CHAPTER 22

OPTIMISTIC PLANNING IN MARKOV DECISION PROCESSES

LUCIAN BUŞONIU[1], RÉMI MUNOS[1], AND ROBERT BABUŠKA[2]

[1]Team SequeL, INRIA Lille-Nord Europe, France
[2]Delft Center for Systems and Control, Delft University of Technology, the Netherlands

Abstract

We review a class of online planning algorithms for deterministic and stochastic optimal control problems, modeled as Markov decision processes. At each discrete time step, these algorithms maximize the predicted value of planning policies from the current state, and apply the first action of the best policy found. An overall receding-horizon algorithm results, which can also be seen as a type of model-predictive control. The space of planning policies is explored optimistically, focusing on areas with largest upper bounds on the value – or upper confidence bounds, in the stochastic case. The resulting *optimistic planning* framework integrates several types of optimism previously used in planning, optimization, and reinforcement learning, in order to obtain several intuitive algorithms with good performance guarantees. We describe in detail three recent such algorithms, outline the theoretical guarantees on their performance, and illustrate their behavior in a numerical example.

Optimistic planning in Markov decision processes. By Buşoniu, Munos, and Babuška

22.1 INTRODUCTION

This chapter considers online algorithms for problems in which a nonlinear, possibly stochastic dynamic system must be optimally controlled in discrete time. Optimality is measured by a cumulative reward signal which must be maximized: the return. Such problems arise in many fields, including artificial intelligence, automatic control, computer science, operations research, economics, medicine, etc. They are often modeled as Markov decision processes (MDPs).

In an MDP, the system is described by a state signal x varying in the state space X, and can be influenced by actions u in the action space U. For the simplicity of notation, X will be considered countable here, but uncountable (e.g. continuous) spaces can easily be handled. Most of the algorithms that will be described in this chapter require that U consists of a small number of discrete actions. The probability that next state x' is reached after action u is taken in state x is $f(x, u, x')$, where $f : X \times U \times X \to [0, 1]$ is a transition probability function describing the dynamics of the system. After the transition to x', a reward $r' = \rho(x, u, x')$ is received, where $\rho : X \times U \times X \to \mathbb{R}$ is the reward function. A control policy $\pi : X \to U$ indicates how the controller should choose actions to interact with the system. Denoting by k the discrete time index, the expected infinite-horizon discounted return of state x under a policy π is:

$$V^\pi(x) = \mathrm{E}_{x_{k+1} \sim f(x_k, \pi(x_k), \cdot)} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \right\} \tag{22.1}$$

where $x_0 = x$, $r_{k+1} = \rho(x_k, \pi(x_k), x_{k+1})$, $\gamma \in (0, 1)$ is the discount factor, and the notation $x_{k+1} \sim f(x_k, \pi(x_k), \cdot)$ means that x_{k+1} is drawn from the distribution $f(x_k, \pi(x_k), \cdot)$ over next states.[1] Other types of return can also be used, such as finite-horizon or averaged over time. We call $V^\pi : X \to \mathbb{R}$ a value function. The goal is to control the system using an optimal policy π^*, so that the value function is maximized for every $x \in X$. This maximal, optimal value function is denoted by V^* and is unique, so it does not depend on the particular optimal policy. It is also useful to consider an action-dependent optimal value function, the optimal Q-function: $Q^*(x, u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \{\rho(x, u, x') + \gamma V^*(x')\}$. Note that optimal control problems are often stated so that a cost is minimized, rather than a return being maximized – but the two formulations are equivalent.

We consider a class of online model-based algorithms that, at each step of interaction, look at the current system state x_k and employ the model to predict the system's response to various *planning policies* – which may be e.g. simple open-loop sequences of actions or more complex, closed-loop selection rules. Note that a planning policy is a different object from the interaction policy π. Exploiting the predictions, an action u_k that is as good as possible is applied, which results in a new state x_{k+1}. The entire cycle then repeats, in a receding-horizon fashion. In systems

[1] Note the "prime" notation, as in e.g. x', is used to generically indicate variables at the next step, whereas if the actual time step is important, it is included as a subscript, e.g. x_{k+1}.

and control, such algorithms are known as model-predictive control [16], while in computer science they are called planning [14]. In this chapter, we mainly draw inspiration from the computer science literature, and to emphasize that we mean the online, feedback control over trajectories consisting of multiple (perhaps infinitely many) steps, we call the algorithms online planning.

The quality of action u_k returned at step k by the planning algorithm is measured by the simple regret, defined as follows:

$$\mathcal{R}(x_k) = \max_{u \in U} Q^*(x_k, u) - Q^*(x_k, u_k) \tag{22.2}$$

that is, the loss incurred by choosing u_k and then acting optimally, with respect to acting optimally immediately, from step k. Note the regret is always nonnegative, and an optimal action choice attains a regret of 0. Theoretical guarantees for planning algorithms are usually given in terms of the regret at any state, independently of k. Then, if an algorithm achieves a regret of ε for every state, the overall suboptimality in terms of discounted return is at most $\frac{\varepsilon}{1-\gamma}$ [11]. This property motivates the use of the simple regret as a performance measure.

The online planning approach is very different from the standard methods for solving MDPs considered in dynamic programming and reinforcement learning. 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. It is generally suboptimal, but useful bounds can be placed on its suboptimality, as will be outlined later in the chapter. While global methods do achieve optimality in some restricted settings, in realistic problems they must also use approximation, thereby sacrificing optimality, as seen elsewhere in this book and specifically reviewed in [4].

We focus on a class of online planning algorithms based on the principle of optimism in the face of uncertainty: given the choice between planning policies having uncertain values, more promising policies are explored first. This optimistic principle has applications in many fields, among which those closest to our focus are optimization, where a representative optimistic method is branch-and-bound; classical planning, with algorithms such as A* [14]; and the exploration-exploitation dilemma in reinforcement learning [20]. In classical planning, optimism relies on *deterministic* upper bounds on the values of incomplete action sequences, whereas in exploration-exploitation, upper *confidence* bounds on stochastic values are used to guide action choices [1]. Upper confidence bounds have also been recently applied to planning [13, 10], but typically without making the connection with the deterministic optimism of classical planning. In this chapter, we integrate both types of optimism into a single framework, in the context of MDPs. To this end, planning is cast as the problem of optimizing returns over planning policies from the current state. This interpretation leads to several intuitive planning algorithms with strong performance guarantees, which have been developed over the last few years [11, 2, 6] and which we review in detail.

Next, Section 22.2 describes the foundations of optimism in online optimization. Section 22.3 forms the core of the chapter, and exploits the ideas of Section 22.2 to introduce the family of optimistic planning algorithms. Methods for three types of

system dynamics are reviewed: deterministic, general stochastic, and stochastic but where each transition can only end up in a finite number of states. Related algorithms from the literature are discussed in relation to optimistic planning in Section 22.4. The behavior of the three methods is illustrated in a numerical example in Section 22.5.

22.2 OPTIMISTIC ONLINE OPTIMIZATION

The idea of optimism can best be understood by looking at the problem of optimizing an unknown function $v : H \to \mathbb{R}$ online, while observing possibly noisy values of this function at chosen points. At each round $t = 1, 2, \ldots$, the algorithm chooses a point $h_t \in H$, and is provided with a random sample $\tilde{v}_t = v(h_t) + w_t$ where w is zero-mean noise, drawn independently for each sample. The goal is not just to find an optimal point $h^* \in \arg\max_{h \in H} v(h)$, but due to the online setting, to find it while sampling points of as high a quality as possible. We will not provide formal guarantees for optimistic optimization in terms of this requirement (although they are available), because we are only interested in these methods as stepping stones for optimistic planning. In that context, a weaker requirement will be sufficient, which only asks the algorithm to return the best point possible after expending some number of samples, without looking directly at the quality of these samples.

We first consider two special cases: noisy evaluations but small discrete set H; and infinite set H but deterministic evaluations. We then proceed to the general case.

22.2.1 Bandit problems

In the case of bandit problems, the observed values are noisy, but H is small and discrete: $H = \{h^1, \ldots, h^M\}$. The name comes from the fact that this case models the optimization of the payoff achieved by pulling the arms of an M-armed slot machine ("bandit"), where each h^j is an arm. Thus, online optimization is a way to solve the exploration-exploitation dilemma: given the information available so far, should the algorithm exploit the arm that seems the best (has the best average observed value), or should another arm be explored to achieve better confidence in its value? This formalism is used especially in reinforcement learning algorithms.

There are many ways to solve bandit problems, and here we will only consider the *upper confidence bound* (UCB) algorithm. At every round t, this algorithm defines a b-value for every h^j:

$$b(h^j) = \frac{\sum_{i=1}^{n(h^j)} \tilde{v}_i^j}{n(h^j)} + \sqrt{\frac{2 \log t}{n(h^j)}} \tag{22.3}$$

where $n(h^j)$ is the number of instances h^j has been chosen up to round t, and $\tilde{v}_i^j$ is the random value (sample) observed at the ith such instance (the b-values of unsampled points, for which $n(h^j) = 0$, are by convention positive infinite). Each b-value is an UCB on $v(h^j)$, that is, with high probability it is an upper bound. Then the algorithm simply makes an *optimistic* choice $\arg\max_{h^j} b(h^j)$ for the next sample. It must be emphasized that the count $n(h^j)$ and thus the b-value $b(h^j)$ depend of course on the

round t. Throughout the chapter, we leave such dependencies on the current round implicit in order to avoid cluttering the notation.

Such an algorithm is optimistic because it treats the UCB as if it were the actual value of the point. The optimistic principle has strong theoretical support in bandit problems [1].

22.2.2 Lipschitz functions and deterministic samples

When the set H is infinite (or just very large), it is impossible to sample every point. Consider the deterministic case, where the algorithm is given the exact value $v(h)$ for each point h it chooses. Let $\ell(h, \bar{h})$ be a metric over H, and assume v is Lipschitz:

$$\left|v(h) - v(\bar{h})\right| \leq \ell(h, \bar{h}) \quad \forall h, \bar{h} \in H$$

where for convenience we do not make the Lipschitz constant explicit. Then for any set of points $\boldsymbol{h} \subset H$ and any point $h \in \boldsymbol{h}$ we have the following upper bound on the supremum of v over $\boldsymbol{h}$ (i.e. this upper bound is larger than $v(\bar{h})$ for any $\bar{h} \in \boldsymbol{h}$):

$$v(h) + \operatorname{diam}(\boldsymbol{h})$$

where $\operatorname{diam}(\boldsymbol{h}) = \sup_{h,\bar{h}\in\boldsymbol{h}} \ell(h, \bar{h})$ denotes the diameter of the set. If multiple samples are available in $\boldsymbol{h}$, say their number is $n(\boldsymbol{h})$, then the bound is refined to:

$$b_{\text{init}}(\boldsymbol{h}) = \min_{i=1,\ldots,n(\boldsymbol{h})} [v(h_i) + \operatorname{diam}(\boldsymbol{h})] \tag{22.4}$$

Using this property, a tree-based algorithm for optimizing v can be derived: Algorithm 22.1, called *optimistic optimization for deterministic functions* (OOD). This method builds a tree $\mathcal{T}$ of policy sets, where each node contains some set $\boldsymbol{h}$ and its children correspond to a partition of $\boldsymbol{h}$. No difference is made between the notation of a node and its corresponding set, and the children of some $\boldsymbol{h}$ are denoted by $\mathcal{C}(\boldsymbol{h})$. OOD starts building the tree from a root node containing the entire space H, and at each step, chooses an *optimistic* leaf by navigating the tree along the maximum-b-value path. The optimistic leaf is expanded, i.e., the corresponding set is split (partitioned) into several child subsets, and an arbitrary point is sampled in each subset. The node splitting procedure should ensure that the diameter decreases with the depth. For example, if each node corresponds to a hyperbox in a Euclidean space, then the optimistic leaf could be split halfway along the longest edge.

B-values b_{init} at the leaves are computed with (22.4) and propagated upwards in the tree, where each parent node gets the largest bound from its children. The minimization in (22.4) is useful because samples inherited by the leaf from its parents may provide a better bound than its own, newly obtained sample. Due to the propagation rule, navigating the tree along the optimistic path is equivalent to directly choosing the leaf with the largest b-value. The optimistic path is nevertheless made explicit to emphasize the connection with later algorithms.

Note the similarity of this algorithm with branch-and-bound optimization. One difference is that branch-and-bound also uses lower bounds to eliminate some nodes from consideration. In Algorithm 22.1, all the nodes remain candidates for expansion, but the algorithm always focuses on the most promising node.

Algorithm 22.1 Optimistic optimization of deterministic functions

initialize tree: $\mathcal{T} \leftarrow \{H\}$
for $t = 1, 2, \ldots$ **do**
  $\boldsymbol{h} \leftarrow$ root H; **while** $\boldsymbol{h}$ is not leaf **do** $\boldsymbol{h} \leftarrow \arg\max_{\boldsymbol{h}' \in \mathcal{C}(\boldsymbol{h})} b(\boldsymbol{h}')$ **end while**
  optimistic leaf found: $\boldsymbol{h}^\dagger \leftarrow \boldsymbol{h}$
  expand $\boldsymbol{h}^\dagger$ into several child subsets $\boldsymbol{h}'$, adding them to $\mathcal{T}$
  sample a point in every new child $\boldsymbol{h}'$
  update b-values upwards in the tree, starting from the leaves:

$$b(\boldsymbol{h}) \leftarrow \begin{cases} b_{\text{init}}(\boldsymbol{h}) \text{ from (22.4)} & \text{if } \boldsymbol{h} \text{ is leaf} \\ \max_{\boldsymbol{h}' \in \mathcal{C}(\boldsymbol{h})} b(\boldsymbol{h}') & \text{otherwise} \end{cases}$$

end for

22.2.3 Lipschitz functions and random samples

A general algorithm, which works for the noisy optimization of Lipschitz functions defined over arbitrary metric spaces, will combine the two types of optimism highlighted above: upper confidence bounds due to the stochastic samples, with diameter-related bounds due to the non-zero size of the sets considered. In particular, at round t, a high-probability upper bound on the supremum of v over some set $\boldsymbol{h}$ is:

$$b_{\text{init}}(\boldsymbol{h}) = \frac{\sum_{i=1}^{n(\boldsymbol{h})} \tilde{v}_i}{n(\boldsymbol{h})} + \sqrt{\frac{2\log(t)}{n(\boldsymbol{h})}} + \operatorname{diam}(\boldsymbol{h}) \tag{22.5}$$

where $n(\boldsymbol{h})$ is as before the number of samples falling inside set $\boldsymbol{h}$, and $\tilde{v}_i$ are the random values of these samples.

The resulting algorithm is called *hierarchical optimistic optimization* (HOO) [3] and is shown in Algorithm 22.2. Like for OOD, the splitting procedure should decrease the diameter. The only difference from OOD is that a parent either inherits the b-value from one of its children, or it uses its own b-value b_{init} if it is better (smaller). The latter situation could not arise in OOD, because the diameters decrease with the depth and sample values are deterministic. Figure 22.1 illustrates how HOO develops the tree and computes the b-values.

22.3 OPTIMISTIC PLANNING ALGORITHMS

In this section, the ideas of optimistic optimization are applied to planning. As already outlined in the introduction, the planning algorithms we consider work online, in a receding-horizon fashion, and use a model of the MDP in the form of the functions f and ρ. At each step k, the model is employed to predict possible behaviors of the system (specifically, state trajectories and ensuing rewards) starting from the current state x_k and responding to various sequences of actions. Using these predictions, the algorithm returns an action u_k that is as close to optimal as possible, where near-optimality is measured by the simple regret (22.2). This action is applied, the system transits to x_{k+1}, and the cycle repeats.

Algorithm 22.2 Hierarchical optimistic optimization

initialize tree: $\mathcal{T} \leftarrow \{H\}$
for $t = 1, 2, \ldots$ **do**
  $\boldsymbol{h} \leftarrow$ root H; **while** $\boldsymbol{h}$ is not leaf **do** $\boldsymbol{h} \leftarrow \arg\max_{\boldsymbol{h}' \in \mathcal{C}(\boldsymbol{h})} b(\boldsymbol{h}')$ **end while**
  optimistic leaf found: $\boldsymbol{h}^\dagger \leftarrow \boldsymbol{h}$
  expand $\boldsymbol{h}^\dagger$ into several child subsets $\boldsymbol{h}'$, adding them to $\mathcal{T}$
  sample a point in every new child $\boldsymbol{h}'$
  update b-values upwards in the tree, starting from the leaves:

$$b(\boldsymbol{h}) = \begin{cases} b_{\text{init}}(\boldsymbol{h}) \text{ from (22.5)} & \text{if } \boldsymbol{h} \text{ is leaf} \\ \min\left\{b_{\text{init}}(\boldsymbol{h}), \max_{\boldsymbol{h}' \in \mathcal{C}(\boldsymbol{h})} b(\boldsymbol{h}')\right\} & \text{otherwise} \end{cases}$$

end for

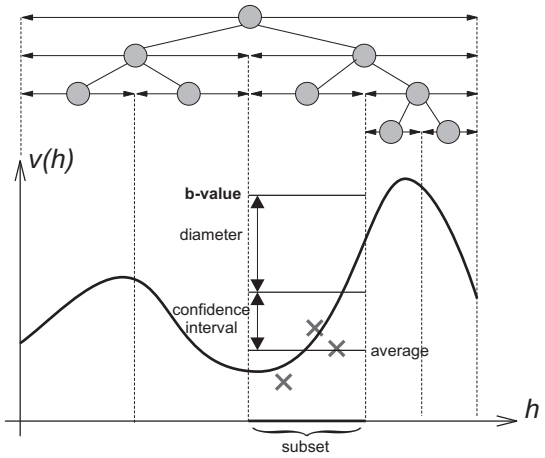

Figure 22.1 Illustration of HOO. Each tree node corresponds to a subset along the horizontal axis. For each leaf subset $\boldsymbol{h}$, the b-value is computed as the sum of the average of the observed samples $\frac{\sum_{i=1}^{n(\boldsymbol{h})} \tilde{v}_i}{n(\boldsymbol{h})}$, the confidence interval $\sqrt{\frac{2\log(t)}{n(\boldsymbol{h})}}$, and the subset diameter $\mathrm{diam}(\boldsymbol{h})$.

To place this problem in the context of optimistic optimization, let us isolate one such step of interaction with the system, and by convention, relabel the current time from k to 0, so that the system state is x_0. Planning should then maximize the expected return starting from x_0:

$$v(h) = \mathrm{E}_{x_{d+1} \sim f(x_d, h(d,x_d), \cdot)} \left\{ \sum_{d=0}^{\infty} \gamma^d r_{d+1} \right\} \tag{22.6}$$

where h is a rule describing how actions are chosen for each state occurring along future trajectories. This equation immediately clarifies the link with the optimization problem discussed in Section 22.2: the function v to optimize is the expected return, and the space over which we optimize consists of all possible h for x_0.

The time step along planned trajectories is denoted d instead of k, for two reasons: to emphasize the difference between planned and real interaction with the system, and because d will be related to the depth in certain trees considered later. The rule h is different from the policy π used while *interacting* with the system, which is the overall result of applying the planning algorithm at each step. We call h a *planning* policy. Since in the sequel we are mostly interested in planning policies, they will often simply be called "policies" for the sake of brevity. When planning policies are discussed together with interaction policies, we will use the long, explicit names of both, to make things clear. Note it is not strictly necessary to have a dependence on time d in the planning policy, since for the discounted infinite-horizon MDPs considered in this chapter, a policy that only depends on the state is sufficient to obtain optimal behavior. Nevertheless, this is the general form of the policy sought by planning algorithms.

Usually, an amount n of computational resources will be specified, and the planning algorithm must return an u_0 with the smallest simple regret given this constraint. The computational units can be number of evaluations of the model functions, number of basic arithmetic operations, etc. In practice, n may often not be specified in advance but may depend e.g. on real-time constraints. This motivates an anytime behavior, which returns the best action possible after any amount n of units spent, without using the actual value of n. The exploration-exploitation tradeoff still appears in the weaker sense of judiciously prioritizing the sampling of planning policies in order to minimize the final, simple regret.

Three planning algorithms will be considered in detail in this section: optimistic planning for deterministic systems (OPD) [11], open loop optimistic planning (OLOP) [2], and optimistic planning for sparsely stochastic systems (OPSS) [6]; the latter two algorithms work for stochastic systems.

All these algorithms require discrete actions, so that $U = \left\{u^1, ..., u^M\right\}$. OPSS also requires that each random transition ends up in one of a finite number of next states. Thus it works for finite-state MDPs, as well as infinite-state MDPs that satisfy the condition, which we call "sparsely stochastic". Furthermore, for all algorithms it is assumed the rewards are bounded in the interval $[0, 1]$, i.e., $\rho(x, u, x') \in [0, 1]$ for any x, u, x'. This assumption is not overly restrictive since, at least for problems that do not have terminal states,[2] any bounded reward function can be normalized to $[0, 1]$ by translation and scaling, without changing the optimal policies.

22.3.1 Optimistic planning for deterministic systems

In the deterministic case, the planning policy is sufficiently specified by an infinite sequence of actions $h = [u_0, u_1, \ldots, u_d, \ldots]$ with $u_d \in U$, since this sequence completely determines the system trajectory. Thus H is the infinitely-dimensional

[2] A terminal state is one from which the system can no longer escape, and from which all actions always obtain zero rewards. Such states can be used to represent e.g. "goal achieved" and "failure" situations.

space U^∞. The value (return) of a policy h is:

$$v(h) = \sum_{d=0}^{\infty} \gamma^d r_{d+1} = \sum_{d=0}^{\infty} \gamma^d \rho(x_d, u_d, x_{d+1}) \tag{22.7}$$

Define the metric (distance between policies) over H:

$$\ell(h, \bar{h}) = \frac{\gamma^{d_{\text{diff}}(h,\bar{h})}}{1-\gamma}$$

where $d_{\text{diff}}(h, \bar{h}) = \min\{d \mid u_d \neq \bar{u}_d\}$, that is, the smallest index where the two policies are different. Intuitively, this distance is the largest difference between the returns provided by the two policies in any possible MDP. Because the rewards are in $[0, 1]$, the returns can differ by at most $\sum_{\bar{d}=d_{\text{diff}}(h,\bar{h})}^{\infty} \gamma^{\bar{d}} \cdot 1 = \frac{\gamma^{d_{\text{diff}}(h,\bar{h})}}{1-\gamma}$.

Optimistic planning for deterministic systems (OPD) will be built starting from OOD (Algorithm 22.1). The sets considered are defined by incomplete sequences of length d, so that each policy in the set matches the sequence up to index $d-1$ and is free to choose any action afterwards: $\boldsymbol{h}_d = \{[u_0, u_1, \ldots, u_{d-1}, \star, \star, ...]\}$ with '$\star$' taken to mean "any action". Note $\boldsymbol{h}_0 = H$ since it imposes no constraints on the policy. A set $\boldsymbol{h}_d$ is split by making the choices for u_d definite, so that we obtain M different children sets $\boldsymbol{h}_{d+1}$, one for every value of $u_d \in \{u^1, ..., u^M\}$. Note that, under the chosen metric, the diameter of each such set is $\frac{\gamma^d}{1-\gamma}$ since any two policies in $\boldsymbol{h}_d$ can differ at the earliest at index d; thus also $\text{diam}(H) = 1/1-\gamma$.

We have not specified how to sample a value in a leaf set $\boldsymbol{h}_d$. In fact, since this value is an infinite sum of rewards, it is not possible to obtain an exact sample. This is fortunately not a problem, since an upper bound on the values of policies in some leaf set $\boldsymbol{h}_d$ can still be obtained from the sequence of rewards simulated so far:

$$b_{\text{init}}(\boldsymbol{h}_d) = \nu(\boldsymbol{h}_d) + \text{diam}(\boldsymbol{h}_d) = \nu(\boldsymbol{h}_d) + \frac{\gamma^d}{1-\gamma}$$

$$\text{where } \nu(\boldsymbol{h}_d) = \sum_{\bar{d}=0}^{d-1} \gamma^{\bar{d}} \rho(x_{\bar{d}}, u_{\bar{d}}, x_{\bar{d}+1})$$

This can be understood using the intuition explained above for the metric ℓ. Thus, expanding a node and computing the rewards obtained for reaching each of its children replaces the usual sampling process. Note that $\nu(\boldsymbol{h}_d)$ is a *lower* bound on the values of all policies in $\boldsymbol{h}_d$.

After performing n expansions, OPD chooses the set that maximizes $\nu(\boldsymbol{h}_d)$ amongst all the sets considered so far, and returns the action u_0 for this set. Notice that in contrast to the optimistic criterion used to build the tree, at the end OPD makes a safe choice, based on lower bounds. With this, the instantiation of OPD as a variant of Algorithm 22.1 is complete. Rather than making this instantiation explicit, we choose to present the algorithm in an alternative, equivalent way that contributes additional intuition and helps in understanding OPSS later.

There exists a one-to-one mapping between the tree of sets $\boldsymbol{h}_d$ and a corresponding tree of *states*, see Figure 22.2. Each path through this latter tree is the trajectory

obtained by applying a considered sequence $u_0, u_1, \ldots, u_{d-1}$; thus each node x_d at depth d corresponds to a set $\boldsymbol{h}_d$. An arc (x_d, x_{d+1}) corresponds to a transition and is labeled by the associated action u_d and reward $r_{d+1} = \rho(x_d, u_d, x_{d+1})$. Expanding a node x is done by simulating the transitions for all M actions, and adding all the resulting next states x' as children $\mathcal{C}(x)$ to the tree. The b-value of a leaf node is:

$$b_{\text{init}}(x) = \nu(x) + \frac{\gamma^d(x)}{1-\gamma} \tag{22.8}$$

where $\nu(x)$ is the return along the path from x_0 to x. Algorithm 22.3 summarizes OPD using this second type of tree.

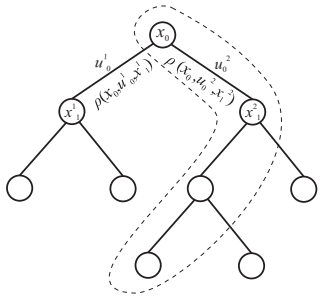

Figure 22.2 Illustration of an OPD tree. Each arc corresponds to a transition. Subscripts are depths, superscripts index the M possible actions/transitions from each node ($M = 2$ in this example). The dashed outline encloses a possible optimistic path.

Algorithm 22.3 Optimistic planning for deterministic systems

initialize tree: $\mathcal{T} \leftarrow \{x_0\}$
for $t = 0, \ldots, n$ **do**
  $x \leftarrow$ root x_0; **while** x is not leaf **do** $x \leftarrow \arg\max_{x' \in \mathcal{C}(x)} b(x')$ **end while**
  optimistic leaf found: $x^\dagger \leftarrow x$
  expand $x^\dagger$: simulate all M transitions from $x^\dagger$, add next states x' to $\mathcal{T}$
  update b-values upwards in the tree, starting from the leaves:

$$b(x) = \begin{cases} b_{\text{init}}(x) \text{ from (22.8)} & \text{if } x \text{ is leaf} \\ \max_{x' \in \mathcal{C}(x)} b(x') & \text{otherwise} \end{cases}$$

end for
output u_0, the first action along path with largest ν

22.3.2 Open-loop optimistic planning

In the stochastic case, there are multiple state outcomes for a given action at some step, so a simple sequence of actions is no longer sufficient to represent a full planning policy. Such a policy would need to depend on the state, having e.g. the form $h(d, x_d)$ in (22.6), which specifies an action choice for each possible outcome x_d at every step d.

Representing such a policy is unfortunately impossible in general, since there can be an infinite number of outcomes. Instead, the *open-loop optimistic planning* (OLOP) algorithm chooses to still optimize only over plain sequences of actions, without explicitly considering the underlying states x_d – hence its "open-loop" nature.[3]

OLOP will be described using the tree-of-sets formalism, since a tree of states does not exist in this case. The space H and metric are the same as in the deterministic case, and splitting has the same meaning of initializing all action choices for the next step. However, the value of a policy h is now the expected return:

$$v(h) = \mathrm{E}_{x_{d+1}\sim f(x_d,u_d,\cdot)}\left\{\sum_{d=0}^{\infty}\gamma^d\rho(x_d,u_d,x_{d+1})\right\} \tag{22.9}$$

so we are dealing with a noisy optimization problem: summing up the rewards along some simulated trajectory only gives a *sample* return. Moreover, to obtain independent return samples, whenever traveling down the optimistic action path in the tree a *new* trajectory must be simulated, rather than relying on the already existing reward samples as in OPD. Formally, each set (tree node) $\boldsymbol{h}_d$ is associated with a cumulative reward $r(\boldsymbol{h}_d)$ and a count $n(\boldsymbol{h}_d)$ of how many reward samples are available for it, with both quantities initialized to 0. Simulating a trajectory $(x_{\bar{d}}, u_{\bar{d}}, x_{\bar{d}+1}, r_{\bar{d}+1})$ for $\bar{d} = 0, \dots, d-1$ provides new reward samples for all the sets $\boldsymbol{h}_{\bar{d}+1} = \{[u_0, u_1, \dots, u_{\bar{d}}, \star, \star, \dots]\}$:

$$\begin{aligned} r(\boldsymbol{h}_{\bar{d}+1}) &\leftarrow r(\boldsymbol{h}_{\bar{d}+1}) + r_{\bar{d}+1} \\ n(\boldsymbol{h}_{\bar{d}+1}) &\leftarrow n(\boldsymbol{h}_{\bar{d}+1}) + 1 \end{aligned}$$

OLOP can be seen as a variant of HOO (Algorithm 22.2) that uses the b-values:

$$b_{\mathrm{init}}(\boldsymbol{h}_d) = \sum_{\bar{d}=0}^{d-1}\gamma^{\bar{d}}\left[\frac{r(\boldsymbol{h}_{\bar{d}+1})}{n(\boldsymbol{h}_{\bar{d}+1})} + \sqrt{\frac{2\log(t)}{n(\boldsymbol{h}_{\bar{d}+1})}}\right] + \frac{\gamma^d}{1-\gamma} \tag{22.10}$$

for some set $\boldsymbol{h}_d$, where t is the number of *trajectories* simulated so far. These b-values are different from what the "vanilla" application of HOO would give, see (22.5): they cleverly exploit the additive structure of the value (return) to compute separate UCBs at each step, whereas vanilla HOO would compute a single UCB for the entire trajectory. The diameter component $\frac{\gamma^d}{1-\gamma}$ remains the same as in the deterministic case.

Algorithm 22.4 summarizes OLOP, in a variant where the computational unit consists of simulating a single transition (rather than M as in OPD), and the final action choice is based on the most-often sampled first action (rather than the largest ν). There is a clear parallel between OLOP as an application of HOO, and OPD as an application of OOD. For instance, OLOP requires a minimum with the b-value b_{init} of the current node when propagating b-values up the tree.

[3]Instead of the theoretical OLOP algorithm of [2], we present here a variant more directly related to the HOO template, and also more amenable to practical implementation.

Algorithm 22.4 Open-loop optimistic planning

initialize tree: $\mathcal{T} \leftarrow \{H\}$
for $t = 1, 2, ...,$ stopping when n transitions have been exceeded **do**
  $\boldsymbol{h} \leftarrow$ root H; **while** $\boldsymbol{h}$ is not leaf **do** $\boldsymbol{h} \leftarrow \arg\max_{\boldsymbol{h}' \in \mathcal{C}(\boldsymbol{h})} b(\boldsymbol{h}')$ **end while**
  optimistic leaf found: $\boldsymbol{h}_d^\dagger \leftarrow \boldsymbol{h}$
  expand $\boldsymbol{h}^\dagger$ into M child subsets $\boldsymbol{h}'$, one for each action, adding them to $\mathcal{T}$
  simulate a trajectory from x_0, using actions in $\boldsymbol{h}_d^\dagger$, reaching some x_d
  from x_d, simulate a transition for each action
  update b-values upwards in the tree, starting from the leaves:

$$b(\boldsymbol{h}) = \begin{cases} b_{\text{init}}(\boldsymbol{h}) \text{ from (22.10)} & \text{if } \boldsymbol{h} \text{ is leaf} \\ \min\left\{b_{\text{init}}(\boldsymbol{h}), \max_{\boldsymbol{h}' \in \mathcal{C}(\boldsymbol{h})} b(\boldsymbol{h}')\right\} & \text{otherwise} \end{cases}$$

end for
output u_0, a first action that was sampled most often

22.3.3 Optimistic planning for sparsely stochastic systems

When each random state transition can only end up in a small number N of next states, it is in fact possible to represent full, closed-loop planning policies $h(d, x_d)$, which also depend on the underlying state. More specifically, a policy can be represented as a vector:

$$h = [u_0, u_1^1, \ldots, u_1^N, u_2^{1,1}, \ldots, u_2^{N,N}, \ldots\ldots]$$

because at step 1 it must provide an action for each of the possible N outcomes; at step 2, for each of the N^2 outcomes, and so on. Exploiting this idea, a *deterministic* algorithm can be built on the foundations of OOD, even though the system is stochastic. We call systems with the property above sparsely stochastic, so the algorithm is *optimistic planning for sparsely stochastic systems* (OPSS). This class includes all finite-state MDPs (where good performance can be expected when N is small), as well as important classes of continuous-state MDPs such as those that arise by combining deterministic dynamics with discrete random variables, or by discretizing in time some continuous-time stochastic systems.

Rather than use the h notation which is cumbersome in this case, OPD is described using a tree of states, like OPD earlier. This tree $\mathcal{T}$ is built starting from a root containing x_0 and iteratively expanding nodes. Each expansion consists of generating and adding all the N one-step successor states of the node-to-expand, for all M actions, see Figure 22.3, left. The b-values of leaf nodes are computed with:

$$b_{\text{init}}(x) = \nu(x) + \frac{\gamma^d(x)}{1-\gamma} \tag{22.11}$$

where $\nu(x)$ is the discounted sum of rewards along the path from x_0 to x. The b-values of nodes higher in the tree maximize *weighted sums* of children's b-values, where the weights are the children's probabilities: $b(x) = \max_{u \in U} \sum_{x' \in \mathcal{C}(x,u)} f(x, u, x') b(x')$. Here, $\mathcal{C}(x, u)$ denotes the set of children reachable from x after performing action u.

Rather than an optimistic path as in OPD and OLOP, OPSS builds an *optimistic subtree* $\mathcal{T}^\dagger$ by starting from the root and selecting at each node x *all* the children associated to an optimistic action:

$$u^\dagger(x) = \arg\max_{u \in U} \sum_{x'} f(x, u, x') b(x')$$

Among the leaves $\mathcal{L}^\dagger$ of this subtree, the node to expand is selected using:

$$x^\dagger = \arg\max_{x \in \mathcal{L}^\dagger} \mathrm{P}(x) \, \frac{\gamma^{d(x)}}{1-\gamma} \tag{22.12}$$

where $\mathrm{P}(x)$ is the probability to reach x, i.e. the product of all probabilities along the path to x: $\mathrm{P}(x) = \sum_{d=0}^{d(x)-1} f(x_d, u_d, x_{d+1})$. After n expansions, the algorithm selects at the root an action:

$$u_0 = \arg\max_{u \in U} \sum_{x'} f(x_0, u, x') \nu(x')$$

where ν-values are propagated up the tree just like b-values, starting from their definition at the leaves given above.

Algorithm 22.5 summarizes OPSS. The computational unit consists in this case of generating all the NM children of a particular state.

Algorithm 22.5 Optimistic planning for sparsely stochastic systems

initialize tree: $\mathcal{T} \leftarrow \{x_0\}$
for $t = 1, \ldots, n$ **do**
  build optimistic subtree $\mathcal{T}^\dagger$
  choose node to expand: $x^\dagger \leftarrow \arg\max_{x \in \mathcal{L}^\dagger} \mathrm{P}(x) \, \frac{\gamma^{d(x)}}{1-\gamma}$
  expand $x^\dagger$: simulate all NM transitions from $x^\dagger$, adding next states x' to $\mathcal{T}$
  update b-values upwards in the tree, starting from the leaves:

$$b(x) = \begin{cases} b_{\mathrm{init}}(x) \text{ from (22.11)} & \text{if } x \text{ is leaf} \\ \max_{u \in U} \sum_{x' \in \mathcal{C}(x,u)} f(x, u, x') b(x') & \text{otherwise} \end{cases}$$

end for
output u_0 maximizing ν

To better understand the choices made in OPSS, let us informally return to the tree-of-policy-sets interpretation. To keep the parallel clear, we denote the tree of sets by $\boldsymbol{\mathcal{T}}$ in contrast to the tree of states $\mathcal{T}$, and always specify explicitly to which tree each entity belongs. Selecting the children of one particular action at each state in $\mathcal{T}$, recursively until reaching some set of leaves $\bar{\mathcal{L}} \in \mathcal{T}$, corresponds to selecting some *single* leaf subset $\boldsymbol{h} \in \boldsymbol{\mathcal{T}}$, with the b-value:

$$\begin{aligned} b(\boldsymbol{h}) &= \sum_{x \in \bar{\mathcal{L}}} \mathrm{P}(x) \, [\nu(x) + \frac{\gamma^{d(x)}}{1-\gamma}] \\ &= \sum_{x \in \bar{\mathcal{L}}} \mathrm{P}(x) \, \nu(x) + \sum_{x \in \bar{\mathcal{L}}} \mathrm{P}(x) \, \frac{\gamma^{d(x)}}{1-\gamma} = \nu(\boldsymbol{h}) + \mathrm{diam}(\boldsymbol{h}) \end{aligned}$$

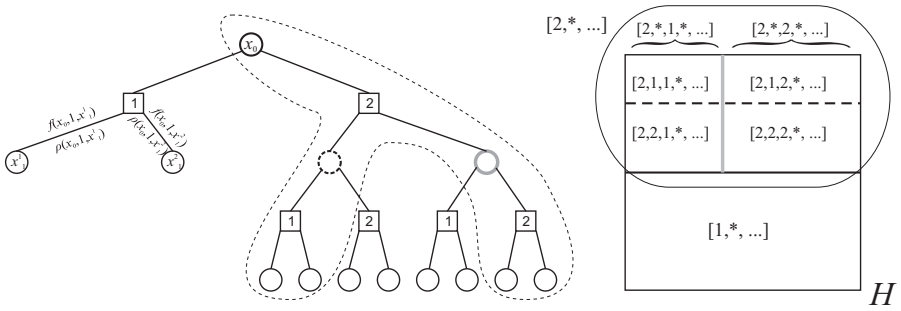

Figure 22.3 Left: illustration of an OPSS tree after three expansions (root, gray node, and dashed node), for $N = M = 2$. The circles are state nodes, and the actions are explicitly represented as square, choice nodes. Transition arcs to next states are labeled by probabilities and rewards. The dashed outline encloses a possible optimistic subtree. Right: corresponding partition of the policy space H. Expanding the root splits H in two sets containing policies of the form $[1, \star, \dots]$ and $[2, \star, \dots]$ respectively (recall a policy has the structure $h = [u_0, u_1^1, u_1^2, \dots]$). Next, expanding the gray node – which corresponds to the second random outcome of action 2 – splits $\{[2, \star, \dots]\}$ into $\{[2, \star, 1, \star, \dots]\}$ and $\{[2, \star, 2, \star, \dots]\}$; note the action for the first random outcome (dashed node) remains undefined. Expanding the dashed node then makes the action definite, thereby splitting *both* sets $\{[2, \star, 1, \star, \dots]\}$ and $\{[2, \star, 2, \star, \dots]\}$, since the dashed node is reached with nonzero probability by both classes of policies.

the standard formula for the b-value. The lower bound $\nu(\boldsymbol{h})$ is the sum of truncated returns up to the leaves in $\mathcal{T}$, weighted by their probabilities. A similar weighted formula holds for the diameter.

Among all these leaf policy sets in $\boldsymbol{\mathcal{T}}$, the optimistic action choices in $\mathcal{T}$ lead to the optimistic set $\boldsymbol{h}^\dagger \in \boldsymbol{\mathcal{T}}$, which has the largest b-value, and expanding the optimistic node $x^\dagger \in \mathcal{T}$ splits $\boldsymbol{h}^\dagger \in \boldsymbol{\mathcal{T}}$ into M children *along the longest edge*, which corresponds to $x^\dagger$. Furthermore, contrary to all algorithms so far, this expansion *also splits many additional sets*: those corresponding to all policies that reach $x^\dagger$ with nonzero probability. Figure 22.3, right illustrates this effect, and more generally how expanding leaves in $\mathcal{T}$ corresponds to splitting sets in $\boldsymbol{\mathcal{T}}$. Due to this multiple-splitting property, there is no one-to-one mapping between $\mathcal{T}$ and $\boldsymbol{\mathcal{T}}$.

22.3.4 Theoretical guarantees

The core theoretical question for optimistic planning algorithms concerns the quality of the action they produce after expending the budget n, in terms of the simple regret (22.2). Available guarantees rely on a measure of the complexity of the problem that can best be understood in terms of a branching factor of some related trees.

For OPD, this tree consists of near-optimal policy sets $\boldsymbol{h}$: $\mathcal{T}^* = \left\{\boldsymbol{h}_d \middle| v^* - \nu(\boldsymbol{h}_d) \leq \frac{\gamma^d}{1-\gamma}\right\}$, where v^* is the optimal return achievable from x_0. So, $\mathcal{T}^*$ includes at every level d the policy sets for which it is impossible to tell, from the rewards achieved up that level, whether they contain an optimal policy. The larger the number of such sets,

the more difficult the problem is for planning. Denoting the asymptotic (as $d \to \infty$) branching factor of $\mathcal{T}^*$ by κ, the simple regret of OPD is:

$$\mathcal{R}(x_0) = \begin{cases} \mathrm{O}(n^{-\frac{\log 1/\gamma}{\log \kappa}}) & \text{if } \kappa > 1 \\ \mathrm{O}(\gamma^{nc}) & \text{if } \kappa = 1 \end{cases}$$

where c is a specific constant [11]. In the ideal case $\kappa = 1$, which means there is a single near-optimal set at every level, and a single optimal planning policy. A regret that decreases exponentially with n is obtained in this case. As κ grows to K, the regret decreases more slowly with n.

OLOP defines the branching factor κ with a similar formula, but allows for technical reasons the sets to be suboptimal by up to $\frac{2\gamma^d}{1-\gamma}$ rather than $\frac{\gamma^d}{1-\gamma}$. Then, roughly speaking (the actual technical statement is more involved) [2]:

$$\mathcal{R}(x_0) \approx \begin{cases} \mathrm{O}(n^{-\frac{\log 1/\gamma}{\log \kappa}}) & \text{if } \gamma\sqrt{\kappa} > 1 \\ \mathrm{O}(n^{-1/2}) & \text{if } \gamma\sqrt{\kappa} \leq 1 \end{cases}$$

Remarkably, when $\gamma\sqrt{\kappa} > 1$ the regret has the same form as for OPD.

Similar guarantees hold for OPSS (these results have not yet been published).

22.4 RELATED PLANNING ALGORITHMS

A recent algorithm in the optimistic planning family is UCB applied to trees (UCT) [13], which has made a great impact in the planning field due to its competitive performance in games such as Go [10]. UCT basically applies UCB at every node in a planning tree similar to that of Figure 22.3. It travels an optimistic path along the tree choosing actions with maximal UCBs (22.3) on the returns obtained from the current node, and sampling states independently. An issue with UCT is that its b-values do not consider the size of the policy sets, which means they are not true upper confidence bounds; equivalently, it can be said that UCT assumes infinite smoothness of the value function it optimizes. A first attempt to address this problem, by introducing a so-called smoothness coefficient in place of the set diameter, was made in [7]. This "bandit algorithm for smooth trees" can be considered a precursor of OLOP. In [17] a continuous-action variant of UCT is introduced, which tackles continuous actions by using HOO instead of UCB in the tree nodes. This still does not introduce information about the size of policy sets.

On the other hand, optimistic planning has a strong relationship to classical planning algorithms such as A* and AO* [14, 18], which also expand planning trees by optimistically prioritizing nodes. By using upper bounds on the values of incomplete action sequences, A* and AO* do consider diameters of policy sets and obey the principles of optimistic optimization (more specifically OOD). OPD can be seen as an extension of A* from goal-based problems to more general, infinite-horizon discounted MDPs, and OPSS as a similar extension of AO*, with the specific rule (22.12) for choosing the particular node to expand. Exploiting UCBs is a strong innovation brought by UCT and OLOP to classical planning.

The key observation here is that, in the context of MDPs, all these algorithms belong to a general framework for optimistic optimization over planning policies. This insight guides essential choices in the algorithms – for instance, the need to always include diameters in the b-values, and the node expansion criterion (22.12) in OPSS, intuitively seen as splitting a set along the longest edge. If these choices are made correctly, strong theoretical guarantees on the performance are achieved in terms of the simple regret, as outlined in Section 22.3.4.

An early, non-optimistic sample-based planning algorithm for MDPs was given by [12]. This algorithm builds a symmetric planning tree by sampling a fixed number of states for each action, at each depth up to some horizon. It is called "sparse sampling" due to the finite number of outcomes sampled, so its sparseness is of a different nature than in OPSS. Sparse sampling was extended by [19] to consider an adaptive horizon. "Forward-search sparse sampling" [22] is an optimistic extension, which does not expand the tree uniformly, but along paths obtained by choosing maximum b-value actions, and then among the successors choosing the one with the maximum *gap* between the b-value and a lower-bound on the state value.

Disturbance trees are very similar to the planning trees used here, and planning algorithms based on them were given e.g. by [8, 9]. The planning algorithm of [15] exploits a different notion of sparse stochasticity than OPSS, in which a small number of the states lead to stochastic outcomes (rather than all states leading to a small number of outcomes, as in OPSS).

22.5 NUMERICAL EXAMPLE

To understand the behavior of OPD, OLOP, and OPSS in practice, they will be applied to the problem of swinging up an underactuated inverted pendulum – a rather simple problem commonly used in the literature on solving MDPs.

The inverted pendulum consists of a mass attached to an actuated link (a rod) that rotates in a vertical plane. The available power is taken insufficient to push the pendulum up in a single rotation from every initial state. Instead, from certain states (e.g., pointing down), the pendulum needs to be swung back and forth to gather energy, prior to being pushed up and stabilized. Finding such a solution is challenging for planning algorithms, because the swing-up must be planned over a longer horizon, and solutions that seem optimal over a short horizon will not work, instead just pushing the pendulum in one direction.

A continuous-time model of the pendulum dynamics is:

$$\ddot{\alpha} = 1/J \cdot [mgl \sin(\alpha) - b\dot{\alpha} - K^2\dot{\alpha}/R + Ku/R]$$

where we assign inertia $J = 1.91 \cdot 10^{-4}\,\text{kgm}^2$, mass $m = 0.055\,\text{kg}$, gravitational acceleration $g = 9.81\,\text{m/s}^2$, length $l = 0.042\,\text{m}$, damping $b = 3 \cdot 10^{-6}\,\text{Nms/rad}$, torque constant $K = 0.0536\,\text{Nm/A}$, and resistance $R = 9.5\,\Omega$. The angle α varies in the interval $[-\pi, \pi]$ rad, with $\alpha = 0$ pointing up, and 'wraps around' so that e.g. a rotation of $3\pi/2$ corresponds to $\alpha = -\pi/2$. The state is $x = [\alpha, \dot{\alpha}]^\top$. The

velocity $\dot{\alpha}$ is restricted to $[-15\pi, 15\pi]$ rad/s, using saturation. The control action $u \in [-3, 3]$ V may be affected by noise as described below. It is discretized into the set $U = \{-3, 0, 3\}$, so that $M = 3$. The sampling time is $T_{\mathrm{s}} = 0.05$ s.

The goal is to stabilize the pendulum in the unstable equilibrium $x = 0$ (pointing up), and is expressed by the *unnormalized* quadratic rewards:

$$r = \rho_{\mathrm{unnorm}}(x, u, x') = -x^{\top} Q_{\mathrm{rew}} x - R_{\mathrm{rew}} u^2$$
$$\text{where: } Q_{\mathrm{rew}} = \mathrm{diag}[5, 0.1],\ R_{\mathrm{rew}} = 1$$

Using the known bounds on the state and action variables, the rewards are normalized (scaled and translated) to the interval $[0, 1]$. The discount factor is $\gamma = 0.95$.

Two versions of the problem are considered. The first variant has deterministic (noiseless) actions, and is used to exemplify the behavior of OPD. In the second variant, an unreliable actuator is modeled that only applies the intended action u with probability 0.6, and applies an action with smaller magnitude, $0.7u$, with probability 0.4 (when the intended action is 0 it remains 0 with probability 1). This corresponds to a sparsely stochastic MDP with $N = 2$, a more difficult problem than the deterministic variant. The results we will present for the stochastic inverted pendulum were first given in [6], while those for the deterministic variant are new.

To obtain a global performance measure, all algorithms are applied in an offline fashion, to find actions for the states on the grid $X_0 = \left\{-\pi, \frac{-150\pi}{180}, \frac{-120\pi}{180}, \ldots, \pi\right\} \times \{-15\pi, -14\pi, \ldots, 15\pi\}$. Since an exact optimal Q-function for the inverted pendulum problem is not known, in order to approximate the simple regret a near-optimal Q-function is computed instead using interpolation-based approximate value iteration [21, 5] with an accurate approximator.

Baseline planning solutions are obtained using uniform planning, which always expands a node having the smallest depth instead of an optimistic node, thus developing uniform, symmetric trees.

Results with OPD. Figure 22.4 shows the results obtained by OPD for the deterministic inverted pendulum, in comparison to uniform planning. The budget n varies in the set $\{100, 200, \ldots, 1000\}$. As expected, OPD works better (obtains smaller regret) than uniform planning, since it expands the planning trees in a smart way, resulting in much deeper trees. The regret of OPD decreases with n but quickly plateaus after $n = 300$, which indicates this budget is already sufficient.[4]

Results with OLOP and OPSS. For OLOP and OPSS, the sparsely stochastic variant of the inverted pendulum is used. For OPSS, the computational budget n

[4]Note that the regret does not decrease monotonically with n. This is because regret can vary nonmonotonically with the planning horizon. For a concrete example, consider a deterministic MDP with state space $\{1, 2, \ldots, 6\}$, two actions $-1, 1$, and additive dynamics $x' = \max(1, \min(6, x + u))$. The rewards obtained upon reaching one of the six states are, respectively, $4, 0, 0, 1, -10, 100$, and the discount factor is 0.5. Then, the optimal action for $x = 3$ is 1. If a fully expanded planning tree of depth (horizon) 1 is used to compute ν-values and choose an action, the action returned will be 1. If the depth is 2, the action changes to -1 due to the -10 reward. For depths larger than 2, due to the 100 reward, the action is again optimal, i.e., 1. The simple regret for $x = 3$ is thus clearly nonmonotonic with respect to the depth.

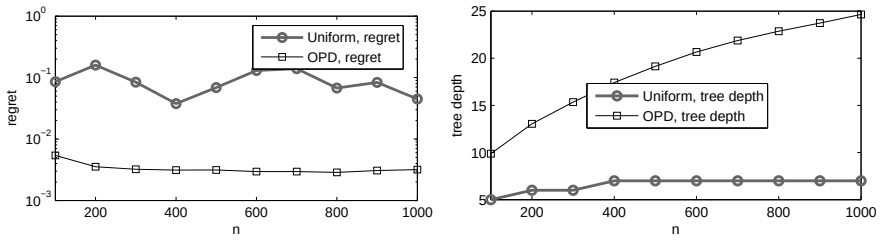

Figure 22.4 Results of OPD, compared to uniform planning: average regret over X_0 (left), average tree depth over X_0 (right).

varies in the set $\{100, 200, \ldots, 1000\}$ like for OPD above. Since each node expansion requires NM transitions, overall $NMn = 6n$ transitions are simulated. Because the computational unit of OLOP consists of simulating a single transition, it is allowed $6n$ transitions (the algorithm may use slightly more transitions to finish the last trajectory).

Figure 22.5 shows the results obtained, compared to the uniform planning baseline. The relationship between OPSS and uniform planning mirrors that between OPD and uniform planning in the deterministic case, although here the performance of OPSS continues to improve as n increases, reflecting the more difficult nature of the stochastic problem. Less expected is that OLOP works poorly, similarly to uniform planning. This likely happens because the computational budgets considered do not allow OLOP to decrease the upper confidence bounds on the returns to informative levels; OLOP may work better with a larger budget.

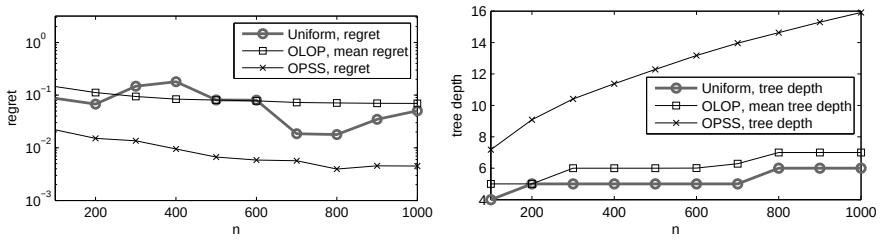

Figure 22.5 Comparison between OPSS and uniform planning: average regret over X_0 (left), average tree depth over X_0 (right). As the results of OLOP depend on particular realizations of stochastic trajectories, this algorithm is run 10 times and mean results are reported (the 95% confidence regions are too tight to be visible at this scale).

To illustrate the online control performance of optimistic planning, OPSS is applied in a receding-horizon fashion starting from the stable equilibrium of the pendulum (pointing down), and $n = 600$. Figure 22.6 shows the resulting trajectory, compared to that given by uniform planning. OPSS swings up the pendulum in one go, whereas uniform planning manages it only after a few attempts.

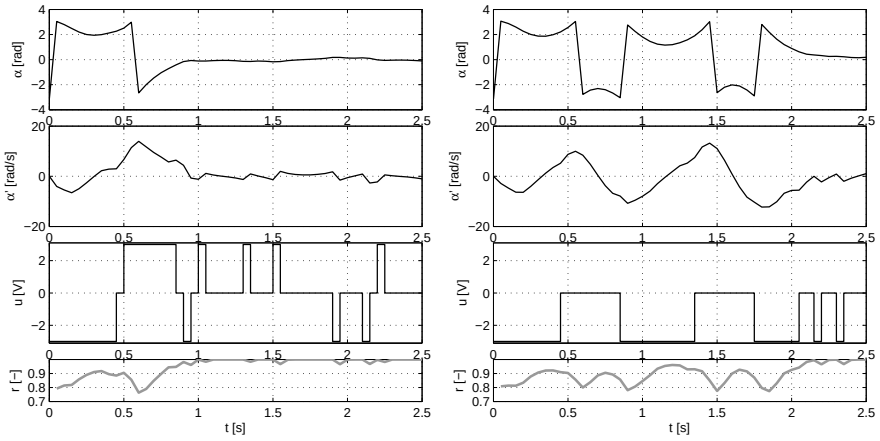

Figure 22.6 Online control results of OPSS and uniform planning.

REFERENCES

1. Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.
2. Sébastien Bubeck and Rémi Munos. Open loop optimistic planning. In *Proceedings 23rd Annual Conference on Learning Theory (COLT-10)*, pages 477–489, Haifa, Israel, 27–29 June 2010.
3. Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. Online optimization in X-armed bandits. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 201–208. MIT Press, 2009.
4. 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.
5. Lucian Buşoniu, Damien Ernst, Bart De Schutter, and Robert Babuška. Approximate dynamic programming with a fuzzy parameterization. *Automatica*, 46(5):804–814, 2010.
6. Lucian Buşoniu, Rémi Munos, Bart De Schutter, and Robert Babuška. Optimistic planning for sparsely stochastic systems. In *Proceedings 2011 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-11)*, pages 48–55, Paris, France, 11–15 April 2011. Special Session on *Active Reinforcement Learning*.
7. Pierre-Arnaud Coquelin and Remi Munos. Bandit algorithms for tree search. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 67–74, Vancouver, Canada, 19–22 July 2007.
8. 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.

9. Boris Defourny, Damien Ernst, and Louis Wehenkel. Planning under uncertainty, ensembles of disturbance trees and kernelized discrete action spaces. In *Proceedings 2009 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-09)*, pages 145–152, Nashville, US, 30 March – 2 April 2009.
10. Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical report, INRIA, 2006.
11. Jean-François Hren and Rémi Munos. Optimistic planning of deterministic systems. In *Proceedings 8th European Workshop on Reinforcement Learning (EWRL-08)*, pages 151–164, Villeneuve d'Ascq, France, 30 June – 3 July 2008.
12. 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.
13. Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings 17th European Conference on Machine Learning (ECML-06)*, pages 282–293, Berlin, Germany, 18–22 September 2006.
14. Steven M. La Valle. *Planning Algorithms*. Cambridge University Press, 2006.
15. Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. Planning for Markov decision processes with sparse stochasticity. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004.
16. J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.
17. Chris Mansley, Ari Weinstein, and Michael L. Littman. Sample-based planning for continuous action Markov decision processes. In *Proceedings 21st International Conference on Automated Planning and Scheduling*, pages 335–338, Freiburg, Germany, 11–16 June 2011.
18. N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing, 1980.
19. Laurent Péret and Frédérick Garcia. On-line search for solving Markov decision processes via heuristic sampling. In *Proceedings 16th European Conference on Artificial Intelligence, ECAI'2004*, pages 530–534, Valencia, Spain, 22–27 August 2004.
20. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
21. John N. Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1–3):59–94, 1996.
22. Thomas J. Walsh, Sergiu Goschin, and Michael L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, Atlanta, US, 11–15 July 2010.