}_t$:

$$\Pi_t^+(i) \in \arg\max_{a \in A} \big[r(x_i, a) + \gamma \sum_{j \in \mathcal{C}(i,a)} p(x_j | x_i, a) b_t(j) \big]. \tag{5.13}$$

We denote by $\mathcal{T}_t^+ = \mathcal{T}^{\pi_t^+}$ the corresponding optimistic subtree of the set of nodes that can be reached when following the optimistic policy, and $\mathcal{L}_t^+$ the leaves of this subtree. Thus for each leaf $j \in \mathcal{L}_t^+$ (of depth $h(j)$) define $p(j)$ as the probability of reaching the leaf j when starting from the root and following policy Π_t^+:

$$p(j) \overset{\text{def}}{=} \prod_{h=0}^{h(j)-1} p(i_{h+1} | i_h, \pi_t^+(i_h)) > 0,$$

Algorithm 3 Optimistic planning in MDP (OP-MDP)

```
Initial state $x_0$, model of $p$ and $r$, budget $n$
Initialize tree: $\mathcal{T}_0 = \{0\}$ (root node is called 0)
for $i = 1, \dots, n-1$ do
    Build optimistic subtree $\mathcal{T}_t^+$ according to (5.13),
    Select leaf $J_t \in \mathcal{L}_t^+$ with largest contribution:
        $$J_t = \arg\max_{j \in \mathcal{L}_t^+} p(j) \frac{\gamma^{h(j)}}{1-\gamma},$$
    Expand $J_t$ (adding $KN$ new leaves)
end for
Return $\arg\max_{a\in A} \big[ r(x_0, a) + \gamma \sum_{j \in \mathcal{C}(0,a)} p(x_j|x_0, a) u_n(j) \big]$.
```

where the $h(j)+1$ nodes ($i_0 \overset{\text{def}}{=} 0, i_1, \dots, i_{h(j)} \overset{\text{def}}{=} j$) is the path from the root to j. Notice that we have $\sum_{j \in \mathcal{L}_t^+} p(j) = 1$.

We call *contribution of a leaf* $j \in \mathcal{L}_t^+$ the quantity $c(j) \overset{\text{def}}{=} p(j) \frac{\gamma^{h(j)}}{1-\gamma}$. OP-MDP selects the leaf of the optimistic subtree with largest contribution: $J_t \in \arg\max_{j \in \mathcal{L}_t^+} c(j)$.

The intuition for that choice is that the diameter (difference between the upper and lower bounds) at the root is the sum of contributions of the leaves $j \in \mathcal{L}_t^+$: $b_t(0) - u_t(0) = \sum_{j \in \mathcal{L}_t^+} c(j)$. Thus expanding the one with largest contribution reduces as much as possible the diameter at the root, thus the accuracy of the value function at the initial state.

5.3.2 Analysis of OP-MDP

For any two policies $\pi, \pi' : \mathcal{T} \mapsto A$, define $\mathcal{T}(\pi, \pi') = \mathcal{T}^\pi \cap \mathcal{T}^{\pi'}$ the set of their common nodes, and $\mathcal{L}(\pi, \pi')$ the set of leaves of $\mathcal{T}(\pi, \pi')$ (with the convention that $\mathcal{L}(\pi, \pi') = \emptyset$ if $\mathcal{T}^\pi = \mathcal{T}^{\pi'}$). Define $\ell(\pi, \pi') \overset{\text{def}}{=} \sum_{j \in \mathcal{L}(\pi, \pi')} c(j)$ the sum of the contributions of $\mathcal{L}(\pi, \pi')$. We have the property that the value function, defined for any $\pi : \mathcal{T} \mapsto A$, as

$$v(\pi) \overset{\text{def}}{=} \sum_{i \in \mathcal{T}^\pi} p(i) \gamma^{h(i)} r(x_i, \pi(x, i)),$$

is Lipschitz w.r.t. ℓ:

$$|v(\pi) - v(\pi')| \leq \ell(\pi, \pi'). \tag{5.14}$$

For any policy-class $\Pi : \mathcal{T} \mapsto A$, define the diameter of Π as

$$\mathrm{diam}(\Pi) \overset{\mathrm{def}}{=} \sup_{\pi,\pi' \in \Pi} \ell(\pi, \pi').$$

Note that from the definition of the contributions, we have that $\mathrm{diam}(\Pi) = \sum_{j \in \mathcal{L}(\Pi)} c(j)$.

Thus one can see OP-MDP as a deterministic optimistic optimization algorithm (see DOO in Chapter 3.3) where at each round t:

- the search space $\mathcal{T}$ is partitioned into policy-classes defined by the current subtree $\mathcal{T}_t$
- an upper bound on each policy-class can be computed with the b-values and the optimistic policy-class Π_t^+ is the one with largest upper-bound
- the diameter of the policy-class Π_t^+ is the sum of contributions of its leaves L_t^+, thus expanding the leaf $J_t \in \mathcal{L}_t^+$ with largest contribution $c(j)$ "splits" the optimistic policy class along its "largest" dimension.

Now the main difference is that we are not directly working on the set of policies but on the set of nodes of the tree (which is no more equivalent). Thus expanding a node has an impact on all the policies containing that node. Thus in order to analyze this algorithm we should not try to characterize the quantity of near-optimal policies, but instead the quantity of nodes that contribute to near-optimal policies.

For any node $i \in \mathcal{T}$, let Π_i be the policy-class $\Pi \ni i$ such that $\min_{j \in \mathcal{L}(\Pi)} c(j) \geq c(i)$ and that has the largest diameter:

$$\Pi_i = \arg \max_{\Pi \ni i; \min_{j \in \mathcal{L}(\Pi)} c(j) \geq c(i)} \mathrm{diam}(\Pi).$$

Finally for any $\epsilon > 0$, define

$$S_\epsilon \overset{\mathrm{def}}{=} \big\{ i \in \mathcal{T}, \mathrm{diam}(\Pi_i) \geq \epsilon, \text{ and } \exists \Pi \ni i, v(\Pi) \geq v^* - \mathrm{diam}(\Pi_i) \big\}.$$

The set S_ϵ represents the set of nodes that (1) belong to a policy-class Π_i with non-negligible diameter and (2) belong to a policy that is diam(Π_i)-optimal. In other words, those are the set of nodes that contribute in a significant way to near-optimal policies.

The paper [34] uses a slightly different definition of S_ϵ (taking into account the number of leaves of Π_i) but the main results stated next are immediate consequences of the analysis undertaken in that paper.

Theorem 5.3. Let $d \geq 0$ be any constant such that $|S_\epsilon| = \tilde{O}(\epsilon^{-d})$, i.e. such that there exists $a, b > 0$, for all $\epsilon > 0$,

$$|S_\epsilon| \leq a(\log(1/\epsilon))^b \epsilon^{-d}. \tag{5.15}$$

Then the loss of OP-MDP after n node expansions, is

$$r_n = \begin{cases} \tilde{O}(n^{-\frac{1}{d}}) & \text{if } d > 0 \\ O(\exp[-(\frac{n}{a})^{\frac{1}{b}}]) & \text{if } d = 0 \end{cases}$$

The full proof of this result can be found in [34]. We now provide a sketch of proof and relate this *near-optimality planning exponent* d to the branching factor $\kappa \in [1, KN]$ of the set of near-optimal nodes, like in previous sections with (5.7) and (5.11).

Define the set of near-optimal nodes $\mathcal{T}^+ \subset \mathcal{T}$:

$$\mathcal{T}^+ \stackrel{\text{def}}{=} \big\{ i \in \mathcal{T}, v(i) \leq v^* - \text{diam}(\Pi_i) \big\},$$

where the value of a node $v(i)$ is the value of the best possible policy containing that node $v(i) \stackrel{\text{def}}{=} \max_{\pi, T^\pi \ni i} v(\pi)$. Then the near-optimality exponent d is related to the branching factor κ of $\mathcal{T}^+$ by $d = \frac{\log \kappa}{\log 1/\gamma}$.

And like for the OPD, the set of near-optimal nodes represents the set of nodes that deserve to be expanded in order to discover the optimal policy. Similarly to OPD, the main intuition for the analysis of OP-MDP is that this algorithms only expands nodes in $\mathcal{T}^+$. Indeed, if at time t, a node J_t is expanded, this means that its contribution is larger than that of any other leaf in $\mathcal{L}_t^+$. Thus $\text{diam}(\Pi_t^+) = \sum_{i \in \mathcal{L}_t^+} c(j) \leq \sum_{j \in \Pi_(J_t)} c(j) = \text{diam}(\Pi_{J_t})$ (by definition of Π_{J_t}). Now since Π_t^+ is the optimistic policy-class, it means that its upper-bound $v(\Pi_t^+) + \text{diam}(\Pi_t^+)$ is larger than v^*. Thus

$$v(J_t) \geq v(\Pi_t^+) \geq v^* - \text{diam}(\Pi_t^+) \geq v^* - \text{diam}(\Pi_{J_t}),$$

which means that $J_t \in \mathcal{T}^+$.

5.3.3 Interesting values of d

The loss is small when d is small (and we obtain exponential rate when $d = 0$), or equivalently when the branching factor κ is close to 1.

Uniform rewards and probabilities The worst possible rate is achieved for $\kappa = KN$ (i.e. the branching factor of $\mathcal{T}^+$ is the same as that of $\mathcal{T}$) and in this case the loss is $r_n = n^{-\frac{\log(KN)}{\log 1/\gamma}}$. This happens when all policies provide the same rewards and the transition probabilities are uniform. In that case OP-MDP reduces to a uniform search, where all nodes of depth up to $\frac{\log n}{\log(KN)}$ are expanded. It may seem surprising that the performance is poor when the problem seems easy, but we should keep in mind that one usually does not know in advance what the difficulty of the problem is (i.e. d or κ are not known by the algorithm although the performance of OP-MDP is expressed in terms of those parameters). If this measure of difficulty of the problem were known, one could design algorithms that would exploit it, like the UCB-AIR algorithm presented in Chapter 1 and discussed in previous Section.

Now, for any n, consider the class of problems where all rewards up to depth $\frac{\log n}{\log(KN)}$ are the same but differ from that depth on. Thus no algorithm can be uniformly better than a uniform planning algorithm on this class of problems. Thus OP-MDP is minimax-optimal on the class of problems characterized by $\kappa = KN$.

Heterogeneous probabilities When the transition probabilities are significantly heterogeneous, the part of the branching factor of $\mathcal{T}^+$ due to the number of next states may be significantly less than N. Indeed, the set of "near-optimal" states $\mathcal{T}^+$ contains states i whose contribution $c(i)$ is sufficiently significant to a near-optimality policy containing i. Thus if the transition probabilities to most of the next states is very small, the corresponding nodes will not be part of $\mathcal{T}^+$. And in cases when the transition probabilities to one next state tends to 1, then this branching factor approaches 1, and the performance of OP-MDP is as good as Optimistic Planning in Deterministic systems (Section 5.1).

Structured rewards In the case of structured rewards (i.e. the rewards along branches corresponding to different actions are heterogeneous), then the part of the branching factor of $\mathcal{T}^+$ due to the number of actions may be significantly less than K. This case was already illustrated in Section 5.1.

Now when the problem has both structured rewards and heterogeneous transition probabilities, then κ can be much less than KN and even close to 1, which provides a loss bound of order $n^{-\frac{\log 1/\gamma}{\log \kappa}}$. Thus like previous optimistic algorithms, the performance of OP-MDP depends on a measure of the quantity of near-optimal nodes, defined by the fact that those are the set of nodes that need to be expanded in order to build a near-optimal policy. The main contribution of this chapter is to show that the right measure of complexity for optimistic planning is defined by $\mathcal{T}^+$ which represents **the set of states that significantly contribute to near-optimal policies**.

5.4 Conclusions and extensions

Generative model OP-MDP requires a full model of the transition dynamics (i.e., for each state-action pair (x, a), a call to the model returns the set of next states y and the exact values of the transition probabilities $p(y|x, a)$). In many situations, only a *generative model* is available: Given (x, a), each call to the model returns a single next state y drawn from the true (but unknown) transition probabilities: $y \sim p(\cdot|x, a)$. This is the case when an agent interacts online with an unknown environment (such as in Reinforcement learning, see [101]) from which he only observes trajectories, or when one uses Monte-Carlo simulations to numerically approximate heavy computations. Thus it would be useful to extend OP-MDP to situations where only a generative model of the transition dynamics (and rewards) is available. Also we would like to cover the case of potentially infinite number of next states (like in [74]) by using a branching factor N (number of next states) that would depend on the node characteristics (such as its contribution) and the numerical budget n. Designing a sound (i.e. enjoying finite-time performance guarantee) optimistic planning algorithm using a generative model is still an open problem. We conjecture that loss bounds in this setting would scale as $O(n^{-1/(d+2)})$ (where d is defined similarly to (5.15)). This research direction is left for future works.

Extensions to POMDPs In a partially observable Markov decision process (POMDP) the state of the system x_t cannot be observed by the agent (see e.g. [68, 18]). However, in each time t, the agent receives an observation y_t determined by observation probabilities $p(y_t|x_t, a_t)$. In a POMDP, the best policy (which maximizes the expected rewards given the uncertainty over the state) can be obtained as a function of the belief state b_t (which is a distribution over the state space X). The literature on the topic is huge and online planning techniques have been developed, such as the point-based value iteration [89, 93]. This method builds a search tree of belief states, using a heuristic best-first expansion procedure which may be combined with branch-and-bound procedure based on computations of upper and lower bounds on the value function. However no theoretical guarantee on the quality of the resulting action in terms of the numerical budget was provided.

Using the work described in the previous chapter one can use OP-MDP to perform the planning. The initial state is the current belief state. The fact that the belief space is large (infinite) is not a problem for this online planning technique. Now, the nodes of the tree that are expanded are the belief states that can be reached from the initial belief given a sequence of actions and transitions (the number of next states is the number of different observations). Thus OP-MDP is an online planning technique with theoretical guarantees that may be advantageously applied here.

In the case a full model of the POMDP is unknown, one can use sampling-based techniques such as the technique (based on UCT) described in [98]. Unfortunately this method does not have finite-time guarantee (since UCT can be arbitrarily poor in some situations, see Section 2.3). This provides an additional motivation for extending the OP-MDP to situations where only a generative model is available.

Bayesian RL In Bayesian Reinforcement learning (see e.g. [48, 106]) some parameters of the Markov decision process are initially unknown and exploration can be performed by using a Bayesian reasoning where one starts with a prior over the unknown parameters and based on the transition and reward samples observed at any time t, a posterior distribution over those parameters can be computed (either in a closed form or using numerical approximation). The so-called Bayesian-adaptive MDP (BAMDP) is a new MDP that enriches the state by the current posterior distribution over the parameters. The interesting property of the BAMDP is that the dynamics are known thus following the optimal action of the BAMDP from the current state provides a good exploration-exploitation strategy (optimal in a Bayesian sense) [48]. The planning problem (of solving the BAMDP) can be addressed using sampling techniques similar to the ones for MDPs of [74], see [107]. Monte-Carlo tree search approaches have been developed also recently, such as in [8, 60]. However, no finite-time guarantees are provided in those works. Now, since the dynamics of the BAMDP are known one could use the OP-MDP planning technique described above to derive loss bounds in terms of the numerical budget allocated to solving the BAMDP (the branching factor of the BAMDP planning tree is the same as in the original MDP, i.e. $A \times N$).

Finally, let us mention the harder problem of solving a POMDP when the parameters of the dynamics or observation function are unknown. An analogous Bayesian approach introduces the Bayesian-Adaptive POMDP (BAPOMDP) [94] and an optimal policy in the BAPOMDP provides a Bayes-optimal exploration in the POMDP. However the planning problem of the BAPOMDP is more challenging because the branching factor now scales with the number of states of the original POMDP (see [94]). Again extending the OP-MDP to handle a possible infinite number of next-states using sampling from a generative model would contribute to the problem.

Final conclusion

The main message of this work is to show that the "optimism in the face of uncertainty" is a simple yet powerful principle that enables to guide the exploration for general learning and optimization problems. It applies when some unknown environment has to be explored while some criterion needs to be optimized.

In the multi-armed bandit problem, an unknown environment (set of arms with unknown distributions) has to be explored while maximizing the sum of rewards. In function optimization under finite numerical budget (e.g. number of function evaluations), the exploration of the space should be optimized in order to return the best possible recommendation of the maximum once the numerical resources are depleted. In both situations, the performance (either in terms of cumulative regret or in terms of loss of the final recommendation) depends on some **measure of complexity of the problem, which expresses how close sub-optimal solutions are to the optimum.**

In multi-armed bandits, the complexity measure is the inverse of the "distance" (i.e. in terms of mean or in Kullback-Leibler divergence) between the distributions of sub-optimal and optimal arms.

In function optimization, the complexity measure is expressed with the quantity of near-optimal solutions (i.e. the near-optimality dimension) measured according to some semi-metric. Another important factor is **our knowledge about the local smoothness of the function around the global optimum**. If this information is known, then it can be used to build efficient algorithms with performance rate independent of the search space dimension. If this is not the case, then one can still build adaptive strategies that can, in some situations, perform almost as well as if this information was known.

Finally we have seen an application to the problem of online-planning which illustrates the benefit of using the specific structure of the problem (rewards, transitions) to design efficient algorithms. In such situations we showed that a relevant complexity measure for the problem of online planning in a MDP is the quantity of states that significantly contribute to the set of near-optimal policies.

References

[1] The computer-go program mogo. `http://www.lri.fr/∼teytaud/mogo.html`.

[2] Y. Abbasi-Yadkori, D. Pal, and Cs. Szepesvári. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems*, 2011.

[3] Y. Abbasi-Yadkori, D. Pal, and Cs. Szepesvári. Online-to-confidence-set conversions and application to sparse stochastic bandits. In *Artificial Intelligence and Statistics*, 2012.

[4] Bruce Abramson. Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:182–193, 1990.

[5] A. Agarwal, D. Foster, D. Hsu, S. M. Kakade, and A. Rakhlin. Stochastic convex optimization with bandit feedback. In *Advances in Neural Information Processing Systems*, 2011.

[6] R. Agrawal. The continuum-armed bandit problem. *SIAM Journal on Control and Optimization*, 33:1926–1951, 1995.

[7] S Agrawal and N. Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, 2012.

[8] John Asmuth and Michael L. Littman. Learning is planning: near bayes-optimal reinforcement learning via monte-carlo tree search. In *Uncertainty in Artificial Intelligence*, 2011.

[9] J.-Y. Audibert, S. Bubeck, and R. Munos. Best arm identification in multi-armed bandits. In *Conference on Learning Theory*, 2010.

[10] J.-Y. Audibert, R. Munos, and Cs. Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410:1876–1902, 2009.

[11] Jean-Yves Audibert and Sébastien Bubeck. Minimax policies for adversarial and stochastic bandits. In Sanjot Dasgupta and Adam Klivans, editors, *Proceedings of the 22nd annual Conference On Learning Theory*, COLT '09, Montreal,Quebec, Canada, jun 2009.

[12] P. Auer, R. Ortner, and C. Szepesvári. Improved rates for the stochastic continuum-armed bandit problem. *20th Conference on Learning Theory*, pages 454–468, 2007.

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

[14] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32:48–77, January 2003.

[15] A. Auger and N. Hansen. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, chapter Theory of Evolution Strategies: A New Perspective, pages 289–325. World Scientific Publishing, 2011.

[16] Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal*, 19:357–367, 1967.

[17] J. S. Banks and R. Sundaram. Denumerable-armed bandits. *Econometrica*, 60:1071–1096, 1992.

[18] Nicole Bäuerle and Ulrich Rieder. *Markov Decision Processes with Applications to Finance.* 2011.

[19] D. A. Berry, R. W. Chen, A. Zame, D. C. Heath, and L. A. Shepp. Bandit problems with infinitely many arms. *Annals of Statistics*, (25):2103–2116, 1997.

[20] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[21] Amine Bourki, Guillaume Chaslot, Matthieu Coulm, Vincent Danjean, Hassen Doghmen, Jean-Baptiste Hoock, Thomas Hérault, Arpad Rimmel, Fabien Teytaud, Olivier Teytaud, Paul Vayssière, and Ziqin Yu. Scalability and parallelization of monte-carlo tree search. In *International Conference on Computers and Games*, 2012.

[22] B. Bouzy and B. Helmstetter. Monte carlo go developments. In Hiroyuki Iida Ernst A. Heinz H. Jaap van den Herik, editor, *Advances in Computer Games*, page 159174. Kluwer Academic Publishers, 2003.

[23] Bruno Bouzy and Tristan Cazenave. Computer Go: an AI oriented survey. *Artif. Intell.*, 132(1):39–103, October 2001.

[24] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon 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 AIin Games*, 4(1), March 2012.

[25] B. Brügmann. Monte carlo go. Technical report, Syracuse University, NY, USA, 1993.

[26] S. Bubeck and R. Munos. Open loop optimistic planning. In *Conference on Learning Theory*, 2010.

[27] S. Bubeck, R. Munos, and G. Stoltz. Pure exploration in multi-armed bandits problems. In *Proc. of the 20th International Conference on Algorithmic Learning Theory*, 2009.

[28] S. Bubeck, R. Munos, G. Stoltz, and Cs. Szepesvári. Online optimization of X-armed bandits. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 22, pages 201–208. MIT Press, 2008.

[29] S. Bubeck, R. Munos, G. Stoltz, and Cs. Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

[30] Sébastien Bubeck. *Bandits Games and Clustering Foundations.* PhD thesis, Université de Lille 1, 2010.

[31] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[32] L. Buşoniu, R. Munos, and R. Babuska. Optimistic planning in Markov decision processes. In Frank Lewis and Derong Liu, editors, *In Reinforcement Learning and Adaptive Dynamic Programming for feedback control.* Wiley, 2011.

[33] Lucian Buşoniu, Robert Babuška, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators.* Automation and Control Engineering. Taylor & Francis CRC Press, 2010.

[34] Lucian Buşoniu and Rémi Munos. Optimistic planning for markov decision processes. In *Proceedings 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, page 182189, 2012.

[35] Lucian Buşoniu, Rémi Munos, Bart De Schutter, and Robert Babuška. Optimistic planning for sparsely stochastic systems. In *2011 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-11)*, Paris, France, 11–15 April 2011. Submitted to special session on *Active Reinforcement Learning.*

[36] A.N. Burnetas and M.N. Katehakis. Optimal adaptive policies for sequential allocation problems. *Advances in Applied Mathematics*, 17:122–142, 1996.

[37] Apostolos N. Burnetas and Michaël N. Katehakis. Optimal adaptive policies for sequential allocation problems. *Advances in Applied Mathematics*, 17(2):122–142, 1996.

[38] E. F. Camacho and C. Bordons. *Model Predictive Control.* Springer-Verlag, 2004.

[39] Olivier Cappé, Aurélien Garivier, Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. Kullback-leibler upper confidence bounds for optimal sequential allocation. Technical report, hal-00738209, 2012.

[40] Alexandra Carpentier and Rémi Munos. Theory meets compressed sensing for high dimensional stochastic linear bandit. In *International Conference on Artificial Intelligence and Statistics*, 2012.

[41] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games.* Cambridge University Press, New York, NY, USA, 2006.

[42] Hyeong Soo Chang, Michael C. Fu, Jiaqiao Hu, and Steven I. Marcus. *Simulation-based Algorithms for Markov Decision Processes*. Springer, London, 2007.

[43] Guillaume Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Maastricht University, 2010.

[44] P.-A. Coquelin and R. Munos. Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*, 2007.

[45] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In LNCS, editor, *Computer Games*, volume 4630, pages 72–83, 2006.

[46] Varsha Dani, Thomas P. Hayes, and Sham M. Kakade. Stochastic linear optimization under bandit feedback. In Rocco A. Servedio and Tong Zhang, editors, *Proceedings of the 21st annual Conference On Learning Theory*, volume 80 of *COLT '08*, pages 355–366, Helsinki, Finland, jul 2008. Omnipress.

[47] Boris Defourny, Damien Ernst, and Louis Wehenkel. Lazy planning under uncertainties by optimizing decisions on an ensemble of incomplete disturbance trees. In S. Girgin, M. Loth, R. Munos, P. Preux, and D. Ryabko, editors, *Recent Advances in Reinforcement Learning*, volume 5323 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2008.

[48] Michael Duff. *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, 2002.

[49] Sarah Filippi, Olivier Cappe, Aurélien Garivier, and Csaba Szepesvari. Parametric bandits: The generalized linear case. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 586–594. 2010.

[50] D. E. Finkel and C. T. Kelley. Convergence analysis of the direct algorithm. Technical report, North Carolina State University, Center for, 2004.

[51] Abraham D. Flaxman, Adam Tauman Kalai, and Hugh Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the 16th annual ACM-SIAM Symposium On Discrete Algorithms*, SODA '05, pages 385–394. SIAM, 2005.

[52] C.A. Floudas. *Deterministic Global Optimization: Theory, Algorithms and Applications*. Kluwer Academic Publishers, Dordrecht / Boston / London, 1999.

[53] J. M. X. Gablonsky. *Modifications of the direct algorithm*. PhD thesis, 2001.

[54] Aurélien Garivier and Olivier Cappé. The KL-UCB algorithm for bounded stochastic bandits and beyond. In *Proceedings of the 24th annual Conference On Learning Theory*, COLT '11, 2011.

[55] S. Gelly, Y. Wang, R. Munos, and O. Teytaud. Modification of UCT with patterns in monte-carlo go. Technical report, INRIA RR-6062, 2006.

[56] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In Zoubin Ghahramani, editor, *International Conference on Machine Learning*, volume 227 of *ICML '07, ACM International Conference Proceeding Series*, pages 273–280, Corvalis, Oregon, USA, jun 2007. ACM.

[57] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175:1856–1875, 2011.

[58] J.C. Gittins. Bandit processes and dynamic allocation indices. *In Journal of the Royal Statistical Society Series B*, 41(2):148–177, 1979.

[59] John C. Gittins, Richard Weber, and Kevin Glazebrook. *Multi-armed Bandit Allocation Indices.* Wiley, 1989.

[60] Arthur Guez, David Silver, and Peter Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems*, 2012.

[61] E.R. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, New York, 1992.

[62] Eric A. Hansen and Shlomo Zilberstein. A heuristic search algorithm for Markov decision problems. In *Proceedings Bar-Ilan Symposium on the Foundation of Artificial Intelligence*, Ramat Gan, Israel, 23–25 June 1999.

[63] J. Honda and A. Takemura. An asymptotically optimal policy for finite support models in the multiarmed bandit problem. *Machine Learning*, 85:361–391, 2011.

[64] Junya Honda and Akimichi Takemura. An asymptotically optimal bandit algorithm for bounded support models. In Adam Tauman Kalai and Mehryar Mohri, editors, *Proceedings of the 23rd annual Conference On Learning Theory*, pages 67–79. Omnipress, June 2010.

[65] R. Horst and H. Tuy. *Global Optimization ? Deterministic Approaches.* Springer, Berlin / Heidelberg / New York, 3rd edition, 1996.

[66] J-F. Hren and R. Munos. Optimistic planning of deterministic systems. In European Workshop on Reinforcement Learning Springer LNAI 5323, editor, *Recent Advances in Reinforcement Learning*, pages 151–164, 2008.

[67] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[68] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[69] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71:291–307, 2005.

[70] Emilie Kauffman, Olivier Cappé, and Aurélien Garivier. On bayesian upper confidence bounds for bandit problems. In *International Conference on Artificial Intelligence and Statistics*, 2012.

[71] Emilie Kauffmann, Nathaniel Korda, and Rémi Munos. Thompson sampling: An asymptotically optimal finite time analysis. In *International Conference on Algorithmic Learning Theory*, 2012.

[72] R. B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, Dordrecht / Boston / London, 1996.

[73] M. Kearns, Y. Mansour, and A.Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markovian decision processes. In *Machine Learning*, volume 49, pages 193–208, 2002.

[74] Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[75] R. Kleinberg, A. Slivkins, and E. Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the 40th ACM Symposium on Theory of Computing*, 2008.

[76] Robert D. Kleinberg. Nearly tight bounds for the continuum-armed bandit problem. In *Proceedings of the 18th conference on advances in Neural Information Processing Systems*, NIPS '04, Vancouver, British Columbia, Canada, dec 2004. MIT Press.

[77] Robert D. Kleinberg, Alexander Slivkins, and Eli Upfal. Multi-armed bandit problems in metric spaces. In *Proceedings of the 40th ACM symposium on Theory Of Computing*, TOC '08, pages 681–690, 2008.

[78] L. Kocsis and Cs. Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML-2006)*, pages 282–293. 2006.

[79] Jussi Kujala and Tapio Elomaa. Following the perturbed leader to gamble at multi-armed bandits. In *International Conference on Algorithmic Learning Theory*, 2007.

[80] Steven M. La Valle. *Planning Algorithms*. Cambridge University Press, 2006.

[81] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.

[82] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The computational intelligence of mogo revealed in taiwan's computer go tournaments. *IEEE Trans. Comput. Intellig. and AI in Games*, 1(1):73–89, 2009.

[83] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[84] Odalric Ambrym Maillard. *Apprentissage séquentiel: Bandits, Statistique et Renforcement*. PhD thesis, Université des Sciences et des Technologies de Lille 1, 2011.

[85] Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. Finite-time analysis of multi-armed bandits problems with kullback-leibler divergences. In *To appear in Proceedings of the 24th annual Conference On Learning Theory*, COLT '11, 2011.

[86] Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[87] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing, 1980.

[88] L. Péret and F. Garcia. On-line search for solving large Markov decision processes. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 2004.

[89] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27:335–380, 2006.

[90] J.D. Pintér. *Global Optimization in Action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications)*. Kluwer Academic Publishers, 1996.

[91] Arpad Rimmel, Fabien Teytaud, and Olivier Teytaud. Biasing monte-carlo simulations through rave values. In *International Conference on Computers and Games*, 2010.

[92] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematics Society*, 58:527–535, 1952.

[93] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[94] Stéphane Ross, Jeoelle Pineau, Brahim Chaib-draa, and Pierre Kreitmann. A bayesian approach for learning and planning in partially observable markov decision processes. *Journal of Machine Learning Research*, 12:1655–1696, 2011.

[95] Paat Rusmevichientong and John N. Tsitsiklis. Linearly parameterized bandits. *Math. Oper. Res.*, 35:395–411, May 2010.

[96] Olivier Sigaud and Olivier Buffet, editors. *Markov Decision Processes in Artificial Intelligence.* Wiley, 2010.

[97] David Silver. *Reinforcement Learning and Simulation-Based Search in Computer Go.* PhD thesis, University of Alberta, 2009.

[98] David Silver and Paul Veness. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, 2012.

[99] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning*, pages 1015–1022, 2010.

[100] R.G. Strongin and Ya.D. Sergeyev. *Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms.* Kluwer Academic Publishers, Dordrecht / Boston / London, 2000.

[101] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[102] Csaba Szepesvári. *Algorithms for Reinforcement Learning.* Morgan & Claypool Publishers, 2010.

[103] William R. Thompson. On the theory of apportionment. *American Journal of Mathematics*, 57:450–456, 1935.

[104] W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285–294, 1933.

[105] Michal Valko, Alexandra Carpentier, and Rémi Munos. Stochastic simultaneous optimistic optimization. In *International Conference on Machine Learning*, 2013.

[106] Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart. *Reinforcement Learning: State of the Art*, chapter Bayesian Reinforcement Learning. Springer Verlag, 2012.

[107] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *International Conference on Machine Learning*, 2005.

[108] Yizao Wang, Jean-Yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Proceedings of the 22nd conference on advances in Neural Information Processing Systems*, NIPS '08, pages 1729–1736, Vancouver, British Columbia, Canada, dec 2008. MIT Press.

[109] Yizao Wang and Sylvain Gelly. Modifications of uct and sequence-like simulations for monte-carlo go. In *IEEE Symposium on Computational Intelligence and Games,*, pages 175–182, 2007.

Acknowledgements

I would like to thank all my students and colleagues who worked with me on the topics presented in this paper, including (by alphabetic order) Jean-Yves Audibert, Sébastien Bubeck, Lucian Buşoniu, Alexandra Carpentier, Pierre-Arnaud Coquelin, Rémi Coulom, Sylvain Gelly, Jean-François Hren, Nathaniel Korda, Odalric-Ambrym Maillard, Amir Sani, Marta Soare, Gilles Stoltz, Csaba Szepesvári, Olivier Teytaud, Michal Valko, and Yizao Wang.

This work was supported by French National Research Agency (ANR) through the project EXPLO-RA $n°$ ANR-08-COSI-004 and by European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement $n°$ 270327.